ALGORITMIKA

Rosa Arruabarrena

UDAKO EUSKAL UNIBERTSITATEA
BILBO, 1997

ISBN: 84-86967-82-1
Lege-gordailua: BI-?-96

Inprimategia: BOAN, S.A. Padre Larramendi 2. BILBO

Azala: Iñaki Morlan eta Rosa Arruabarrena

Banatzaileak: UEU. General Concha 25, 4. BILBO telf. 4217145
Zabaltzen: Igerabide, 88 DONOSTIA

Coro eta Prontxio, nire gurasoei, eta Bingeni

AURKIBIDEA

ESKERRAK

Atal honen bitartez eskerrak eman nahi dizkiet liburu honen argitaratzean lagundu didaten guztiei. Fruitu honen hazia Luis Mari Alonso eta Jesus Bermudez irakasleekin "Algoritmika"-ko irakasgaiaren prestaketan igarotako elkarrizketa luzeetatik sortu zen; haziaren ernetzetik fruitua gorpuzterainoko prozesuan Informatika Fakultateko Lengoaia eta Sistema Informatikoak saileko zenbait irakaslek lagundu didate (edukian aurrez aipatutako bi irakasleak; hiztegi espezifikoaren eraketan Xabier Arregi, Xabier Artola, Arantxa Diaz de Illarraza, Kepa Sarasola, Montse Maritxalar, Arantxa Irastorza eta Ana Arruarte irakasleek; Iñaki Pikabeak zalantzaren aurrean sarritan bere aholkua emateagatik; Tomas A. Perez bulegokideak softwareari eta Macintosh-i buruzko hamaika galderatan laguntzeagatik; nire ikasleei ere eskerrak eman nahi dizkiet liburuko lehenengo hurbilketen irakurleak eta aholkulariak izateagatik); fruituaren borobiltzea eta azalaren sorrera testuaren gainbegiraleei (Xabier Arregi, Joseba Makazaga eta Montse Maritxalar) zor diet; fruituaren ugalketa eta banaketa, berriz, UEUk (M. Karmen Menika, Izaro Gurrutxaga eta Nekane Intxaurtza) ahalbidetu du.

AITZINSOLASA

Testu honen helburuak hiru dira:

– Maiz agertzen diren problemak ebazten dituzten algoritmo klasikoen zerrenda bat aztertzea.

– Informatikari askorentzat ezezagunak diren algoritmoen diseinurako oinarrizko teknika desberdinak azaltzea.

– Algoritmoen eraginkortasunaren analisia egiteko behar diren erremintak azaldu eta adibide bidez hornitu. Irakurleak trebetasuna lor dezan lehenengo ariketak ematea. Aldi berean, informatikaria lortutako emaitzen egokitasunari buruz galdetzen ohitzea: Soluzio ona da? Onena? Hoberik topa daiteke?

Helburu horiek lortzeko eta irakurlea eroso senti dadin, gomendagarria litzateke irakurlea:

– programazioan esperientzia edukitzea, konkretuki errekurtsioa eta datu--egituren maneiua menperatuz.

– matematikan jakituna: matematika diskretuan eta kalkulu matematikoan (logaritmoak, limiteak, deribatuak, integralak eta batuketak progresio aritmetiko eta geometrikoak).

– indukzio bidezko frogaketekin ohitua egotea.

Zein irakurle motari zuzendua dagoen eta helburuak zein diren finkatuta, horiek lortzeko liburuko edukinaren antolaketa honakoa izan da:

Lehenengo gaian *Algoritmika* zer den eta bere ikasketak zertarako balio duen jasotzen dut. Motibazio eta sarrera gaia da. Testuan zehar algoritmoen eraginkortasun teorikoa neurtzeko erabiliko ditudan kontzeptuak eta propietateak laburbildu ditut bertan. Algoritmoek kontsumatzen dituzten baliabideen artean, nagusienak eta kontuan hartuko direnak, exekuzio-denbora eta kontsumitutako memoria-espazio estra izango dira.

Bigarrengo gaian ordenazio algoritmo klasikoen bilduma bat dago bi sortatan banatuta: osagaien arteko konparaketa bidezko ordenazio algoritmoak eta

ordenazioa denbora linealean egin dezaketen algoritmoak. Arrazoi desberdinek eragiten dute ordenazio algoritmoen ikasketa. Alde batetik, irakurleak ezagutzen duen problema da eta aurreko gaian azaldutako kontzeptu berriak praktikan jartzeko guztiz aproposa da. Bestalde, problema bera hamaika metodo ezagun jarraituz ebatz daiteke. Hori dela eta, bibliografian eta informatikako liburuetan sarritan agertzen diren metodo edo algoritmoak ikusteko une ezin hobea da. Bestalde, metodo bakoitzak behar duen denbora eta memoria-espazio estra kalkulatu ondoren problema bera ebazteko zein metodo zein baldintzapetan den onena azter dezakegu eta erantzuna arrazoitu. Azkenik, eta aurrekoak bezain garrantzitsua den beste arrazoi bat ere badago: batezbesteko kasuko analisia erabat kalkulagarria duen oso algoritmo gutxi dago. Hauetako algoritmo batzuk gai honetako bilduman daudenez gero, aurreko gaian azaldutako batezbesteko analisia kontzeptua sendotzeko probetxuzkoak gertatzen dira; testuko ondorengo algoritmoentzat analisi mota hau hainbesteko zorroztasunez ezingo da egin.

Hirugarren gaian grafoa datu-mota aztertzen dut. Datu-egitura honek ezinbesteko garrantzia du problema erreal asko grafoak erabiliz modela baitaiteke. Gaian grafo bat emanik hura korritzeko metodo ezberdinak aztertuko ditut: sakonerako korritzea, zabalerako korritzea eta atzera-jotzezko algoritmoak. Metodoek bai grafo zuzenduentzat bai ez-zuzenduentzat balio dute.

Ondorengo gaien filosofia desberdina da. Bertan programatzeko teknikak irakasten dira. Laugarren, bosgarren eta seigarren gaiek metodo bana aurkezten dute: *teknika jalea, zatitu eta irabazi teknika* eta *programazio dinamikoa* teknika hurrenez hurren. Gai bakoitzaren hasieran metodoa bere orokortasunean aurkezten dut. Ondoren, metodoaren aplikazio praktikoak diren adibideak datoz. Adibideen arteko ordena ez da edozein: zerrendaketa adibide errazenetatik zailenetara doa. Irakurleak ez du inolaz ere pentsatu behar teknikak automatikoki eta zuzenean aplikatuz problemari soluzioa emango diola. Aldiz, problemak identifikatu egin behar dira eta metodo orokorrak problema bakoitzari egokitu. Edozein problema ezin da edozein metodo jarraituz ebatzi. Irakurleari lehendabizi metodo orokorrak ondo uler ditzala gomendatzen diot, horretarako adibideetan metodoa nola egokitu den arreta handiz begiratuz. Adibideek ezinbesteko garrantzia dute eta zer esanik ez ariketen ebazpenak.

Benetan problemen ebazpena eraginkorrean interesaturik dagoenari ahalik eta ariketa gehien norbera (edota taldeka) egiten saia dadila aholkatzen diot. Hasieran gogorra badirudi ere, trebetasuna praktikarekin lortzen da. Lehenengo ariketak gai bakoitzaren amaieran erantsitako atalean aurki daitezke. Halere, ehundaka ariketa dago bibliografiako edozein liburutan eta algoritmikako liburuetan.

1. GAIA

OINARRIZKO KONTZEPTUAK

1.1. ERAGINKORTASUN ALGORITMIKOA

1.1.1. Zenbait kontzeptu eta azalpen

⇨ Problema bat ebatzi behar dugunean, bi kasu desberdinen aurrean egon gaitezke:

– problema aztertu ondoren, hura ebatziko duen algoritmoa idatzi behar da: algoritmo baten edo bestearen eraikuntza gure irizpideek eta helburuek mugatzen dute.
– problema ebazten duten zenbait algoritmo ezagutzen dugu: algoritmo desberdinak aztertu eta hoberena aukeratu behar dugu. Baina, zein da hoberena?

Kasu bietan maiz kontrajar daitezkeen jarraiko helburu hauek topa ditzakegu:

– algoritmoa ulerterraza, azkar kodegarria eta zuzengarria izateko nahia: metodologiak.
– algoritmoa eraginkorra izateko nahia: baliabideen erabilera egokia.

Bestalde, ondokoa ere kontutan eduki beharko da:

– Zenbat aldiz erabiliko da algoritmoa?
– Sarreren tamaina nolakoa izango da gehienetan ?

Ondorioz, ondorengo galdera hauek bururatzen zaizkigu:

– Eraginkortasunaren hobekuntzan saiatzeak merezi al du?
– Noiz inbertitu behar da programazio-kostuan?
– Programazio-kostuaren eta exekuzio-kostuaren arteko oreka interesatzen zaigu?

Eraginkortasunean inbertitu nahi denean programazio-kostuan inbertituko dugu, hau da, baliabideen eskasian gaudenean edota haien erabilera egokiak ezinbesteko garrantzia duenean.

⇨ Algoritmo bat edo beste aukeratzean ez dugu ahaztu behar:

– algoritmoa inplementatzen duen programa zenbat aldiz erabiliko den,
– zer tamainako sarrerekin erabiliko den,
– algoritmoa eraginkorra izanik, ulergaitza den,
– algoritmoa azkarra izanik, memoria-espazio handia behar duen (eta alderantziz).

⇨ Merkatura ateratzen diren konputagailuak gero eta azkarragoak badira, zergatik aztertzen dira algoritmoak datorren konputagailu belaunaldira itxaron beharrean?

Adibidez: demagun algoritmo baten inplementazio batek $10^{-8}\ 2^n$ s. hartzen dituela:

Sarrerako parametroen tamaina	Sarrera tamaina horrek beharko duen exekuzio-denbora
n=10	$10^{-8}\ 2^{10}$ s. = 0.1 s.
n=20	10 min.
n=30	>1 egun
n=38	≈1 urte

Eta algoritmoa sarrera handiagoak ebazteko beharko bagenu? Ondorengo belaunaldira itxaronez gero, demagun 100 aldiz azkarragoa den konputagailu bat lortuko genukeela; kasu horretan, gure algoritmoaren inplementazioak $10^{-6}\ 2^n$ s. hartuko lituzke. Eta horrek ez luke aurrerakuntza handirik suposatuko: urtebete osoz egikaritzen utziz n=45 tamainako instantziak besterik ez baitziren ebatziko. Soluzio bakarra: algoritmo azkarragoren bat bilatzea (edo asmatzea).

⇨ Eraginkortasuna behar duen zenbait algoritmo:

• Bilaketa,
• Sailkapena, Ordenazioa (fitxategiak, taulak, egiturak)
• Denbora-errealeko sistemak (uholde, trafiko, zentral elektrikoen kudeaketetan, lurrikaren kudeaketan...)
• DBen atzipena (galderen optimizazioan, galdera-lengoaiak...)
• Sistema Eragileak (Baliabideen kudeaketa...)
• Lehentasun dinamikoko sistemak (hilaren kudeaketa, prozesuen kudeaketa...)
• Lengoaia interpretatuak
• ...

1.1.2. Algoritmoen eraginkortasuna mugatzeko zenbait estrategia

Algoritmoen konplexutasuna aztertzeko hiru estrategia desberdin jarrai ditzakegu:

Enpirikoa (*a posteriori*): makina konkretu batean inplementatu ondoren, probak eginez lortzen diren emaitzak aztertuko ditu.

Teorikoa (*a priori*): sarreraren tamaina kontutan hartuz beharko diren baliabideak matematikoki kalkulatzen ditu aldez aurretik.

Hibridoa

⇨ *Estrategia teorikoak abantaila handiak dakartza:*

1- ez dago konputagailu, programazio-lengoaia ez eta programatzailearen trebetasunaren menpe.
2- algoritmo ez-eraginkorren programazio- eta exekuzio-denbora aurrezten du.
3- eraginkortasuna edozein tamainako intantziarentzat azter dezake .

1.1.3. Algoritmoek darabiltzaten baliabideak

Algoritmo baten eraginkortasuna aztertzeko, algoritmoak behar duen zenbait baliabide aztertu behar da:

– espazioa: memoria gehitzeak exekuzio-denbora jaistea suposatzen du.
– sarreraren tamaina: hau handitzeak exekuzio-denbora gehitzea dakar.
– exekuzio-denbora: faktore desberdinen menpe dago:
  • algoritmoaren sarrera,
  • algoritmoaren konplexutasuna denborarekiko,
  • programak erabiltzen duen espazioa,
  • konpilatzaileak sortutako kodearen kalitatea,
  • erabilitako makinaren aginduen izaera eta abiadura.

Eraginkortasuna aztertzeko garaian, eta kalkuluak gehiago ez zailtzearren, lehenengo bi faktoreak bakarrik izango ditugu kontutan (hala ere, zenbait kasu konkretutan hirugarrena ere hartuko da).

⇨ *Sarreraren tamaina:*

Parametro errealen tamaina da. Problema desberdinetan parametroen tamainatzat gauza desberdinak hartzen dira:

– zerrendekin, fitxategiekin: instantziaren osagai kopurua,
– zenbaki handiekin: instantziak adierazteko behar diren biten kopurua,
– ...

Exekuzio-denbora sarreraren tamainaren funtzio bezala neurtuko da.

1.1.4. Inbariantzaren printzipioa. Eraginkortasunaren unitatea

Inbariantzaren printzipioa:

> *"Algoritmo baten bi inplementazio desberdinen eraginkortasunen arteko diferentzia konstante biderkatzaile batean datza."*

$$T_1(n) \leq k \cdot T_2(n) \qquad \text{n handia denean}$$

Printzipio honek ondorio garrantzitsu batzuk dakartza:

– edozein konputagailu eta programatzaileren trebetasunetarako balio du.
– eraginkortasunaren hobekuntza algoritmoaren aldaketa baten bidez bakarrik lortuko da.
– algoritmoen eraginkortasun teorikoak EZ du UNITATERIK, konstante biderkatzaile baten funtziopean ematen baita.

1.1.5. Algoritmoen ordena

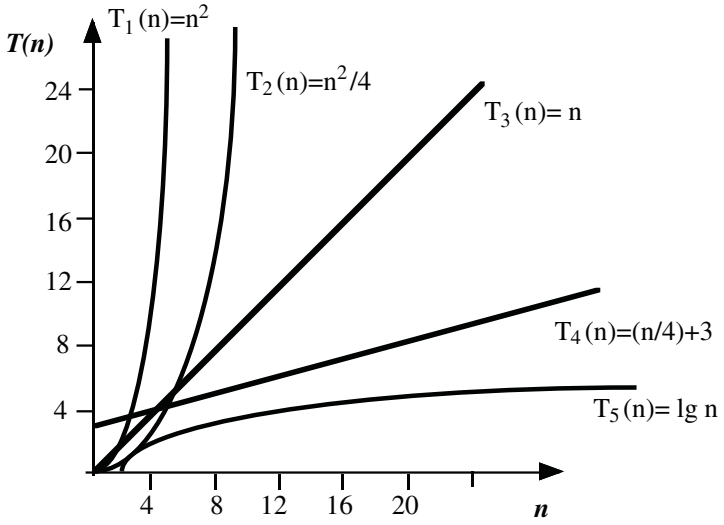
T(n)-ren bidez, sarrera *n* tamainakoa duen algoritmo baten *exekuzio-denbora* adieraziko da. Algoritmoak hartzen duen *denbora* **f(n)-ren ordenakoa** dela esango da baldin eta soilik baldin k konstanteak existitzen badu, non instantzia bakoitzeko problema $k * \mathbf{f(n)}$ segundotan (μsegundotan, minututan...) gehienez ebazten baita. *k* konstanteari konstante biderkatzailea deritzo.

Segundotan dela esatea erabat arbitrarioa da konstantearen balioa aldatzea besterik ez baita beharko unitatea beste bat izateko. Bestalde, gaur egun makina estandarrik ez dagoenez, ezingo da bat eredutzat hartu (ik. RAM). Honen guztiaren ondorioz, eraginkortasunak unitaterik ez duela esan dezakegu.

⇨ *Maiz agertzen diren ordenak:*

n -ren ordenakoa	→ algoritmoa	*lineala*	da
n^2		*koadratikoa*	
n^3		*kubikoa*	
n^k		*polinomikoa*	
a^n		*esponentziala*	

non aurreko *k* eta *a* konstante egokiak diren.



[1.1] irudia

[1.1] irudiko T_1 eta T_2 funtzioak koadratikoak dira, T_3 eta T_4 linealak eta T_5, berriz, logaritmikoa.

⇨ *Konstante biderkatzaileak:*

Demagun problema bera ebazten duten bi algoritmok n^2 egun eta n^3 s. behar dituztela *n* tamaina duten sarrerak ebazteko. *n* tamaina duten instantzietarako teoriak lehenengo algoritmoa bigarrena baino azkarragoa, hobea, eraginkorragoa dela adierazten du:

Algoritmo koadratikoa kubikoa baino ASINTOTIKOKI ERAGINKORRAGOA da.

Teoriak hori badio ere, koadratikoa:

20 miloi urte baino denbora gehiago beharko luketen
sarreretarako soilik izango litzateke eraginkorragoa!

Zenbait kasutan konstante biderkatzailea kontutan ez hartzeak bere arriskua du.

Halere, ez dira kontutan izango irakasgai honetan, konstanteak mugatzeko inplementazio bakoitzaren xehetasun espezifikoetan sartu beharko genukeelako. Hortaz, sarrerako instantzien tamainak behar bezain handiak izango direla suposatuko dugu.

HELBURUA: algoritmo jakin baten ordena kalkulatzea beste algoritmo batzuren ordenekin konparatu ahal izateko non beste algoritmo hauek ere problema

bera ebazten duten. Honela eraginkorrena zein den muga dezakegu (azkarrena, memoria kudeaketan hobena; oro har, baliabideen kudeatzaile onena mugatu dezakegu.

1.1.6. RAM konputagailu eredua eta oinarrizko eragiketa

⇨ *RAM (Randon Access Machine) eredua:*

Gure algoritmoen denbora-ordena, eta oro har ordena, kalkulatzeko haien inplementazioak zein makina motan egikarituko liratekeen kontutan hartu beharko litzateke. Konputagailu estandarrik ez dagoenez, haiek helburu orokorreko prozesadore bakarra duen makina batean egikarituko liratekeela suposatuko dugu irakasgai honetan. Konputagailua RAM eredua izango da. Eredu honetan aginduak bata bestearen jarraian egikarituko dira, eta ez da eragiketa konkurrenterik kontsideratuko.

⇨ *Oinarrizko eragiketa:*

Oinarrizko eragiketa bere exekuzio denbora konstante batez mugaturik dagoen eragiketa da, non konstante hori *inplementazio konkretu* baten menpe dagoen.

Oinarrizko eragiketak izango dira:
– osokoen gaineko eragiketak: batuketa, kenketa, zatiketa, biderkaketa,
– koma higikorrezko zenbakien gainekoak,
– konparaketak, asignazioak.

Kontuz! Oso handiak diren osokoen maneiua, adibidez, ezin da oinarrizko eragiketa kontsideratu. Ondorioz, eragiketa bat oinarrizkoa den ala ez mugatzeko sen ona erabili beharko dugu: eragigaien tamaina arrazoizkoa den bitartean eragiketa oinarrizkoa kontsidera daiteke.

1.2. NEURRI ASINTOTIKOAK, HAZKUNTZA-TASAK ETA PROPIETATE BATZUK

1.2.1. T(n)-A. $T_b(n)$ eta $T_t(n)$

⇨ T(n). Zer da?

> *T(n)* *n* tamaina duen sarrera batentzat algoritmoak hartzen duen exekuzio-”denboraren” neurketa da.

HOTS:

• Unitateak zehaztu gabe daude.

• Neurketa funtzioa da. Zehazki mintzatuz, konputagailu ideal batek, RAMek, exekuzio garaian egindako lana neurtuko du.

T(n)-ren kalkulua ondorengo bi puntu hauetan oinarritzen da:

a) oinarrizko eragiketa bakoitza exekutatzen den aldi kopuruan (maiztasuna)

b) oinarrizko eragiketa bakoitzak hartzen duen denboran (konstantea definizioz).

Aurreko a eta b informazioen biderkaketak behar den exekuzio-denbora osoa ematen du.

Halere, T(n) neurketa funtzioa dela eta, zenbait kasutan exekuzio-denbora neurtu ordez beste baliabideen kontsumoa kontrolatzea interesatzen zaigu.

• *n* tamaina duen sarrera batentzat egin ohi diren beste neurketa tipiko batzuk honako hauek dira:

– exekutatu beharko dituen oinarrizko eragiketen zenbaketa:

– sarrerako osagaien artean egiten diren alderaketen kopuruaren zenbaketa,

– egiten diren dei errekurtsiboen zenbaketa,

– ...

Analisi kasu bakoitzean interesaturik gauden horren neurketa egingo dugu. Gauza desberdinak neurtzeko erabiliko denez, kasu bakoitzean zer neurtzen ari garen esan behar da.

Adibidez, demagun ondorengo programen zatiak ditugula:

```
X:= X+Y;
Y:= 3*Y+8;
```

```
for I in 1.. N loop
            X:= X+Y;
end loop;
```

```
for I in 1 .. N  loop
  for J in 1 .. N loop
    X:= X+Y;
  end loop;
end loop;
```

1) T(n) = zein da denbora-ordena? = (Zenbat aldiz egikaritzen da oinarrizko E eragiketa bakoitza) $*$ (Zenbat denbora hartzen du E oinarrizko eragiketak)

$T(n)= k_1 + k_2 = k_3$	$T(n)= k_1 * n$	$T(n)= k_1 * n^2$
1en ordenakoa	ordena lineala	ordena koadratikoa
(ordena konstantea)		

2) T(n)= zenbat batuketa egikaritzen da?

T(n)= 2	T(n)= 1 ∗ n	T(n)= 1 ∗ n^2
1en ordenakoa	ordena lineala	ordena koadratikoa

Ordenen interpretazioa zuzen egin behar da. Beti zer neurtzen ari garen argi eduki behar dugu.

⇨ *Batezbesteko kasua eta kasu txarrenaren analisia:* $T_b(n)$ *eta* $T_t(n)$.

Normalki, algoritmo batek tamaina desberdineko sarrerentzat hartzen dituen exekuzio-denborak desberdinak izatea gertatzen bada, tamaina bereko bi instantzia desberdinarentzat ere gauza bera gertatu ohi da. Kasu hauetan, algoritmoa sarreraren egoerarekiko sentikorra dela esaten da.

Algoritmo berdin baten eraginkortasuna hainbat kasu desberdinetan azter dezakegu: T(n) neurketa funtzioak exekuzio-denbora neurtzen duela suposatzen dugunean.

Kasu txarrenak denbora gehien hartzen duen tamaina bateko instantziak bakarrik aztertuko ditu. Noiz erabiliko dugu analisi hau? Erantzun--denborak garrantzi handia duenean erabiltzea interesgarria litzateke. Idazkera: $T_t(n)$ eta zalantzarik ez badago T(n).

Batezbesteko kasua: Algoritmoak oso instantzia desberdinekin erabiltzen badira beste analisi hau interesgarria gerta liezaguke.
$T_b(n)$: *n* tamaina duten sarrera guztien batezbesteko exekuzio-denbora adieraziko du.

Normalki batezbesteko kasua kalkulatzea zaila gertatzen da sarreren banaketa zein den aldez aurretik jakin behar baita eta sarrera guztien probabilitateek ez baitute zertan berdinak izan behar.

1.2.2. O(n), Ω(n) eta Θ(n) hurbilketa asintotikoak

f: N → R* emanik:

<u>O larria</u>

$$\mathbf{O(f(n))} = \{\ t: N \to R^* \mid \exists k\ \exists n_0\ (\ (k \in R^+) \wedge (n_0 \in N) \wedge \forall n\ (n \geq n_0 \Rightarrow\ T(n) \leq k\ f(n))\ )\ \}$$

O(f(n)) -"**f(n) ordena**" - *n* balio <u>handietarako</u> f(n) funtzioen multiplo erreal batek goitik mugatutako T(n) funtzio-multzoa da; hau da, atalase-balio batetik aurrera.

Oro har, gure helburua f(n) funtzio guztietatik goi-bornerik txikiena lortzea izango da, non $T(n) \in O(f(n))$.

Omega larria: Ω

$$\Omega(f(n)) = \{ t: N \to R^* \mid \exists k\ \exists n_0\ (\ (k \in R^+) \wedge (n_0 \in N) \wedge \forall n\ (n \geq n_0 \Rightarrow T(n) \geq k\ f(n))\ )\ \}$$

Ω (f(n)) - "**f(n) Omega**" - *n* balio handietarako f(n) funtzioen multiplo erreal batek behetik mugatutako T(n) funtzio-multzoa da; hau da, atalase-balio batetik aurrera.

Oro har, gure helburua f(n) funtzio guztietatik behe-bornerik handiena lortzea izango da, non $T(n) \in \Omega(f(n))$.

Teta larria: Θ

$$\Theta(f(n)) = \{t: N \to R^* \mid \exists k\ \exists d\ \exists n_0\ (\ (k \in R^+) \wedge (d \in R^+) \wedge (n_0 \in N) \wedge \forall n\ (n \geq n_0 \Rightarrow k\ f(n) \leq T(n) \leq d\ f(n))\ )\ \}$$

Ordena zehatza honela ere defini daiteke:

$$f(n) \in \Theta(g(n)) \Leftrightarrow f(n) \in \Omega(g(n)) \text{ y } f(n) \in O(g(n)),$$
$$\Theta(f(n)) = O(f(n)) \cap \Omega(f(n))$$

1.2.3. Hazkuntza-tasak. L'Hopitalen erregela

Maizen agertzen diren bornatze funtzioak konplexutasun ordena gorakorrean:

$$O(1) \subset O(\lg n) \subset O(n) \subset O(n \lg n) \subset O(n^2) \subset O(n^3) \subset \ldots \subset O(n^k) \subset O(a^n) \subset O(n!) \subset O(n^n)$$
$$\text{y } O(k\ f(n)) = O(f(n))$$

HOTS:
n guztientzat, a eta b konstanteentzat eta a>1 betetzen denean funtzio esponentzialen hazkuntza-tasa polinomialarena baino handiagoa da; hau da, $\lim_{n\to\infty} \frac{n^b}{a^n} = 0$ (edo beste modu batean $n^b \in o(a^n)$).

⇨ *Algoritmo desberdinen artean algoritmo eraginkorrenaren muga:*

Zenbaitetan, algoritmo desberdinen exekuzio-denboren funtzioak konparatzea komenigarria izaten da, alderatzen diren guztietatik eraginkorrena zein den

mugatzearren. Alderaketa egin ahal izateko bai L'Hopital-en Erregela bai eta limiteak ere erabil ditzakegu:

f,g: N → R* suposatuz:

$$1)\ \lim_{n\to\infty}\frac{f(n)}{g(n)}\in R^{+}\Rightarrow O\big(f(n)\big)=O\big(g(n)\big)$$

$$2)\ \lim_{n\to\infty}\frac{f(n)}{g(n)}=O\Rightarrow O\big(f(n)\big)\subset O\big(g(n)\big)$$

3) L'Hopitalen Erregela:

Baldin

a) $\lim_{n\to\infty} f(n) = \lim_{n\to\infty} g(n) = 0$, edo

$\lim_{n\to\infty} f(n) = \lim_{n\to\infty} g(n) = \infty$ betetzen bada,

b) $n \in [n_0, \infty)$ → $x \in [n_0, \infty)$
$f(n), g(n)$ $f^*(x), g^*(x)$
zenbaki arruntetatik zenbaki errealetara heda daiteke,

c) $f^{*\prime}(x)$ eta $g^{*\prime}(x)$-k tarte berri honetan existitzen badute ($f^*(x)$ eta $g^*(x)$ deribagarriak $[n_0, \infty)$ tartean), eta

$$d)\ \exists \lim_{x\to\infty}\frac{f^{*\prime}(n)}{g^{*\prime}(n)}$$

orduan

$$\lim_{x\to\infty}\frac{f(n)}{g(n)}=\lim_{x\to\infty}\frac{f^{*\prime}(n)}{g^{*\prime}(n)}$$

1.3. ALGORITMO ITERATIBOEN ANALISIA

Agindu baten edo agindu multzo baten exekuzio-denbora sarreraren tamainaren eta aldagai batzuren menpe badago ere, sarreraren tamaina izango dugu bakarrik kontuan.

⇨ *Algoritmo baten eraginkortasunaren analisia aztertzeko faseak.*

1) Algoritmoak behar dituen datu-multzoak finkatu,
2) Erabiltzen diren oinarrizko eragiketak finkatu,
   (oinarrizko eragiketa bakoitzaren exekuzio-denbora konstantea da)
3) Oinarrizko eragiketa bakoitza exekutatzen den aldi kopurua finkatu,
4) Beste eragiketak eta beraien kostua finkatu, eta
5) Aurreko emaitzak erabiliz algoritmo osoaren kostua mugatu.

Eragiketen kostua nola mugatzen da?.

⇨ *Erregela orokorrak*

1 Parametroek sarreraren tamaina ematen dute,

2 Asignazioa, alderaketa eta sarrera/irteera eragiketen denbora 1en ordenakoak dira; oro har, oinarrizko eragiketak.
Salbuespenak: array-en tamaina sarrerako tamainaren funtziopean dagoenean...

3 Agindu-sekuentzia baten denbora baturaren erregelaren bidez lortuko da,

4 IF-THEN aginduaren exekuzio-denbora:
THENen ataleko aginduen denbora **+** baldintzaren denbora (normalki, 1en ordenakoa).

IF-THEN-ELSE aginduen exekuzio-denbora:
Baldintzaren denbora **+ max** { THENen ataleko aginduen denbora,
ELSEren ataleko aginduen denbora }

5 Iterazio baten exekuzio-denbora:
$(T_{gorputza} + T_{konparaketa}) * \text{iterazio-kopurua}$

6 Azpiprograma-deiak (ez-errekurtsiboak): azpiprogramaren exekuzio--denbora + parametro pasatzeak hartzen den denbora.

Algoritmo errekurtsiboen exekuzio-denbora eta ordenaren kalkulua ez da horrela kalkulatzen.

⇨ *Bi erregela berezi*

<u>Baturaren erregela</u>

• $T_1(n)$ eta $T_2(n)$ P_1 eta P_2 bi programa-atalen exekuzio-denborak direlarik hurrenez hurren, eta
• $T_1(n) \in O(f(n))$, $T_2(n) \in O(g(n))$ emanik,

P_2 programa P_1en jarraiean jartzean lortzen den programaren exekuzio--denbora, $\mathbf{T_1(n)+T_2(n)}$, **max{ f(n), g(n) }**en ordenakoa izango da. Hau da,

$$\mathbf{T_1(n) \in O(f(n)) \ \wedge \ T_2(n) \in O(g(n)) \Rightarrow T_1(n) + T_2(n) \in O(max(f(n),g(n)))}$$

$$O(f(n)+g(n)) = O(max(f(n),g(n)))$$
$$\Theta(f(n)+g(n)) = \Theta(max(f(n),g(n)))$$
$$\Omega(f(n)+g(n)) = \Omega(max(f(n),g(n)))$$

Biderkaketaren erregela

• $T_1(n)$ eta $T_2(n)$ P_1 eta P_2 bi programa-atalen exekuzio-denborak direlarik hurrenez hurren, eta
• $T_1(n) \in O(f(n))$, $T_2(n) \in O(g(n))$ emanik,

P_2 programa P_1tik $T_1(n)$ aldiz dei egiten bazaio programa osoak hartzen duen denbora $\mathbf{T_1(n)*T_2(n)}$ da eta exekuzio ordena aldiz $\mathbf{O((f(n)\bullet g(n)))}$. Hau da,

$$\mathbf{T_1(n) \in O(f(n)) \ \wedge \ T_2(n) \in O(g(n)) \Rightarrow T_1(n) * T_2(n) \in O((f(n)\bullet g(n)))}$$

$$O(f(n)*g(n)) = O((f(n)\bullet g(n)))$$

1.4. ALGORITMO ERREKURTSIBOEN ANALISIA

Ondorengo pausoak eman behar dira:

1) Algoritmoaren konplexutasuna errekurtsio-ekuazio baten bidez adierazi.
2) Errekurtsio-ekuazioa metodoren bat aplikatuz ebatzi.
   Bi metodo ikusiko dira: errekurtsibitatearen hedapen metodoa eta ekuazio karakteristikoaren metodoa. Funtzio-sortzaileen metodoa ez dugu hemen aztertuko.

Exekuzio-denbora errekurtsiboa den ekuazio baten bidez adierazten da: **errekurtsio-ekuazioa.** Metodo hauen helburua errekurtsio-ekuazioaren beste ekuazio baliokide bat, errekurtsiboa ez dena, bilatzean oinarritzen da. Ondoren, ekuazio berri honi aurretik ikusi diren erregelak aplikatuko zaizkio algoritmo errekurtsiboaren ordena lortuz.

Oharra: algoritmoaren dei errekurtsiboek egindako lana lortu nahi da: zenbat dei egiten den, edo zenbat aldiz egikaritzen den eragiketa jakin bat, edo...

1.4.1. Errekurtsibitatearen hedapen metodoa

Kasu orokorreko ekuazioan agertzen diren t_n-ak hedatu egiten dira kasu nabarira iritsi arte. Kasu nabaria garatu ondoren, ekuazio berritik algoritmo errekurtsiboaren ordena lortuko da.

Adibidea: $T(n) = 2\,T(n-1)+ 2\ \ a \qquad n>1$ (kasu nabaria $T(0) = b$)

$T(n) = 2\,T(n-1)+ 2\ \ a =$

$= 2\,(2\,T(n-2)+ 2a) + 2a = 2^2\,T(n-2)+ 2^2a + 2\,a =$

$= \ldots$ indukzio bidez

$= 2^i\,T(n-i)+ 2^ia + \ldots +2^2a + 2\,a =$ $n - i = 0$

$$= 2^n T(0) + a\sum_{m=1}^{i} 2^m = 2a\left(2^i - 1\right) = (b + 2a)2^n - 2a \ \in\ O(2^n)$$

⇨ *Aldagai aldaketaren metodoa:*

Aurrekoaren kasu berezia da. Zenbaitetan errekurtsio-ekuazioaren hedapenak aldagai aldaketa bat egitea eskatzen du kasu nabarira iritsi ahal izateko.

Adibidea: $T(n) = 4\,T(n/2) + n \qquad n>1$ emanik, metodoa aplikatuz:

$T(n) = 4\,T(n/2) + n = 4\,(\,4\,T(n/2^2) + n/2) + n = 4^2\,T(n/2^2) + 3\,n \qquad n>2$

$= 4^2\,(4\,T(n/2^3) + n/2^2\,) + 3\,n = 4^3\,T(n/2^3) + 7\,n \qquad n>3$

$= 4^3\,(4\,T(n/2^4) + n/2^3\,) + 7\,n = 4^4\,T(n/2^4) + 15\,n \qquad n>4$

$= \ldots$ indukzio bidez

$= 4^i\,T(n/2^i) + (2^i\text{-}1)\,n \qquad n>i$

Aldagaiaren aldaketa bat eginaz: $n=2^k$ eta $i=k$, gero $k=\log_2 n$, kasu nabariraino iristea lortzen da:

$T(n) = 4^k\,T(n/2^k) + (2^k\text{-}1)\,n = 2^{2k}\,T(n/2^k) + (2^k\text{-}1)\,n = n^2\,T(\underline{1}) +$
$(n-1)\,n = d\,n^2 + n^2 - n = (d+1)\,n^2 - n \qquad \Rightarrow \quad T(n) \in O(n^2)$

1.4.2. Ekuazio karakteristikoaren metodoa

Ondorengo kasuak aztertuko ditugu:

a) Errekurtsio homogeneoak. Honen barnean beste bi kasu dauzkagu:
   – erro guztiak desberdinak – zenbait erro berdinak

b) Errekurtsio ez homogeneoak. Berriz ere aurreko bi kasuak:
   – erro guztiak desberdinak – zenbait erro berdinak

a) Errekurtsio homogeneoak

Errekurtsio-ekuazioak jarraiko itxura dauka:

$a_0\,t_n + a_1\,t_{n-1} + \ldots + a_k\,t_{n-k} = 0$

Non t_i: bilatu nahi diren balioak diren, eta $a_0,a_1,\ldots,a_k$: konstanteak
(beste izen batez ere ezaguna da aurreko ekuazioa: *diferentzietan lineala den ekuazioa*.)

$t_n = X_n$ ordezkapena eginaz, (non X konstante bat den):

$$a_0 X_n + a_1 X_{n-1} + ... + a_k X_{n-k} = 0$$

Ondoren, n=k eginaz:

$$a_0 X_k + a_1 X_{k-1} + ... + a_k = \mathbf{0}$$ EKUAZIO KARAKTERISTIKOA

Ekuazio hau ebazteko, bere erroak besterik ez dira bilatu behar: $r_1, r_2, ..., r_k$

– Erro desberdinak:

Exekuzio-denboraren ekuazio berria: $t_n = \sum_{i=1}^{n} k_i r_i^n$

– Zenbait erro berdinak:

Exekuzio-denboraren ekuazio berria: $t_n = \sum_{i=1}^{d} \left(\sum_{j=0}^{m_i - 1} k_{ij} n^j \right) r_i^n$

non

$r_1, r_2, ..., r_d$: erro desberdinak diren, eta

m_i: r_i erroaren anizkoitasuna izango den (errepikatzen den aldi kopurua).

b) Errekurtsio ez homogeneoak

Errekurtsio-ekuazioak ondorengo itxura dauka:

$$a_0 t_n + a_1 t_{n-1} + ... + a_k t_{n-k} = \mathbf{(b_1)^n p_1(n) + (b_2)^n p_2(n) + ...}$$

non b_i: konstanteak diren, eta

$p_i(n)$: g_i mailako polinomioak.

Kasu honetan ebatzi behar dugun ekuazio karakteristikoa ondorengo hau da:

$$(a_0 X_k + a_1 X_{k-1} + ... + a_k) (X - b_1)^{g1+1} (X - b_2)^{g2+1} ... = 0$$

EKUAZIO KARAKTERISTIKOA

Ekuazio honen erro guztiak bilatu ondoren, errekurtsio homogeneoetan bezala jokatu behar da (erro guztiak desberdinak badira formula mota bat erabiliko da, eta erroren bat behin baino gehiagotan agertzen bada beste formula bat erabiliko da).

⇨ Konstanteen muga

Demagun algoritmo errekurtsibo batetik honako ekuazio hau lortu dugula:

$$t(n)=\begin{cases}0 & n=0 \vee n=1\\ 1+t(n-1) & n\geq 2\end{cases}$$

Ekuazio karakteristikoaren metodoa aplikatu ondoren t(n)= k1 +k2 n lortuko dugu. Ondoren, konstanteak mugatu beharko ditugu. Horretarako jarraitu beharko dugun prozesua ondokoa izango da:

– Definizio errekurtsiboa zenbat gai errekurtsiboren funtziopean definitzen den ikusi: a_k gai balitu, a_k kasu nabari beharko genituzke. Gure adibideak bakarra duenez, kasu nabari batekin nahiko litzateke, nahiz eta bi eduki (n=0 eta n=1).

– Ekuazio karakteristikoan dagoen konstante kopurua ikusi: b_k balitu, b_k ekuazioz osatutako sistema lineala ebatzi beharko litzateke; hots, ekuazio bat, konstante-inkognita bakoitzeko. Gure adibidean, $t(n)=K_1+K_2n$, bi ditugunez, 2 ekuazio beharko ditugu.

– Ekuazioak idazteko:

- Errekurtsio-ekuazioa n-ren zein kasutarako definitzen den ikusiz (demagun n≥c dela), errazena n-ren balio horretatik hastea da. Gure kasuan, definizio errekurtsiboa n≥2rentzat definituta dagoenez eta 2 ekuazio behar ditugunez (K_1 eta K_2 bi inkognitak ditugu eta), n=2 eta n=3 balioak erabiliko ditugu.

- Errekurtsio-ekuazioan n-ren ordez aurreko puntuan erabakitako balioak ordezkatuz kalkulatu behar diren konstante-inkognita haina balio lortuko dugu. Balio hauei oinarrizko kasuak deritzaie. Gure kasuan:

  t(2)= 1+ t(1)= 1+0=1
  t(3)= 1+ t(2)= 1+1=2

  Oinarrizko kasuak kalkulatzeko t(n)ren kasu nabariak erabili behar dira maiz. Gure adibideko kasu nabariak n=0 eta n=1 dira, baina n=1 behar dugu.

  Bestalde, posiblea eta zuzena litzateke n-ren balio hain txikiak hautatu beharrean beste batzuk hartzea, adibidez gure adibidean n=5 eta n=8. Baina, t(4) eta t(7) balioak ezagunak izatea eskatuko digute t(5) eta t(8) ekuazioak, hurrenez hurren, honek lan gehiago suposatuko duelarik.

- Ondoren, oinarrizko kasuetako balioak eta ekuazio karakteristikoko metodoan lortutako ekuazioan n-ren balio egokiak ordezkatu ondoren lortzen diren ekuazioak berdindu behar dira. Adibidearekin jarraituz:

$n=2\ (1) \quad 1= k_1+2\ k_2$
$n=3\ (2) \quad 2= k_1+ 3\ k_2$

- Bukatzeko, ekuazio-sistema ebatzi behar da metodo ezagunen bat erabiliz. Hala nola, matrize bidezko ebazpena, aldagai ordezkapena, aldagai berdinketa edo sinplifikazio metodoa erabil dezakegu. Lortzen diren balioak soluzioan ordezkatu behar dira ondoren:

$k_1 = -1 \quad k_2= 1$

Kontu handiz egin behar da prozesu hau. Oinarrizko kasuek ezin dute edozein izan. Hemen irakurleari txikienak kalkulatzea gomendatzen zaio, kalkulatzeko errazenak dira eta. Hala eta guztiz ere, oinarrizko kasuetan erabiltzen diren kasu nabariak ere oinarrizko kasu onargarriak dira. Honela, adibidean, n=1 eta ondorioz t(1) erabil genezake. Horrela egin izan bagenu emaitza bera lortuko genuke:

$n=1\ (1)$	$t(1)=0= k_1+ k_2$	$k_1= -1$	$t(n)= n-1$
$n=2\ (2)$	$t(2)=1= k_1+2\ k_2$	$k_2= 1$	

Berriz ere, kontuz! n=0 kasu nabaria oinarrizko kasutzat hartu izan bagenu, emaitza okerrak lortuko genituzke eta:

$n=0\ (1)$	$t(0)=0= k_1$	$k_1= 0$	$t(n)= k_1+k_2\ n=\mathbf{0}$!!!
$n=1\ (2)$	$t(1)=0= k_1+ k_2$	$k_2= 0$	

1.5. OINARRIZKOA DEN ZENBAIT KONTZEPTU MATEMATIKOREN ZERRENDA

Esponentzialak ($a\neq0$)

$a^0 = 1$	$(a^m)^n = a^{mn}$
$a^1 = a$	$(a^m)^n = (a^n)^m$
$a^{-1} = 1/a$	$a^m a^n = a^{m+n}$

Logaritmoak ($\forall a,b>0$)

1 - $\log_b$ funtzioa zuzenki gorakorra da (baldin $X_1>X_2$ orduan $\log_b X_1 > \log_b X_2$)
2 - $\log_b 1 =0$
3 - $\log_b b^a = a$
4 - $\log_b{}^k n = (\log_b n)^k$

5 - $\log_b (X_1{*}X_2) = \log_b X_1 + \log_b X_2$
6 - $\log_b (X_1/X_2) = \log_b X_1 - \log_b X_2$
7 - $\log_b (X^a) = a \log_b X$

8 - $X_1 \log_b X_2 = X_2 \log_b X_1$

9 - $\log_{b1} X = (\log_{b2} X) / \log_{b2} b_1$

10 - $\log_b (1/n) = -\log_b n$

11 - Guretzat eta idazkera sinplifikatzearren: $\log_2 X_1 = \lg_2 X$

$$n \le 2^{\lceil \lg n \rceil} < 2n \qquad \frac{n}{2} < 2^{\lfloor \lg n \rfloor} \le n$$

10.- $\lg e = 1.44$, $\ln 2 = 0.7$ eta $\lg 10 = 3.32$

Stirling-en hurbilketa

$$n! = \sqrt{2\prod n}\left(\frac{n}{e}\right)^n \left(1+\Theta\left(\frac{1}{n}\right)\right) \qquad \sqrt{2\prod n}\left(\frac{n}{e}\right)^n \le \sqrt{2\prod n}\left(\frac{n}{e}\right)^{n+\left(\frac{1}{12n}\right)}$$

Probabilitatea

Demagun egoera batean ekintza edo esperimentu batek $s_1, s_2, ..., s_k$ k irteeren arteko edozein eman dezakeela. s_i irteera bakoitzari zenbaki erreal bat erlazionatzen diogu $p(s_i)$, non honi s_i-ren probabilitatea deritzogun eta ondorengo hau betetzen duen:

1.- $\forall i (1 \le i \le k \Rightarrow 0 \le p(s_i) \le 1)$
2.- $p(s_1)+...+p(s_k)=1$

Batuketa-formulak

1.- Jarraiko zenbaki osokoen batuketa:

$$\sum_{i=a}^{b} i = \frac{(a+b)(b-a+1)}{2} \qquad \sum_{i=1}^{n} i = \frac{(1+n)n}{2}$$

2.- Karratuen batura:

$$\sum_{i=1}^{n} i^2 = \frac{2n^3 + 3n^2 + n}{6}$$

3.- 2ren potentziak:

$$\sum_{i=a}^{b} 2^i = 2^{b+1} - 2^a$$

4.- Batuketa geometrikoak (progresio geometrikoak):

$$\sum_{i=0}^{k} \frac{1}{2^i} = 2 - \frac{1}{2^k} \qquad \sum_{i=0}^{k} a^i = \frac{a^{k+1} - 1}{a - 1}$$

5.- Nahasketak:

$$\sum_{i=0}^{k} i2^i = (k-1)2^{k+1} + 2$$

Batuketak integralak erabiliz

1.- Demagun *f* funtzioa jarraia eta beherakorra eta *a* eta *b* zenbaki osokoak. Orduan:

$$\int_{a}^{b+1} f(x)dx \leq \sum_{i=a}^{b} f(i) \leq \int_{a-1}^{b} f(x)dx$$

2.- Demagun *f* funtzioa jarraia eta gorakorra eta *a* eta *b* zenbaki osoak. Orduan:

$$\int_{a-1}^{b} f(x)dx \leq \sum_{i=a}^{b} f(i) \leq \int_{a}^{b+1} f(x)dx$$

1.6. APLIKAZIO PRAKTIKOA: ELEMENTU BATEN BILAKETA TAULA BATEAN

1) Suposa dezagun N luzera duen taula bat dugula *(taula ez dago sailkaturik)*. X balioa emanik, algoritmo batek X balio hori dagoen taulako posizioaren indizea itzultzen du, eta X taulan ez badago, berriz, 0 itzultzen du.

   Algoritmo honek zenbat alderaketa egingo du *X* balioarekin ?
   a - kasu txarrenean, eta
   b - batezbesteko kasuan.

2) Suposa dezagun *N* luzera duen eta *sailkaturik* dagoen taula bat dugula. *X* balioa emanik, algoritmo batek *X* balio hori dagoen taulako posizioaren indizea itzultzen du, eta *X* taulan ez badago, berriz, 0 itzultzen du.
   Algoritmo honek zenbat alderaketa egingo du *X* balioarekin?
   a - kasu txarrenean, eta
   b - batezbesteko kasuan.

3) Aurreko bi ariketetan taulako osagaien arteko alderaketen kopurua zenbatzen da, agindu bakoitzak behar dun denbora batu ordez. Hori egitea algoritmoak egindako lanaren erakusle ona al da? Zer esanahi du honek?

4) Algoritmo batek *Bilaketa Dikotomikoa* metodoarekin aurreko arazo bera ebazten du. Alderaketen kopurua kaxkarragoa dela badakigu, baina zein da zehazki kopuru hori?
   a - kasu txarrenean, eta
   b - batezbesteko kasuan.

ARIKETAK: OINARRIZKO KONTZEPTUAK

1- Jarraiko espresioak egiazkoak ala faltsuak diren adierazi. Arrazoitu erantzuna:

a) $n^2 \in O(n^3)$
b) $n^3 \in O(n^2)$
c) $2^{n+1} \in O(2^n)$
d) $(n+1)! \in O(n!)$
e) $T(n)= 3n^3 + 2n^2 \in O(n^3)$
f) $T(n)= 3^n \notin O(2^n)$
g) $\log_a n! \in \Theta(n \log_a n)$
h) $\forall a,b\ (a,b >1 \Rightarrow \log_a n \in \Theta(\log_b n))$
i) $\forall f\ (\ f: N \to R^+ \Rightarrow\ (\ f(n) \in O(n) \Rightarrow (f(n))^2 \in O(n^2))\ )$
j) $\forall f\ (\ f: N \to R^* \Rightarrow\ (\ f(n) \in O(n) \Rightarrow\ 2^{f(n)} \in O(2^n))\ )$
k) Izan bitez $t(n)= 5n+10$ eta $f(n)= n^2+1$ bi funtzio.
k=1, *k*=2 edo *k*=3 hartzen baditugu, $\forall n\ (n \geq n_0 \Rightarrow t(n) \leq k\ f(n))$ bete dadin, zein dira hurrenez hurreneko n_0 atalase-balioak? Aurki bedi behar bezain handia den *k*-ren balio bat, M, ondorengoa bete dadin: $\forall n\ (n \in N \Rightarrow t(n) \leq M\ f(n))$
l) Izan bitez $t(n)= n+1$ eta $f(n)= n^2$ bi funtzioak. *k*=1 edo *k*=2 hartzen baditugu, zein dira hurrenez hurreneko n_0 atalase-balioak $\forall n\ (n \geq n_0 \Rightarrow t(n) \leq k\ f(n))$ bete dadin? Posiblea litzateke behar bezain handia litzatekeen *k*-ren balio bat aurkitzea, M, $\forall n\ (n \geq n_0 \Rightarrow t(n) \leq k\ f(n))$ propietatea beteko lukeena?

2- Frogatu bedi jarraiko O(f(n))ren definizioa aurrez emandakoaren baliokidea dela:

$$\mathbf{O(f(n))} = \{\ t: N \to R^* \mid \exists k\ (\ (k \in R^+) \wedge \forall n_0\ (\ n_0 \in N \Rightarrow \forall n\ (n \geq n_0 \Rightarrow\ t(n) \leq k\ f(n))\ )\ )\ \}$$

baldin eta f(n) beti hertsiki positiboa den $\forall n \in N$. Hau da, n_0 atalase-balioa ez da printzipioz beharrezkoa, nahiz eta praktikan erabilkorra izan.

3- Iragankortasun propietatea froga bedi:

$$\left.\begin{array}{l} f(n) \in O(g(n)) \\ g(n) \in O(h(n)) \end{array}\right\} \Rightarrow f(n) \in O(h(n))$$

Θ, Ω eta *o* ordenentzat ere propietatea betetzen da.
Froga bedi ondorengoa ere: baldin $g(n) \in O(h(n))$ orduan $O(g(n)) \subseteq O(h(n))$

4- $\forall f, g: N \Rightarrow R^*$, frogatu:

a) $O(f(n)) = O(g(n)) \Leftrightarrow f(n) \in O(g(n))$ eta $g(n) \in O(f(n))$,
b) $O(f(n)) \subset O(g(n)) \Leftrightarrow f(n) \in O(g(n))$ eta $g(n) \notin O(f(n))$

5- Froga bitez ondorengo propietateak edozein direla f eta g: $N \Rightarrow R^*$

a) baturaren erregela; i.e., $O(f(n)+g(n)) = O(\max(f(n),g(n)))$
(Θrentzat eta Ωrentzat ere egiazkoa dela gogoratu)
b) $f(n) \in O(g(n)) \Leftrightarrow g(n) \in \Omega(f(n))$,
c) $f(n) \in \Theta(g(n)) \Leftrightarrow g(n) \in \Theta(f(n))$.

6- Ondorengo arrazonamenduan non dago akatsa?

$$\sum_{j=1}^{n} j = 1+2+...+n \in O(1+2+...+n) = O\left(\max(1,2,...,n)\right) = O\left(n\right)$$

7- Limiteen herraminta erabiliz bi funtzio aldera ditzakegu. Izan bitez:
$f, g: N \Rightarrow R^+$. Froga bitez:

a) $\lim_{n\to\infty} \frac{f(n)}{g(n)} \in R^+ \Rightarrow O(f(n)) = O(g(n))$

b) $\lim_{n\to\infty} \frac{f(n)}{g(n)} = 0 \Rightarrow O(f(n)) \subset O(g(n)) = O\left(g(n) \pm f(n)\right)$

c) frogatu aldi berean jarraikoa gerta daitekeela:

$O(f(n)) = O(g(n))$ eta $\neg\exists \lim_{n\to\infty} \frac{f(n)}{g(n)}$

d) frogatu aldi berean jarraikoa gerta daitekeela:

$O(f(n)) \subset O(g(n))$ eta $\neg\exists \lim_{n\to\infty} \frac{f(n)}{g(n)}$ eta $O\left(g(n)\right) \neq O\left(g(n) - f(n)\right)$

e) $\lim_{n\to\infty} \frac{f(n)}{g(n)} \in R^+ \Rightarrow f(n) \in \Theta(g(n))$

f) $\lim_{n\to\infty} \frac{f(n)}{g(n)} = 0 \Rightarrow f(n) \in O(g(n))$ baina $f(n) \notin O(g(n))$

g) $\lim_{n\to\infty} \frac{f(n)}{g(n)} = +\infty \Rightarrow f(n) \in \Omega(g(n))$ baina $f(n) \notin \Theta(g(n))$

$O(f(n))=O(g(n))$ eta $\neg\exists \lim_{n\to\infty} \frac{f(n)}{g(n)}$

8- L'Hopital-en erregela edota 4 eta 7 ariketetako propietateak erabiliz ondorengoa frogatu:

$\log n \in O(\sqrt{n})$ baina ez $\sqrt{n} \notin O(\log n)$

9- Ondorengo funtzioak goranzko ordenean zerrenda bitez. Ordena berdinekoak direnak nolabait hala direla azaldu:

n	2^n	n lg n	ln n
$n-n^3+7n^5$	lg n	$\sqrt{n}$	e^n
$n^2+\lg n$	n^2	2^{n-1}	lg lg n
n^3	$(\lg n)^2$	n!	$n^{1+\varepsilon}$ y $0<\varepsilon<1$
n lg n	$n^2/\lg n$	n^{1+x}	$(1+x)^n$
lg n!	$(n^2+8n+\lg^3 n)^4$		

10- Ondorengo programa zatien analisia egin bedi kasu txarrenerako:

```
a) — Lista 0-z hasieratu
   for IND in LISTEN_FREKUENTZIA'RANGE loop
           LISTEN_FREKUENTZIA(IND) := 0;
   end loop;
b) — Handienaren posizioa aurkitu
   IND := 1;
   for ZENB  in 2..LISTA'LAST loop
           if  LISTA(ZENB) > LISTA(IND) then
                   IND := ZENB;
           end if
   end loop;
```

c)
```
-- Taula batean bilaketa lineala
AURKITUA := false;
IND := 0;
while IND < LISTA'LENGTH and  not AURKITUA loop
         IND := IND +1;
         if  ELEMENTUA = LISTA(IND) then
                  AURKITUA := true;
         end if
end loop;
```

11- Ondoan dagoen BILATUBIDEA funtzioak A eta B adabegien artean LUZ luzerako bideak existitzen dun ala ez adierazten du.

```
function  BILATUBIDEA (LUZ: in INTEGER; A,B : in ADABEGIA)
                     return BOOLEAN is
        K: integer;
begin
       if  LUZ= 1  then BILATUBIDEA :=ALBOKO(A, B)
       else
           for   K  in 1 .. ADABEGI_KOP loop
                 if ALBOKO(A, K) or else
                    BILATUBIDEA(LUZ-1, K, B)
                 then return true;
                 end if;
           end loop;
           return false;
       end if;
end BILATUBIDEA;
```

Suposatuz ALBOKO eragiketaren konplexutasuna 1 ordenakoa dela, esan ondoko ekuazioen artean zein den BILATUBIDEA funtzioaren ordena adierazten duena. Aukera bakoitza arrazoitu.

$$T_1(LUZ) = \begin{cases} k_1 \bullet ADABEGI\_KOP & LUZ = 1 \\ k_2 + k_3 \bullet ADABEGI\_KOP \bullet T_1(LUZ-1) & LUZ > 1 \end{cases}$$

$$T_2(LUZ) = \begin{cases} k_1 & LUZ = 1 \\ k_2 \bullet ADABEGI\_KOP \bullet T_2(LUZ-1) - k_3 & LUZ > 1 \end{cases}$$

$$T_3(LUZ) = \begin{cases} k_1 & LUZ = 1 \\ k_2 + ADABEGI\_KOP \bullet \left(k_3 + T_3(LUZ-1)\right) & LUZ > 1 \end{cases}$$

12- T(I) ≤ N (1 ≤ I ≤ N) betetzen duen T(1..N) taula eta X zero ez den digitua emanda, ondoan ematen den algoritmoaren konplexutasuna kalkulatu. Algoritmo honek K aldagaian T taularen elementu positibo guztien X digituaren agerpen kopurua bueltatzen du.

```
begin
(1)for I in T'RANGE loop
(2)     if T(I) > 0 then
(3)             B := T(I); K := 0;
(4)             while B> 0 loop
(5)                if (B rem 10 = X) then
(6)                    K := K + 1;
                   end if;
(7)                B := B/10;
                end loop;
(8)             TEXT_IO.PUT (INTEGER'IMAGE'T(I) &"-an: "&
                                INTEGER'IMAGE(K)&" aldiz ");
        end if;
   end loop;
end;
```

13- Ondorengo programa errekurtsiboen T(n)-a eta ordena kalkula ezazu:

a)

```
function FAKTORIALA (N: in INTEGER) return INTEGER is
begin
  if  N= 0  then
        return 1;
  else
        return  N * FAKTORIALA(N-1);
  end if;
end FAKTORIALA;
```

b)

```
function FIBONACCI (N: in INTEGER) return INTEGER is
begin
  if  N=1  then
        return N;
  else
        return  FIBONACCI(N-1) + FIBONACCI(N-2);
  end if;
end FIBONACCI;
```

c)

```
procedure HANOI (N: in INTEGER; I,J:  in DATU_1_2_3 ) is
  -- N-ek diskoen kopurua adierazten du.
begin
  if N> 0 then
         HANOI (N-1, I, 6-I-J)
         TEXT_IO.PUT (DATU_1_2_3'IMAGE(I)
                      &"->"&DATU _1_2_3'IMAGE(J));
         HANOI (N-1, 6-I-J,J);
  end if;
end HANOI ;
```

14- t(n)-ak emanik haien ordenak kalkula itzazu:

a) $t(n) = \begin{cases} 2\,t(n-1) + n + 2^n & n \geq 1 \\ 0 & n=0 \end{cases}$

b) $t(n) = \begin{cases} 2\,t(n-1) + (n+5) + 3^n & n>1 \\ 1 & n=1 \\ 0 & n=0 \end{cases}$

c) $t(n) = \begin{cases} 2\,t(n-1) + 3^n & n>1 \\ 1 & n=1 \\ 0 & n=0 \end{cases}$

d) $t(n) = \begin{cases} 3\,t(n-1) + (n+5) + 3^n & n>0 \\ 0 & n=0 \end{cases}$

e) $t(n) = \begin{cases} 3\,t(n-1) + 3^n & n>0 \\ 0 & n=0 \end{cases}$

15- Ekuazio karakteristikoaren metodoa erabiliz errekurtsio-ekuazioen erroak aurkitu:

a) $t(n) = 5\,t(n-1) - 6\,t(n-2) + 4\,3^n$ $\quad n>1$
$t(1)= 36 \qquad t(0)= 0$

b) $t(n) = 3\,t(n-1) - 2\,t(n-2) + 2^{n-1} + 2\,3^n$ $\quad n>1$
$t(1)= 29 \qquad t(0)= 9$

c) $t(n) = t(n-2) + 4\,n$ $\quad n>1$
$t(1)= 4 \qquad t(0)= 1$

d) $t(n) = 3\,t(n-1) + t(n-2) - 3\,t(n-3) + 16\,n + 8\,3^n$ $\quad n>2$
$t(2)=117 \qquad t(1)= 10 \qquad t(0)= -1$

e) $t(n) = 3\,t(n-1) - 2\,t(n-2) + 3\,2^{2n-1}$ $\quad n>1$
$t(1)= 12 \qquad t(0)= 0$

Emaitzak:

a) $t(n) = 12\ n\ 3^n$ b) $t(n) = n\ 2^n + 3^{n+2}$

c) $t(n) = (n+1)^2$ d) $t(n) = (9\ n-2)\ 3^n + 2n^2 - 10\ n + 1$

e) $t(n) = 4(4^n-1)$

16- Izan bedi $b \geq 2$ arrunta, $f: N \rightarrow R^*$ funtzio b-armonikoa, eta $t: N \rightarrow R^*$ asintotikoki beherakorra ez den funtzio bat, non

$$t(n) \in \Theta(f(n) \mid n \text{ b-ren potentzia den})$$

$t(n) \in O(f(n))$ betetzen dela froga bedi. Bi adibidetaz baliaturik, bai "t(n) asintotikoki ez beherakorra" dela bai eta "$f(bn) \in O(f(n))$" emaitza orokorrean lortzeko baldintza beharrezkoak direla frogatu.
Propietateak egiazkoa izaten jarraitzen du Θ-ren ordez O jartzen denean.

17- $\forall n \geq n_0$ $f(n) \geq g(n)$ eta $f(n), g(n) \in \Theta(h(n))$ baldintzetan egonik, zein da f(n)-g(n)ren ordena?

18- Froga bedi: $\forall a,b (a,b>1 \Rightarrow \lg_a n \in \Theta(\lg_b n))$. Baldin $a \neq b$, orduan egia da $2^{\lg_a n} \in \Theta(2^{\lg_b n})$?

19- Honako errekurtsio-ekuazioa ebatz bedi. Zein da bere ordena?

$$T(n) = \begin{cases} a & \text{baldin } n = 1 \\ 2T\left(\frac{n}{4}\right) + \lg n & \text{baldin } n > 1 \end{cases}$$

20- (a atala)
Ondorengo funtzio errekurtsiboak, koefiziente-taula eta X aldagaiaren balioa emanik:

X

KOEF

a_0	a_1	a_2 ...	a_{n-1}

$$\Rightarrow \quad a_{n-1} X^{n-1} + a_{n-2} X^{n-2} + \ldots + a_1 X^1 + a_0$$

polinomioaren balioa zein den kalkulatzen du:

```
Koef: Arrunten_Taula(0..N-1):= HASIERATU_K;
X: Polinomio_Aldagaia:= HASIERATU_X;
function POLI_EBAL (Y: Arrunten_Taularen_Indizea)
                                        return Pol_Balioa is
   X_Ber_Y: ...
begin
```

```
    if  Y = Koef´FIRST  then  return Koef(Koef´FIRST);
    else    X_Ber_Y:= X;
            for  Ind  in Koef´FIRST+2..Y  loop
                    X_Ber_Y:=X_Ber_Y * X;
            end loop;
            return (Koef(Y * X_Ber_Y)+ POLI_EBAL(Y-1);
    end if;
end POLI_EBAL;
```

✏ POLI_EBAL(Koef´LAST) deiak zenbat eragiketa aritmetiko (+,-,*,/) egiten ditu? Hau da, $T_1(n)$ = zenbat eragiketa aritmetiko egikaritzen dira?

(b atala)

Aurreko ataleko zenbakia erraz hobetu daiteke. Honela, eta ** berreketa eragile estandarra erabili gabe, emaitza berdina ekoiztuko duen eta T(n) hobea duen algoritmoa idatzi eta $T_2(n)$ berria txikiagoa dela frogatu.

21- (a atala)

Ondorengo funtzio errekurtsiboak Xen M. berredura kalkulatzen du:

```
X: Oinarria := HASIERATU_X;
function X_BER (M: Arrunta) return Arrunta is
begin
    if  M = 0  then  return 1;
    elsif  M = 1  then  return X;
    else
            if  M mod 2 = 0
            then  return X_BER(M/2) * X_BER(M/2);
            then  return X* X_BER(M/2) * X_BER(M/2);
    end if;
end X_BER;
```

Zein da algoritmoaren ordena zehatza?

Iruzkina: ekuazio karakteristikoaren metodoa erabiltzen duenak, hark eskatzen dituen konstanteen balioak bilatzeko, oinarrizko kasuak N=1etik aurreragokoak erabil itzazu (1 barne).

(b atala)

Aurreko ataleko algoritmoak kalkulu berdinak errepikatzen dituela ikusiz, balio bera kalkulatzen duen algoritmo errekurtsibo berria idaztea eskatzen da,

baina berreketa (**) eragiketa estandarra erabili gabe. Bertsio berriak biderkaketa kopuru gutxiago egin beharko du. Algoritmo berriaren ordena hobea dela frogatu.

Iruzkina: aurreko atalekoa.

2. GAIA

ORDENAZIO ALGORITMOAK

2.1. SARRERA

Ikasgai honetan hainbat ordenazio algoritmo ikusiko dugu. Algoritmo hauek lista bateko elementuak ordena jakin baten arabera berrordenatzen dituzte. Listako elementu bakoitzari identifikatzaile bat dagokio, *giltza* izenekoa. Elementu hauek linealki ordenaturik dagoen multzo bateko partaideak direnez gero, bi giltzen arteko alderaketa egin daiteke; bietatik handiena zein den edo biak berdinak ote diren determina dezakegu. Berez, giltzaren balio propioa ez zaigu interesatzen, bai ordea beste giltzekiko duen ordena erlatiboa. Listako elementu bakoitzak giltzez at informazio gehiago izan dezake. Ordenazioa egitean giltzaren balioa ez ezik, erregistro berean dagoen beste informazio guztia ere berrordenatuko dugu behar den moduan. Hala eta guztiz ere, ondorengo algoritmoetan giltzari buruz bakarrik mintzatuko gara, eta gainontzeko informazioari buruz ez dugu aipamenik egingo.

Zergatik ikasten dira ordenazio algoritmoak?

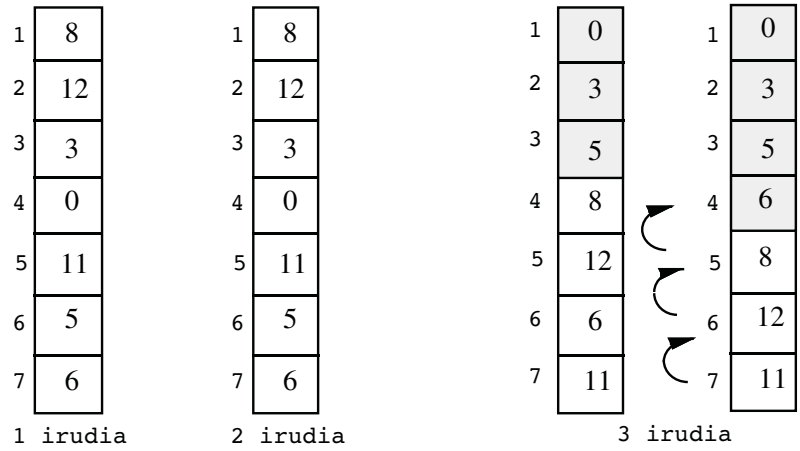
- Ordenazioa maiz egiten denez, algoritmo hauek asko erabiltzen dira.
- Problema bera, ordenazioa, algoritmo desberdinek ebatz dezakete. Hauen azterketak baliagarriak gerta dakizkiguke algoritmo bat beste batzuren artean aukeratu nahi dugunean, bai eta emaniko algoritmo bat hobetu nahi denean ere.
- Ordenazio algoritmoen kasu txarrenaren eta batezbesteko kasuen kalkulua ez da beste problemetan bezain zaila, lortzen diren bornaketak nahiko onak baitira.

Ikasgai honetako algoritmoek goranzko ordena jarraituz ordenatuko dituzte giltzak. Bestalde, Ada programazio-lengoaiatik hurbil idatzi diren algoritmoetan, adibidez, nahiz eta taulak oso-osorik parametro bezala pasa, hauen pasatzeak denbora edo memoria-espazio aldetik desabantailarik ez dakarrela suposatuko dugu. Hau kontsideratzea zentzuzkoa da gaur egungo konpiladore gehienek erreferentzia bidez pasatzen baitituzte.

2.2. BURBUILA ETA AUKERAKETA

2.2.1. Burbuila

Prozesua:



Algoritmo honek sarrerako taulako osagaiak ordenatzeko *n-1* iterazio egin behar ditu. 1 irudiak prozesuaren hasierako egoera azaltzen du. Lehenengo biran, sekuentziako osagai txikiena taulako aurreneko posiziora azaleratzen da, eta ondorengo bietarako osagai hori bere posizio erlatibo zuzenean gelditu denez, ez da kontuan hartuko ez bere balioa ez eta haren posizioa ere. Honela, prozesu bera gelditzen diren *n-1* osagaiekin errepikatzen da. 2 irudian *i* iterazio ondoren taularen egoera nolakoa izango den jasotzen da: lehengo *i* osagaiak ordena gorakorra jarraituz ordenatuta daude eta taulako beste osagaiak baino txikiagoak dira. Azkeneko irudian, berriz, *i+1* iterazioan (*i+1*)garren osagai txikiena nola azaleratzen den ageri da.

Algoritmoa:

```
procedure BURBUILA (S: in out OS_SEK) is
begin   — Ss= <a1,a2, ...,an>
(1)     for  I  in S'RANGE  loop
                — permutazioa(Ss, S) ∧ ordena_<(S(S'FIRST,...,I))
(2)             for J  in reverse I+1..S'LAST  loop
(3)                     if  S(J-1) >S(J) then
(4)                             TRUKEA(S(J-1),S(J));
                        end if;
                        end loop;
                — permutazioa(Ss, S) ∧ ordena_<(S(S'FIRST,...,I))
                end loop;
— permutazioa(Ss, Si) ∧ ordena_<(Si)
end BURBUILA;
```

Algoritmoaren ordenaren analisia:

a) TRUKEAri eginiko deiak: $\Theta(1)$ ordena hartzen du, prozedura horren gorputza elkarren segidan dauden hiru asignazio aginduez soilik osaturik baitago:

- Asignazio bakoitzaren ordena, konstantea: $\Theta(1)$, eta
- Baturaren erregela erabiliz, hiru asignazioen denboraren ordena: $\Theta(\max(1,1,1)) = \Theta(1)$

b) IF-THEN (3,4), FOR(2,4) eta FOR(1,4) aginduak habiatuta daudenez, barrutik kanpora joango gara.

d) IF-THEN (3,4) eragiketa: baldintza $\Theta(1)$ ordenakoa dugu; kasu txarrenean THEN atalaren aginduak exekutatzen dira, non $\Theta(1)$ ordenakoak diren *a* ataleko kalkuluen arabera:

$\rightarrow$ IF-THEN osoa $\Theta(1)$ ordenakoa dugu.

e) FOR(2,4) eragiketa: 2 eta 6 puntuen arteko aginduak (n-i) aldiz errepikatzen dira, non *n*-k taulako osagaien kopurua adierazten duen.

f) Biderkaketaren erregela aplikatuz: $\Theta((n-i) * 1) = O(n-i)$

g) FOR(1,4) eragiketa: kanpoko begizta n aldiz egikaritzen da, beraz:

$$T(n) = \sum_{i=1}^{n} (n-i) = (n-1) + (n-2) + ... + 1 + 0 = \frac{0 + (n-1)}{2} \bullet n = \frac{n^2 - n}{2}$$

$$T(n) \in \Theta\left(n^2\right)$$

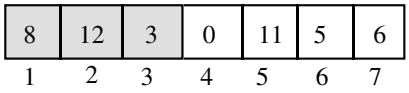
*Burbuila*ren algoritmoan sarreraren egoerak ez du eraginik. Hau da, sarrera edozein izanik ere, hartzen duen denbora beti ordena berekoa da.

Bestalde, algoritmoek exekutatzen dituzten aginduen kopurutik kanpo, hau da, egindako lanetik at, algoritmoen eraginkortasuna neurtzeko algoritmoek beren helburua lortzeko erabiltzen duten memoria-espazio *estra* kalkulatzea ere interesgarria izan ohi da. Neurketa berri hau ahal dugunean egingo dugu edota haren kalkulua interesgarria gerta dakigukenean.

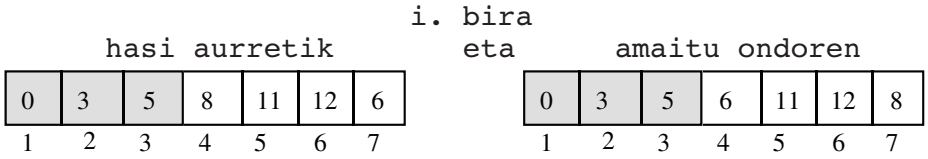
Aldiz, *Burbuila* algoritmoak *n* osagaiez osaturiko taula bat ordenatzeko ez du memoria-espazio estrarik behar.

2.2.2. Aukeraketa

Prozesua:



Prozesuaren i.go bira egiten hasterako lehenengo (i-1) posizioetan sarrerako taulako (i-1) osagai txikienak ordena gorakorrean daude. Orduan, Min_j eta Min_b aldagaietan i. indizea eta S(i) balioak metatzen dira hurrenez hurren. Bira honetan, gelditzen diren (n-i) osagaien arteko baliorik txikiena Min_b aldagaian metatzen da eta balio horren posizioa, berriz, Min_j aldagaian. Ondoren, Min_j eta i posizioetako balioak trukatzen dira, honela, lehenengo i posizioetan sarrerako i osagai txikienak ordena gorakorrean ordenaturik lortzen direlarik.



Algoritmoa:

```
procedure  AUKERAKETA  ( S: in out OS_SEK) is
   Min_J: INDIZEA;
   Min_B: INTEGER;
begin      — S_s= <a_1,a_2, ...,a_n>
(1)        for  I  in  S'FIRST.. S´LAST-1  loop
                  — permutazioa(S_s, S) ∧ ordena_<(S(S'FIRST,...,I-1))
(2)               Min_J := I;
(3)               Min_B := S(I);
(4)               for  J  in reverse  I+1..S´LAST loop
(5)                     if  S(J) ≤ Min_B then
(6)                           Min_J := J ;
(7)                           Min_B :=S(J);
                        end if;
                  end loop;
(8)               S(Min_J) := S(I);
(9)               S(I):= Min_B;
           end loop;
— permutazioa(S_s, S_i) ∧ ordena_<(S_i)
end AUKERAKETA;
```

Oro har, *Burbuila*ren egitura jarraitzen duenez eta beste prozedura-deirik ez dagoenez gero, algoritmo honen eta aurrekoaren ordena berdina izango dela egiazta dezakegu:

Algoritmoaren ordenaren analisia:

a) 6,7 puntuko asignazioek denbora konstantea behar dute: bakoitzaren ordena 1 da.

b) Baturaren erregela erabiliz, jarraian dauden bi eragiketen ordena: $\Theta(\max(1,1)) = \Theta(1)$

d) IF-THEN (5,7) eta FOR(4,7) aginduak habiatuta daudenez, barrutik kanpora joango gara.

e) IF-THEN (5,7) eragiketa: baldintza $\Theta(1)$ ordenakoa dugu; kasu txarrenean THEN atalaren aginduak exekutatzen dira, non $\Theta(1)$ ordenakoak diren b ataleko kalkuluen arabera:

$\rightarrow$ IF-THEN osoa $\Theta(1)$ ordenakoa dugu.

f) FOR(4,7) eragiketa: barneko agindu bakarra, (n - (i+1) + 1) aldiz egikaritzen da. Biderkaketaren erregela aplikatuz, FOR-ak osoki hartzen duen denboraren ordena lortzen dugu: $\Theta((n-i) \bullet 1) = \Theta(n-i)$

g) 8 eta 9 asignazioak $\Theta(1)$ ordenakoak dira.

h) 2 , 3, 8 eta 9 asignazio bakoitzaren ordena 1 da.

i) (2), (3), FOR(4,7) , (8) eta (9) agindu-sekuentziak hartzen duen denboraren ordena, baturaren erregela aplikatuz lortzen dugu: $\Theta(\max(1,1,(n-i),1,1)) = \Theta(n-i)$

j) FOR(1,7) eragiketa: kanpoko begizta ((n-1) - 1 + 1) aldiz errepikatzen da, beraz:

$$T(n) = \sum_{i=1}^{n}(n-i) = (n-1)+(n-2)+...+1+0 = \frac{0+(n-1)}{2} \bullet n = \frac{n^2-n}{2}$$

$$T(n) \in \Theta\left(n^2\right)$$

Esan bezalaxe, *Burbuila* eta *Aukeraketa* algoritmoen ordena berdina da. Oraingo honetan ere, *Aukeraketa*n sarrerako egoerak ez du zerikusirik.

Ez du memoria-espazio estrarik ere behar. Ordenazioa sarrerako taularen gainean egikaritzen da.

2.3. TXERTAKETA. KASU TXARRENA ETA BATEZBESTEKO KASUA

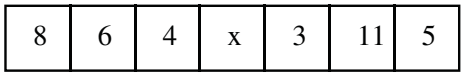
Prozesua:

Ordenazioa egiteko era naturala eta orokorra da. Bestaldetik, kasu txarrenaren eta batezbesteko kasuaren analisia egitea ez da zaila. Maiz, beste ordenazio algoritmo azkarragoen barnean erabili ohi da.

Ordenazio prozesua *n* giltza dituen S sekuentzia batekin hasten da, non giltza horiek ausaz ordenaturik dauden. Suposa dezagun sekuentziaren segmentu bat ordenatu dugula. Pauso orokorrean, ordenatuta dagoen segmentu horren luzera gehitu egiten da. Horretarako, ondorengo giltza ordenaturiko segmentuan eta dagokion lekuan txertatzen da.

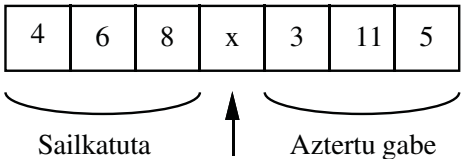
Adibidez:

Suposa dezagun zazpiko luzera duen ondorengo lista ordenatu nahi dugula:

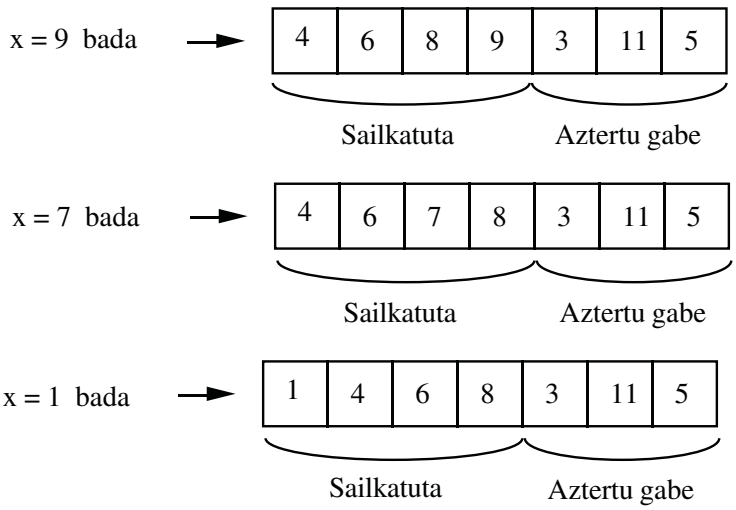


Sailkatu gabeko giltzak

Suposa dezagun, zenbait giltza ordenatu ondoren, ondorengo egoeran aurkitzen garela:



Oraingo honetan, x giltza, jadanik ordenatuta dagoen segmentuan txertatzeak, ondorengo egoerara igarotzea ekarriko du:



Aurreko prozesua ondorengo algoritmoan aurrematen da:

Algoritmoa:

```
procedure  TXERTAKETA  (S: in out OS_SEK) is
   X:INTEGER; J: INDIZEA; JARRAITU: BOOLEAN;
begin      — Ss= <a1,a2, ...,an>
(1)        for  I  in  S'FIRST+1..S'LAST  loop
                  — permutazioa(Ss, S)  ∧ ordena_<(S(S'FIRST,...,I-1))
(2)               X:= S(I);
(3)               J:= I-1;
(4)               JARRAITU := true;
(5)               while  J>S'FIRST-1  and  JARRAITU  loop
(6)                        if  S(J)>X  then
(7)                                 S(J+1) := S(J);
(8)                                 J:= J-1;
                           else
(9)                                 JARRAITU := false;
                           end if;
                  end loop;
(10)              S(J+1) := X;
                  — permutazioa(Ss, S) ∧ ordena_<(S(S'FIRST,...,I))
           end loop;
— permutazioa(Ss, Si) ∧ ordena_<(Si)
end TXERTAKETA;
```

Adibidean zehar ikusi den bezala, txertaketa algoritmoak giltzen arteko alderaketa asko egiten du. Hori dela eta, algoritmoaren eraginkortasuna aztertzeko garaian giltzen arteko alderaketen kopurua neurtzea (exekutaturiko aginduen kopuruaren ordez) algoritmoak egindako lanaren adierazgarri ona litzateke (eta neurtzeko errazagoa).

Algoritmoaren denbora-ordenaren analisia:

Kasu txarrena:

I-ren balio bakoitzerako while begiztaren barnean egin daitezkeen giltzen alderaketen kopururik handiena *I-1* da. Ondorioz, guztira:

$$T(n)=\sum_{i=2}^{n}(i-1)=1+2+...+n-1=\frac{1+(n-1)}{2}\bullet(n-1)=\frac{n^2-n}{2}$$

$$T_t(n)\in\Theta\left(n^2\right)$$

Sarreran dauden giltzak beheranzko ordena jarraituz ordenaturik badaude, sarrera hori Txertaketa algoritmoaren kasu txarrena dela egiazta daiteke.

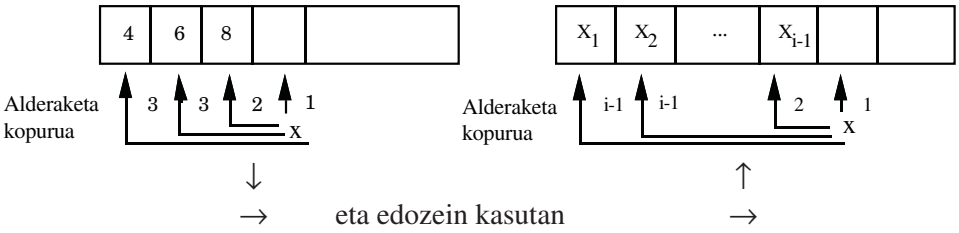
Batezbesteko kasua:

Analisia erraztearren, giltza guztiak ezberdinak direla eta giltzen permutazio guztiak ekiprobableak direla suposatuko dugu.

Lehenbizi, giltza bat jadanik ordenatuta dagoen segmentuan txertatzeko egiten diren alderaketen kopurua finkatuko dugu; hau da, *while* begiztan egiten diren batezbesteko iterazioen kopurua *I*-ren balio bakoitzarentzat.

x ordenatu nahi dugun *I*. giltza bada, *I* posizio ezberdinetan koka dezakegu bere balioaren arabera. Ondorioz, *x*-ek posizio konkretu batean kokatzeko duen probabilitatea 1/*I* da (hau betetzen da, baldin eta *x* aurrez azaldu ez bada. Aurrez azaldu izan balitz eta ondorioz algoritmoak erabaki bat hartu izan balu, ezingo atekeen esan zuzenean aurreko *I*-1 giltzekiko aleatorioa zela).

Ondorengo irudiek, giltza bakoitzari dagokion posizioa lortzeko egin beharko liratekeen alderaketen kopurua adierazten digute:



Hau guztia kontutan izanik, *for* aginduaren iterazio bakoitzeko *x* txertatzeko egin behar diren batezbesteko konparazio kopurua:

$$\sum_{j=1}^{i-1}\frac{1}{i}+\frac{1}{i}(i-1)=\frac{1}{i}\sum_{j=1}^{i-1}j+\left(1-\frac{1}{i}\right)=\frac{1}{i}\left(\frac{1+(i-1)}{2}(i-1)\right)+\left(1-\frac{1}{i}\right)=$$

$$=\frac{1}{i}\frac{i}{2}(i-1)+1-\frac{1}{i}=\frac{i-1}{2}+1-\frac{1}{i}=\frac{i}{2}-\frac{1}{2}-\frac{1}{i}=\frac{i+1}{2}-\frac{1}{i}$$

Honela, *for*-aren ondorioz egin behar diren txertaketa guztiak eransten badizkiogu:

$$T_b(n)=\sum_{i=2}^{n}\left(\frac{n+1}{2}-\frac{1}{i}\right)=\frac{1}{2}\sum_{i=2}^{n}(i+1)-\sum_{i=2}^{n}\frac{1}{i}=\frac{1}{i}\left(\frac{3+(n+1)}{2}(n-1)\right)-\sum_{i=2}^{n}\frac{1}{i}=$$

$$=\frac{n+4}{2}\ \frac{n-1}{2}-\sum_{i=2}^{n}\frac{1}{i}=\frac{n^2+3n-4}{2}-\sum_{i=2}^{n}\frac{1}{i}=$$

$\sum_{i=2}^{n} \frac{1}{i} \approx \ln n$ denez, eta ordena txikiagoko batugaiak ez baditugu kontutan hartzen: $T_b(n) \approx \frac{n^2}{4} \in \Theta(n^2)$

Bi kasuen analisia egin ondoren lortu ditugun emaitzak ondorengo hauek dira:

$$\Rightarrow T_t(n) = \frac{n(n-1)}{2} \in \Theta(n^2)$$

$$T_b(n) \approx \frac{n^2}{2} \in \Theta(n^2)$$

Hainbat ondorio:

Tankerako zenbait algoritmo aztertu ondoren, teorema hau egiaztatzen da:

Teorema 2.3.4.1:

Giltzen alderaketen bidez sailkatzen duen eta alderaketa bakoitzaren ondoren gehienez alderanzketa (inbertsio) bat ezabatzen duen edozein algoritmok gutxienez n(n–1)/2 konparazio egin beharko ditu kasu txarrenean eta batezbesteko kasuan berriz, gutxienez n(n–1)/4, non n giltzen kopurua den.

Definizioa:

a_1, a_2, ...,a_n n elementu dituen permutazio bat izanik eta $\forall$.i,j ($1 \le i < j \le n$) eta $a_i > a_j$ betetzen badira, orduan (a_i,a_j) bikotea *alderanzketa* bat da.

2.4. ORDENAZIO-AZKARRA, QUICKSORT, ETA BATERAKETA, MERGESORT: ZATITU ETA IRABAZI

Bi algoritmoek 'zatitu eta irabazi' teknika jarraitzen dute. Teknika pauso hauetan azal dezakegu:

➪ N tamainako problema tamaina txikiagoa duten problema bereko zenbait azpiproblematan banatzen du (kasu honetan, ordenatu beharreko lista, luzera txikiagoa duten zenbait azpilistatan banatzen da),

➪ txikiagoak diren instantziak errekurtsiboki ebazten ditu metodoren bat erabiliz,

➪ azkenean, azpiproblemek itzulitako emaitza guztiak konbinatuz, jatorrizko sarreraren emaitza lortzen du.

Quicksort eta *Mergesort* problema zatitzeko moduan eta ondoren emaitza partzialak (ordenaturiko azpilistak) konbinatzeko moduan desberdintzen dira.

2.4.1. Ordenazio-azkarra algoritmoa: QUICKSORT

Hoare-ren ordenazio algoritmoa izenez ere ezaguna da.

2.4.1.1. QUICKSORT: betiko algoritmo errekurtsiboa

Prozesua:

Estrategia honek ordenatu behar diren giltzak berrantolatzen ditu, balio jakin bat baino 'txikiagoak' diren giltzak 'handiagoak' diren giltzen aurrean kokatuz. Ondoren, Quicksort-ek 'txikiagoak' eta 'handiagoak' azpilistak errekurtsiboki ordenatzen ditu. Azkenean, jatorrizko lista bera baina goranzko ordena jarraituz ordenatuta itzultzen du.

Izan bedi S giltzen sekuentzia (taula batez inplementaturik dagoena) eta izan bitez *S'FIRST* eta *S'LAST* sailkatzen ari den azpisekuentziaren lehenengo eta azkeneko sarreren indizeak hurrenez hurren. Hasiera batean, izan bitez *S'FIRST* =1 eta *S'LAST*=n, non n giltzen kopurua den. *Banatu* algoritmoak azpisekuentziatik giltza bat aukeratuko du. Giltza honen balioaren arabera (izan bedi *x* bere balioa) sarrera berrordenatzen du banaketa-puntu deritzon indize bat aurkituz. Indize honek ondoko baldintzak betetzen ditu:

- $\forall i\ (S'FIRST \leq i < Banaketa\_Puntua \rightarrow S(i) < x\ )\ \wedge$
- $S(Banaketa\_Puntua) = x\ \wedge$
- $\forall j\ (Banaketa\_Puntua < j \leq S'LAST \rightarrow\ x \leq S(j)\ )$

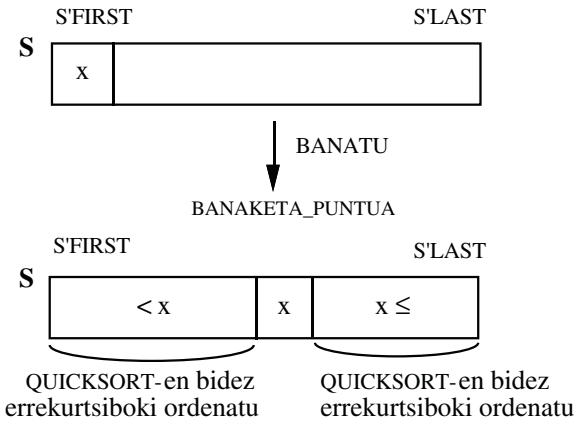
Honenbestez, *x* taulan dagokion posizio zuzenean (erlatiboan) kokatzea lortu da. *x* balioari BANAKETA-BALIOA deritzogu eta ondorengo ordenazio pausoetan ez da kontutan izango:

Algoritmoa:

```
procedure QUICKSORT ( S: in out OS_SEK) is
    Banaketa_Puntua:PUNTUA:INDIZEA;
begin — Ss= <a1,a2, ...,an>
    if  S'LENGTH >1  then
            BANAKETA(S, Banaketa_Puntua);
            — permutazioa(Ss, S) ∧
            —denak_<(S(S'FIRST..Banaketa_Puntua-1),S(Banaketa_Puntua)) ∧
            — denak_>(S(Banaketa_Puntua+1)..s'LAST),S(Banaketa_Puntua))
            QUICKSORT (S(S'FIRST.. BANAKETA-PUNTUA-1));
            — permutazioa(Ss, S) ∧ ordena_<(S(S'FIRST..Banaketa_Puntua))
            QUICKSORT (S(Banaketa_Puntua+1.. S'LAST));
            — permutazioa(Ss, S) ∧ ordena_<(S(S'FIRST..Banaketa_Puntua)) ∧
            — ordena_<(S(Banaketa_Puntua..S'LAST))
    end if;
— permutazioa(Ss, Si) ∧ ordena_<(Si)
end QUICKSORT ;
```

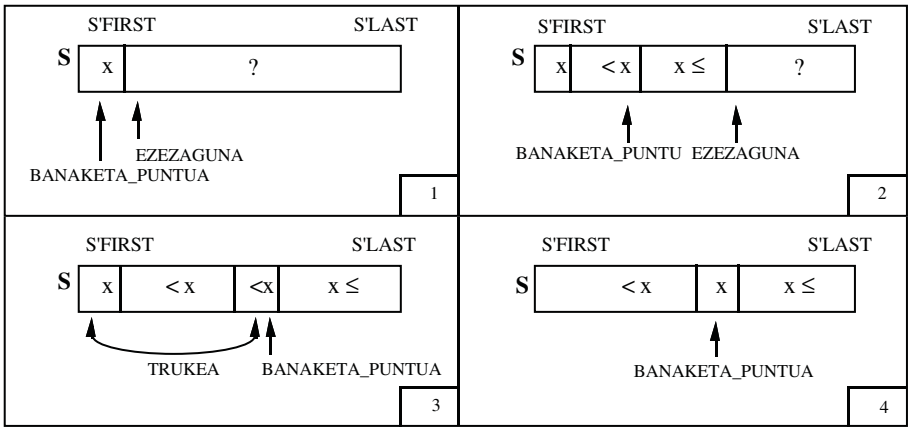
Sekuentziaren banaketaren kalkulua:

Prozesua:



Giltzen alderaketa eta lekualdaketa BANAKETA prozedurak egingo du. BANAKETA egiteko ere zenbait estrategia desberdin dago, eta bakoitzak bere aldeko eta aurkako puntuak ditu. Guk estrategia bakar bat aztertuko dugu.

BANAKETA algoritmoak S(S'FIRST) eta S(S'LAST)-en arteko edozein giltza aukera badezake ere, gauzak erraztearren, x = S(S'FIRST) hartuko dugu. Gainontzeko giltzak, BANAKETA prozedura exekutatzen ari den bitartean elkarren segidako 3 sekuentziatan banatzen dira: x baino txikiagoak direnak taularen aurrealdean, haien segidan, x baino handiagoak direnak, eta azkenik taularen amaieran kokatuta, x-ekin oraindik alderatu ez direnak (x-ekiko duten erlazioa oraindik ezezaguna da).



Hasieran giltza guztiak 3. sekuentzian daude. Izan bedi *y* iterazio bakoitzeko ondorengo giltza ezezaguna; BANAKETA prozedurako begiztan *y* *x*-ekin alderatzen da. Baldin $y < x$ orduan 2. sekuentziako lehenengo giltza eta *y* trukatzen dira, bestela, $x \leq y$, *y* dagoen tokian gera daiteke. Giltza ezezagun guztiak *x*-ekin alderatu ondoren, *x* berez taulan dagokion posizio erlatiboan kokatzen da. Hau lortzeko, 1. sekuentzian dagoen azkeneko giltza eta *x* balioa trukatzen dira.

Algoritmoa:

```
procedure BANAKETA (S: in out OSOEN_SEK; BP: out INDIZEA) is
   X: INTEGER; BP_LAG: INDIZEA;
begin
   X := S(S'FIRST);
   BP_LAG:= S'FIRST;
   for  EZEZAGUNA in  S'FIRST+1 .. S'LAST  loop
          if  S(EZEZAGUNA) < X then
                BP_LAG:= BP_LAG+1;
                TRUKEA(S(BP_LAG), S(EZEZAGUNA));
          end if;
   end loop;
   TRUKEA_POS(S'FIRST, BP_LAG);
   BP := BP_LAG;
end BANAKETA;
```

Algoritmoaren ordenaren analisia:

Algoritmo honek, edozein kasutan, lehenengo osagaia gainontzeko guztiekin alderatzen duenez, guztira (n-1) alderaketa egiten ditu sekuentziako osagaien artean. Beraz, lineala da.

Quicksort algoritmoaren ordenaren analisia:

Banaketa algoritmoaren ordena ezaguna denez (kasu honetan ordena taulako osagaien arteko alderaketen kopuruak ematen digu), orain, Quicksort algoritmoaren ordena kalkula dezakegu. Quicksorten kasu desberdinak aztertuko ditugu:

Kasu txarrenaren analisia:

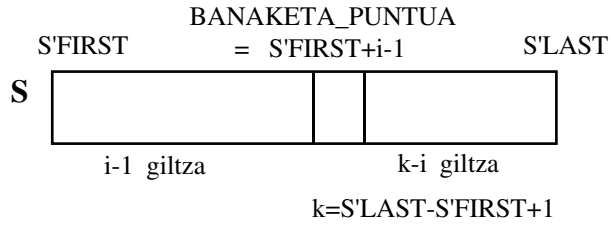
BANAKETA algoritmoak giltza bakoitza *x*-ekin alderatzen du. Beraz, lantzen ari garen taularen sekzioan *k* giltza badaude, *k-1* alderaketa egingo ditu. Gainera, zenbat eta handiagoak izan taulako segmentuak, orduan eta handiagoa izango da egin beharko diren alderaketen kopurua. Ondorioz, S(S'FIRST) banatzen ari den sekzioko giltzarik txikiena bada, orduan Banaketa_Puntua = S'FIRST da, eta azken batean egingo den gauza bakarra lista bitan banatzea da: sekzio hutsa batetik (*x* baino txikiagoak direnak) eta *k-1* giltzez osatutako sekzioa bestetik (*x* baino handiagoak direnak bestetik). Honela, BANAKETA algoritmoari deitzen zaion bakoitzean gauza bera gertatzen bada (alegia, *x* giltzarik txikiena izatea), egiten diren alderaketen kopurua ondorengo hau da:

$$T_t(n) = \sum_{k=2}^{n} (k-1) = \frac{n(n-1)}{2} \in \Theta\left(n^2\right)$$

Hala bada, QUICKSORT algoritmoa kasu txarrenean BURBUILA eta TXERTAKETA algoritmoak bezain txarra dugu. Ordenazio algoritmo honen kasurik txarrena harrigarri samarra da, sarrera beheranzko edo goranzko ordenan dagoenean gertatzen baita.

Batezbesteko kasuaren analisia:

Aurreko atalean ikusi dugun bezala, ordenazio algoritmoak alderaketa bakoitzaren ondoren alderanzketa bat ezabatzen badu giltzen permutazio batetik, orduan gutxienez $(n^2-n)/4$ alderaketa egin beharko ditu batezbestean permutazio bat ordenatzeko. Quicksortek berriz ez du murriztapen hau. BANAKETA algoritmoak giltza bat sekzio luze batean zehar mugi dezake: trukatze batek 2n-2 inbertsio ken ditzake. Portaera honi esker lortu du bere izena.



Giltza guztiak desberdinak direla suposatuko dugu, bai eta giltzen permutazio guztiak ekiprobableak direla ere. Izan bedi *k* ordenatu beharreko sekzioko giltzen kopurua, eta izen bedi $T_b(k)$ eginiko giltzen arteko konparaketen batezbesteko kopurua *k* luzerako listetan. Suposa dezagun, BANAKETA algoritmoa exekutatzen den ondorengo aldian, *x i*.go posizioan kokatzen duela. BANAKETA algoritmoak, k-1 alderaketa egin ditu; ondoren ordenatu beharko diren bi azpisekuentziek *i-1* eta *k-i* giltza dituzte hurrenez hurren. *i* banaketa-puntuaren posizioa edozein izan daiteke, posizio guztiak ekiprobableak baitira (beraz, 1/k). k =n egiten badugu, ondorengo erlazioa lortzen dugu:

$$T_b(n) = (n-1) + \sum_{i=1}^{n} \frac{1}{n} \left(T_b(i-1) + T_b(n-1) \right) \quad \text{baldin } n \geq 2$$

$$T_b(1) = T_b(0) = 0$$

Batugaiak aztertuz, eta sinplifikatu ondoren, erlazio hau lortzen dugu:

$$T_b(n) = (n-1) + \frac{1}{n} \sum_{i=2}^{n-1} 2T_b(i)$$

Errekurtsio-ekuazio hori orain arte ikusitako errekurtsio-ekuazioak baino konplexuagoa da. Ekuazioa askatzeko hauetako teknikaren bat erabil dezakegu:

– Emaitzaren itxura jakinda, honen suposaketa egin dezakegu eta ondoren indukzio bidez baieztatu. Teknika hau guztiz egokia gerta daiteke algoritmoa errekurtsiboa bada,

– Injenio pixka bat erabili.

Biak ikusiko ditugu.

• Nola jakin $T_b(n)$-k zer itxura izango duen? Honi buruz suposizio bat osa ahal izateko, kontsidera dezagun QUICKSORTek nahiko ondo lan egiten duen kasu bat. Suposa dezagun BANAKETA exekutatzen den aldi bakoitzean, honek lista luzera berdina duten bi azpilistatan banatzen duela. Estimazio bat egitea lagungarri gerta dakiguke QUICKSORTek batezbesteko kasuan zein azkar lan egiten duen jakiteko. Honela, bi azpisekuentzien luzera n/2 dela estima dezagun (ez zaigu axolako n/2 osokoa den ala ez). Egiten diren alderaketen kopurua ondorengo erlazioan deskribatzen da:

$$Q(n) \approx n - 1 + 2Q\left(\frac{n}{2}\right)$$

$$Q(1) = 0$$

Errekurtsio-ekuazioa garatuz ondorengo erlazio hau lortzen dugu:

$$Q(n) \approx n-1+2Q\left(\frac{n}{2}\right)= n-1+2\left(\frac{n}{2}-1\right)+4Q\left(\frac{n}{4}\right)= n-1+n-2+n-4+8Q\left(\frac{n}{8}\right)=$$

$$\approx in-\sum_{j=1}^{i} 2^{j-1}+2^{i}Q\left(\frac{n}{2^{i}}\right)\approx$$

$$= n \lg n - n + 1 \in O(n \lg n) \qquad (1)$$

Ondorioz: $Q(n) \in O(n \lg n)$

Honela, sekuentzia bitan banatzen den aldi bakoitzean S(S'FIRST) balio zentraletik hurbil badago, QUICKSORTek egindako alderaketen kopurua O(n ln n) ordenakoa izango litzateke. Kopuru hau $O(n^2)$ baino dexente hobea da. Baina permutazio guztiak ekiprobableak badira ere, ba al dago hainbeste kasu batezbesteko ordena hobetzen duenik? Hau da, batezbesteko ordena $O(n^2)$ baino hobea da? Halaxe gertatzen dela frogatuko dugu.

Teorema

$n \geq 1$ denean, algoritmoak hartzen duen batezbesteko denbora, $T_b(n)$, $O(k\ n\ \ln\ n) = O(n\ \ln\ n)$ ordenakoa da, non k konstantea den eta n sarreraren tamaina den.

Hau da: $T_b(n) \leq k\ n\ \ln\ n$. (2)

Frogaketa: n gainean indukzioa erabiliz

– n= 1 denean: $T_b(1)= 0$ eta $k\ 1\ \ln\ 1= 0$.
– n> 1 denean: orain errekurtsio-ekuazioa eta indukzio hipotesi garatua
$\forall i\ (1 \leq i < n \rightarrow T_b(i) \leq k\ i\ \ln\ i)$ erabiliz:

$$T_b(n) = n-1+\frac{2}{n}\sum_{i=2}^{n-1} T_b(i) \leq n-1+\frac{2}{n}\sum_{i=2}^{n-1} k\ i \ln i$$

(1). Hemendik aurrerantzean lg n laburtzapena erabiliko dugu 2 oinarriarentzat. Hala ere, gogora bedi logaritmo guztiak ordena berekoak direla, oinarri batetik besterako bihurketa konstante bat biderkatuz lortzen baita.

(2). Jakinik $\log_2 n = (\ln n)/(\ln 2)$ betetzen dela, eta (ln 2) azken batean konstantea denez, k = (ln 2)•k' eginik, logaritmoak bi oinarrian erabili ordez logaritmo naturalak erabil ditzakegu frogaketako kalkuluak sinplifikatuz.

k konstante batentzat. Batura integral baten bidez borna dezakegu:

$$\sum_{i=2}^{n-1} k\, i \ln i \le \int_{2}^{n} x \ln x \, dx$$

Atalak integratuz:

$$\int_{2}^{n} x \ln x \, dx = \left[\frac{x^2}{2}\ln x\right]_2^n - \left[\frac{x^2}{4}\right]_2^n = \left[\frac{x^2 \ln x}{2} - \frac{x^2}{4}\right]_2^n = \frac{n^2 \ln n}{2} - \frac{n^2}{4} - 2\ln 2 + 1$$

Beraz,

$$T_b(n) \le n - 1 + \frac{2k}{n}\left[\frac{n^2 \ln\, n}{2} - \frac{n^2}{4} - 2\ln 2 + 1\right] = kn \ln n - 1 + \left[1 - \frac{k}{2}\right] + \frac{2k}{n}(1 - 2\ln 2)$$

$T_b(n) \le k\ n \ln n$ betetzen dela azaltzeko, 3. eta 4. batugaiak negatiboak edo zero direla adieraztearekin nahikoa da. 3. batugaia zero edo eta hau baino txikiagoa da k≥2-rentzako. 4. batugaia, berriz, beti negatiboa da ln 2= 0.7 baita. Ondorioz, k=2 bada, $T_b(n) \le k\ n \ln n$ betetzen dela frogatu dugu: $T_b(n) \le 0.7\ n \ln n$.

Edo, gauza bera, baina logaritmoa bi oinarriakoa bada: $T_b(n) \le 1.4\ n \log_2 n$.

• QUICKSORTen portaera batezbesteko kasuan finkatu badugu ere, itzul gaitezen ekuazio errekurtsibora eta saia gaitezen emaitza bera lortzen baina oraingoan garapena zuzenki eginaz:

$$T_b(n) = n - 1 + \frac{2}{n}\sum_{i=2}^{n-1} T_b(i) \qquad (1)$$

$$T_b(n-1) = n - 2 + \frac{2}{n-1}\sum_{i=2}^{n-2} T_b(i) \qquad (2)$$

(1). ekuazioko batukariari (2). ekuazioko batukaria kentzen badiogu, batugai gehienak sinplifikatu egiten dira. Bestalde, batugaiak faktore desberdinez biderkaturik daudenez gero, algebra konplexuago bat beharko genuke. Guk informalki ondorengo hau kalkulatuko dugu:

$$n \times Ek(1) - (n-1) \times Ek(2)$$

Hau da:

$$nT_b(n)-(n-1)T_b(n-1)=n(n-1)+2\sum_{i=2}^{n-1}T_b(i)-(n-1)(n-2)+$$

$$+2\sum_{i=2}^{n-2}T_b(i)=2T_b(n-1)+2n-2$$

Kalkuluak egiten jarraituz:

$$\frac{T_b(n)}{n+1}=\frac{T_b(n-1)}{n}=\frac{2n-2}{n(n+1)}$$

Orain, izan bedi T_{aux}:

$$T_{aux}(n)=\frac{T_b(n)}{n+1} \quad \text{eta} \quad T_{aux}(1)=0$$

Honela, T_{aux}-en errekurtsio-ekuazioa:

$$T_{aux}(n)=T_{aux}(n-1)+\frac{2n-2}{n(n+1)}$$

Azkenik, espero genuen bornaketa lortzen dugu:

$$T_{aux}(n)=\sum_{i=2}^{n}\frac{2i-2}{i(i+1)}\approx 2\ln n$$

$$T_b(n)\approx 2(n+1)\ln n\approx 1.4(n+1)\lg n\in O(n\lg n)$$

Quicksort algoritmoaren memoria-espazio estra:

Algoritmoa errekurtsiboa dela eta, dei errekurtsiboek parametroen pilaren tamaina handituko dutela erraz ikus daiteke. Zenbat eta dei errekurtsibo gehiago habiatuta egon, orduan eta handiagoa izango da pilaren tamaina. Kasu hori Banaketa_Puntua sekuentziaren ertz batean kokatzen denean eta ondorengo deia zati handienarekin egiten denean gertatzen da (posibleak liratekeen dei-zuhaitz guztietatik, aipatutako kasu horretan osatzen den zuhaitzak izango du adar luzeena, adar horrek pilaren tamainarik handiena emango duelarik).

Gaiaren hasieran esan den bezalaxe, taularen kopiarik ez dela egingo suposatuko dugu, konpiladoreak seguruenik sekuentzia erakusten duen erakusle bat pasako baitu; hots, erreferentzia bidez pasako du. Suposa dezagun, hain zuzen,

erakusleak pilan beharko den memoria-espazioa *a*-ren tamainakoa dela. Ondorioz:

MEE(n)= a + MEE(n-1)

Faktoriala funtzioaren lana kalkulatzen duen errekurtsio-ekuazioaren berdina da. Beraz, ordena lineala da.

2.4.1.2. QUICKSORT iteratiboa

Bertsio iteratiboak pila egitura erabiliko du, prozesua honakoa izanik: banaketari dei egin ondoren, ordenatu behar den taula bi azpitartetan banatzen da. Prozedura iteratiboak bigarrengo azpitartearen lantzea atzeratzeko azpitarte hori pilan metatzen du, eta ondoren zuzenean lehengo azpitartearekin jarraitzen du lanean. Lantzen ari den azpitarteak osagairik ez duenean, pilatik azpitarte berriaren indizeak ateratzen ditu eta banaketarekin jarraitzen du.

```
procedure QUICKSORT_IT (S: in out OS_SEK; BP: out INDIZEA) is
   X: INTEGER;
   Banaketa_Puntua,Hasiera,Amaiera: INDIZEA;
   P: PILA_MOTA;
begin
   PILA.PILA_HUTSA(P);
   PILA.PILARATU(P, S'FIRST, S'LAST);
   while not PILA.HUTSIK_DAGO_P?(P)  loop
          PILA.PILATIK_ATERA(P, Hasiera, Amaiera);
          while  Hasiera < Amaiera  loop
                   BANAKETA( S(Hasiera..Amaiera), Banaketa_Puntua);
                   PILA.PILARATU(P, Banaketa_Puntua+1, Amaiera);
                   Amaiera:= Banaketa_Puntua-1;
          end loop;
   end loop;
end QUICKSORT_IT;
```

Aurreko algoritmoa beti lehengo azpitartearekin saiatzen da lehendabizi. Hau ez da oso azkarra. Hobea da tarte handienarekin lanean jarraitzea, eta tarte txikienaren indizeak pilaraztea. Prozesu hau jarraituz gero, pilaren tamainaren hazkuntza gehienez logaritmikoa izatera irits daitekeela froga dezake irakurleak.

```
procedure QUICKSORT_IT (S: in out OS_SEK; BP: out INDIZEA) is
   X: INTEGER;
   Banaketa_Puntua,Hasiera,Amaiera: INDIZEA;
   P: PILA_MOTA;
begin
   PILA.PILA_HUTSA(P);
```

```
    PILA.PILARATU(P, S'FIRST, S'LAST);
    while not PILA.HUTSIK_DAGO_P?A(P)  loop
          PILA.PILATIK_ATERA(P, Hasiera, Amaiera);
          while  Hasiera < Amaiera  loop
                BANAKETA( S(Hasiera..Amaiera), Banaketa_Puntua);
                if  (Banaketa_Puntua - Hasiera) >
                     (Amaiera - Banaketa_Puntua) then
                     PILA.PILARATU(P, Hasiera, Banaketa_Puntua-1);
                     Hasiera:= Banaketa_Puntua+1;
                else
                     PILA.PILARATU(P, Banaketa_Puntua+1, Amaiera);
                     Amaiera:= Banaketa_Puntua-1;
                end if;
          end loop;
    end loop;
end QUICKSORT_IT;
```

Bigarren Quicksort iteratiboaren memoria-espazioa $\Theta(\lg n)$ da.

2.4.1.3. Ondorioak

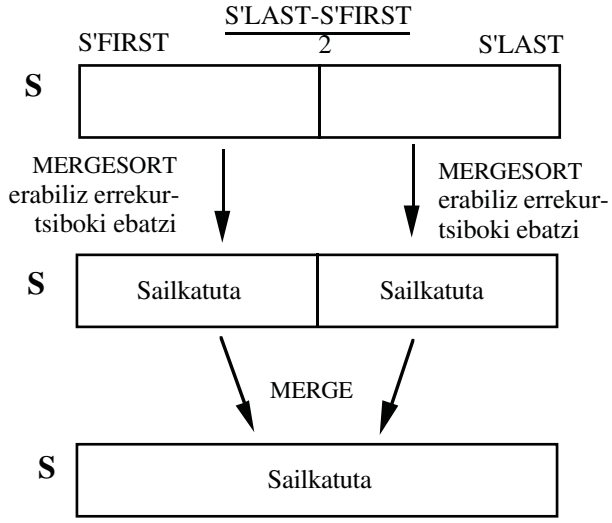
Praktikan, *n* handia bada ere batezbesteko kasuan QUICKSORTen inplementazioak nahiko azkar exekutatzen dira. Hauen erabilera oso hedatuta dago. Baina kasu txarrenean, QUICKSORT motela da. BURBUILAk eta TXERTATUk bezala, kasu txarrenean hartzen duten denbora, $T_t(n)$, $\Theta(n^2)$ ordenakoa da. QUICKSORTek batezbesteko kasuan hartzen duen denbora O(n ln n) ordenakoa da. Gure ondorengo galderak hauek izan daitezke: ba al dago ordenazio algoritmikorik kasu txarrenean hartzen duen denbora $\Theta(n \ln n)$ ordenakoa dena? Edo eta, $O(n^2)$ ordena baino txikiagoa? Oraingoz, zatitu eta irabazi teknika erabiliz hobera egin dugu batezbesteko kasuan.

Ikus dezagun teknika bera erabiliz kasu txarrenean orain arte lortutako goi bornea txikitu dezakegun ala ez.

2.4.2. Bateraketa bidezko ordenazioa: MERGESORT

Mergesorten prozesua:

Berriz ere, ordena gorakorrean sekuentzia bat ordenatzea da gure helburua. Sarrerako sekuentzia tamaina berdina duten bi azpisekuentziatan banatzen du. Sekuentzia berriak errekurtsiboki ebazten ditu. Honela, ordena gorakorrean ordenaturiko bi azpisekuentzia lortzen ditu. Ondoren, Merge izeneko algoritmoari pasatzen dizkio bi azpisekuentziak, honek ordena gorakorrean dagoen sekuentzia bakar bat itzuliz. MERGESORTen gakoa dagoeneko ordenatuta dauden listen bateraketan dago.



Merge algoritmoa

Mergesorten giltzarria honakoa da: gorantz ordenaturik dauden L1 eta L2 bi lista emanik, biak elkartu egiten ditu goranzko L3 sekuentzia osatzeko.

MERGE edo BATERATU algoritmo batek egindako lana ez da giltzen alderaketen kopuruaren bidez neurtu behar.

```
procedure  MERGE1  (L1,L2: in OS_SEK; L3: out OS_SEK)  is
    Ind_L1: INTEGER: = L1'FIRST;
    Ind_L2: INTEGER: = L2'FIRST;
    Ind_L3: INTEGER: = L3'FIRST;
begin
    while  Ind_L1≤L1'LENGTH and Ind_L2≤L2'LENGTH  loop
          if  L1(Ind_L1)<L2(Ind_L2)  then
                L3(Ind_L3):= L1(Ind_L1);
                Ind_L1:=Ind_L1 +1;
          else
                L3(Ind_L3):=L2(Ind_L2);
                Ind_L2:=Ind_L2 +1;
          end if;
              Ind_L3:=Ind_L3 +1;
    end loop;
    if Ind_L1>L1'LENGTH then
          –move L2(Ind_L2),...,L2(L2'LENGTH) to
          –        L3(Ind_L3),...L3(L1'LENGTH+L2'LENGTH)
          for I in Ind_L2.. L2'LENGTH loop
```

```
                L3(Ind_L3):=L2(I);
                Ind_L3:=Ind_L3+1;
        end loop;
    else
        J:= Ind_L3;
        — move L1(Ind_L1),...,L1(L1'LENGTH) to
        —        L3(Ind_L3),...L3(L1'LENGTH+L2'LENGTH)
        for I in Ind_L1.. L1'LENGTH loop
                L3(Ind_L3):=L1(I);
                Ind_L3:=Ind_L3+1;
        end loop;
    end if;
    return L3;
end MERGE1;
```

Azter dezagun MERGE1en portaera kasu txarrenean.

Ordenatuta dauden bi listetako giltza bana alderatzen direnean, aldi bakoitzean gutxienez giltza bat kopiatzen da L3 lista berrian, mugituriko giltza berriro sekulan aztertuko ez balitz. Azkeneko alderaketa egin ondoren, gutxienez bi giltza (alderatutako bi giltzetatik txikiena eta gelditzen diren gainontzeko giltzak) L3 lista berrira mugitzen dira. Kasu txarrena, N1+N2-1 alderaketa egiten denean gertatzen da, hain zuzen, L1(N1) eta L2 (N2) L3-ren azken bi posiziotan kokatzen direnean. N1=N2 bada, orduan kasu txarrenean 2N1-1 alderaketa egiten da (hots, N1=L1'LENGTH, N2=L2'LENGTH eta N1+N2=N3=L3'LENGTH ditugu). Hala ere, L3-ra mugitutako giltzen kopurua N3 da beti.

Baina, zer gertatzen da batezbesteko kasuan? Ez da hainbesteko giltza alderaketa egiten, baina L3-an kopiatzen diren giltzen kopurua berdin mantentzen da. Beraz, badirudi alderaketen kopurua neurtzea ez dela MERGEk egiten duen lanaren adierazgarri ona.

Algoritmo bera beste inplementazio honen bidez ere eraman dezakegu aurrera:

```
procedure  MERGE  (L1,L2: in OS_SEK; L3: out OS_SEK)  is
    Ind_L1: INDIZEA:= L1'FIRST;
    Ind_L2: INDIZEA:= L2'FIRST;
begin     — ordena_<(L1) ∧ ordena_<(L2)
    for  Ind_L3  in 1..L1'LENGTH+L2'LENGTH  loop
         if  (Ind_L1<L1'LAST) and (Ind_L2<L2'LAST)  then
              if  L1(Ind_L1)<L2(Ind_L2)  then
                   L3(Ind_L3):=L1(Ind_L1);
              else
                   L3(Ind_L3):=L2(Ind_L2);
```

```
                end if;
            elsif  (Ind_L1>L1'LAST)  then
                L3(Ind_L3):=L2(Ind_L2);
            else
                L3(Ind_L3):=L1(Ind_L1);
            end if;
        end loop;
        — ordena_<(L3)
    end MERGE;
```

Hemen, MERGEk egiten duen lana erraz kalkula dezakegu:

For-aren gorputza konstante batez muga dezakegu asignazioak eta alderaketa kopuru mugatu bat besterik ez baita egiten. For-a, berriz, n aldiz egiten da. Beraz, MERGEk hartzen duen denbora: $T(n) = k_* n \in \Theta(n)$, non n=N1+N2 den. Hots, kasu honetan egindako mugimenduen kontaketa ere egindako lanaren adierazgarri ona litzateke.

QUICKSORTen barneko BANAKETA algoritmoak lortzen dituen bi azpisekuentziek normalki ez dute luzera berdina izaten, eta hau ez da komenigarria. MERGESORTek berriz, ordenatu beharreko sekuentzia luzera berdina duten bi azpisekuentziatan banatzen du, eta ondoren, bi azpisekuentziak bakarka sailkatzen ditu (eta noski, errekurtsiboki). Ondoren, ordenatutako azpisekuentziak elkartzen ditu sekuentzia ordenatu bakarra lortuz. Sekuentzia bi azpisekuentziatan banatzeko prozesuak ez du orain giltza alderaketarik egiten. Giltza alderaketa guztiak MERGE algoritmoan egiten dira. MERGE algoritmoa auzokideak diren taula baten bi azpisekzioak elkartzeko eraldatu dela suposatuko dugu. Honela, MERGEK, jarraian eta taula berean dauden tamaina berdineko bi azpisekuentzia ordenatuak sekuentzia ordena berdineko bihurtzen ditu, emaitza taula berean itzuliz.

Mergesort algoritmoa:

```
procedure  MERGESORT (S: in out OS_SEK)  is
    S_LAG: OS_SEK(S'RANGE);
    — Sren sekuentziaren luzera 2ren potentzia dela suposatzen da
begin — Ss= <a1, ...,an,b1,...,bn>= Sa·Sb
    if  S'LENGTH>1  then
        MERGESORT (S(S'FIRST..((S'LAST-S'FIRST)/2)));
        — permutazioa(Sa, S(S'FIRST..((S'LAST-S'FIRST)/2))) ∧
        — ordena_<(S(S'FIRST,...,(S'LAST-S'FIRST)/2))
        MERGESORT (S(S'LAST-S'FIRST)/2).. S'LAST));
        — permutazioa(Sa, S(S'FIRST..((S'LAST-S'FIRST)/2))) ∧
        — ordena_<(S(S'FIRST,...,(S'LAST-S'FIRST)/2))∧
```

```
            — permutazioa(Sb, S(S'LAST-S'FIRST)/2).. S'LAST)) ∧
            — ordena_<(S(S'LAST-S'FIRST)/2).. S'LAST))
            MERGE (S(S'FIRST..(S'LAST-S'FIRST)/2),
                   S(S'LAST-S'FIRST)/2).. S'LAST),
                   S_LAG);
            S(S'RANGE):= S_LAG(S'RANGE);          — Ada lengoaiaren mugak
        end if;
    — permutazioa(Ss, Si) ∧ ordena_<(Si)
    end MERGESORT ;
```

Mergesort algoritmoaren ordenaren analisia:

Algoritmoa errekurtsiboa denez, egindako lana errekurtsio-ekuazio batez emango dugu. Lana, oraingoan, alderaketen kopurua neurtuz kalkulatu ordez, mugimenduen kopurua zenbatuz kalkulatu dugu. Beste posibilitate bat denbora kalkulatzea litzateke (hots, oinarrizko eragiketak bider hauetako bakoitzak hartuko lukeen denbora).

$$T(n) = T\left(\left\lfloor \frac{n}{2} \right\rfloor\right) + T\left(\left\lceil \frac{n}{2} \right\rceil\right) + n = 2T\left(\frac{n}{2}\right) + n$$

$$T(1) = 0$$

QUICKSORTen kasurako egin genuen analisi informalaren antzekoa da.

Edo eta errekurtsioa hedatuz:

$$T(n) = 2T\left(\frac{n}{2}\right) + n = 2\left(2T\left(\frac{n}{4}\right) + \frac{n}{2}\right) + n = 4T\left(\frac{n}{4}\right) + 2n =$$

$$= 4\left(2T\left(\frac{n}{8}\right) + \frac{n}{4}\right) + 2n = 8T\left(\frac{n}{8}\right) + 3n =$$

$$= 2^i T\left(\frac{n}{2^i}\right) + in \qquad \text{i. pausoan}$$

2i = n eginik; hau da, i = lg n

$$= nT(1) + n \lg n \in \Theta(n \lg n)$$

Beraz, MERGESORT algoritmoak bai kasu txarrenean bai batezbesteko kasuan Θ(n lg n) ordenako denbora behar du. Hau da, orain arteko algoritmo onena dugu.

Baina, MERGESORTek, espazio estra behar du.

Mergesort algoritmoaren memoria-espazio estra:

Berriz ere, taula erakusten dion erakusle bat besterik ez duela konpilatzeak pasako suposatuko dugu analisia egiteko. Bestalde, demagun dei-errekurtsiboak kudeatzeko behar den pilako espazioa *a* dela eta S_LAG taula laguntzaileak behar duen espazioa *n* (sarrerako n osagaien permutazio ordenatua jaso behar baitu). Beraz, MERGESORTek kontsumituko duen memoria-espazio estra momentu batean gehienez honakoa da::

$$MEE(n)= a + n + MEE(n/2) \in \Theta(n)$$

2.5. HEAPSORT

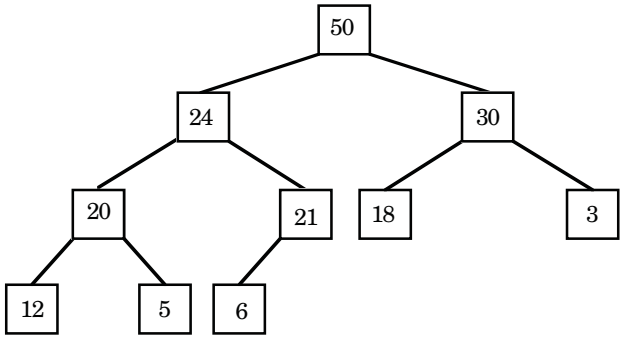
2.5.1. Meta datu-egitura

Meta, taula baten gainean eraginkorki inplementaturik dagoen zuhaitz bitar berezia da. Egitura honek aplikazio ugari du: famatua den ordenazio algoritmo batean, Willias-en ordenazioan eta lehentasun dinamikoa duen hainbat listaren inplementazio eraginkorretan.

Zuhaitz bitar bat **ia-betea** dela esango dugu, bere barneko adabegi bakoitzak zehazki bi ume baditu, ezkerra eta eskuina, gerta daitekeelarik azkenaurreko mailan kokaturik dagoen barne-adabegi berezi batek ume bakar bat izatea, ezkerra hain zuzen. Gainera, hosto guztiak azken edo azkenaurreko mailan kokaturik daude, eta kasu honetan, azkenaurreko mailan kokaturik dagoen edozein hosto, ez da sekulan maila berdineko barne-adabegi baten ezkerraldean egongo. Barne--adabegi berezi honek existitzen badu, azkenaurreko mailako beste edozein barne--adabegi baino eskuinaldeago kokaturik egongo da.

Meta, ia-betea den zuhaitza da, non adabegi bakoitzak *giltza* deritzon informazio bat erlazionaturik duen. Bi motako metak daude: maximoen metak eta minimoen metak. Honela:

Meta batean, barne-adabegi bakoitzeko giltza bere umeen giltzak baino handiagoa edo berdina bada, orduan meta *maximoen meta* da. Hau, maximoen meten propietatea bezala ezagutzen da.



Maximoen meta baten adibidea

Aldiz, meta bateko barne-adabegi bakoitzeko giltza bere umeen giltzak baino txikiagoa edo berdina bada, orduan meta *minimoen meta* da.

Atal honetan maximoen metak bakarrik erabiliko ditugu.

Metaren adierazpena: Mota honetako zuhaitzak, taula baten bidez adieraz daitezke, k sakonera mailan dauden adabegiak taulako 2^k, $2^k+1,\ldots,2^{k+1}-1$ posizioetan biltegiratzen baditugu (azkeneko mailan ezik, izan ere maila hori osorik ez egotea gerta daiteke).

$\forall i>1$-etarako, M(i)-an adierazirik dagoen adabegiaren gurasoa $M\left(\left\lfloor \frac{i}{2} \right\rfloor\right)$ da; eta honen umeek existituz gero, M(2 i) eta M(2 i +1) dira.

Aurreko adibideko meta, taula baten bidez honela adieraz dezakegu:

50	24	30	20	21	18	3	12	5	6

Beraz, i posizioan metatutako balioaren gurasoa $\left\lfloor \frac{i}{2} \right\rfloor$ posizioan dago eta ezkerreko eta eskuineko umeak 2i eta 2i+1 posizioetan hurrenez hurren.

Ikus dezagun adabegien gaineko **hainbat eragiketa**:

Metaren propietatearen ondorioz, egitura hau eragiketa batzuk egiteko guztiz aproposa da. Zenbait eragiketaren artean ondorengo hauek aipa ditzakegu: maximoa aurkitu, erroa kendu, adabegi bat erantsi edo eguneratu, taula bat emanik meta eraiki, e.a. Hain zuzen, lehentasun dinamikoa duen lista baten adierazpen eraginkor bat aurrera eraman ahal izateko behar diren eragiketak dira, eta adabegiaren balioak dagokion gertakizunaren lehentasuna adierazten du. Kasu hauetan, erroan lehentasun handiena duen gertakizuna egongo da, lehentasun hau dinamikoki alda daitekeelarik.

• Adabegi bat eguneratu ondoren, metaren propietatea eraginkorki eraberritzea datu-egitura honen oinarrizko ezaugarria izango da. Honela, adabegi baten balioa bere gurasoaren balioa gainditzeraino handitzen bada, bi balio horiek trukatu egin beharko ditugu eta prozesu bera egiten jarraitu, metaren propietatea berreskuratu arte. Hau gertatzen denean, eguneratu den balioa bere posizio berrira arte *azaleratu* dela esaten da. Ondorengo algoritmoak meta bateko azkeneko osagaia azaleratzen du:

```
procedure   AZKENA_AZALERATU    (SM: in  META; G: in GILTZA;
                                 IM: out META);
    Ind_K, Ind_J: INDIZEA;
    M_LAG: META(SM´FIRST..SM'LAST+1); — Ada prog. lengoaiaren mugak.
    —  SM metari G osagaia sakonera mailan eta azkeneko hostoaren senide bezala erasten
    —  dio lehendabizi. Ondoren, G azalareratzen du SM-ko osagaiek eta G osagaiak meta
    —  osatzen duten arte. Meta berria IM-n itzultzen du.
    —  SM-ko osagaiek meta propietatea betetzen dute.
begin
    ERANTSI(SM, G, M_LAG);      — M_LAG taulan lehengo osagaiak
                                — SMkoak dira (eta ordena berean)
                                — eta azkena G da.
    Ind_K:= M_LAG'LAST;
    loop
          Ind_J := Ind_K;
          if  M'FIRST < Ind_J  and then  S(Ind_J/2)< S(Ind_K)
          then  Ind_K:= Ind_J/2;
          end if;
          TRUKEA(M(Ind_J), M(Ind_K));
          exit when Ind_J = Ind_K;
    end loop;
    IM:= M_LAG;
end AZKENA_AZALERATU;
```

• Eguneratutako adabegiak, berriz, umeren baten balioa baino balio txikiago bat eskuratzen badu, oraingo honetan adabegi honen balioa bere umeen baliorik handienarekin trukatu beharko da, eta gainera, metaren propietatea berreskuratu arte prozesu bera beherantz hedatu beharko da. Hau gertatzen denean, eguneratu den balioa bere posiziora arte *hondoratu* dela esaten da.

```
procedure HONDORATU (M: in OS_SEK) is
    J, Ume_Max: INDIZEA ; H_Giltza : OSAGAIA;
    — prozedura honek erro adabegia metaren propietatea berreskuratu arte
    — hondoratzen du.
```

```
begin
    J := M'FIRST; H_Giltza:= M(J);
    while  2·J ≤ M'LAST  loop
          — J adabegiaren umerik handiena bilatzen du.
          Ume_Max := 2·J ;
          if  2·J < M'LAST  and then  M(2·J+1) > M(2·J)  then
                Ume_Max := 2·J+1;
          end if;
          if  H_Giltza < M(Ume_Max) then
                M(J) := M(Ume_Max);
                J := Ume_Max;
          else
                exit  when  true;
          end if;
    end loop;
    — adabegia dagokion posizioraino hondoratu da.
    M(J) := H_Giltza;
end  HONDORATU;
```

• Eguneratu:

```
procedure META_EGUNERATU (M: in out META; I: in INDIZEA;
                          G: in GILTZA; );
```

Sarrera: G giltza, M meta eta bertako I indizea.

Prozesua: M meta bat izanik, M(I)-ri G balioa ematen zaio; ondoren, metaren propietatea berreskuratu G balioa hondoratu edo azaleratu egiten da, azkenean M-n meta lortuz .

• Taula bat emanik meta bat eraikitzeko bi aukera ikusiko ditugu:

1 hurbilketa:

Ideia: k osagai dituen meta bati osagai bat atzetik eransten badiogu, eta ondoren hau, azaleratzen badugu, k+1 dituen meta bat lortzen da. Metak osagai bat duen unetik, aurreko prozesua n-1 aldiz errepikatzen bada, n osagai dituen meta bat lortuko dugu:

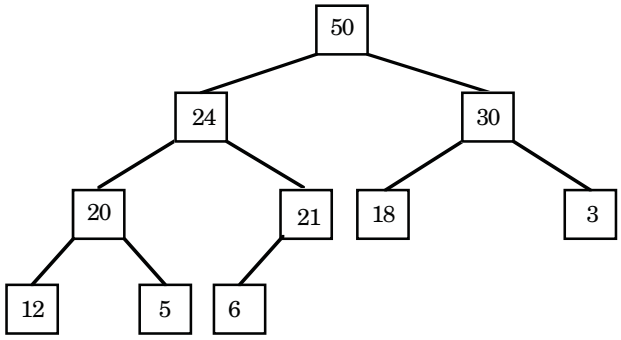
```
procedure  META_ERAIKI1 (T:in  OS_SEK: M:out META) is
begin
    for  K in T'FIRST+1..T'LAST loop
          AZKENA_AZALERATU(T(T'FIRST,..,K-1),T(K),T(T'FIRST,..,K));
    end loop;
end META_ERAIKI1;
```

2 hurbilketa:

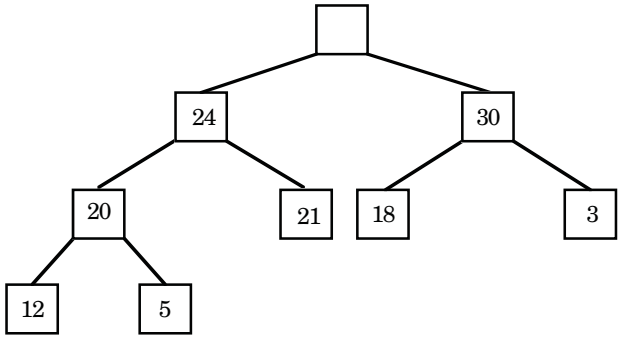
Ideia: taulak n osagai baditu, azkeneko n/2 osagaiek eraikiko dugun metaren hostoen posizioak adierazten dituzte. Hostoak adabegi bakarreko zuhaitzak dira eta metaren propietatea betetzen dute. Hori dela eta, metaren barne-adabegiak aztertuko ditugu, eta alderantzizko ordenean gainera. Barne-adabegi bat aztertzen denean, bere ezkerreko eta eskuineko zuhaitzak metak badira eta barne-adabegi hori dagokion posizio erlatiboraino hondoratzen badugu, meta berri bat lortuko dugu. Barne-adabegi guztiekin alderantzizko ordenean prozesu hau errepikatuz, hau da, n/2 aldiz, n osagai dituen meta bat eraikiko dugu .

```
procedure  META_ERAIKI2 (T:in  OS_SEK: M:out META) is
    T_LAG: OS_SEK:=T;
begin
    for I  in reverse  T_LAG'FIRST.. T_LAG'LAST/2  loop
          HONDORATU(T_LAG(I..T_LAG'LAST));
    end loop;
    M:=T_LAG;
end META_ERAIKI2;
```

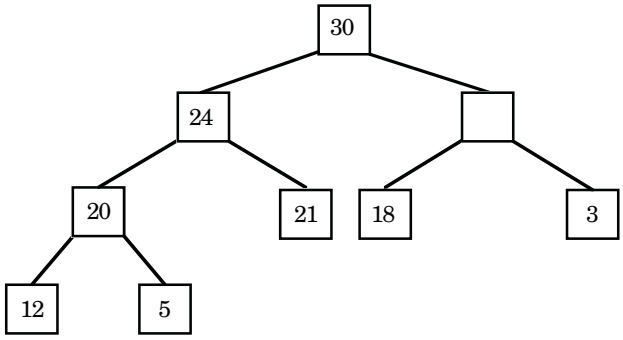
• Meta bati adabegi berri bat eransten zaionean, hau ezkerretik eskuinaldera hutsik dagoen lehenengo lekuan kokatu behar da; ezabatzeko garaian berriz, eskuinaldeen dagoen adabegia bakarrik ezaba dezakegu. Bai eranstean eta bai ezabatzean gelditzen den egiturak meta izan behar du.



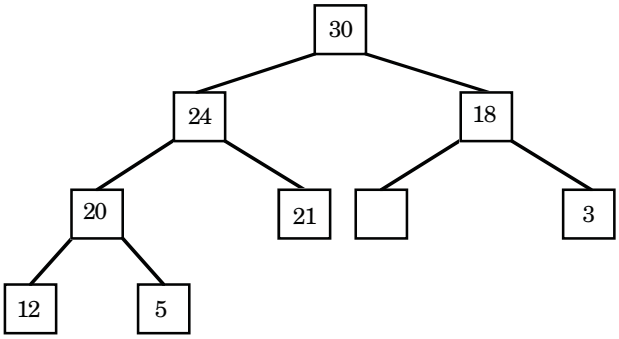
Hasierako meta



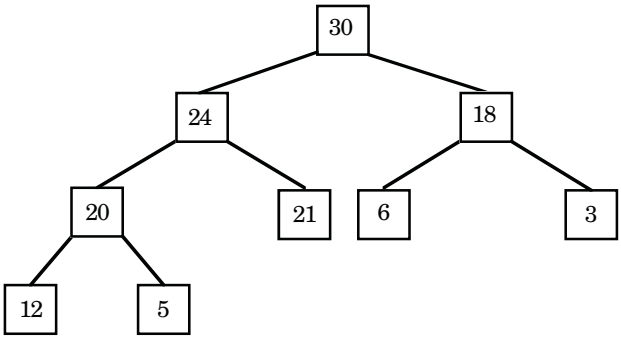
Erroa kentzen da; azkeneko mailan eskuinean dagoen adabegia kentzen da (I aldagaia, kasu honetan 6)



Hutsik gelditu den adabegiaren ume handiena hondoratzen ari garen balioa baino handiagoa denez, gurasoaren posiziora azaleratzen da, bere lekua hutsik geldituz.



Hutsik gelditu den adabegiaren umerik handiena I giltza baino handiagoa denez, gurasoaren posiziora azaleratzen da, bere lekua hutsik geldituz.



Hutsik dagoen adabe-gia hosto bat denez, I alda-gaiaren balioarekin betetzen da.

2.5.2. HEAPSORT algoritmoa: meta egitura bidezko ordenazioa

Willians-ek asmatu zuenez, bere izenez ere ezagutzen da.

Prozesua:

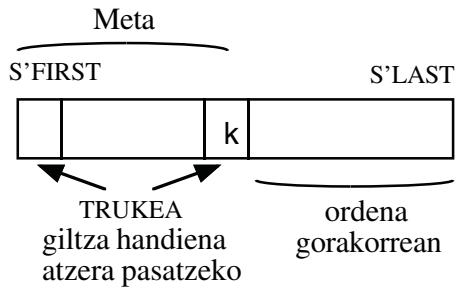
Meta egiturak giltzak ordenatzeko baliagarriak dira. Honela, egitura honetatik abiatuz, ordenaturik dagoen lista bat lortu nahi bada: lehendabizi metako erroa listaren lehenengo posizio hutsean kokatu behar da; ondoren meta egitura berreskuratu, eta berriz ere prozesu bera errepikatu meta egitura hutsa izan arte. Estrategia honek meta eraikitzea eskatzen du eta ondoren, behin eta berriro, meta egitura berreskuratzeko giltzen berrantolamenduak egitea. Hasiera batean, estrategia hau ez dela eraginkorra izango badirudi ere, lortutako emaitzek aurkakoa

diote, kasu txarrenean O(n lg n) ordenakoa baita. Heapsortek ideia hau aurrera eramaten du. Algoritmo honen egitura orokorra:

```
alg HEAPSORT (...)
begin
    Meta_eraiki;
    for  I  in  1 .. M'LAST  loop
          Erroa_kopiatu;
          Azken_mailako_eskuinaldekoena_ezabatu;
          Berrantolatu;
    end loop;
end HEAPSORT;
```

Algoritmoa:

```
procedure HEAPSORT(S: in out OS_SEK;) is
begin
    — meta eraiki2
    for I  in reverse  S'FIRST.. S'LAST/2  loop
          HONDORATU(S(I..S'LAST));
    end loop;
    — erroa ezabatu eta dena berrantolatu.
    for META_LUZERA  in reverse  S'FIRST+1..S'LAST loop
          TRUKEA(S(S'FIRST)),S(META_LUZERA));
          HONDORATU(S(S'FIRST..META_LUZERA-1));
    end loop;
end HEAPSORT;
```



Algoritmoaren ordenaren analisia:

Kasu txarrenaren analisia.

Bi bigiztez osaturik dago. Aztertu dezagun lehendabizi metaren bilakaera. Izan bedi $s = \lfloor \lg n \rfloor n$ adabegi dituen meta baten sakonera. Zenbait meta aztertu ondoren, Hondoratu algoritmoak kasu txarrenean egiten duen giltzen alderaketen kopurua lan egiten den azpizuhaitzaren sakoneraren bikoitza dela ikus daiteke; hau da, *s* ken azpizuhaitzaren erroaren maila.

$$\sum_{m=0}^{s-1} 2(s-m)(\text{m. mailan dauden adabegien kopurua}) = \sum_{m=0}^{s-1} 2(s-m)2^m$$

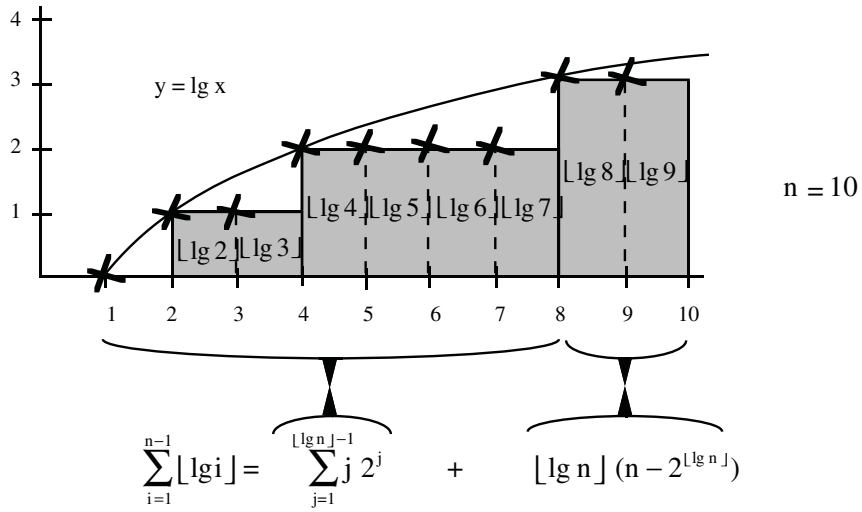
$$2\sum_{m=0}^{s-1} 2(s-m)2^m = 2^{s+2} - 2s - 4 \in \Theta(n)$$

Beraz, metaren bilakaerak denbora lineala hartzen du!

Aztertu dezagun orain bigarren begizta. *k* adabegi dituen meta batean Hondoratu algoritmoak gehienez $2\lfloor \lg k \rfloor$ alderaketa egiten ditu. n-1 aldiz errepikatu behar da prozesua, lehenengo aldian metak (n-1) adabegi dituelarik eta azkeneko aldian, berriz, bakarra. Beraz, ezabapen guztietan zehar egiten den giltzen alderaketa kopurua gehienez

$$2\sum_{k=1}^{n-1} \lfloor \lg k \rfloor = 2 \sum_{k=T'FIRST}^{T'LAST-1} \lfloor \lg k \rfloor$$

izango da. Baina, batura hau kalkulatu ahal izateko ondorengo grafikoa aztertu behar dugu:(n=10 deneko kasua aztertzen da). Ondorioz, meta handienak (n-1) osagai izango ditu:



Kalkulatu behar dugun batukaria grafikoan gurutze baten bidez markaturik agertzen diren balioen baturaren berdina da. Berriz, $s = \lfloor \lg n \rfloor$ izanik, kalkulatu nahi dugun batukaria honako hau da:

$$\sum_{j=1}^{s-1} j2^j + s(n - 2^s)$$

Azkenik, zenbait kalkulu egin ondoren, lortzen dugun emaitza hau da:

$$2\left(s2^s - 2^s + 1\right) + s\left(n - 2^s\right) = ns - 2^s + 2 \in \Theta(n \lg n)$$

Beraz, eta baturaren erregela erabiliaz, jarraian dauden bi bigiztek hartzen duten denboraren ordena, begizta bakoitzak hartzen duen denboraren maximoa izango da. Hau da, Heapsort algoritmoaren ordena hori kasu txarrenari dagokio.

2.6. EMAITZEN ALDERAKETA

Aztertutako alderaketa bidezko ordenazio algoritmo nagusien portaera--analisien emaitzak ondorengo taulan laburbildu ditugu:

Algoritmoa	**Kasu txarrena**	**Batezbesteko**	**Memoria-espazioa**
Txertaketa	$\frac{n^2}{2} \in \Theta(n^2)$	$\frac{n^2}{4} \in \Theta(n^2)$	Lekuan
Quicksort	$\frac{n^2}{2} \in \Theta(n^2)$	$O(n \lg n)$	Espazio estra: *lg n*-ren proportzionala
Mergesort	$\Theta(n \lg n)$	$\Theta(n \lg n)$	Espazio estra: Mergea egiteko *n*-ren proportzionala
Heapsort	$\Theta(n \lg n)$	$2(n-1)\lg n \in \Theta(n \lg n)$	Lekuan

Nahiz Mergesort algoritmoa kasu txarrenean onenatik hurbil egon, badaude alderaketa gutxiago egiten dituzten algoritmoak. Bestalde, memoria-espazio estra behar du algoritmo horrek. Aldiz, Heapsort algoritmoa bai kasu txarrenean bai batezbesteko kasuan bere ordena O(n lg n) da eta, gainera, memoria espazio estraren beharrik ez du. Beraz, *n* behar hainbat handia den kasuetarako aztertu ditugun algoritmoen artetik onena dugu Heapsort algoritmoa. Bestalde, alderaketa bidezko ordenazio algoritmoek kasu txarrenean Ω(n lg n) ordenako alderaketa kopurua egin behar dutenez gero (kopuru hori datorren azpiatalean frogatuko dugu), Kasu txarrenean Heapsort algoritmoa baino algoritmo hoberik ez dugula sekulan aurkituko esanaz amaitzen dugu alderaketen bidezko ordenazio algoritmoen azterketa.

2.7. ALDERAKETA BIDEZKO ORDENAZIO ALGORITMOEN BEHE-BORNE MINORANTEA

Orain arte ikusitako algoritmo guztiek propietate interesgarri bera betetzen dute: *Ordenazio ordena sarrerako osagaien arteko alderaketek bakarrik mugatzen*

dute. Propietate hau betetzen duen edozein algoritmo *Alderaketa Bidezko Ordenazio* algoritmoa dela esan ohi da.

Honela, ikusitako alderaketa bidezko ordenazio algoritmoen artean badugu *n* elementu ordenatzeko O(n lg n) denbora-ordena behar duen zenbait algoritmo. Bateraketa eta Metaketa ordenazio algoritmoak kasu txarrenean goi-borne horretara iristen dira. Gainera posible litzateke algoritmo hauentzat *n* elementu lituzketen eta Ω(n lg n) denbora-ordena hartuko luketen sekuentzia bana ekoiztea.

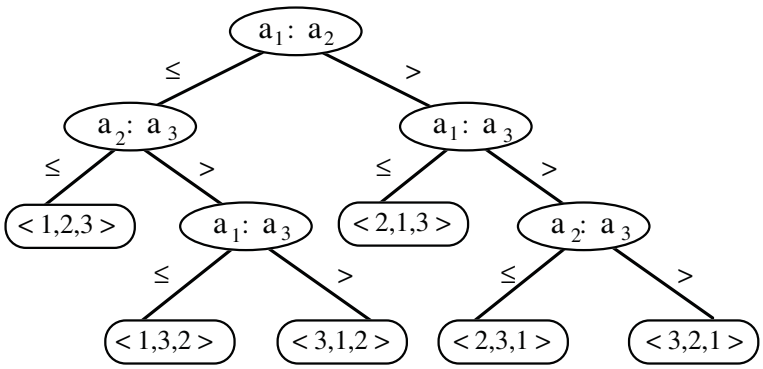
Baina, posible al litzateke beste alderaketa bidezko ordenazio algoritmo bat aurkitzea non honi legokiokeen denbora-ordena aurrekoei dagozkienak baino hobea izango litzatekeen?

Esan dugun bezala, alderaketa bidezko ordenazio batean $<a_1,\ldots,a_n>$ sekuentzia bat emanik, osagaien arteko ordena erlatiboa lortzeko, sekuentziako elementuen arteko alderaketak egingo dira soilik. Horrela, a_i eta a_j elementuak izanik eta bien arteko ordena erlatiboa mugatzeko, algoritmoan hauetako galderaren bat egingo da: $a_i<a_j$, $a_i\leq a_j$, $a_i>a_j$, $a_i\geq a_j$ edo $a_i=a_j$. Balioak zehazki zein diren ez dugu aztertuko, ez eta elementuen arteko ordenari buruz informazio gehiago eman diezagukeen beste metodorik erabiliko ere.

Atal honetan, eta orokortasuna galduko ez dugula jakinik, sarrerako sekuentziako osagai guztiak desberdinak direla suposatuko dugu. Ondorioz, ez da $a_i=a_j$ alderaketarik egingo. Bestetik, a_i eta a_j elementuen arteko ordena erlatiboa mugatzeko $a_i<a_j$, $a_i\leq a_j$, $a_i>a_j$ eta $a_i\geq a_j$ alderaketek emango diguten informazioarekiko baliokideak izango ditugu. Ondorioz, alderaketa guztiek $a_i\leq a_j$ itxura izango dute.

Erabaki-zuhaitzen eredua:

Alderaketa bidezko ordenazio algoritmo guztiak e*rabaki-zuhaitz*etan oinarritzen direla esan ohi da. Alderaketa bidezko ordenazio algoritmo batek emandako sarrera batekiko egingo lituzkeen alderaketak adierazten ditu erabaki-zuhaitzak. Kontrola, datuen mugimendua eta bestelako xehetasunak ez dira kontutan hartzen. Honela, adibidez, *txertaketa* algoritmoari dagokion erabaki-zuhaitza honako hau dugu:



Erabaki-zuhaitzeko barne-adabegi bakoitzean alderatzen diren sarrerako sekuentziako bi elementu ditugu: a_i eta a_j . Hosto bakoitzak sarrerako sekuentziaren permutazio bat izango du. Ordenazio algoritmoaren egikaritzapen bakoitza, zuhaitzaren errotik hosto baterainoko korritzea izango da. Barne-adabegietan, $a_i \leq a_j$ alderaketa egingo da eta honen emaitza egiazkoa izatea gertatuko balitz, orduan ezkerreko azpizuhaitzak mugatuko lituzke ondoren egin beharko liratekeen alderaketak. Bestela, $a_i > a_j$ gertatzen denean, eskuineko azpizuhaitzak mugatuko lituzke. Hosto batera iristerakoan, algoritmoak sarrerako osagaien arteko ordena erlatiboa mugatua izango du. Sekuentziak *n* osagai dituenean, zuhaitzak *n!* hosto izango ditu.

Kasu txarreneko behe-borne minorantea:

Algoritmoak egin beharko dituen alderaketen kopuru handiena erabaki-zuhaitzeko errotik hasi eta hosto bateraino iristen den adar luzeenak ematen du; hau da, zuhaitzaren sakonerak. Erabaki-zuhaitzen sakoneren behe-bornaketa, alderaketa bidezko algoritmoen exekuzio-denboraren behe-bornaketa da. Ondorengo teoremak bornaketa hau finkatzen du.

Teorema:

n elementu ordenatzen dituen erabaki-zuhaitz baten sakonera gutxienez *n lg n* ordenakoa da; hau da, $s \in \Omega(n \lg n)$ da.

Frogaketa:

Demagun *n* elementu ordenatzen dituen eta *s* sakonera duen erabaki-zuhaitza dugula. *n* elementuek *n!* permutazio ematen dituztenez, non permutazio bakoitza sarrerako elementuen ordenazio posible bat den, zuhaitzak *n!* hosto izango ditu derrigorrez. Bestalde, *s* sakonera duen edozein zuhaitz bitarrak ezin duenez 2^s baino hosto gehiago izan, orduan:

$$n! \leq 2^s$$

erlazioa dugu, eta bertan logaritmoak aplikatuz

$$\lg n! \leq s$$

funtzio monotono gorakorra lortzen dugu, eta *s*-ren behe-bornea lortzeko posibilitate desberdinak ditugu:

1) Orain, Stirling-en hurbilketa $n! > (n/e)^n$ eta $e = 2.71828...$ direla jakinik

$$s \geq \lg n! \geq \lg (n/e)^n = n \lg n - n \lg e$$
$$s \in \Omega(n \lg n)$$

2) edo, lg n! = lg (n (n-1) (n-2) ... 1) erlazio ezaguna aplikatuz:

$$s \geq \lg n! \geq \lg (n\,(n-1)\,(n-2)\,\ldots\,1) = \sum_{i=1}^{n} \lg i = \sum_{i=2}^{n} \lg i \geq \int_{1}^{n} \lg x\,dx = n\lg n - 1.5n$$

$s \in \Omega(n \lg n)$

Korolarioa:

Bateraketa eta Metaketa ordenazioak, asintotikoki onenak diren *alderaketa bidezko algoritmoak* ditugu.

Frogaketa:

Bi algoritmoren exekuzio-ordenak O(n lg n) goi-borneak dituzte eta hauek aurreko teoremaren kasu txarreneko Ω(n lg n) behe-borne minorantearekin bat datoz.

2.8. DENBORA LINEALA BEHAR DUEN ZENBAIT ORDENAZIO ALGORITMO

2.8.1. Kontaketa algoritmoak

Sarrerako elementu bakoitzaren balioa [1,k] tarte itxian dagoela suposatzen du kontaketa algoritmoak. $k \leq n$ gertatzen denean, eta oro har $k \in O(n)$ betetzen denean, orduan algoritmoaren exekuzio-ordena *n*-ren lineala da.

Ideia:

Sarrerako elementu bakoitzeko hura baino txikiagoak edo berdinak diren elementu kopuruaren mugan dago algoritmoaren gakoa. Gero, informazio horri esker elementu bakoitza zuzenki bere posizio erlatiboan koka daiteke. Adibidez, demagun 17 elementu ditugula *x* baino txikiagoak direnak; orduan *x*-k 18. posizioan egon beharko luke irteeran. Balio berdina duten elementuak baleude, eskema hau pixka bat aldatu beharko litzateke balio berdinak ez baititugu posizio berean kokatu nahi.

Algoritmoa inplementatzerakoan, sarrerako sekuentzia *A* dela eta bere osagai kopurua, A'LENGTH, *n* dela suposatuko dugu. A taulaz at, beste bi taula gehiago ere erabiliko ditu: B(1..n) taulan emaitza uzteko eta D(1..k) tarteko soluzioak biltegiratzeko.

Algoritmoa:

```
procedure KONTAKETA (A: in OS_SEK; K: in INTEGER; B: out OS_SEK) is
    D: OS_SEK(1..K):= (others => 0);
begin
```

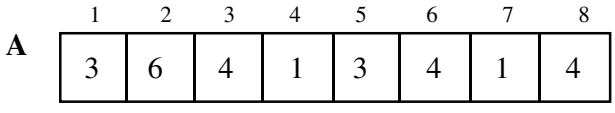
```
(1) for  J  in  A'RANGE  loop
                D(A(J)):= D(A(J)) +1;
          end loop;
          — D(j) posizioak A taulan j balio bera duen zenbat osagai dagoen
          — biltegiratzen du
(2) for  I  in  D'FIRST+1..D'LAST  loop
                D(I):= D(I) +D(I-1);
          end loop;
          — D(i) posizioak A taulan i balio bera edota  txikiagoa duen
          — zenbat osagai dagoen biltegiratzen du
(3) for  J  in reverse  A'RANGE  loop
                B(D(A(J))):= A(J);
                D(A(J)):= D(A(J))-1;
          end loop;
    end loop;
end KONTAKETA;
```

Prozesua:

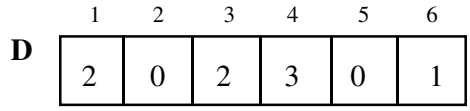
D taula 0-z hasieratu ondoren, sarrerako elementu guztiak banan-banan aztertzen dira 1 begiztan. Sarrerako elementu baten balioa *j* bada, orduan D(J) balioa 1ez gehitzen da. Ondorioz, 1 begizta amaitu ondoren D(J)-k sarrerako A taulan J balioko zenbat balio dagoen biltegiratzen du, non J $\in$ [1,K] betetzen den. 2 begiztak, berriz, I [1,K] tarte itxiko balio bakoitzeko A taulan I balioko edo balio hori baino txikiagoko zenbat elementu dagoen zehazten du.

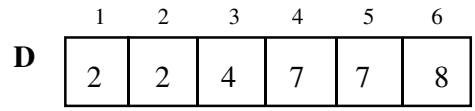
Azkeneko begiztan, sarrerako osagai bakoitza irteerako B taulan dagokion posizio erlatiboan kokatzen da. Ako osagai guztiak desberdinak balira, orduan A(J) balio bakoitzeko D(A(J)) balioak, A(J)-ri irteerako taulan legokiokeen posizioa adieraziko luke. A(J) baino txikiagoak edo berdinak diren D(A(J)) elementu ezberdin daude. Baina, elementu guztiak ez balira desberdinak, D(A(J)) 1ez gutxitu beharko litzateke A(J) B-n ipintzen den aldi bakoitzean; honek, eta sarrerako taulan A(J)-ren balio berdineko beste elementu bat balego, aurreko A(J) balioa B taulan ezarritako posizioaren aurreko posizioan kokaraztea eragingo luke.

Adibidea:

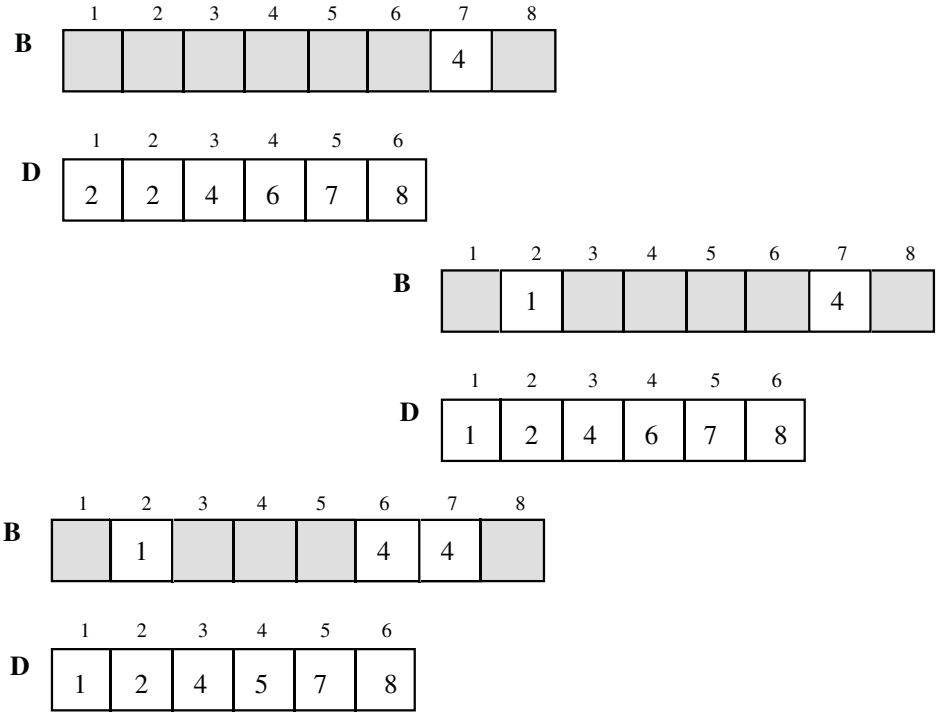


A taulako osagai bakoitza zenbaki positibo osokoa da eta [1,k] tarte itxikoa. D(j) posizioak *A* taulan *j* balioko zenbat osagai dagoen biltegiratzen du.

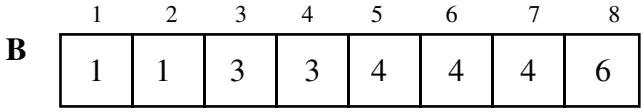




D(j) posizioak *A* taulan *j* balioko edo balio txikiagoko duten zenbat osagai dagoen biltegiratzen du.



3 begizta 1, 2 eta 3 aldiz egikaritu ondoren B eta D taulen egoera.



B taulak A taulako osagaien ordenazioa biltegiratzen du *kontaketa* algoritmoaren amaieran.

Counting Sort algoritmoaren denbora-ordenaren analisia:

Kontaketa algoritmoaren gorputzean 3 begizta ikusten baditugu ere, berez lau daude: D taularen hasiera ez da ahaztu behar.

Lau bigizten gorputzak asignazio aginduz edota *if* eta asignazio aginduz osaturik daude. Edozein kasutan gorputzek beharko duten exekuzio-denbora konstantea izango da: *i* begiztaren gorputzeko konstantea a_i dela suposatuko dugu. Bestaldetik, lau begiztak jarraian agertzen direnez gero, algoritmoaren denbora--ordena zein den jakitearren batuketaren erregela aplikatu besterik ez dugu egin behar:

$$T(n) \in \Theta(a_1 k + a_2 n + a_3 (k-1) + a_4 n) = \Theta(\max(k, n, k, n))$$

Exekuzio gehienetan $k \in \Theta(n)$ betetzen da. Kasu horietan *kontaketa* algoritmoaren exekuzio-denborak $T(n) \in \Theta(n)$ betetzen du.

Kontaketa algoritmoaren Θ(n) denbora-ordenak aurreko atalean ikusitako Ω(n lg n) behe-bornaketa minorantea auzitan jartzen al du? Ez, *Kontaketa* algoritmoa ez da alderaketa bidezko algoritmoa: kodean ikus daitekeen bezalaxe, ez da sarrerako elementuen arteko alderaketarik egiten. Kontaketako kodean, ordea, sarreraren taulako balio errealak erabiltzen dira B-n indexatzeko.

Counting Sort algoritmoaren memoria-espazio estraren analisia:

Ordenatu behar den A taulaz at, beste bi taula lagungarri behar ditu: D(1..K) eta B(A'RANGE) (= B(1..N), kasu honetan). B taula beharrezkoa da, ezin da-eta emaitza aldi berean A-n itzuli (A-ko balioak behar dira B osatzeko).

Egonkortasun propietatea:

Balio bera duten sarrerako zenbakiak irteerako sekuentzian ordena berean azaltzen dira.

Kontaketa algoritmoak propietate interesgarri hau betetzen du. Propietatea interesgarria gertatzen da batez ere ordenatu behar den zenbaki bakoitzarekin batera informazio asko mugitu behar denean.

Propietatearen garrantzia ondorengo atalean ikusiko da.

2.8.2. Berredura-ordenazioa: RADIX SORT

Demagun lehen bezala, zenbakien sekuentzia bat emanik hau ordenatu nahi dugula. Suposa dezagun konkretuki sekuentzia hori kasu batean honako hau dela:

< 329, 457,657,839,436,720,355>

Nola ordenatuko genuke ez bada alderaketa bidezko ordenazio algoritmorik erabili nahi? Honaino ezagutzen dugun beste algoritmo bakarra kontaketa algoritmoa da. Zentzurik izango al luke hemen kontaketa algoritmoa zuzenean aplikatzeak? Ez. D(1..839) taula beharko genuke metodoa zuzenean aplikatu ahal izateko! 839 osagai lituzkeen taula bat erabiliko genuke 7 zenbaki ordenatzeko!

Oraingo honetan, zenbakiak digituz osaturik daudela eta hauetako bakoitzaren balioa [0, Zenbakiaren_oinarria-1] tartekoa dela jabetzean dago gakoa; hots, ordenatu behar diren zenbakien oinarria jakitean. Radix Sort algoritmoak egingo duen prozesua ondorengoa da: lehendabizi, *d* digitu dituzten *n* zenbakiak lerroz lerro jartzen ditu. Ondoren, zenbaki guztien pisu txikieneko digituaren balioarekiko ordenatzen ditu zenbakiak, horretarako egonkorra den algoritmoren bat erabiliz. Prozesu berdina pisu txikieneko digitutik hasi eta pisu handieneko digituraino errepikatuz ordenatzen ditu *n* zenbakiak.

Adibidea:

329		720		720		329
457		355		329		355
657		436		436		436
839	$\Rightarrow$	457	$\Rightarrow$	839	$\Rightarrow$	457
436		657		355		657
720		329		457		720
355		839		657		839

Algoritmoa:

```
type  ZENB_D is array ( 1..D)  of DIGITUA;
type MATRIZEA_N_D  is array (1..N) of  ZENB_D;
procedure BERREDURA is (M: in out MATRIZEA_N_D ;
                        OINARRIA: in INTEGER) is
begin
    for  ZUT in 0..OINARRIA-1 loop
          KONTAKETA_MATRIZEA_N_D ( M, ZUT );
    end loop;
end BERREDURA;
```

Denbora-ordenaren analisia:

Radix-ek hartzen duen denbora-ordena *kontaketa* algoritmoak hartzen duen denboraren araberakoa da, non $\Theta(n+k)=\Theta(\max(n,k))$ dela arrazoitu den. Bestetik, sailkatu behar diren zenbakiak d digituz osaturik daudenez, gure algoritmoan ordena $\Theta(d\ n+d\ k)$ izango da; hau da, $\Theta(\max(d\ n,d\ k))$. Baina, $k \in \Theta(n)$ betetzen bada eta normalki sailkatu behar ditugun zenbakien digitu kopurua finkoa denez, d konstantea litzateke eta ondorioz algoritmoaren ordena n-en lineala: $\Theta(n)$.

k ere konstantea litzateke ordenatu nahiko genituzkeen zenbakien oinarria beti berdina balitz. Hala ere, informatika munduan maiz 2,8, 16 eta 10 erabiltzen ditugu.

ARIKETAK: ORDENAZIO ALGORITMOAK

1- n giltza desberdinez osatutako sekuentzia izanik, ondoren ematen den Txertaketa bidezko Ordenazio algoritmoaren aldaera azter bedi:

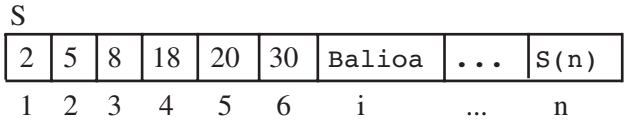
$\forall i\ (2 \le i \le n \rightarrow S(1) \le S(2) \le ... \le S(i-1))$ sekuentzian S(i) zuzenean bere posizio erlatiboan txertatzen da. Bilaketa Dikotomikoa erabiltzen da S(i)ren posizio zuzena $S(1) \le S(2) \le ... \le S(i-1)$ sekuentzian mugatzeko.

```
procedure  TXERTAKETA  (S: in out OS_SEK) is
    POS: INDIZEA;
    LAG: OSOA;
begin      — Ss= <a1,a2, ...,an>
    for I in S'FIRST+1..S'LAST loop
          — permutazioa(Ss, S) ∧ ordena_<(S(S'FIRST,...,I-1))
          POS_BILAKETA(S(S'FIRST..I-1), S(I), POS)
          LAG:= S(I);
          — POS eta I-1 posizio arteko baliok POS+1 eta I posizioetara mugitu
          for K in reverse POS..I-1 loop
                S(K+1):= S(K);
          end loop;
          S(POS):= LAG;
          — permutazioa(Ss, S) ∧ ordena_<(S(S'FIRST,...,I))
          end loop;
— permutazioa(Ss, Si) ∧ ordena_<(Si)
end TXERTAKETA;
```

Hau da, Txertaketa Algoritmoak S(i) osagaia zein posiziotan txertatu behar duen mugatzeko ondorengo algoritmo hau erabiltzen du:

```
Alg POS_BILAKETA(T, Bal,P)
— Bal ∉ Multzoa_Egin(T)
  baldin  T'Length=1  orduan
     baldin  T(T'First) < Bal
     orduan P:=T'First+1
     bestela  P:=T'First
  bestela
     baldin  T((T'length)/2) < Bal
     orduan POS_BILAKETA(T(((T'length)/2)+1)..T'Last),Bal,P)
     bestela POS_BILAKETA(T(T'First..((T'length)/2)-1)),Bal,P)
bukatu alg
```

Adibideak:



POS_BILAKETA(S(1..6), 1) → P=1
POS_BILAKETA(S(1..6), 3) → P=2
POS_BILAKETA(S(1..6), 9) → P=4
POS_BILAKETAS(1..6), 37) → P=7

Txertaketa bidezko Ordenazio metodoaren aldaera berri honetan ondorengo hau aztertzea eskatzen da:

a) Giltzen arteko alderaketak:
   Zein da kasu konkretu txarrena? Zein da kasu txarren horren exekuzio denboraren ordena?
b) Giltzen mugimenduak:
   Zein da kasu konkretu txarrena? Kasu txarren horretan, zenbat giltzen mugimendu egiten da guztira?
c) Zein da algoritmoaren ordena kasu txarrenean?

2.- Muga bedi kasu txarrenean TXERTAKETA algoritmoak (sekuentziako osagaien artean) egiten dituen:

a) alderaketen kopurua, eta
b) trukaketa kopurua.

3- $<e_1, e_2, ...,e_n>$ n osagaiez osatutako bektore bat ordenatzeko, MergeSorten parekoa den estrategia bat erabiltzen da, baina oraingoan bektorea 2 osagai dituen n/2 zatitan banatzen da. Algoritmoa idatzi gabe, haren eraginkortasuna kalkulatzeko:

– exekuzio-denbora neurtzen duen ekuazioa idatzi bedi bertako batugai bakoitzaren jatorria azalduz, eta
– ordena kalkula bedi.

4- a) Aska bedi ondorengo errekurtsio ekuazioa:

$$t(n)=\begin{cases} d & \text{baldin } n \le a \\ \sqrt{n} \bullet t(\sqrt{n}) + bn & \text{baldin } n > a \end{cases}$$

suposatuz *n*-ren balioak a^{2^k} modukoak direla eta $k \ge 0$.
b) Nahasketa (Mergesort) algoritmoaren beste hurbilpen bat azter dezagun: Ordenatu nahi den taula oraingoan $\sqrt{n}$ tamainako zatitan banatzen da.

Emaitza algoritmoa errekurtsiboki aplikatu ondoren lortutako $\sqrt{n}$ taulak nahastuz (Merge) lortzen da. Aurreko errekurtsio ekuazioak *nahasketa* (Mergesort) algoritmoaren hurbilpen berriak hartuko lukeen exekuzio-denbora deskribatzen al du? Zergatik? Errekurtsio-ekuazio zuzena idatzi eta arrazoitu.

5- *Ordenazio-azkarra (Quicksort)* algoritmoa egoki bedi estekadura bikoitzeko listak ordena ditzan. Bestalde, ezaguna da *n* osagai dituen taula bat ordenatzeko *Ordenazio-azkarrak* (gelan ikusitakoa) $T_t(n) \in O(n^2)$ eta $T_b(n) \in O(n \lg n)$ denbora-ordenak hartzen dituela kasu txarrenean eta batezbesteko kasuan hurrenez hurren. Emaitza horiek kontutan izanik, eztabaida bedi bertsio berriaren exekuzio-ordena.

(OHARRA: Estekadura bikoitzeko listaren datu-mota honela dago definituta:

```
type  ADABEGIA;
type  ERAK_ADABEGIA is access ADABEGIA;
type  ESTEKADURA_BIKOITZEKO_L  is
            record
                  LEHENENGOA: ERAK_ADABEGIA;
                  AZKENEKOA: ERAK_ADABEGIA;
            end record;
type  ADABEGIA is
            record
                  DATUAK: DATUAK_MOTA;
                  ONDORENGOA: ERAK_ADABEGIA;
                  AURREKOA: ERAK_ADABEGIA;
            end record;                                    )
```

Beste alderaketa bidezko ordenazio algoritmoak estekadura bikoitzeko listak ordena ditzaten egoki al daitezke? Haien ordenekin zer gertatzen da?

6- Ondorengo ariketak Heapsort algoritmoaren bertsio berri bat aztertzen du. Hain zuzen, bertako Metaren eraiketaren prozesua aldatzen da.

```
procedure HEAPSORT (S: in out SEK) is
begin
    META_ERAIKI2 (S'FIRST, S'LAST);
    for  Heaparen_Luzera  in reverse  S'FIRST+1 .. S'LAST loop
        S_TAULAN_TRUKEA(S'FIRST, Heaparen_Luzera);
        ERROA_HONDORATU(S'FIRST, Heaparen_Luzera -1);
    end loop;
end HEAPSORT;
```

(a atala)

(a atala)

Meta datu-mota abastraktua abiapuntutzat harturik eta taula bat emanik, taula hori meta bilakatuko duen META_ERAIKI2 algoritmoa idaztea eskatzen da.
Baina, haren eraiketarako AZKENA_AZALERATU eragiketaz bakarrik balia daiteke programatzailea. AZKENA_AZALERATU prozeduraren zehaztapena ondorengoa da:

procedure AZKENA_AZALERATU (Hasi,Buka: **in** SEK_INDIZEA);
— Sarrera: Hasi eta Buka-1 indizeek S taulan meta bat mugatzen dute.
Buka indizean dagoen balioa S-n azaleratu nahi den osagaia da.
— Irteera: Hasi eta Buka-1 indizeek S taulan meta bat mugatzen dute.
— Eragina: S(Hasi, Buka) maximoen metan, metaren propietatea berreskuratu arte S(Buka) balioa azaleratzen du.

(b atala)

AZKENA_AZALERATU prozeduraren kodea honako hau bada:

```
procedure AZKENA_AZALERATU (Hasi,Buka: in SEK_INDIZEA) is
    Ind_K, Ind_J: SEK_INDIZEA;
begin
    Ind_K:= Buka;
    loop
          Ind_J := Ind_K;
          if  Hasi < Ind_J  and then  S(Ind_J/2)< S(Ind_K)
          then  Ind_K:= Ind_J/2;
          end if;
          S_TAULAN_TRUKEA(Ind_J, Ind_K);
          exit when Ind_J = Ind_K;
    end loop;
end AZKENA_AZALERATU;
```

Egindako lana neurtzeko eragiketa adierazgarri bezala S taulako osagaien arteko alderaketak hartzen baditugu (non S taulak hasieran n osagai dituen), META_ERAIKI2 kasu txarrenean zein ordena zehatzekoa da?

Heapsorten zein bertsio da azkarragoa: (a) atalekoa edo ‘Hondoratu’ prozedura erabiliz lineala dena? Erantzuna arrazoitu bedi gehienez 5 lerro erabiliz.

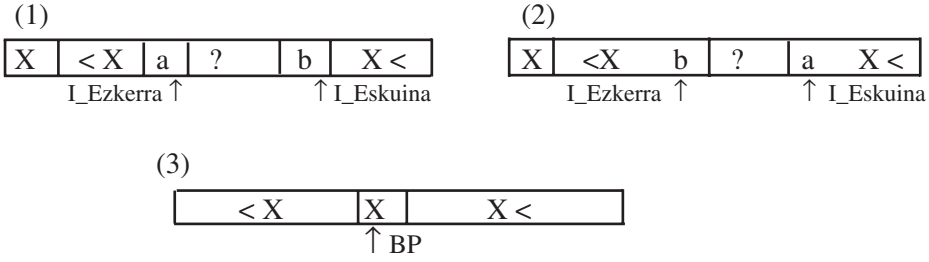
7- Problema hau QUICKSORTek dei egiten duen BANAKETA prozeduraren beste aukera bat da.

```
procedure QUICKSORT (Hasi1, Buka1: in SEK_INDIZE) is
    Banaketa_P: SEK_INDIZE;
begin
    if  Hasi1<Buka1  then
          BANAKETA2 (Hasi1, Buka1, Banaketa_P);
          QUICKSORT(Hasi1, Banaketa_P-1);
          QUICKSORT(Banaketa_P+1, Buka1);
    end if;
end QUICKSORT;
```

Prozesuaren bertsio berria:

X = S(Hasi). X-en balioaren arabera S sekuentziako balioak bi azpisekuentziatan banatzeko modu egoki bat, S(Hasi, Buka) behin bakarrik korritzea da, baina sekuentziaren bi ataletatik hasita. Bi indize-aldagai behar

dira, I_Ezkerra eta I_Eskuina. Indize hauek Hasi eta Buka+1 balioekin hasieratzen dira hurrenez hurren. I_Ezkerra inkrementatu egin behar da S(I_Ezkerra)>X gertatu arte eta, aldiz, I_Eskuina dekrementatu S(I_Eskuina)<=X gertatu arte (1 irudia ikusi). Orduan S(I_Ezkerra) eta S(I_Eskuina) trukatu egiten dira (2 irudia). Prozesuak (I_Ezkerra < I_Eskuina) baliozkoa den bitartean jarraitzen du. Azkenean, S(Hasi) eta S(I_Eskuina) permutatu egiten dira X bere posizio zuzen erlatiboan kokatzeko, hots, BP posizioan (3 irudia).



Prozesuaren kodea ondorengo prozeduran jaso da:

```
procedure BANAKETA2 (Hasi,Buka: in SEK_INDIZEA; BP: out SEK_INDIZEA)is
     I_Ezkerra: SEK_ INDIZEA:= Hasi;
     I_Eskuina: SEK_ INDIZEA:= Buka+1;
     X: SEK_BALIOA:= S(Hasi);
begin
     loop
          I_Ezkerra:=I_Ezkerra+1;
          exit when S(I_Ezkerra)>X or I_Ezkerra >=Buka;
     end loop;
     loop I_Eskuina:= I_Eskuina-1;
          exit when S(I_Eskuina)<=X;
     end loop;
     while  I_Ezkerra < I_Eskuina  loop
          S_TAULAN_TRUKEA(I_Ezkerra, I_Eskuina);
          loop
               I_Ezkerra:=I_Ezkerra+1;
               exit when S(I_Ezkerra)>X
          end loop;
          loop
               I_Eskuina:=I_Eskuina-1;
               exit when S(I_Eskuina)<=X;
          end loop;
     end loop;
     S_TAULAN_TRUKEA(Hasi, I_Eskuina);
     BP:= I_Eskuina;
end BANAKETA2;
```

Galderak: n osagai dituen sekuentzia bat emanik:

a- BANAKETA2 algoritmoan eta kasurik txarrenean, Sko osagaien artean zenbat trukaketa egiten da?

b- QUICKSORT algoritmoan eta kasurik txarrenean, Sko osagaien arteko trukaketa kopuruaren ordena zein da?

c- S-ko osagaien arteko zenbat alderaketa egiten du kasu txarrenean BANAKETA2k? Hots, zein da BANAKETA2(S'FIRST,S'LAST,BanaketaP) deiak Sko osagaien artean egiten duen alderaketa kopurua kasu txarrenean?

d- QUICKSORT algoritmoan eta kasurik txarrenean, Sko osagaien arteko alderaketa kopuruaren ordena zein da?

e- QUICKSORTek sarrerako sekuentzia banatzeko goian azaldu den bertsioa erabiliko balu, kasu txarrenean aldatuko al litzateke bere ordena? Eta batezbesteko kasuan? Gehienez 10 lerrotan erantzuna argudia bedi.

8- Gorantz ordenatutako gizakien izenak dituzten $G_1(1..N)$ eta $G_2(1..N)$ bi bektore ditugu. Aldi berean bi bektoreetan agertzen diren gizakien izenak kalkulatuko dituen programa idatzi behar da. Gizakien bektoreen luzera *n* denean, soluzioak har dezakeen denbora-ordena $T_t(n) \in \mathbf{o}(n \lg n)$ da (hots, n lg n baino hobea izatea eskatzen da).

9- Ondorengo ordenazio algoritmoa kontutan izanik:

```
procedure   HERENAK_ORDENATU  (B: in out BEKTOREA;J: in INTEGER) is
    K: INTEGER:= ⌊(J-I+1)/3⌋;
begin
    if B(I) > B(J) then TRUKEA(B(I),B(J)); end if;
    if J-I+1 <= 2  then return; end if;
    HERENAK_ORDENATU (B, I, J-K);
    HERENAK_ORDENATU (B, I+K, J);
    HERENAK_ORDENATU (B, I, J-K);
end HERENAK_ORDENATU ;
```

a) *n* osagai dituen bektore bat sailkatzeko beharko duen exekuzio-denbora, haren ordena eta memoria-espazio estra kalkula bitez.

b) Aldera bitez *n* luzerako bektore batentzat HERENAK_ORDENATU-ren eta beste sailkapen algoritmo klasikoen exekuzio-denborak (hots, ordenak); hain zuzen: *burbuila* (BubbleSort), *txertaketa* (InsertionSort), *bateraketa* (MergeSort), *metaketa* (HeapSort) eta *ordenazio_azkarra* (QuickSort) algoritmoen ordenekin aldera bedi.

Behaketa:
– HERENAK_ORDENATU algoritmoaren exekuzio-ordena kalkulatzeko $n=(3/2)^k$ dela kontsidera bedi.

$$-\log_b a = \frac{\log_d a}{\log_d b}$$

$$-\log_3 2 = 0.63092$$

10- Demagun PARTIKETA funtzioa honela definituta dagoela:

```
type BEKTORE  is array(INTEGER range <>) of POSITIVE;
function PARTIKETA (B: BEKTORE ) return INTEGER  is
begin
    for I in  B'RANGE  loop
          if  Σ (j=B©FIRST, I-1) B(j) = Σ (j=1, B©LAST) B(j)  then return  I end if;
    end loop;
    return 0;
end PARTIKETA;
```

Zenbaki osoko positiboen B bektorea emanik, PARTIKETA funtzioak ondorengo baldintza betetzen duen I (B bektoreko) posizioa itzultzen du:

• I posizioaren ezkerraldean dauden B-ko osagaien batura (B(I) balioa kanpo) I posizioaren eskuinaldean dauden B-ko osagaien (B(I) balioa barne) baturaren berdina da.

Aurreko baldintza betetzen duen posiziorik ez badago bektorean, funtzioak 0 itzuli beharko du.

– Frogatu azaldutako ezaugarriak betetzen dituen posizio bakar bat egon daitekeela gehienez.
– Programaren exekuzio-denbora eta denbora-ordena kalkula bitez.
– O(n) ordena hartuko duen algoritmo bat idatz bedi.
– O(lg n) denbora hartuko lukeen algoritmorik aurkitzea posible ote litzatekeen laburki eztabaidatu.

11- Ondorengo algoritmoak osokoen bektore bat emanik, bertako osagai handiena eta txikiena zein diren erabakitzen du. Kasu txarrenean algoritmoak osagaien arteko ZENBAT ALDERAKETA egiten duen kalkula bedi.

```
proz  MAX_MIN (T[i..j], Max, Min)
    — i ≤ j
hasi
    baldin   T[i] ≤ T[j]
      orduan Max := T[j]; Min:= T[i];
      bestela  Max := T[i]; Min:= T[j];
    end baldin;
    baldin   i+1 ≤ j-1
      orduan MAX_MIN (T[i+1..j-1], Lag_Max, Lag_Min);
      baldin Max < Lag_Max
          orduan  Max := Lag_Max;
      end baldin;
      baldin Lag_Min < Min
          orduan  Min:= Lag_Min;
      end baldin;
    end baldin;
end proz;
```

12- a) Dagoeneko ezaguna den bilaketa dikotomikoa errekurtsiboki kalkulatzen du ondorengo algoritmoak:

(X osagaia bektorean ez badago, orduan 0 indizea itzuliko du)

```
proz  BILAKETA_DIKO (T[i..j], X, Ind)
hasi
    baldin  i< j orduan
          k:= (i+j) div 2;
          baldin  T[k] = X
          orduan  itzuli Ind:= k;
          bestela
                baldin  T[k] < X
                orduan  BILAKETA_DIKO(T[k+1..j], X, Ind);
                bestela BILAKETA_DIKO(T[i..k-1], X, Ind);
                end baldin;
          end baldin;
    bestela
          baldin  i=j  bai eta  T[i] = X
          orduan  itzuli  Ind:= i;
          bestela  itzuli  Ind:= 0;
          end baldin;
    end baldin;
end proz;
```

Algoritmoaren bi inplementazio desberdin kontsidera bitez, parametroen pasatzea batean erreferentzia bidez egiten dena eta bestean aldiz balio (edo kopia) bidez. Bi inplementazioen denbora-eragikortasunean diferentziarik ba al dago?

b) MergeSorten, lehen azaldutako bi parametroen pasatze aukeren artean ba al dago denbora-eraginkortasun diferentziarik?

3. GAIA

GRAFO EGITURA MANEIATZEN DUTEN ALGORITMOAK

3.1. SARRERA

3.1.1. Zenbait kontzeptu eta azalpen

Problema asko grafoen bidez adieraz ditzakegu. Kasu hauetan, problemari dagokion grafoaren korritzeren batek problemaren emaitza ematen du. Baina grafoen korritzeak esplizituki edo inplizituki egin daitezke. Gai honen hasieran korritze esplizitu bi azalduko dira: sakonerako eta zabalerako korritzeak. Hauen ondoren, eta gaia amaitzeko, beste korritze inplizituen adibide batzuk aztertuko ditugu. Grafoak darabiltzaten algoritmo ezagun asko dago, eta gai honen helburua haien sarreratxo bat egitea besterik ez da. Ondorengo gaietan algoritmo gehiago aztertuko ditugu.

Hemendik aurrera *grafo* hitza bi zentzu desberdinetan erabiliko dugu:

-1- Grafo bat konputagailuaren memorian dagoen datu-egitura izan daiteke. Kasu honetan, erpinak erregistroen bidez eta ertzak erakusleen bidez adieraziko dira. Egituraren gainean beharko ditugun eragiketak ondorengo hauek izango dira: *erpin bat markatu* (memoriako bit bat aldatu), *auzokidea den erpin batera pasatu* (erakusleren bat aztertzea suposatuko du).

-2- Beste kasuan, grafoak ez du esplizituki existitzen: inplizituki dago. Jokoak maiz grafo abstraktuen bidez adierazten dira: erpin bakoitzak jokoaren egoera bat adierazten du eta bi erpinen arteko ertzak (edo arkuak) lehenengo egoeratik bestera igarotzeko egin daitekeen jokaldi legala adieraziko du. Mota honetako grafoek berez ez dute konputagailuaren memorian existitzen. Normalki, momentu konkretu batean momentu horretako jokoaren egoera gordetzen da soilik, eta ez une horretaraino aztertu diren egoera guztiak. *Erpin bat markatzea* oraingoan, egoera berdina bi aldiz ez aztertzearren beharrezkoak diren neurriak hartzea izango da, eta *auzokidea den erpinera pasatu,* aldiz, posizio edo egoera batean egonik jokaldi legal bat eginez beste posizio edo egoera batera igarotzea izango da.

Baina grafoa datu-egitura bat izanik, edo eta abstrakzio bat izanik, grafoa korritzeko teknikak gutxi gora-behera berdinak dira. Ondorengo ataletan ez dugu desberdinduko zein kasutan gauden.

Grafoen adierazpide desberdinak:

Liburuaren helburua ez da Grafo datu-mota abstraktuen definizioa ematea. Baina, irakurleak, dagoeneko hauen ezagupen minimo bat eduki dezan komenigarria kontsideratzen dugu. Horretarako, eta beharra balu, datu-mota abstraktuen diseinuaz eta inplementazioaz aritzen diren liburu desberdinetara jo dezan gomendatzen diogu irakurleari. Adibidez, auzokideen listen bidezko eta pisu-auzokidetza auzokidetza matrize bidezko adierazpideak erabilienak direla eta, hauei buruz aipatzen dena irakurtzea gomendatzen zaio. Adierazpide hauek zenbait aldiz irudi bidez azaltzen dira.

3.2. SAKONERAKO KORRITZEA: Grafo ez-zuzendua

Izan bedi, G grafo ez zuzendu bat: G=<ERP,ERT>.

```
type ERPINA_DESK is                          — grafoaren erazagupen posible bat.
        record
            INFORM: INFORMAZIOA;
            AUZOKIDEAK: ERPINEN_L;
        end record;
type GRAFOA is array (POSITIVE range 1..P) of ERPINA_DESK;
```

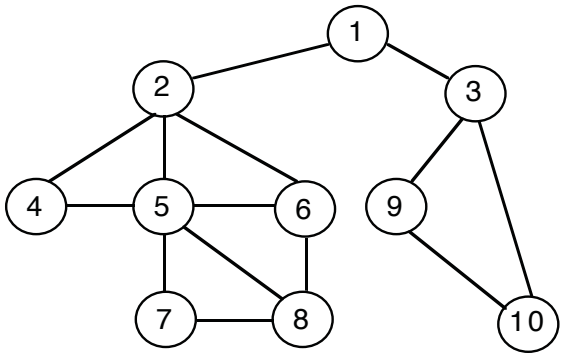
Grafoko erpin guztiak aztertu nahi ditugu. Suposa dezagun, erpin bat aztertu ondoren hura marka dezakegula. Hasieran, ez dago erpin markaturik.

Grafoa sakoneran korritu nahi bada, grafoko erpin bat abiapuntutzat aukeratu behar da, izan bedi A (A $\in$ ERP), eta dagoeneko aztertua izan dela adierazteko erpina markatzen da. Ondoren, honen auzokidea den eta oraindik markatu gabeko erpinen bat balego, hura abiapuntu berritzat aukeratzen da eta prozedurari errekurtsiboki dei egiten zaio. Honek kontrola itzultzen duenean, A-ren auzokidea den eta oraindik markatu gabeko beste erpinen bat balego, berriz ere A-ren auzokide hori abiapuntutzat harturik errekurtsiboki dei egiten zaio. Prozesu bera behien eta berriro errepikatzen da. A-ren auzokide guztiak markaturik daudenean, A-tik abiaturiko sakonerako korritzea amaitzen da.

Grafoan oraindik markatu gabeko erpinik balego, hauetakoren bat aukeratzen da eta bera abiapuntutzat harturik dei errekurtsibo bat egiten da. Ondoren, aurreko prozesu bera egiten da. G-ko erpin guztiak markaturik gelditu arte errepikatzen da prozesu osoa.

Adibidea:

Demagun erpinak zenbakiturik dituen ondorengo grafo hau dugula:



Erpin baten auzokideak zenbakitze ordena jarraituz aztertzen direla suposatuko dugu; beraz, grafoko erpinak sakonerako korritzean ordena honetan aztertuko dira:

• 1 erpina abiapuntutzat aukeratzen bada:

1.- S_K(1) Lehenengo deia.
2.- S_K(2) Dei errekurtsiboa.
3.- S_K(4) Dei errekurtsiboa.
4.- S_K(5) Dei errekurtsiboa.
5.- S_K(6) Dei errekurtsiboa.
6.- S_K(8) Dei errekurtsiboa.
7.- S_K(7) Dei errekurtsiboa. (Irtenbiderik ez.)
8.- S_K(3) 7,8,6,5,4 eta 2 erpinen auzokide guztiak markatuta daude. 3 erpina markatu gabeko 1en auzokidea.
9.- S_K(9) Dei errekurtsiboa.
10.- S_K(10) Dei errekurtsiboa (irtenbiderik ez). Grafoko erpin guztiak markatuta daude.

• 5 erpina abiapuntutzat aukeratzen bada:

1.- S_K(5) Lehenengo deia.
2.- S_K(2) Dei errekurtsiboa.
3.- S_K(1) Dei errekurtsiboa.
4.- S_K(3) Dei errekurtsiboa.
5.- S_K(9) Dei errekurtsiboa.
6.- S_K(10) Dei errekurtsiboa (irtenbiderik ez).
7.- S_K(4) 10,9,3,1 eta 2 erpinen auzokide guztiak markatuta daude. 4 erpina markatu gabeko 5en auzokidea.
8.- S_K(6) Dei errekurtsiboa (irtenbiderik ez). 4 erpinaren auzokide guztiak markatuta daude. 3 erpina markatu gabeko 5en auzokidea.
9.- S_K(8) Dei errekurtsiboa.
10.- S_K(7) Dei errekurtsiboa (irtenbiderik ez). Grafoko erpin guztiak markatuta daude.

Algoritmo errekurtsiboaren inplementazioa:

```
procedure S_K (G: in GRAFOA; A: in ERPINA) is
    A_AUZOKIDEEN_KOPIA: ERPINEN_L;
    LEHENA: ERPINA;
begin
    A_AUZOKIDEEN_KOPIA:= AUZOKIDEAK(G, A);
    while not(HUTSIK_DAGO_L? (A_AUZOKIDEEN_KOPIA)) loop
        LEHENA:= LEHENENGOA_L(A_AUZOKIDEEN_KOPIA);
        HONDARRA_L(A_AUZOKIDEEN_KOPIA);
        if not MARKATUA_DAGO(BISITAK, LEHENA) then
            MARKA_IPINI(BISITAK, LEHENA, AZTERTUA);
            S_K(G, LEHENA);
        end if;
    end loop;
end S_K;

procedure SAKONERAKO_KORRITZEA (G: in GRAFOA) is
    AZTERTUA: constant boolean := TRUE;
    BISITAK:TAULA(ERPIN_KOP_ESPARRUA):= (others=>not(AZTERTUA));
begin
    for ERPIN_BAT in ERPIN_KOP_ESPARRUA loop
        if not MARKATUA_DAGO(BISITAK, ERPIN_BAT) then
            MARKA_IPINI(BISITAK, ERPIN_BAT, AZTERTUA);
            S_K(G, ERPIN_BAT);
        end if;
    end loop;
end SAKONERAKO_KORRITZEA;
```

Sakonerako korritzearen izena, erpin bat abiapuntutzat harturik horri dagokion S_K deiarekin bukatu aurretik bere azpikoei dagozkien S_K dei errekurtsibo guztiak egitetik datorkio. Errekurtsibitatea irtenbide gabeko puntu batera iristen denean amaitzen da; hau da, abiapuntutzat aukeratu den erpinaren erpin auzokide guztiak markaturik daudenean. Orduan, maila altuagoko posibilitateak aztertuak izan daitezen kontrola itzultzen da.

Algoritmoaren ordenaren analisia:

Kalkula dezagun P erpinez eta T ertzez osaturiko grafo baten sakonerako korritze algoritmoak hartzen duen denboraren ordena:

– Erpin bakoitza behin bakarrik aztertzen da $\Rightarrow$ S_K algoritmoari P dei errekurtsibo egingo zaizkio.

– Erpin bakoitzean bere auzokideak markatuta dauden edo ez ikusi behar da. Grafoa auzokideen listekin adierazirik badago, gure kasuan bezala, azterketa hori Tren proportzionala izango da.

• S_Kren dei errekurtsiboak egiteko behar den denboraren ordena: O(P)
• Markatuta dauden edo ez jakiteko behar den denboraren ordena: O(T)

$\Rightarrow$ O(max(P,T))

Lortutako ordena zentzuzkoa da. Grafoa modu batera edo bestera adierazten bada ere:

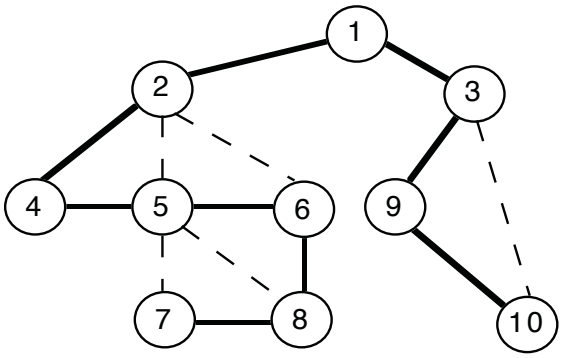
- P erpin aztertu behar dira, eta
- grafoko ertzetan zehar mugitzerakoan ertz bakoitzetik 2 aldiz pasako gara (behin norabide bakoitzeko). Beraz, 2T 'salto' edo azterketa egin behar dira.

Eginiko lan horrek guztiak hartzen duen denboraren ordena:

$\Rightarrow$ O(P+2T) = O(max(P,T))

Grafo konexuak/ez-konexuak sakonerako korritzean duen eragina:

Grafo konexu baten sakonerako korritzeari grafoko hedapen zuhaitz bat dagokio. Abiapuntu erpina hedapen zuhaitzaren erroan bilakatzen da. Eta grafoa korritzean erpinak atxitzeko erabili den ertz bakoitzari hedapen zuhaitzeko arku bat dagokio. Korritzean erabili ez diren ertzek, berriz, ez dute arku ordezkaririk zuhaitzean.



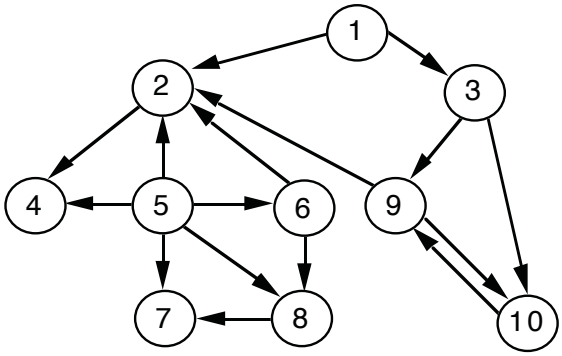
Grafoa konexua ez bada, ez du hedapen zuhaitz bakar bat sortzen sakonerako korritzean, baizik eta hedapen zuhaitzez osaturiko basoa, zuhaitz bat osagai konexu bakoitzeko.

3.3. SAKONERAKO KORRITZEA: Grafo zuzendua

Grafo ez-zuzenduetan ikusitako algoritmo bera erabili badezakegu ere, beharrezkoa da auzokidetza kontzeptua argitzea:

Grafo zuzenduetan Y erpina X erpinaren auzokidea da (X,Y) arkuak existitzen badu; baina (Y,X) arkuak ez badu existitzen, X erpina ez da Y erpinaren auzokidea izango.

Adibidea:



Berriz ere, erpinen auzokideak aztertzean zenbakitze ordena jarraituz egingo da. Ondorioz, aurreko grafoa sakoneran korritzean erpinak ordena honetan aztertuko dira:

• 1 erpina abiapuntutzat aukeratzen bada:

1.-	S_K(1)	Lehenengo deia.
2.-	S_K(2)	Dei errekurtsiboa.
3.-	S_K(4)	Dei errekurtsiboa (irtenbiderik ez).
4.-	S_K(3)	Dei errekurtsiboa. 1en auzokidea
5.-	S_K(9)	Dei errekurtsiboa.
6.-	S_K(10)	Dei errekurtsiboa (irtenbiderik ez).
7.-	S_K(5)	Beste abiapuntu berri bat
8.-	S_K(6)	Dei errekurtsiboa.
9.-	S_K(8)	Dei errekurtsiboa.
10.-	S_K(7)	Dei errekurtsiboa (irtenbiderik ez).
		Grafoko erpin guztiak markaturik daude.

Algoritmoaren inplementazioa:

Emaniko auzokideen definizio hori kontutan izanik, grafoa sakoneran korritzeko aurreko S_K eta SAKONERAKO_KORRITZEA algoritmo berak erabil

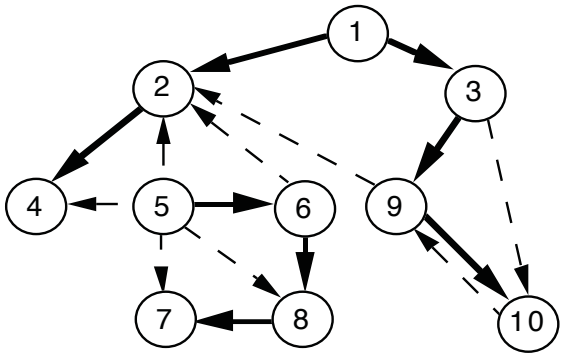
ditzakegu. Baina orain, algoritmoaren portaera desberdina dela argi gelditzen da adibidean.

Algoritmoaren ordenaren analisia:

Aurreko ataleko arrazonamendu bera jarraituz, algoritmoaren ordena O(max(P,A)) da, non P eta A-k grafoko erpin eta arkuen kopuruak adierazten dituzten hurrenez hurren.

Grafo konexuak/ez-konexuak sakonerako korritzean duen eragina:

Grafoa zuzendua denean, G= <ERP,ARK>, nahiz eta hau konexua izan korritzean erabilitako arkuek hedapen zuhaitz bat baino gehiago duen baso bat osa dezakete. Begira bedi gure adibidea:



3.4. ZABALERAKO KORRITZEA: Grafo ez-zuzendua

Grafo bat sakoneran korritzen dugunean, X erpin bat aztertu ondoren beti X-en auzokidea den beste erpin bat aztertzen saiatzen da, eta gero, beste honen auzokidea den beste erpin bat, eta horrela behin eta berriro. Zabalerako korritzean, berriz, X aztertu ondoren bere auzokide guztiak aztertzen dira eta ondoren, soilik ondoren, urrutiagoko erpinak.

Sakonerako korritzea naturalki errekurtsiboki egiten bada ere, ez da gauza bera gertatzen zabalerako korritzearekin. Bien arteko berdintasunak eta desberdintasunak aztertzearren lehendabizi sakonerako korritzea modu iteratiboan azalduko dugu.

Sakonerako korritze iteratiboa:

Pila egitura eta honen ‘pilaratu’, ‘pilatik atera’ eta ‘gailurra’ eragiketak erabili behar dira. Pila honek, “FIFO” estrategia inplementatuko du.

```
procedure S_K2 (X: in ERPINA) is
    P: PILA_DATU_MOTA;
    Y: ERPINA;
begin
    P:= HUTSA_P;
    PILARATU(P,X);
    while not( HUTSIK_DAGO_P?(P)) loop
       while EXISTITZEN_DA_GAILURRAREN_AUZOKIDE_BAT_
                  _MARKATU_GABEA(P) loop
          GAILURRAREN_AUZOKIDE_BAT_MARKATU_GABEA(P,Y);
          MARKA_IPINI(Y, AZTERTUA);
          PILARATU(P,Y)                          — X-ek pilan darrai
       end loop;
       PILATIK_KENDU(P);
    end loop;
end S_K2;
```

Zabalerako korritze iteratiboa:

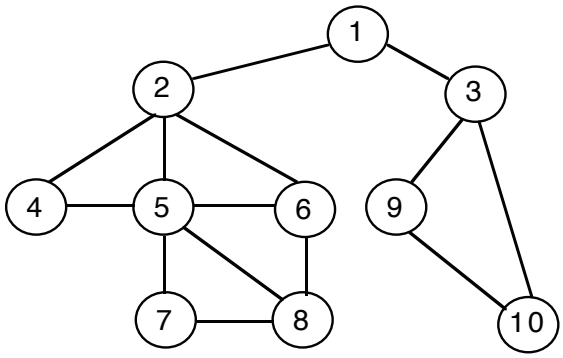
Grafoa zabaleran korritu nahi badugu, oraingoan behar den datu-egitura ilara izango da. Egitura honen burua ‘ilarari gehitzea’ eta ‘ilaratik kentzea’ eragiketak erabiliko ditugu.

```
procedure Z_K (G: in GRAFOA; X: in ERPINA) is
    ILARA: ILARA_DATU_MOTA:=HUTSA_I;
    Y_REN_AUZOKIDEAK: ERPINEN_L;
    Y,Z: ERPINA;
begin
    ILARARI_GEHITU(ILARA, X);
    while not( HUTSIK_DAGO_I?(ILARA)) loop
       Y:= BURUA(ILARA);
       ILARATIK_KENDU(ILARA);
       Y_EN_AUZOKIDEAK:= AUZOKIDEAK(G, Y).;
       while not (HUTSIK_DAGO_L?(Y_REN_AUZOKIDEAK)) loop
          Z:= LEHENA_L(Y_REN_AUZOKIDEAK);
          HONDARRA_L(Y_REN_AUZOKIDEAK);
          if not MARKATUA_DAGO?(BISITAK, Z) then
                MARKA_IPINI(BISITAK, Z, AZTERTUA);
                ILARARI_GEHITU(ILARA, Z);
          end if;
       end loop;
    end loop;
end Z_K;
```

Bai S_K2 eta Z_K prozedurei ondorengo programa nagusitik deituko zaie:

```
procedure  ZABALERAKO_KORRITZE_IT (G: in GRAFOA) is
      BISITAK:TAULA(ERPIN_KOP_ESPARRUA):= (others=>not(AZTERTUA));
begin
      for ERPIN_BAT in ERPIN_KOP_BARRUTIAK(G) loop
            if  not MARKATUA_DAGO (BISITAK, ERPIN_BAT)  then
                  MARKA_IPINI(BISITAK, ERPIN_BAT, AZTERTUA);
                  Z_K(G,ERPIN_BAT);            — edo S_K2(ERPIN_BAT);
            end if;
      end loop;
end  ZABALERAKO_KORRITZE_IT;
```

Adibidea:



Erpin baten auzokideak zenbakitze ordena jarraituz aztertzen dira.

• 1 erpina abiapuntutzat harturik:

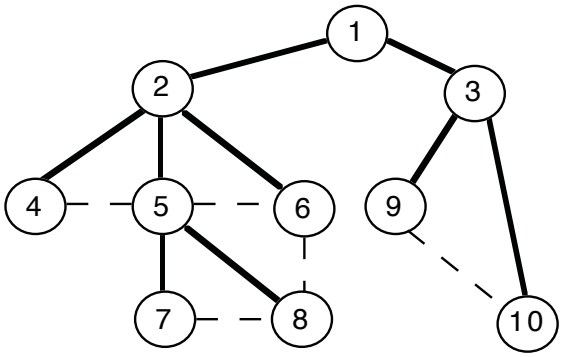
Ordena	*Aztertu*	*Ilara*
1.-	1	2,3
2.-	2	3,4,5,6
3.-	3	4,5,6,9,10
4.-	4	5,6,9,10
5.-	5	6,9,10,7,8
6.-	6	9,10,7,8
7.-	9	10,7,8
8.-	10	7,8
9.-	7	8
10.-	8	-

Algoritmoaren ordenaren analisia:

Sakonerako korritzeak hartzen duen denboraren ordena kalkulatu den bezalaxe, zabaleran korritzeak hartzen duen denboraren ordena O(max(P,T)) da.

Grafo konexuak/ez-konexuak zabalerako korritzean duen eragina:

Zabalerako korritzeari ere hedapen zuhaitz bat dagokio:



Grafo zuzenduen zabalerako korritzea:

Erpin auzokideen definizioa argi azalduz gero, zabalerako korritzea erraz egoki diezaiekegu grafo zuzenduei.

Noiz erabili zabalerako korritzea:

Korritze-mota hau batez ere infinituak diren grafoak partzialki aztertu nahi direnean edo eta bi punturen arteko biderik motzena aurkitu nahi denean erabiltzen da.

3.5. GRAFOEN BI KORRITZE INPLIZITU

3.5.1. Atzera-jotzezko algoritmoak: BACKTRACKING

Funtsean, sakonerako korritzea da.

Grafo zuzenduak aztertzeko erabiltzen diren algoritmoak ditugu, non korritzearen teknika inplizituki emana datorren. Normalki, grafoak aziklikoak izaten dira, zenbaitetan zuhaitzak ere izanik. Soluzioa aurkitzeko bilaketa metodoa zehatza da. Printzipio hau ondorengo problema ezagunei aplika diezaiekegu: labirinto bat zeharkatzen duen eta alferrikako bueltak ematen ez dituen bideen

zerrendaketa, 8 erregina 8x8-ko xake taula batean elkar jan gabe nola koka daitezkeen mugatzen duen posizioen zerrenda, Rubricken kaxaren egoera nahasi bat emanik kaxaren aldeak koloreka ordenatzen dituen mugimenduen sekuentzia mugatu... Problema hauek maiz jokoak izan ohi dira, edo eta jokoak balira bezala kontsidera genitzake. Halere, jokoak ez ezik beste hamaika problema ebazteko ere balioko du (ez ahaztu sakonerako korritzea dela).

Joko bakoitzeko ondorengo puntu hauek definitu behar dira (beste problema batzuentzat ere orokor daiteke):

<u>*Atzera-jotzeko algoritmoak soluzio posible guztiak bilatzen ditu*</u>:

– *Jokoaren egoerak*: une konkretu batean problemaren egoera deskribatzen du.

– *Mugimendu posibleen multzoa*: jokoaren edozein egoeratik beste batera pasatzeko egin daitezkeen mugimendu guztien multzoa.

– *Legezko mugimendua (onargarria)*: egoera jakin batean mugimendu bat onargarria edo posiblea den ala ez mugatzea beharrezkoa izango da.

– *Mugimendua erantsi*: soluzio oso bat biltegiratzeko, tarteko soluzioari pausoz pauso legezko mugimenduak erantsi beharko dizkiogu.

– *Mugimendua ezabatu*: diren soluzio guztiak kalkulatu nahi baditugu, atzera jotzea beharrezkoa da

– *Soluzioa da*: egoera berria gure problemaren soluzioetako bat da?

```
M_Sek:  Mugimenduen sekuentzia

prozedura  SAIOA  (E1: Egoera)
hasi
    Mugimendu_guztien_multzoa_hasieratu
    errepikatu baldin Mugimendu_posibleren_bat_gelditzen_da
        hasi
            Ondorengo_posibilitatea_aukeratu(M)
            baldin Legezkoa_da(E1,M)  orduan
                Mugitu(E1,M, E2)
                Mugimendua_erantsi(M_Sek, (E1,M,E2))
                baldin  Soluzioa_da (E2)  orduan
                    Soluzioa_idatzi (M_Sek)
                bestela  SAIOA(E2)
                Mugimendua_Ezabatu((E1,M,E2), M_Sek)
            bukatu
        bukatu
bukatu
```

Atzera-jotzezko algoritmoak soluzio bat bilatzen du:

Algoritmoan aldaketa kaxkar bat besterik ez da egin behar:

```
M_Sek:  Mugimenduen sekuentzia

prozedura  SAIOA  (E1: Egoera; Soluzio_bat: boolean)
hasi
    Mugimendu_guztien_multzoa_hasieratu
    errepikatu baldin (Mugimendu_posibleren_bat_gelditzen_da)
                                        ∧ (Soluzio_bat = false)
        hasi
            Ondorengo_posibilitatea_aukeratu(M)
            baldin Legezkoa_da(E1,M)  orduan
                Mugitu(E1,M, E2)
                Mugimendua_erantsi(M_Sek, (E1,M,E2))
                baldin  Soluzioa_da (E2)  orduan
                    hasi
                        Soluzio_bat:= true
                        Soluzioa_idatzi (M_Sek)
                    bukatu
                bestela  SAIOA(E2, Soluzio_bat)
                Mugimendua_Ezabatu((E1,M,E2), M_Sek)
            bukatu
        bukatu
bukatu
```

Lehenengo deia: SAIOA(Hasiera_egoera, false);

⇨ *Atzera-jotzeko algoritmoak soluzio onena bila dezake:*

Aurreko algoritmoak soluziorik onena bila dezan egoki daiteke.

Atzera-jotzeko algoritmoek behar duten denboraren ordenaren analisia:

Grafoan bilaketa exhaustiboa egiten dute. Baina:

– Grafoan zenbat arku dagoen ez dakigu: egoera bakoitzean mugimendu guztiak ez dira legezkoak,

– Grafoan zenbat erpin, egoera desberdin, dagoen ez dakigu beti;

Analitikoki kalkula ezin daitekeen problema da.

Oharra:

Ordena kalkulatzea posible izango liteke zenbat egoera desberdin aztertu beharko liratekeen ziur jakingo bagenu. Adibidez: 8 erreginen probleman

errenkadak zuhaitzaren mailak badira eta kalkulatu behar duguna erregina bakoitza 8 zutabeetatik zein zutabetan ezarri behar den bada, orduan 8^8 erreginen kokapen desberdinetatik soluzioak zein den mugatu beharko genuke. Posibilitate guztiak aztertuko bagenitu ordena hau izango litzateke: $\Theta(8^8)$. Baina lehenengo soluzioa bakarrik bilatu nahiko bagenu, goi bornea besterik ezingo genuke eman: $O(8^8)$. *n* erreginentzat berriz, n^n soluzio posibleak aztertu beharko lirateke. Baina *n*-ren balio horrentzat problemak soluziorik ez izatea gerta liteke. Hala ere, soluzio guztien analisiari legokiokeen ordena zehatza $\Theta(n^n)$ izango litzateke.

3.5.2. Ordenazio topologikoa

Adibide batez baliatuko gara problema zein den aurkezteko.

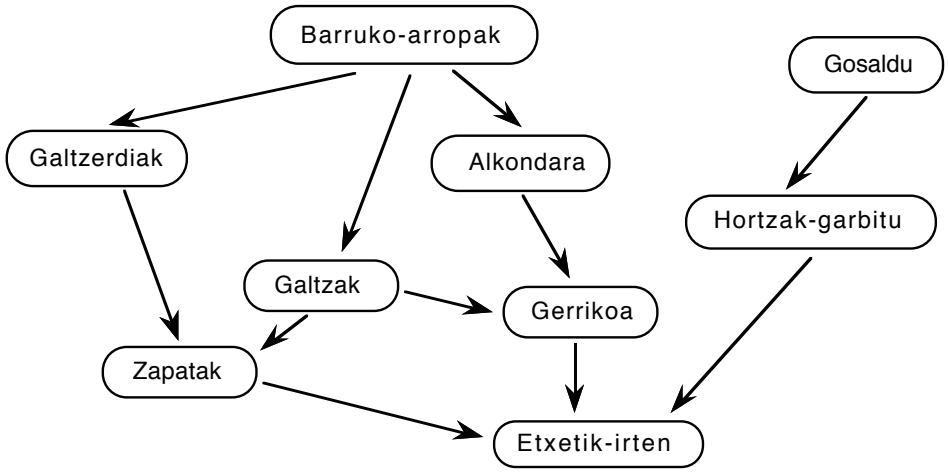
Egunero, lanera joan aurretik hamaika gauza egin behar dugu: hortzak garbitu, barruko arropak jantzi, galtzerdiak, zapatak, alkandora, gerrikoa eta galtzak jantzi, gosaldu eta irten. Prestaketa horien artean, begibistakoa da, derrigorrezkoa dela pauso batzuk beste batzuen aurretik egitea: galtzerdiak jantzi ondoren jantziko ditugu zapatak, hortzak gosaldu ondoren garbituko ditugu,... Halere, zenbait prestaketa edozein ordenan egin dezakegu: gosaldu jantzi ondoren egin dezakegu edo alderantziz. “Lanera joan aurreko prestaketak” ataza osoa eraginkorki egikari dezagun, zein ordenean sekuentziatu beharko genituzke azpiataza desberdinak?

Gure problema ebazteko azpiatazen zerrendaketa bat egin beharko dugu. Baina edozein zerrendaketak ez du balioko: azpiatazen menpekotasun ezagunak zerrendaketan mantendu eta jaso beharko dira.

Problema grafo zuzendu baten bidez modelatzen da. Azpiataza bakoitza grafoko erpin bat izango da. (X,Y) bi erpinen arteko arkuak X azpiataza Y azpiataza baino lehenago egin behar dela adieraziko du. Arkuek erpinen arteko prezedentziak definitzen dituzte; hau da, erpinen arteko *ordena partziala*. Problemak soluzioa izan dezan grafoak aziklikoa izan behar du.

Problema modelatzen duen grafoaz baliatuz, honek erpin bateko menpekotasunak jasotzen baititu, erpinen arteko menpekotasunak betetzen dituen azpiatazen zerrendaketa bat lortu nahi da; hau da, *ordena total* bat.

Adibidea:



Adibidean ikusten den bezala, azpiatazen menpekotasunek ez dute zertan ordenazio total bakarra eman behar: arku bakoitzak bi azpiatazen arteko menpekotasunak definitzen baditu ere, oraindik nolabaiteko askatasuna gelditzen zaigu zerrendaketa bat edo beste eraikitzeko. Hala eta guztiz ere, derrigorrezkoa da grafoa aziklikoa izatea ordenazio totalen batek existi dezan.

Adibideak eman ditzakeen bi ordenazio total:

– Gosaldu, barruko arropak, galtzerdiak, galtzak, zapatak, alkandora, gerrikoa, hortzak garbitu, etxetik irten.

– Barruko arropak, gosaldu, hortzak garbitu, alkandora, galtzak, gerrikoa, galtzerdiak, zapatak, etxetik irten.

Azpiatazen arteko menpekotasunak metatzen dituzten grafoen beste zenbait propietate ezagun:

– Grafoa zuzendua eta aziklikoa da.

– Erpin edo azpiataza bakoitzak guraso bat edo gehiago eduki ditzake. Gurasoen kopurua erpinera iristen diren arkuen kopuruaren berdina da: d^-.

– Erpin edo azpiataza bakoitzetik arku kopuru finitu bat aterako da. Kopuru honek erpinaren auzokideak, umeak, zenbat den ematen du: d^+.

– Grafo hauetan beti dago gutxienez erpinen bat $d^- = 0$ duena, hau da, haren aurretik derrigorrez egin behar den azpiatazik ez duena (ez du aurrekaririk). $d^- = 0$ duen erpinari *erroa* deritzogu.

– Bestalde, grafo hauetan beti dago gutxienez erpinen bat $d^+=0$ duena, ziklorik ez baitago eta ondorioz menpekotasun sekuentzia posible guztiek amaiera dutelako. $d^+ =0$ duen erpinari *hostoa* deritzogu.

– Grafo zuzendu azikliko bati erro bat kentzen badiogu, lortzen den grafo berriak gutxienez erro bat izango du beti. Nahiz eta prozesu bera behin eta berriro aplikatu, erpinak ditugun bitartean grafo berriek ere erroak edukiko dituzte.

Zerrendaketa nola egin? Algoritmoaren kalkulu prozesua:

Erroak zerrendaketan lehendabizi ager daitezkeen azpiatazak dira, hauetaraino arkurik ez baita iristen, eta ondorioz, ez baitute beste atazen ondoren derrigorrez azaldu beharrik. Demagun erro bakarreko grafo bat dugula. Erro hau, eraikiko den zerrendan, azalduko den lehenengo azpiataza izango da. Baina, honen ondoren zer? Erroa erantsia izan bada, orain erroaren umeak (hots, auzokideak) erro bihurtu badira, zerrendara gehi genitzake. Orokortuz, erro bat baino gehiago badugu une batean, erro baten eransketak erro horren erpin auzokideetan eragingo dio zerrendari. Auzokide bakoitzari orain aurrekari bat gutxiago gelditzuko zaio bere aurretik sekuentziara eransteko. Hain zuzen, erpinera beste arkurik iristen ez bazen, orain, erpina erro bihurtu da. Prozesua, erpinak gelditzen diren bitartean egikaritu behar da.

Grafoen zein adierazpide erabiltzea komeni da?

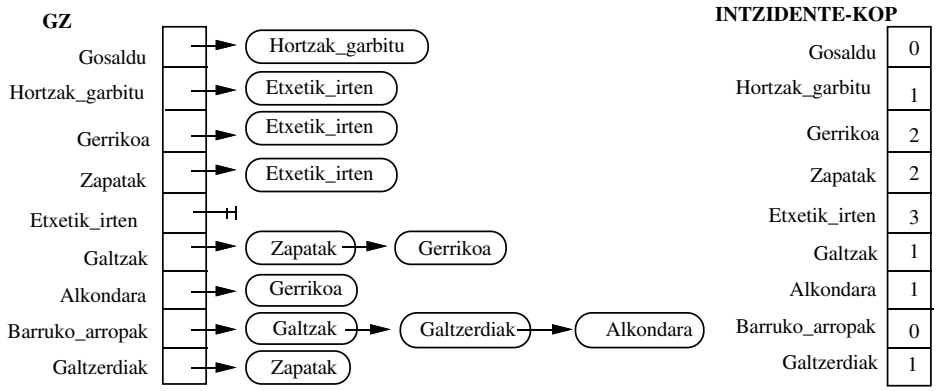
Erpin bakoitzeko gorde behar den informazioa honako hau da:

– Erpin bakoitzeraino iristen den arku kopurua, d^-, INZIDENTE_KOP taulan jasoko da. Informazio honi esker, zein erpin erro bihurtzen den jakin ahal izango dugu (bestalde, momentuoro gelditzen den grafoko erroak ERROAK izeneko aldagai batean metatu beharko dira).

– Erpin bakoitzaren auzokideak zein diren gorde behar da. Auzokideen lista bat nahikoa izango da erpin bakoitzeko.

Auzokideen lista bidezko adierazpidea erabil dezakegu.

“Lanera iritsi aurretiko prestaketak” adibidea adierazpide hau erabilita honela gelditzuko litzateke:



Erroak diren erpinak edozein ordenan erants diezazkiokegu zerrendari. Hori dela eta ERROAK aldagaia pila, ilara, lista edo multzo egituraz inplementa. Gure soluzioan pila aukeratu dugu.

Algoritmoa :

```
pocedure ORDENAZIO_TOPOLOGIKOA    (GZ:in out GRAFO_ZUZENDUA;
                                   ORD_TOTALA:out ERPINEN_L) is
    ERROAK:PILA; HURRENGO_ERPINA,AUZOKIDE_BAT:ERPINA;
    ORD_L:ERPINEN_L:=HUTSA_L;
    INZIDENTE_KOP: TUALA(ERPIN_KOP_BARRUTIA(GZ)):= (others=>0);
begin
    PILA_SORTU(ERROAK);
    ARKU_INZIDENTEEN_KOPURUA_MUGATU(GZ, INZIDENTE_KOP);
    for ERPINA in ERPIN_KOP_BARRUTIA(GZ) loop
        if ERROA_DA?(INZIDENTE_KOP, ERPINA) then
            PILARATU(ERROAK,ERPINA);
        end if;
    end loop;
    while not PILA_HUTSIK_DAGO?(ERROAK) loop
        HURRENGO_ERPINA:= GAILURRA(ERROAK);
        PILATIK_ATERA(ERROAK);
        ERANTSI_ATZETIK(ORD_L, HURRENGO_ERPINA);
        UMEAK:= AUZOKIDEAK(GZ, HURRENGO_ERPINA);
        while not HUTSIK_DAGO_L?(UMEAK) loop
            AUZOKIDE_BAT:= LEHENA_L(UMEAK);
            HONDARRA_L(UMEAK);
            TXIKITU_1(INZIDENTE_KOP, AUZOKIDE_BAT);
            if ERROA_DA?(INZIDENTE_KOP, AUZOKIDE_BAT) then
                PILARATU(ERROAK,AUZOKIDE_BAT);
            end if;
        end loop;
    end loop;
    if ADABEGI_KOP(ORD_L)= ERPIN_KOP(GZ) then
```

```
            ORD_TOTALA:= ORD_L;
        else
            ORD_TOTALA:= HUTSA_L;
            TEST_IO.PUT("Grafoak zikloak ditu");
        end if;
    end ORDENAZIO_TOPOLOGIKOA;
```

Eta INZIDENTE_KOP taula 0z hasieratuta dagoela suposatzen bada, erpin bakoitzairaino zenbat arku iristen den jasotzeko, arku guztiak korritu eta non amaitzen diren ikusi beharko da:

```
ARKU_INZIDENTEEN_KOPURUA_MUGATU (GZ, INZIDENTE_KOP):
for ERPINA in ERPIN_KOP_BARRUTIA(GZ) loop
    UMEAK:= AUZOKIDEAK(GZ, ERPINA);
    while not HUTSIK_DAGO_L?(UMEAK) loop
        GEHITU_1(INZIDENTE_KOP, LEHENA_L(URTEAK));
        UMEAK_L:= HONDARRA_L(UMEAK);
end loop;

ERROA_DA?(INZIDENTE_KOP, X):
INZIDENTE_KOP(X)=0
```

Ordenaren analisia:

Demagun grafoak P erpin eta A arku dituela.

(Grafoaren errepresentazioak O(P+A) memoria-espazio erabiliko du.)

Erpin bakoitzeraino iristen diren arkuen kopurua INZIDENTE_KOP taulan kalkulatzeko, lehendabizi taula 0z hasieratu behar da (O(P)), ondoren balio hauek banan bana arku guztiak tratatzen diren heinean eguneratu behar direlarik (O(A)). Honek, INZIDENTE_KOP taulako balioen kalkuluak, O(P+A) denbora-ordena eskatuko du.

Une jakin batean, grafoak zenbat erro duen jakiteko, grafoko erpin guztiak aztertu behar dira. Gainera, erpin bakoitzeko hura erroa den ala ez ikusi behar da (kalkulu honentzat denbora konstante bat beharko dugu). Beraz, ERROAK pilaren hasieraketa honek O(P) denbora-ordena beharko du.

Bestalde, erro bakoitzeko hura pilatik kendu, kanpoko zerrendara erantsi (aurreko hauek egiteko denbora konstante bat beharko da) eta, ondoren, bere auzokideak aztertu behar dira. Demagun pilatik hartzeko, kentzeko, soluzio listara gehitzeko eta auzokideak kopiatzeko (erakusle baten kopia egiteko), behar den denbora b_1 konstantea dela. Agindu sekuentzia hau erpin bakoitzekiko behin errepikatzen denez, guztira $P * b_1$ denbora beharko da. Baina, aldi berean, erpin

bakoitzaren auzokide bakoitzeko, hots, arku bakoitzeko, UMEAK auzokideen listatik auzokide bat lortu behar da, honen hondarra kalkulatu eta arku inzidenteen kopuruari 1 kendu, eta, erro bihurtu bada, ERROAK pilan pilaratu behar da. Demagun honek hartzen duen denbora b_2 konstantea dela, eta grafoak A arku dituenez, guztira $A * b_2$. Ondorioz, begiztaren ordena $O(P * b_1 + A * b_2) = O(P+A)$ izango da.

Amaitzeko, korritze berezi honen ordena lineala da, hots, $O(\max(P,A))$.

ARIKETAK: GRAFOEN KORRITZEAK

1- G=(ERP, ERT) grafo ez-zuzendu baten osagai konexuak kalkulatzeko partiketa egitura erabiltzen duen ondorengo algoritmoa ematen da:

```
proz OSAGAI_KONEXUAK (G)
     — Partiketaren hasieraketa
hasi
     for a ∈ ERP erpin bakoitzarentzat egin Multzoa_egin(v)
     end for;
     for (x,y) ∈ ERT ertz bakoitzarentzat egin
          baldin BILATU(x) ≠ BILATU(y) orduan
               BATERATU(ETIKETA(x),ETIKETA(y))
          end baldin;
     end for;
end proz;
```

Grafoak K osagai konexu baditu, zenbat BILATU eragiketa egingo da? Eta zenbat BATERATU eragiketa? Emaitza |ERT|, |ERP| eta K-ren funtziopean eman.

2- (Xj,Yj) jatorri puntutik (Xn,Yn) helmuga puntura zenbat bide posible dagoen mugatzen duen algoritmo bat idatzi bedi. Puntuen koordenadak zenbaki arruntak direla suposatu. Bestalde, (X,Y) puntu batetik beste puntu batera joateko eman daitezkeen pausoak (X+1,Y) eta (X,Y+1) bakarrik izango dira. Algoritmoaren exekuzio-denbora aztertu.

3- G=(Erpinak, Arkuak) grafo zuzenduentzat beste adierazpide berezi bat, |Erpinak|x |Arkuak| inzidentzi matrize bidezko adierazpidea da. IM inzidentzi matrizeko IMij posizioan ager daitekeen balioa

$$IM_{ij} = \begin{cases} +1 & \text{j arkua i erpinetik irtetzen da} \\ -1 & \text{j arkua i erpinera iristen da} \\ 0 & \text{bestela} \end{cases}$$

izan daiteke. Adierazpide konkretu hau duen grafo bat sakoneran korritzen duen algoritmoa idatzi bedi. Zein da algoritmoaren ordena zehatza?

Algoritmoak egindako lana aztertuz argudia bedi.

Zabalerako korritzearekin gauza bera egin bedi.

4- Grafo bat *bipartitua* da baldin eta soilik baldin bere erpinen multzoa bi azpimultzo disjuntutan partitu badaiteke, non grafoan ez dagoen ertzik erpinen azpimultzo bereko bi erpin elkartzen dituenik.

Bila ezazu grafo konexu eta ez-zuzendu bat *bipartigarria* den ala ez mugatzen duen eta ertzen kopuruan lineala izango den algoritmo bat.

5- Zuhaitz baten *diametroa* edozein erpin bikote arteko distantzia guztietatik distantzia handiena da. Zuhaitz baten diametroa kalkulatzen duen algoritmo lineal bat eraikitzea eskatzen da.

6- n erpin eta a arku dituen grafo zuzendu azikliko bateko erpinei [1..n] tarteko zenbaki desberdinak esleitzen dien algoritmo bat idaztea eskatzen da, asignazio metodoak ondorengoa izan behar duelarik: $\forall$v, w grafoko erpinak izanik

baldin v $\neq$ w eta baldin v -tik w -raino bideak badu
orduan ZENBAKIA[v] < ZENBAKIA[w]

Zein da algoritmoaren ordena zehatza? Erantzun biak arguia bitez.

7- Grafoa auzokideen matrize bidez adierazirik baletor, zein litzateke sakonerako korritzea kalkulatuko lukeen algoritmoaren exekuzio-ordena?

8- Auzokideen listen bidez adierazitako grafo zuzendu bat emanik, zenbat denbora beharko litzateke erpin bakoitzeko d+, out-degree, kalkulatzeko? Eta d–, in-degree, kalkulatzeko?

9- Matrize bidezko adierazpidea erabiltzen denean grafo gehienek $Q(|Erpinak|^2)$ denbora behar dute. Halere, salbuespen batzuk badaude. Bestalde, grafo bateko erpin bat ISURLEKU erpina da, baldin eta $d^-=|Erpinak|-1$ eta $d^+=0$ baditu. Froga bedi grafo batek ISURLEKU erpinik duen edo ez mugatzeko O(|Erpinak|) denbora behar duen algoritmo batekin aski dela, nahiz grafoa auzokide matrize bidez adierazirik eman.

10- a) Grafo ez-zuzendu batek ziklorik ote duen mugatuko duen algoritmo bat idaztea eskatzen da.

b) Grafo zuzendu batek ziklorik ote duen mugatuko duen algoritmo bat idaztea eskatzen da.

c) Aurreko ataletan sakonerako korritzea erabili baduzu, berridatz itzazu algoritmoak zabalerako korritzea erabiliz problema berdinak ebatzi ditzaten, eta alderantziz. Zein estrategia nahiago duzu? Arrazoi sendorik baduzu horretarako? Zein?

11- G=(Erpinak, Ertzak) grafoa zuhaitza ote den mugatzen duen algoritmoa idatzi. Grafoa konexua balitz, algoritmo berdina erabiliko zenuke? Horrela ez balitz, aurrebaldintza bezala grafo konexua eskatuko lukeen algoritmoa idatzi.

12- G grafo ez-zuzendu batean v eta w erpinen arteko bide motzena kalkulatu nahi da, non distantzia bideko erpinen kopuruak ematen duen (ad. hegazkin bidezko bidai batean ahalik eta geldiunerik gutxien mugatzeko). Ikusitako korritze algoritmoen artean, zein aukeratuko zenuke? Zergatik?

4. GAIA

ALGORITMO JALEAK

4.1. SARRERA

Optimizazio algoritmo tipikoak dira. Pauso bakoitzean hautagai-multzo batetik hautagai bat aukeratzen duten algoritmoak dira. Optimizazio problema askotan programazio dinamikoko teknika erabiltzea posiblea bada ere, eraginkortasun aldetik ez da onena. Algoritmo jaleen teknikak beti momentuan hoberen dirudien aukera egiten du; hau da, momentuoro optimizazio lokala egiten du, honek soluzio optimo orokorrera gidatuko duelakoan. Hala ere, ez du beti soluzio optimoa ematen, baina zenbait problematan bai, eta nahiko algoritmo sinpleak lortzen direnez, eraginkortasun aldetik onak dira. Teknika jalea, zenbait problema ezagunetan erabili ohi da: konputagailu batean ataza multzo bat exekutatzeko sekuentzia onena bilatu, grafo bateko bide motzena aurkitu... Arazo hauetan guztietan normalki osagai hauek aurki ditzakegu:

– hautagaien multzoa (prozesa ditzakegun atazak, zeharkatu behar diren grafoko erpinak)

– jadanik aukeratu eta onartu diren hautagaien multzoa, non aurrekoaren azpimultzoa den.

– hautagaien azpimultzo bat gure arazoaren soluzioa den ala ez erabakitzen duen funtzioa (nahiz eta soluzio hoberena ez izan).

– *prozedura hauteslea:* oraindik aukeratu ez diren hautagaietatik soluzio onenaren bideratzailea den hautagaia aukeratuko duen prozedura (hautagai bat aukeratu zen unean onena bazirudien ere, ondoren onena ez zela ikustea gerta liteke).

– *helburu funtzioa*: soluzio bati dagokion balioa edo kostua itzultzen duen funtzioa. Soluzioetan maximoa edo minimoa aurkitu nahi dugu, oro har, onena. Beraz, helburu funtzioa maximizatu edo minimizatu egin beharko dugu.

– aukeratutako azken hautagaia tarteko emaitzari erantsiko ote zaion erabakitzen duen funtzioa. Honela, tarteko emaitzari aukeratutako azken hautagaia eransterakoan lortuko litzatekeen multzo berria *osogarria* balitz, orduan tarteko emaitzari hautagaia benetan eranstea onartuko da, bestela ez. Bestalde, multzo bat *osogarria* dela esango dugu, baldin eta multzoari

hautagai gehiago eratsiz gero soluzio bat (nahiz eta onena ez izan) eskura badezakegu, hau da, eraikitzen ari den multzoa soluzio onenaren bideratzailea bada.

Optimizazio arazo bat ebatzi nahi dugunean, honen soluzioa den eta helburu funtzioa optimizatzen duen hautagaien azpimultzo bat aurkitu behar da.

Algoritmo hauek pausoz pauso egiten dute lan. Hasieran, aukeratutako hautagaien azpimultzoa hutsa da. Ondoren, eta pauso bakoitzean, oraindik hautatu gabeko eta haietatik onena den hautagaia aukeratzen da. Aukera egiteko *funtzio hauteslea* erabiltzen da. Lortzen den azpimultzoa osogarria ez bada, hautagaia ez da onartzen, eta ondorengo saiaketetan ere ez da kontutan izango. Baina osogarria bada berriz, dagoeneko aukeratutako hautagaien multzoari eransten zaio, eta hemendik aurrera beti soluzio multzoko partaidea izango da. Azpimultzoari hautagai berria erantsi ondoren, lortzen den multzo berria, arazoaren soluzioa den ala ez egiaztatzen da. Algoritmo jaleak soluzio onena beti lortzen badu, orduan algoritmo jalea zuzena da.

Algoritmo jaleen eskema generikoa:

```
funtzioa JALEA (H: Hautagai-multzoa) itzuli Hautagai-multzoa da
    S: Hautagaien-multzoa;
hasi
    Multzo_hutsa(S);    {Soluzioa S multzoan eraikiko da}
    errepika  not( Soluzioa?(S)) and  not(Multzo_hutsa?(H))  egin
          Hautesle-prozedura(H,x);     — x hautatu jalea
          baldin  Osogarria?(Erantsi(S,x)) orduan
                S:= Erantsi (S,x)
          bukatu baldin;
    bukatu errepika;
    baldin  Soluzioa?(S)  orduan return(S)
    bestela
          Multzo_hutsa(S);     {Ez dago soluziorik}
          return (S)
    bukatu baldin;
bukatu JALEA;
```

Nondik datorkie algoritmo hauei izena? Pausu bakoitzean zatirik onena jaten dute etorkizunaz arduratu gabe. Optimizazio lokala egiten dute, baina behin hartutako erabakirik ez dute sekula desegingo. Honela, soluzioa izango den hautagaien multzoari hautagai bat eransten badio, hautagaia bukaeraraino multzoko partaidea izango da, eta une batean onartzen ez dena ez da sekula onartuko, ere bere onarpena ez baita berriz berkontsideratuko.

Prozedura *hauteslea* eta *helburu* funtzioa erlazionatuta egon ohi dira, zenbaitetan funtzio bera izanik. Zenbait kasutan bi funtzio *hautesle* egon daitezkeela iruditzen bazaigu ere, ona bakarra izango da.

Algoritmo jaleen itxura oso sinplea izan ohi da. Zenbaitetan hain dira sinpleak, soluzio onena BETI kalkulatzen ote duten zalantzan jartzen dela. Bestetan, aldiz, zenbait proba, konplexu edo ez, egin ondoren, algoritmoek zuzenak dirudite. Bai bata eta bai besteagatik, komenigarria da algoritmo jaleen zuzentasuna frogatzea. Informatikariontzat, sarrera onargarri guztientzat algoritmo jaleek zuzenak izan daitezen soluzio onena kalkulatu beharko dute.

Algoritmo jaleen zuzentasuna frogatzeko errezeta bat:

(Derrigorrezkoa da optimizazio problemak sarrera onargarri guztientzat gutxienez soluzio optimo bat izatea.)

☞ Algoritmo jaleen gakoa hautaketan dago. Normalki, algoritmo hauen zuzentasuna hautaketa *segurua* eta *induktiboa* dela frogatzean oinarritzen da:

Hautaketa *segurua* da, baldin eta hautaketak aukeratzen eta onartzen dituen hautagaiak barnean dituen soluzio optimo batek existitzen badu. Edozein soluzio optimotik abiatuta hautatua barnez duen soluzio optimo bat eraikiz frogatu ohi da propietate hau.

Hautaketa *induktiboa* da baldin eta S jatorrizko H problemaren eta bere barne HJ hautaketa duen soluzio optimo bat izanik orduan S-{HJ} jatorrizko H problemaren pareko problemaren soluzio optimo bat bada eta pareko azpiproblemaren soluzio optimoak {HJ}-rekin bateragarriak badira.

"{HJ}-rekin bateragarriak" espresioak jatorrizko H problemaren pareko azpiproblema soluzio optimo bati {HJ} erantsiz gero H-ren soluzio optimo bat lortzen dela esan nahi du.

Bi propietate hauek betetzen direla frogatu ondoren, algoritmo jalearen zuzentasunaren frogaketa borobiltzeko oraindik honako hau frogatu behar da:

☞ S soluzio hoberenaren bideratzailea dela egindako hautaketen kopuruan indukzioa eginez frogatu behar da; hau da:

S (aukeratu eta onartutako) hautagaien 'soluzio' multzoa da eta H jatorrizko problemaren hautagaien multzoa. Frogaketarako (1) eta (2) propietateak beharko dira.

– Oinarrizko kasua: $|S|=1$.
  Jakinik (1): $\exists$ OPT soluzio optimo bat dago (gutxienez). $S \subseteq OPT \Rightarrow$ S soluzio hoberenaren bideratzailea da; hau da, osogarria da.
  Jakinik (2): $\exists$ H_1 azpiproblema H problemaren pareko azpiproblema da eta bere soluzio optimo bat OPT-S da.

– Indukzio hipotesia:

|S|=n-1 eta S soluzio hoberenaren bideratzailea da.

H_{n-1} azpiproblema H problemaren pareko azpiproblema da eta S-rekin bateragarriak diren soluzio optimoak ditu.

– Kasu orokorra:

Algoritmo jaleak H_{n-1} azpiproblema ebazteko *n*. hautaketa jalea, HJ_n, onartzen duenean:

Jakinik (1): Izan bedi OPT_{n-1} H_{n-1} problemaren soluzio optimo bat bere barne algoritmo jaleak erabakitako lehenengo hautagaia duena, hots, HJ_n (*n*. hautagai onartua).

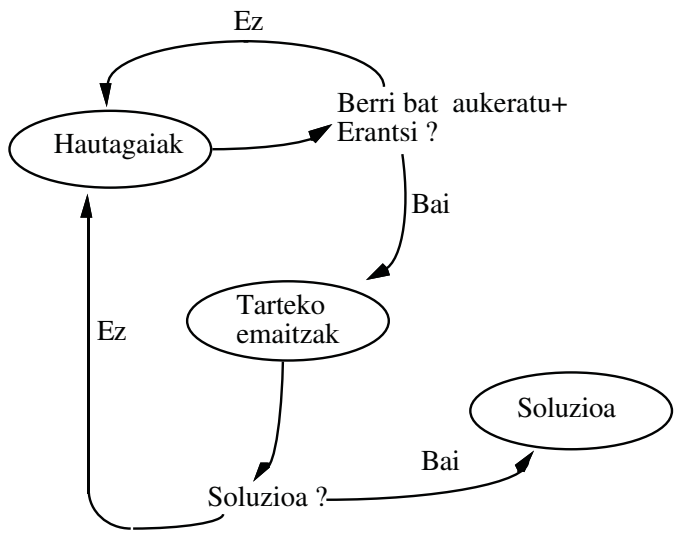
OPT_{n-1} S-rekin bateragarria dela badakigu (i.h.): $OPT_{n-1} \cup S$ H-ren soluzio optimo bat da eta hautaketa jaleak onartutako *n*. hautagaia barnean duena gainera.

$\Rightarrow$|S|=n eta S soluzio hoberenaren bideratzailea da.

Jakinik (2): OPT_{n-1}-$\{HJ_n\}$ H_n azpiproblemaren soluzio optimo bat (H_{n-1} azpiproblemaren parekoa, bai eta Hrena ere) eta bere soluzio optimoak $\{HJ_n\}$-kin bateragarriak dira (H_{n-1} ebazteko, eta ondorioz, H ebazteko) (hurrengo pausorako dena prestatuta utzi behar da).

4.2. ALGORITMO JALEAK GRAFOETAN

Nahiz eta hemendik aurrera grafo ez-zuzenduetaz hitz egin, grafo zuzenduentzat azterketa antzekoak egin daitezke.



Algoritmo jaleen prozesua

Hautagaien hasierako multzoa grafoko hautagaiez (ertzez, arkuz, edo erpinez) osaturik dagoen multzoa izango da. Algoritmo jale batek hautagaiak ordena konkretu batean aukeratuko ditu. Aukeratutako hautagaia onartzen badu, erdibideko soluzioari erantsiko dio, bestela ez. Hautagai hori, edozein kasutan, ez da berriz ere berkontsideratuko. Arazo bera ebazten duten algoritmo jaleak hautagaiak kontsideratzeko ordenean desberdintzen dira.

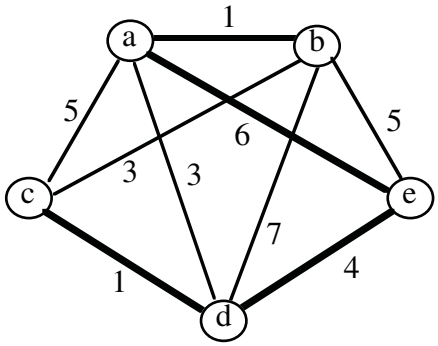
Ondorengo azpiatalen ikusiko ditugun algoritmoek grafo bat emanik haren hedapen zuhaitz minimo bat kalkulatzen dute, horretarako denek teknika desberdinak erabiltzen dituztelarik.

4.2.1. Hedapen zuhaitz minimoak

Demagun G = <Erpinak, Ertzak, Pisuak> grafo konexu, pisudun eta ez--zuzendua. Ertz bakoitzak pisu positibo bat izango du erlazionaturik: ertzaren luzera edo kostua.

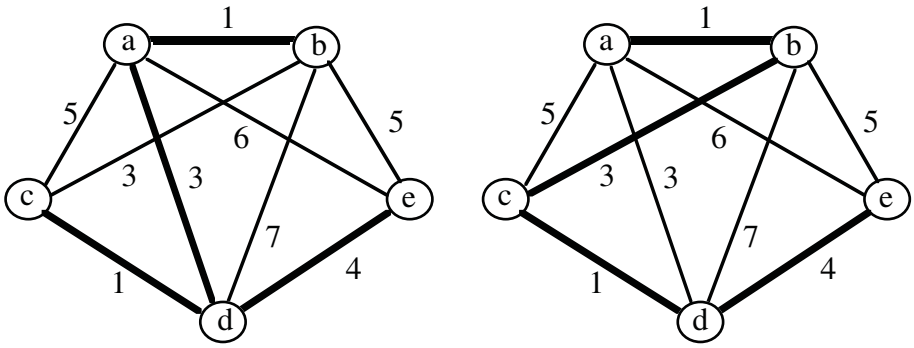
Helburua: G-ko erpin guztiak konektatzen dituen eta haien pisuen batura minimizatzen duen ERT multzoa mugatzen da. Grafo berria <Erpinak,ERT,PISU> *hedapen zuhaitz minimoa* dela esango dugu.

Grafo bateko hedapen zuhaitz eta hedapen zuhaitz minimoen adibideak:



Grafoa:
Erpinak = {a, b, c, d, e}
Ertzak = {(a,b), (a,c), (a,e), (a,d)
(b,c), (b,d), (b,e), (c,d),
(d,e) }
Pisuak = { ... }
Markaturiko hedapen zuhaitza:
Erpinak = { a, b, c, d, e}
Ertzak = {(a,b), (a,e), (c,d), (d,e) }
Pisuak = { ... }

Hedapen zuhaitz minimoa hedapen zuhaitza da, non bere ertzen pisuen batura beste hedapen zuhaitzena baino txikiagoa edo berdina den.



Grafoko bi hedapen zuhaitz minimo dira.

Zenbait aplikazio:

Adabegi-multzo bateko adabegiak modurik 'merkeenean' konektatu nahi ditugun arazoetan algoritmo jaleak erabil ditzakegu: erpinak (hemendik aurrera grafoetako adabegiak honela izendatuko ditugu) hiriak, zentral elektrikoak, konputagailuak, lantegiak eta abar izan daitezke eta hauek konektatzen dituzten ertzak berriz bideak, kableak, telefono hariak, eta abar.

Adib.: Saltzailearen problema: Saltzaile batek ondorengo hau jakinaz
- Erpinak: bisitatu behar dituen hiriak
- Ertzak: hiri batetik bestera joateko dauden bide desberdinak
- Pisuak: bide bakoitzaren kostua (luzera, bidean eman behar den denbora edo gasolinaren kostua)

Hiri guztiak kostu merkeenean bisitatu nahi ditu. Zer bide jarraitu beharko luke? Grafoaren hedapen zuhaitz minimoa jarraitu beharko luke.

Hedapen zuhaitz minimoak zenbait bideratze algoritmoren azpiproblema garrantzitsuak dira.

4.2.2. Kruskal

4.2.2.1. Partiketa datu-mota

Demagun zenbakituriko P objektu ditugula, 1-etik P-ra. Objektu hauek multzo disjuntuetan bildu nahi ditugu. Une bakoitzean objektu bakoitzak multzo bateko, eta bakar bateko, partaidea izan behar du: adierazi nahi dugun egitura *partiketa* da eta multzo bakoitzak baliokidetasun-klase bat adieraziko du. Multzo bakoitzean multzoko etiketa izango den objektu kanoniko bat aukeratuko dugu. Hasieran, P objektuak P multzotan banatzen ditugu; beraz, multzo bakoitzak objektu bat bilduko du. Objektu (bakar) hori multzoko etiketa izango da. Eragiketa

honez gain, egitura hau maneiatuko duten beste bi eragiketa sakonkiago aztertuko ditugu:

– Bilatu: objektu bat emanik, hau zein multzotako partaidea den mugatuko du, multzoko etiketa itzuliz.

– Bateratu: bi multzo desberdinetako etiketak emanik, bi multzoak bat egiten ditu.

Bi eragiketak eraginkorki inplementa al ditzakegu?

• 1. Soluzioa:

Suposa dezagun ondorengo definizio eta erazagupen hauek ditugula:

– Multzoko etiketatzat multzoko osagairik txikiena aukeratzen dugu. Adib.: {7,16, 3, 9} multzoa bagenu, bere etiketa "3" litzateke, eta

– PARTIKETA(1..P): P objektuak zenbakituta ditugunez gero, multzoaren erazagupena P osagai dituen osoen taula baten bidez egin dezakegu. Taulako *i* posizio bakoitzean *i* objektuaren multzoko etiketa biltegiratuko dugu.

Eragiketen inplementazioa ondorengo moduan egin liteke:

```
function  BILATU1 (PARTIKETA: in P_PARTIKETA_MOTA; X: in OSAG_ETIK)
                    return OSAG_ETIK is
begin
    return  PARTIKETA(X);
end BILATU1;
procedure BATERATU1 (PARTIKETA: in out P_PARTIKETA_MOTA;
                        E1,E2: in OSAG_ETIK) is
    I,J: OSAG_ETIK;
begin
    — E1 eta E2 etiketak dituzten multzoak bat egiten ditu.
    I:= E1; J:= E2;
    if  I>J  then
          TRUKEA(I,J);
    end if;            — bi multzoen etiketa berria I
    for  K  in  1..P  do
          if PARTIKETA(K)=J then PARTIKETA(K):= I; end if;
    end loop;
end BATERATU1 ;
```

Partiketa bat sortu ondoren, hau maneiatzeko bi funtzio hauek erabiliko ditugu; beraz, *n* luzera duen Bilatu eta Bateratu eragiketen sekuentzia batek hartzen duen exekuzio-denbora aztertzea interesatzen zaigu:

– P_PARTIKETA_MOTA taula bateko osagai baten eguneratzea edo bilaketa eragiketak oinarrizko eragiketak direla kontsideratzen badugu, BILATU_1-en exekuzioak hartzen duen denbora-ordena $\Theta(1)$ izango da eta BATERATU_1-en ordena berriz $\Theta(P)$. Kasu txarrenean jarraian eginiko n eragiketek hartuko duten denboraren ordena $\Theta(n\ P)$ izango da.

• 2. Soluzioa:

Saia gaitezen aurrekoa hobetzen. P_PARTIKETA_MOTA taula bakarra erabiltzen badugu ere, multzo bakoitza zuhaitz baten bidez (eta taula bakar baten gainean) adieraz dezakegu. Ondorengo hitzarmena hartuko dugu adierazpide berrian:

– baldin PARTIKETA(i) = i, orduan *i* objektua bere multzoko etiketa da eta aldi berean dagokion zuhaitzaren erroa.

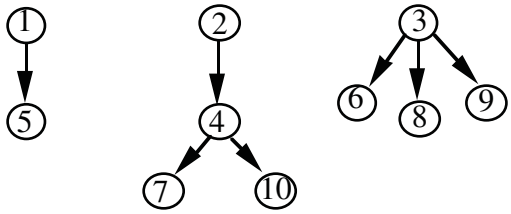
– bestela, PARTIKETA(i) = j ≠ i, i eta j objektuak multzo berean daude eta multzo horri dagokion zuhaitzean, *j* adabegia *i*-ren gurasoa da.

Adibidea:

Ondorengo taulak

1	2	3	2	1	3	4	3	3	4
1	2	3	4	5	6	7	8	9	10

zuhaitz hauek adierazten ditu:



Aldi berean, zuhaitz hauek {1, 5}, {2, 4, 7, 10} eta {3, 6, 8, 9} multzoak adierazten dituzte hurrenez hurren (irudietan, erakusleek arkuen norabidea zuhaitzetan adierazten dute eta ez taulakoa).

Bi multzo bat egiteko, elkartzeko, taulako balio bat eguneratzea nahikoa da. Baina, objektu bat zein multzotako partaidea den jakitea zailagoa da:

```
function BILATU2 (PARTIKETA: in P_PARTIKETA_MOTA; X: in OSAG_ETIK)
                    return OSAG_ETIK is
    I:OSAG_ETIK:= X;
    — X partaidea den multzoko etiketa itzultzen du.
begin
    while  PARTIKETA(I) /=I   loop
            I:= PARTIKETA(I);
    end loop;
    return (I)
end BILATU2;

procedure BATERATU2 (PARTIKETA: in out P_PARTIKETA_MOTA;
                        E1,E2: in OSAG_ETIK) is
    — E1 eta E2 etiketak dituzten multzoak bat egiten ditu.
begin
    if  E1<E2  then                    — Multzo berriaren etiketa
          PARTIKETA(E2):= E1;          — bat egindako bi multzoetako
    else  PARTIKETA(E1):= E2;          — objektu txikiena da: hots,
    end if;                            — bi etiketetatik txikina.
end BATERATU2 ;
```

Oraingo honetan ere, *n* BILATU$_2$ eta BATERATU$_2$ eragiketa jarraian exekutatzen baditugu, sekuentzi horren exekuzio denbora zein ordenakoa izango da kasu txarrenean eta hasiera egoera batetik hasten garela suposatuz? (berriz ere, taularen posizio baten eguneratzea eta bilaketa eragiketak oinarrizko eragiketak direla suposatuko dugu). Analisia:

– BATERATU$_2$ funtzioaren ordena $\Theta(1)$-koa da: edozein kasutan bi osoen arteko alderaketa bat eta ondoren taulako posizio baten eguneratzea egiten da. (*n* BATERATU$_2$ eragiketa exekutatuko bagenitu ere, hauen exekuzio denboraren ordena $\Theta(n)$ litzateke).

– BILATU$_2$ funtzioaren azterketa: zuhaitz bateko objektu bat bilatu nahi dugunean, objektu honen zuhaitzeko maila zenbat eta handiagoa izan orduan eta denbora gehiago beharko da BILATU$_2$ funtzioak objektu hori aurkitzeko. Hain zuzen, *m* mailan dagoen objektu bat zein multzokoa den erabaki nahi dugunean BILATU$_2$ hartzen duen denboraren ordena $\Theta(m)$ da.

BATERATU$_2$ eta BILATU$_2$ -ren ordenak zein diren jakin ondoren eta *n* dei jarraian exekutatzen baditugu hasierako egoera batetik hasita, zein izango da *n* luzera duen BATERATU$_2$ eta BILATU$_2$ eragiketez osaturiko sekuentzia txarrena (*x* denbora gehiena hartzen duena)?

Suposa dezagun programa baten agindu-sekuentzia ondorengo hau dela:

1. BATERATU$_2$(n/2+1,n/2)
...
(n/2-1). BATERATU$_2$(3,2)
(n/2). BATERATU$_2$(2,1)
(n/2+1) BILATU$_2$(n/2+1)
..
n BILATU$_2$(n/2+1)

programa honek hartzen duen denboraren ordena: $\Theta((n/2)$ Bateratu$_2$ -ren ordena)+ $\Theta((n/2)$ Bilatu$_2$ $(n/2+1)$-en ordena) = $\Theta(n/2) + \Theta((n/2)(n/2+1)) = \Theta((n/2)(n/2+1)) = \Theta((n/2)^2)) = \Theta(n^2)$

P eta n balioak antzekoak direnean ez dugu ezer irabazi BILATU$_1$ eta BATERATU$_1$-ekiko. Baina, BATERATU$_2$ funtzioaren k deien ondoren, k sakonera duen zuhaitz bat lortzea gerta dakiguke, eta ondoren, BILATU$_2$ funtzioari azkeneko mailan dagoen objektu bat bilatzeko dei egingo bagenio, honen exekuzioak kasu txarrenean beharko zukeen denboraren ordena $\Theta(k)$-koa litzateke. Hau dela eta, zuhaitzen sakonera handiegia ez izatea gomendagarria da. Beraz, zuhaitzen sakonera murriztu beharko dugu.

• 3. Soluzioa:

Multzoetako etiketak multzoko objekturik txikienak izan zitezen erabaki bagenuen ere, azter dezagun beste zenbait posibilitate. Estrategia:

s_1 eta s_2 sakonera duten bi zuhaitz bateratu nahi baditugu, gomendagarria litzateke sakonera txikiena duen zuhaitza sakonera handiagoa duen zuhaitzaren erroaren haurra bihurtzea. Horrela jokatuz gero, lortzen den zuhaitz berriaren sakonera $\max(s_1,s_2)$ izango da baldin eta $s_1 \neq s_2$ bada; bestela, $s_1=s_2$, s_1+1ekoa izango da. Estrategia honekin zuhaitzen sakonerak ezin du oso azkar hazi.

Lema:

Estrategia hau erabiltzen badugu hasierako egoeratik hasi, eta bateratu eragiketen edozein sekuentzi exekutatu ondoren, k adabegi dituen zuhaitz baten sakonera $\leq \lfloor \lg k \rfloor$ dela froga daiteke.

Frogaketa: indukzio bidez.

– Suposa dezagun adabegi bat dugula; orduan zuhaitzaren sakonera $s_1=0$ izango da. Beraz, betetzen da $0 \leq \lfloor \lg 1 \rfloor$

– Indukzio hipotesi hedatua: $\forall i\ (1 \leq i < n \rightarrow s_i \leq \lfloor \lg i \rfloor)$ i adabegi dituzten zuhaitzentzat propietatea betetzen da.

– Suposa dezagun orain:

1- z_1 eta z_2 zuhaitzek k_1 eta k_2 adabegi dituztela hurrenez hurren,
2- $k_1 < n$ eta $k_2 < n$ eta $k_1+k_2 \geq n$ betetzen dela,
3- s_{k_1} eta s_{k_2} z_1 eta z_2 zuhaitzen sakonerak direla, eta beraz
4- i.h. aplikatuz $s_{k1} \leq \lfloor \lg k_1 \rfloor$ eta $s_{k2} \leq \lfloor \lg k_2 \rfloor$ beteko da.

z_1 eta z_2 elkartzerakoan zuhaitz berriak k_1+k_2 adabegi izango ditu. Baina, bere sakonerak, betetzen al du $s_{k_1+k_2}$, propietatea? $s_{k_1+k_2} \leq \lfloor \lg(k_1+k_2) \rfloor$ betetzen du? Aztertu dezagun. Bi kasu desberdin gerta dakizkiguke :

(a) $s_{k_2} \neq s_{k1}$ ($s_{k1} < s_{k_2}$ edo $s_{k_2} < s_{k_1)}$:

Elkartzen ditugun bi zuhaitzen sakonerak desberdinak dira. Suposa dezagun $s_{k_1} < s_{k_2}$. Beraz, sakonera txikiena duen zuhaitza, s_{k_1}, beste zuhaitzaren errotik zintzilikatu ondoren lortzen den zuhaitz berriaren sakonera $s_{k_1+k_2}$ sakonera handiena duen zuhaitzaren sakoneraren berdina da:

$$s_{k_1+k_2} = s_{k_2} \leq \lfloor \lg k_2 \rfloor \leq \lfloor \lg(k_1+k_2) \rfloor \quad \text{beraz} \quad s_{k_1+k_2} \leq \lfloor \lg(k_1+k_2) \rfloor$$

($s_{k_2} < s_{k_1}$ kasuaren frogaketa berdin egiten da)

(b) $s_{k_1} = s_{k_2}$

Elkartzen diren zuhaitzen sakonerak berdinak dira. Kasu honetan lehenengo zuhaitza bigarrenaren errotik zintzilika dezakegu. Ondorioz:

$$s_{k_1+k_2} = s_{k_2} + 1 \leq \lfloor \lg k_2 \rfloor + \lfloor \lg 2 \rfloor = \lfloor \lg 2 + \lg k_2 \rfloor = \lfloor \lg 2k_2 \rfloor = \lfloor \lg(k_2+k_2) \rfloor \leq$$

Bestetik $k_2 \leq k_1$ gertatzen dela suposa dezakegu (frogaketan ez dugu orokortasunik galtzen). Honen bidez, nahiz eta biek sakonera berdina eduki, z_1-ek z_2-k baino adabegi gehiago dituela (edo berdin) suposatzen dugu; hau da, z_1 zuhaitzaren zabalera handiagoa da.

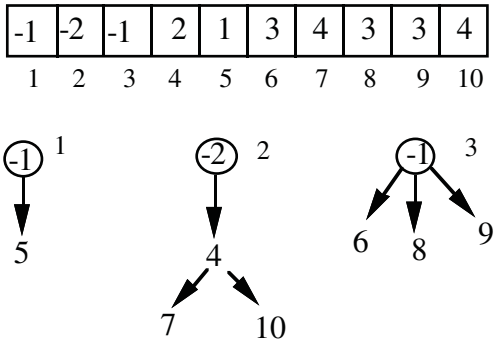
$s_{k_1+k_2} = \ldots \leq \lfloor \lg 2k_2 \rfloor = \lfloor \lg(k_2+k_2) \rfloor \leq \lfloor \lg(k_1+k_2) \rfloor$, beraz, hau ere betetzen da.

Baina, alderantziz ere zintzilika ditzakegu, eta kasu horretan egingo genukeen suposaketa $k_1 \leq k_2$ izango litzateke.

$$s_{k_1+k_2} = s_{k_1} + 1 \leq \lfloor \lg k_1 \rfloor + \lfloor \lg 2 \rfloor = \lfloor \lg 2 + \lg k_1 \rfloor \leq \lfloor \lg k_1 \rfloor \leq \lfloor \lg(k_1+k_2) \rfloor$$

Beraz, zuhaitzaren sakoneren funtziopean bateraketak egingo baditugu, zuhaitzen sakonerak nonbait biltegiratuta edukitzea komeniko litzaiguke. Hauek,

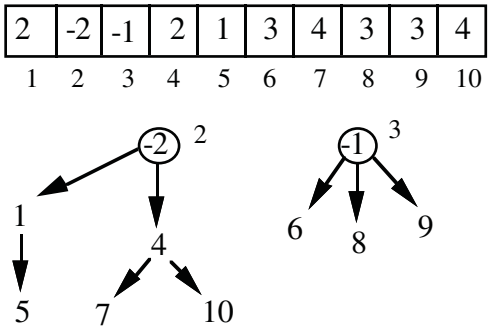
sinuz aldatuta zuhaitz bakoitzaren erroan biltegiratuko bagenitu, bateraketaren prozesua erraztuko genuke. Orain, azpimultzo bakoitzeko etiketa zein izango litzateke? Bertako partaideen artetik txikiena? Ez. Multzo bakoitzari dagokion zuhaitzaren erroa da etiketa. Hau da, PARTIKETAn bere posizioan zuhaitzaren sakonera biltegiratuta duen osagaia.



Eragiketak honela inplementa ditzakegu:

```
function BILATU3 (PARTIKETA: in P_PARTIKETA_MOTA; X: in OSAG_ETIK)
                    return OSAG_ETIK is
    I: OSAG_ETIK:= X;
    — x partaidea den multzoko etiketa itzultzen du.
begin
    while  PARTIKETA(I) >0  loop
            I:= PARTIKETA(I)
    end loop;
    return (I)
end BILATU3;
procedure BATERATU3 (PARTIKETA: in out P_PARTIKETA_MOTA;
                        E1,E2: in OSAG_ETIK) is
    — E1 eta E2 etiketak dituzten multzoak bat egiten ditu.
begin
    if  PARTIKETA(E1)=PARTIKETA(E2)  then
              PARTIKETA(E1):= PARTIKETA(E1)-1;
              PARTIKETA(E2):= E1;
    elsif  PARTIKETA(E1)<PARTIKETA(E2)  then
              PARTIKETA(E2):= E1;
    else   PARTIKETA(E1):= E2;
    end if;
end BATERATU3;
```

Aurreko adibidea izanik $BATERATU_3(1,2)$ egikarituko bagenu, lortuko genukeen partiketa berria:



Hau da: [2]={2,1,4,5,7,10} eta [3]={3,6,8,9} multzoak eta etiketak.

Kasu honetan ere *n* eragiketen sekuentzia batek hartzen duen denboraren ordenaren analisia egin dezakegu hasierako egoera batetik hasten garela suposatuz. Oraingoan berriz kasu txarrenean lortzen dugun emaitza: $\Theta(n \lg n)$ da.

Frogaketa (kasu txarrena):

– (n/2) BATERATU$_3$ eragiketa exekutatu ondoren, aurreko lema aplikatuz, lortzen den zuhaitzaren sakonera $\lfloor \lg((n/2)+1) \rfloor$ izango da ((n/2)+1 adabegi izango baititu). (n/2) elkartu eragiketa hauek egiteko behar den denbora-ordena $\Theta(n/2)$-koa da.

– Ondoren beste (n/2) BILATU$_3$ eragiketa egiten baditugu, azkeneko mailan dagoen balio bat bilatu nahirik, (n/2) bilatu. Eragiketen sekuentziak behar duen denbora-ordena: $\Theta((n/2) \lfloor \lg((n/2)+1) \rfloor = \Theta(n \lg n)$.

– eta beraz *n* eragiketen sekuentziak behar duen denbora-ordena baturaren erregela erabiliz $\Theta(n \lg n)$-koa da.

Beste ordena hobea aurkitu nahi izanez gero, beste zenbait hobekuntza bilatu eta aurkitu beharko genuke (ikusi 'Algorítmica' Brassard eta Bratley-n liburua, edo eta 'Computer Algorithms' Sara Baase-ren liburua).

4.2.2.2. Kruskal-en algoritmoa

Azalduko dugun Kruskalen algoritmoak grafo konexuekin lan egingo du. Grafo ez-konexuekin funtziona dezan, algoritmoan zenbait aldaketa egin beharko genuke.

Prozesua:

Hasieran, ertzen ERT multzoa hutsik dago. Banan-bana ERT multzoari ertzak eransten zaizkio. Edozein unetan, G-ko erpinak eta ERT-eko ertzak erabiliz

osaturiko ‘grafo partzialak’ zenbait osagai konexu izango du (hasieran, G-ko erpin bakoitza osagai konexu bat da). Osagai konexu bakoitzeko ertzek osagai horretako erpinentzat hedapen zuhaitz minimo bat osatzen dute. Azpizuhaitz hauek grafoaren hedapen baso bat osatzen dutela esango dugu.

ERT-i ertzak eransten dizkiogun neurrian, osagai konexuen tamainak handitzen doaz eta osagai konexuen kopurua txikitzen. Algoritmoaren exekuzioa amaitzen denean, ERT-en G-ko erpin guztiak erlazionatzen dituen osagai konexu bakar bat dago. G-ko hedapen zuhaitz minimoa izango da.

Gero eta handiagoak diren osagai konexuak osatzeko, G-ko ertzak goranzko ordena jarraituz ordenaturik daudela kontsideratuko dugu. Behin eta berriro aztertu gabe gelditzen diren ertzen artetik, pisu txikiena duen ertza aukeratzen da: ***e*** . **e** ertzak:

– ERT-eko bi osagai konexu ukitzen baditu, ***e*** ERT multzoari eransten zaio eta bi osagai konexuak elkartzen dira osagai konexu bat lortuz.

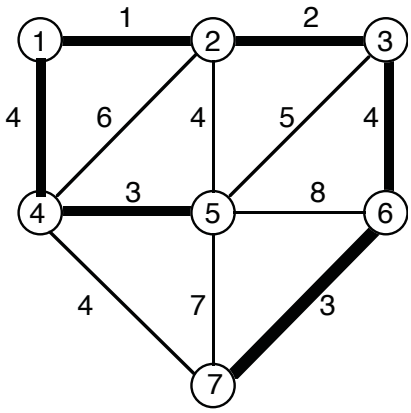
– Bestela, ez da ertza onartzen, osagai konexu berdin bateko bi erpin elkartuko bailituzke ziklo bat sortuz.

G-ko erpin guztiak dituen osagai konexu bakar bat gelditzen denean, algoritmoaren egikaritzapena amaitzen da.

Algoritmoaren lehenengo hurbilketa:

```
algoritmoa KRUSKAL (G=<Erpinak,Ertzak,Pisuak>;ERT: Ertz-multzoa)
    ...
hasi
    – hasieraketak
    L:= Pisuen_goranzko_ordena_jarraituz_sailkatu (Ertzak);
    P := Erpin_kopurua (Erpinak);
    Multzo_hutsa_ert (ERT);        — hedapen zuhaitzaren ertzak biltegiratzen ditu
    — begizta jalea
    errepika  (Ertz_kopurua(ERT) ≠ P-1) hasi
          Pisu_txikieneko_ertza_aukeratu(L,(x,y)); — lehenengo ertza lortu
          Aukeratutako_ertza_kendu(L,(x,y));       — lehenengo ertza kendu
          baldin not(Ziklorik_eransten_du?(Erantsi_ert(ERT,(x,y)))
          orduan Erantsi_ert (ERT, (x,y));
          bukatu baldin;
    bukatu errepika;
bukatu KRUSKAL;
```

Adibidea:



1 grafoa:

Grafo horretako ertzak pisuen balioen goranzko ordena jarraituz sailkatu ondoren lortzen den sekuentzia hau da: (1,2), (2,3), (4,5), (6,7), (1,4), (2,5), (3,6), (4,7), (3,5), (2,4), (5,7) eta (5,6). Sekuentzia honentzat algoritmoak exekutatzen dituen pausoak berriz:

Pausoa	*Kontsideraturiko ertza*	*Osagai konexuak*
hasiera		{1} {2} {3} {4} {5} {6} {7}
1	(1,2)	{1, 2} {3} {4} {5} {6} {7}
2	(2,3)	{1, 2, 3} {4} {5} {6} {7}
3	(4,5)	{1, 2, 3} {4, 5} {6} {7}
4	(6,7)	{1, 2, 3} {4, 5} {6, 7}
5	(1,4)	{1, 2, 3, 4, 5} {6, 7}
6	(2,5)	ez da ertza onartzen
7	(3,6)	{1,2,3,4,5,6,7}

ERT-ek onartu diren ertzak metatzen ditu; hau da, (1,2), (2,3), (4,5), (6,7), (1,4) eta (4,7). Irudian marra zabalez grafoaren hedapen zuhaitza adierazi da. Guztiaren pisua 17 da.

Algoritmoaren zuzentasunaren azterketa:

Edozein grafo konexu eta pisudun auzokide listen bidezko adierazpenarekin adierazirik dagoenean Kruskalen algoritmoak haren h.z.m. kalkulatzen duela frogatuko dugu. Gai honen hasieran emandako azalpenak kontuan izanik, hemen hautaketa segurua eta induktiboa dela frogatuko dugu soilik. Ondoren, algoritmoak eraikitzen duen S soluzio multzoaren kardinalitatean indukzioa eginez S multzoa

osogarria dela ez da azalduko, frogaketa orokorra (alegia, edozein algoritmo jalerentzat) gaiaren hasieran ere eman da eta. Beraz:

(1) Hautaketa segurua da?

Hautagaiak, H, grafoko ertzak dira. Eta Kruskalek onartzen duen lehenengo hautagaia, HJ, bi osagai konexu desberdin konektatzen dituen e=(x,y) pisu txikieneko ertza da. Bestalde, demagun soluzio optimo posible bat OPT dela. Orduan, bi gauza gerta daitezke:

1- $e \in OPT \Rightarrow$ Hautaketa jalea segurua da

2- $e \notin OPT$. Kasu honetan OPT S-n eralda daitekeela frogatuko dugu, non S-ko ertzek ere h.z.m bat osatzen duten eta haien pisuen batura minimoa den. Eraldaketa prozesua honela egin dezakegu:

– OPT-ri e ertza eransterakoan ziklo bakar bat sortuko da, bertako ertzek h.z.m. bat osatzen baitzuten. Demagun z sortu den zikloko pisu handiena duen ertza dela. z ertza kenduz gero, zikloa hautsiko genuke, berriz ere h.z. bat lortuz, erpin guztiak konektaturik geldituko bailirateke. Multzo berriko ertzak, beraz,

$$S= (OPT-\{z\})\cup\{e\}$$

– Baina, S soluzio optimo bat al da? Hots, bertako ertzen pisuen batura minimoa da? Bai. e gelditzen ziren hautagaietatik pisu txikiena zuena zen eta z aldiz zikloan pisu handiena zuena. Ondorioz,

$$\sum_{er\in S} \text{pisua}(er) \leq \sum_{er\in OPT} \text{pisua}(er)$$

Baina OPT optimoa denez, derrigorrez bi batukarien batura berdina da. (Derrigorrez, pisua(e)=pisua(z) beteko da.)

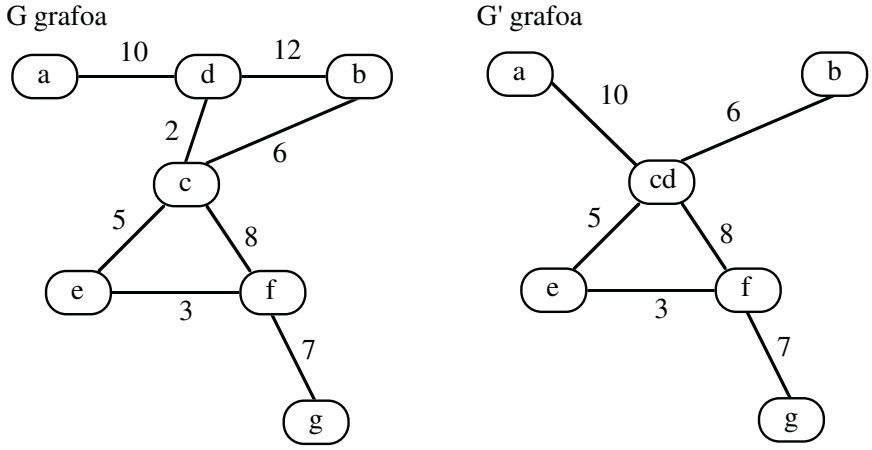
Beraz, $e \in S$ eta S h.z.m. bat da.

(2) Hautaketa induktiboa da?

G Jatorrizko problemaren pareko azpiproblema bat, G', bilatu behar da, non honen soluzio optimoa, S', Kruskalek onartutako lehenengo hautagaiarekin bateragarria den. Azpiproblema berria zein den erraz definitu ahal izateko grafoko ertzak eta hauen pisuak lista batean dauzkagula suposatuko dugu (honek frogaketa sinplifikatzeko balioko du): G=<Erpinak, Ertzak+Pisuak_Listan>

Pareko azpiproblema:

Ideia: Kruskalek onartutako lehengo ertza (x,y) izan bada, (x,y) ertzak elkartzen dituen erpinek bat egiten dute, xy erpin berria emanaz (kontzentratu egiten dira erpin bakar batean), eta erpin hauetara zihoazen ertzak berrizendatu egin beharko ditugu, gainera, erpinen batetik bai x-era eta bai y-ra ertzak ailegatuko balira, pisu txikienekoa bakarrik gorde beharko litzateke grafo berrian.



Irudietan (x,y) ertza 2 pisua duen (c,d) ertza da.

G'=<Erpinak-{x,y})∪{xy}, E+P_Lista_Berrian>

E+P_Lista_Berrian= Ertzak+Pisuak_Listan

$$- \{((x,h),p) \mid \forall h(h \in \text{Erpinak} \wedge ((x,h),p) \in \text{Ertzak+Pisuak\_Listan})\}$$

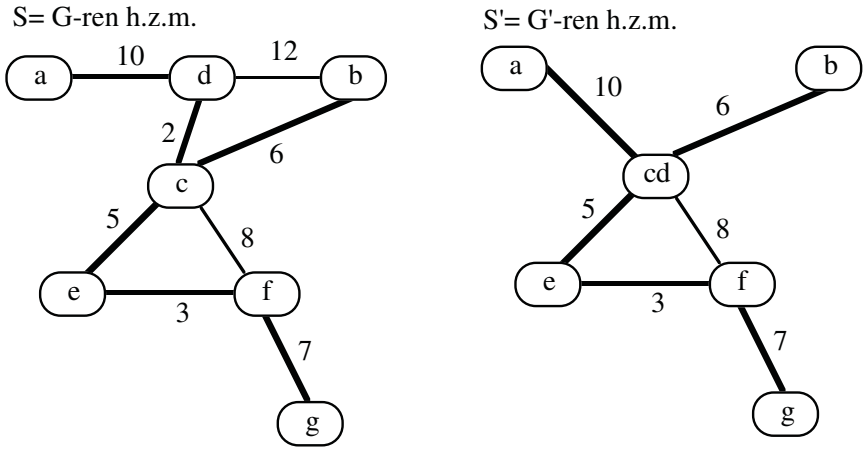
$$- \{((h,y),p) \mid \forall h(h \in \text{Erpinak} \wedge ((h,y),p) \in \text{Ertzak+Pisuak\_Listan})\}$$

$$\cup \{(xy,h),p) \mid \forall h(h \in \text{Erpinak} \wedge ((x,h),p) \in \text{Ertzak+Pisuak\_Listan} \wedge \neg \exists j((h,y),j) \in \text{Ertzak+Pisuak\_Listan})\}$$

$$\cup \{(xy,h),p) \mid \forall h(h \in \text{Erpinak} \wedge ((h,y),p) \in \text{Ertzak+Pisuak\_Listan} \wedge \neg \exists j((x,h),j) \in \text{Ertzak+Pisuak\_Listan})\}$$

$$\cup \{(xy,h),p) \mid \forall h\ (h \in \text{Erpinak} \wedge ((x,h),p_1) \in \text{Ertzak+Pisuak\_Listan} \wedge ((h,y),p_2) \in \text{Ertzak+Pisuak\_Listan} \wedge p=\min(p_1,p_2))\}$$

Orain, G' Gren 'azpigrafo berezi' bat da. Eta G' grafoa emanik problema bera ebatzi behar dugu orain. G'ren h.z.m. S' lortuko bagenu, hots, S' G'ren h.z.m. bat, S'ko xy erpina x eta y erpinetan desplegatuz G-ko h.z.m. lortuko genuke, S' eta e=(x,y) bateragarriak baitira.



Desplegatze prozesua egiteko G grafoan S-ko ertzak eta e ertza nola dauden ikusiz egin behar da. Halere, atzerako pausoa beti egin daiteke. Beraz, Hautaketa induktiboa da.

Algoritmoaren inplementazioa:

– Osagai konexu bakoitzeko erpinak metatzeko aldi berean zenbait multzo maneiatu behar dugu. Multzo bakoitzak tarteko soluzioan konektatuta dauden erpinak biltegiratuko ditu. Gainera, erpin bakoitza multzo bakar batean egongo dela badakigu.

– Bestalde, erdibideko soluzioari ertz bat eransteko honek ziklorik sortzen duen edo ez jakin behar dugu. Eragiketa hori inplementatu nahi badugu, ertz horren erpin bakoitza zein multzotan dagoen *bilatu* beharko dugu eta ondoren hauek berdinak diren ala ez galdetu. Ondoren, eta multzo desberdinetan badaude soilik, (= ez du ziklorik sortuko), ertz hori soluzioari eransten zaio. Beraz, bi erpin horiek dituzten multzoak *elkartu* beharko ditugu.

– Egitura eta eragiketa hauek partiketa egitura erabiliz eraginkorki inplementa ditzakegu:

- BILATU (Os_Konexuak, X):
  P partiketan X erpina zein osagai konexutan dagoen kalkulatuko du, haren etiketa itzuliz.

- BATERATU (Os_Konexuak, ET_1, ET_2):
  P partiketako ET_1 eta ET_2 etiketak dituzten multzoak bateratzen ditu.

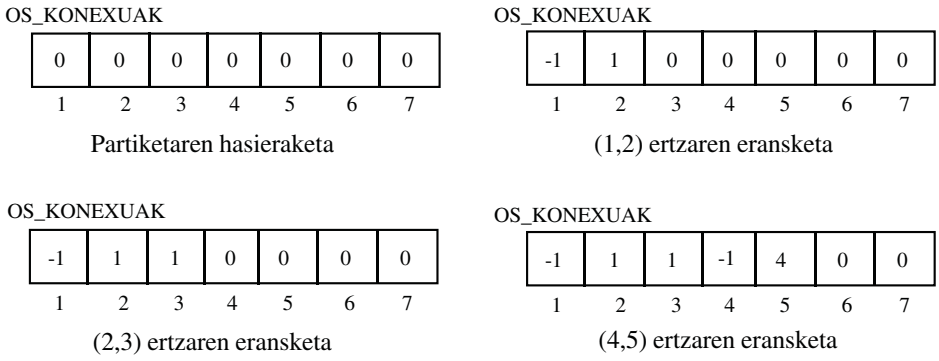
– BILATU (Os_Konexuak, X) eta BATERATU (Os_Konexuak, ET_1, ET_2) eragiketak eraginkorrak izatea beharrezkoa da. Helburu hori lortzeko partiketa egituraren hirugarren inplementazioa erabiliko dugu.

– Grafoa (ertza, pisua) osagaiez osaturiko taula bidez adieraztea gomendagarria da, pisuen matrizearen adierazpena erabili gabe.

```
procedure KRUSKAL (G: in GRAFOA; ERT: out Ertz_multzoa)  is
    ERT_LAG: Ertz_multzoa;
    OS_KONEXUAK: P_PARTIKETA_MOTA;
    P: INTEGER;
    X_osagaia_barne,Y_osagaia_barne:ETIKETA
begin
    — hasieraketa
    L:= Pisuen_goranzko_ordena_jarraituz_ordenatu (ERTZAK(G));
    P := ERPIN_KOP (G);
    Multzo_hutsa_ert (ERT_LAG);   — hedapen zuhaitzaren ertzak biltegiratzen ditu

p_multzo_hasieratu_bakoitza_erpin_ezberdin_batekin(OS_KONEXUAK);
    — begizta jalea
    while  (ERTZ_KOP (ERT_LAG) ≠ p-1)  loop
          kontsideratu_ez_den_pisu_txikieneko_ertza (L, (x,y));
                                   — lehenengoa lortu eta kendu
          X_osagaia_barne:=BILATU3(OS_KONEXUAK, X);
          Y_osagaia_barne:=BILATU3(OS_KONEXUAK, Y);
          if Desberdinak(OS_KONEXUAK,X_osagaia_barne, Y_osagaia_barne)
          then
              BATERATU3(OS_KONEXUAK, X_osagaia_barne, Y_osagaia_barne);
              Erantsi_ert(ERT_LAG, (x,y));
          end if;
    end loop;
    ERT:= ERT_LAG;
end KRUSKAL;
```

Aurreko adibidea osatzen:



OS_KONEXUAK

-1	1	1	-1	4	-1	6
1	2	3	4	5	6	7

(6,7) ertzaren eransketa

OS_KONEXUAK

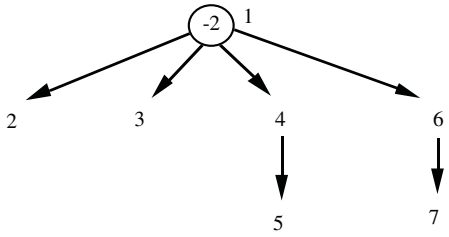
-2	1	1	1	4	-1	6
1	2	3	4	5	6	7

(1,4) ertzaren eransketa

OS_KONEXUAK

-2	1	1	1	4	1	6
1	2	3	4	5	6	7

(3,6) ertzaren eransketa



Algoritmoaren ordenaren azterketa:

Grafoak *p* erpin eta *a* ertz dituela suposatuz, algoritmoaren exekuzio--denboraren hurbilketa bat honela kalkula dezakegu:

Eragiketen konplexutasunaren ordena ondorengo hau da:

1.- Ertzak sailkatzeko $\Theta(a \lg a)$. Eta *p* eta *a* erlazionatzen dituen inekuazio hau $p-1 \leq a \leq (p(p-1))/2$ dela eta, ordena hau gauza bera da: $\Theta(a \lg p)$.

2.- Inplementazioaren arabera, erpin kopurua mugatzeko behar den denboraren ordena $\Theta(1)$ edo $\Theta(p)$ izango da.

3.- Ertzak metatzeko multzo huts bat sortzeak hartzen duen denboraren ordena $\Theta(1)$ da.

4.- p multzoak hasieratzeko: $\Theta(p)$.

5.- Kasu txarrenean: partiketaren analisia bergogoratuz, hasierako egoeratik abiatuz eta *n* luzera duen $BILATU_3$ eta $BATERATU_3$ eragiketez osaturiko sekuentzia batek hartzen duen denboraren ordena $\Theta(n \lg n)$ da.

Kruskalen algoritmoak ertzak banan bana aukeratzen ditu. Aukeratutako ertz batek baldintza batzuk betetzen baditu, orduan ertza erdibideko soluzioari erantsiko dio. Prozesu bera soluzio hoberena lortu arte errepikatzen du. Beraz, kasu txarrenean ertz guztiak aukeratu eta aztertu beharko ditu. Kasu horretan, a ertz eta p erpin baditugu 2a bilatu eragiketa eta (p-1) elkartu eragiketa exekutatuko

dira. Guztira, 2a+(p-1) $BILATU_3$ eta $BATERATU_3$ eragiketa. Sekuentzia honen egikaritzapenak hartuko duen denboraren ordena $\Theta((2a+(p-1))\ lg\ (2a+(p-1))) = \Theta(a\ lg\ a)$ da. Gainera, grafoa konexua denez, $(p-1) \leq a \leq p(p-1)/2$ betetzen da eta $\Theta(a\ lg\ a)=\Theta(a\ lg\ p)$ berdinketa beteko da.

6.- 1..5 => baturaren erregela aplikatuz => $\Theta(a\ lg\ p)$, kasu txarrenean.

Kasu txarrenean desberdintasunik ez badago ere, gomendagarria litzateke ertzak tontor egitura batean biltegiratuta edukitzea. Honela, hasieraketak hartuko lukeen denboraren ordena O(a) izatea posible izango litzateke. Baina orduan, *while* begiztaren barneko "kontsideratu_ez_den_pisu_txikieneko_ertza(L, (x,y))" eragiketa exekutatzeko behar den denboraren ordena O(lg a) izango litzateke (O(lg a) =O(lg p)). Honek, hedapen zuhaitz minimoa osatzeko oraindik kontsideratu gabeko ertzen kopurua handia denean abantaila asko dakartza; kasu horretan, emaniko algoritmoak sailkapena egiten denbora galduko luke eta.

Kruskalen algoritmoak pauso bakoitzean soluzio onenera gidatzen duen ertz bat aukeratzen du. Baina ertz honek dagoeneko hautatuta dauden ertzekin dituen erlazioak algoritmoak ez ditu kontutan hartzen eta ezartzen duen baldintza bakarra ertz berriak erdibideko soluzioan ziklorik ez osatzea da. Honela, hedapen zuhaitz minimoak (osagai konexuak) era anarkiko batean eraikitzen dira hedapen zuhaitz minimo bakar bat osatu arte.

4.2.3. Prim-en algoritmoa

Prim-en algoritmoak, erpin bat, edozein, abiapuntutzat harturik grafoaren hedapen zuhaitz minimoa eraikiko du. Horretarako osagai konexu bakarra erabiliko du. Honen hazkuntza, hedapena, naturalki egingo du. Osagai konexua grafoaren hedapen zuhaitz minimoaren azpizuhaitz bat izango da momentu oro. Osagai konexuaren hedapena osagai konexu hori grafoaren hedapen zuhaitz minimoa izatera iristen denean amaitzen da.

Prozesua:

Hasieran, ERP erpinen multzoak erpin bat dauka (edozein erpin, ez du axola) eta ERT ertzen multzoa hutsa da. Pauso bakoitzean algoritmoak ertz bat (x,y) bilatzen du non bere pisua minimoa den eta $x \in$ Erpinak-ERP eta $y \in$ ERP diren eta ERT multzoari gehitzen dio. Honela ERT-eko ertzek ERP-eko erpinentzat hedapen zuhaitz minimoa osatzen dute momentu oro. Prozesu bera DESBERDINAK(ERP ,Erpinak) betetzen den bitartean errepikatzen da.

Adibidea:

Gorago azaldutako 1 grafoa kontutan edukiz:

Pausoa	*Kontsideraturiko ertza*	*ERP*
hasieraketa		{1 }
1	(2, 1)	{1, 2 }
2	(3, 2)	{1, 2, 3 }
3	(4, 1)	{1, 2, 3, 4 }
4	(5, 4)	{1, 2, 3, 4, 5 }
5	(7, 4)	{1, 2, 3, 4, 5, 6}
6	(6, 7)	{1, 2, 3, 4, 5, 6, 7 }

Beraz, algoritmoak itzultzen duen emaitza: ERT = { (2,1), (3,2), (4,1), (5,4), (7,4), (6,7) }

Algoritmoaren inplementazioa

Algoritmo honen inplementazio sinple bat lortu nahi badugu, datuentzat zein adierazpen interesatuko litzaiguke? Demagun G grafoaren erpinak P direla eta gainera 1etik p-ra zenbakitu ditugula: Erpinak = {1,2, ..., P}. G matrize simetriko baten bidez ertz bakoitzaren pisua (balio ez negatiboa) adieraziko dugu. (i,j) ertzen bat egongo ez balitz, hura G(i,j) = SYSTEM.MAX_INT bidez adieraziko genuke. Matrize honi pisuen matrizea deritzo.

Algoritmoak bi taula erabiliko ditu: AUZOKIDE eta PISU_MIN taulak. Grafoak izango dituen erpin kopuru adina osagai izango dituzte (edo kopuru hori baino bat gutxiago edukitzearekin ere ondo leudeke).

– AUZOKIDE(i): i ∈ Erpinak-ERP erpin bakoitzarentzat, AUZOKIDE(i) espresioak *i* erpinetik ERP-ekoa den eta hurbilen duen erpina biltegiratzen du; hau da, (i,AUZOKIDE(i)) ertza momentu horretako ERT multzoa (eta beraz, bertako erpinak, ERP) eta *i* erpina modu merkeenean konektatzen dituen ertza da, eta

– PISU_MIN(i): (i, AUZOKIDE(i)) ertzaren pisua da. Bestalde, *i* erpina ERP-ekoa bada orduan bere PISU_MIN(i) = -1 da.

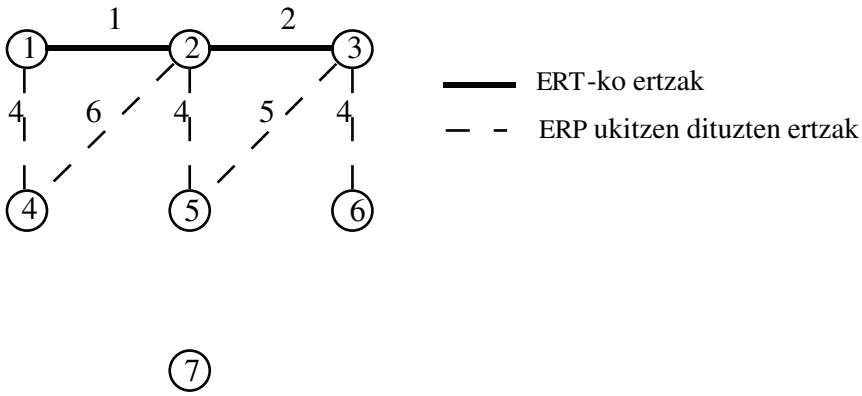
ERP multzoa {1}-ekin hasieratzen bada ere, balio hau ez da esplizituki mantentzen, AUZOKIDE(1) eta PISU_MIN(1) ez baitira erabiltzen.

Algoritmoa ondoko hau izango da:

```
type MATRIZEA is array (POSITIVE range <>, POSITIVE range <>)
                       of INTEGER;
procedure  PRIM  (G: in MATRIZEA;  ERT: out Ertz-multzoa) is
    AUZOKIDE: TAULA_ERPINAK(G´FIRST(1)..G'LAST(1));
    PISU_MIN: TAULA_PISUAK(G´FIRST(1)..G'LAST(1));
```

```
    K:ERPINA; MIN: PISUA; ERT_LAG:Ertz-multzoa;
begin
    — hasieraketa, ERPen 1. erpina dago soilik
    Multzo_hutsa_ert(ERT_LAG);
    for  I   in  G´FIRST(1) +1..G'LAST(1)  loop
          AUZOKIDE(I) := 1;
          PISU_MIN := G(I,1);
    end loop;
    for IND in  G´LENGTH(1)-1  loop
                                          — Grafoko h.z.m'ak behar dituen
                                          — ertz kopuru adina bira kopuru.
          MIN := SYSTEM.MAX_INT;          — Edozein ertzek izan dezakeen
                                          — baino pisu handiagoa.
          for J in G´FIRST(1)+1..G'LAST(1) loop
                                          — Lehenengo posizioa ez da erabiltzen
             if 0≤PISU_MIN(J)<MIN then    — IND bira batean
                MIN:=PISU_MIN(J);         — erantsi daitekeen ertz
                K:= J;                    — merkeenaren pisua.
             end if;
          end loop;
          Erantsi_ert(ERT_LAG,(K,AUZOKIDE(K));— Soluzioari erantsi.
          PISU_MIN(K):= -1;               — K ERP multzoari eratsi
          for J in G´FIRST(1)+1..G'LAST(1)loop
                                          — ERTri ertz berria erantsi
            if  G(K,J)<PISU_MIN(J) then   — ondoren pisu minimoen
                PISU_MIN(J):= G(K,J);     — taula eguneratu behar da,
                AUZOKIDE(J):= K;          — eta ondorioz auzokideen
            end if;                       — taula ere bai.
          end loop;
    end loop;
    ERT:= ERT_LAG;                        — Ada lengoaiagatik
end PRIM;
```

Algoritmoaren erdibideko pausu baten exekuzioa (aurreko adibidea):



Beraz, 4 pisua duen (1,4) ertza soluzioari erantsiko zaio.

AUZOKIDE

	1	2	1	2	3	1
1	2	3	4	5	6	7

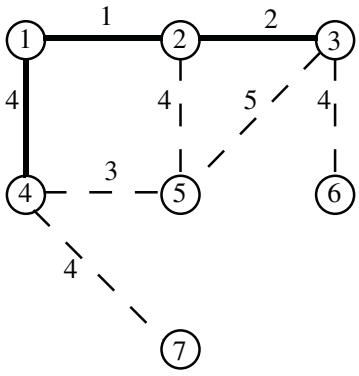
ERP ukitzen duten eta pisu txikieneko grafoko ertzak

PISU_MIN

	-1	-1	4	4	4	∞
1	2	3	4	5	6	7

Ertz horien pisuak

Algoritmoaren zuzentasunaren azterketa:



AUZOKIDE

	1	2	1	4	3	4
1	2	3	4	5	6	7

PISU_MIN

	-1	-1	-1	3	4	4
1	2	3	4	5	6	7

Kruskal algoritmoaren pareko frogaketa egin beharko litzateke hemen. Irakurleak frogaketa hura kasu berri honetarako egoki lezake.

Algoritmoaren ordenaren azterketa:

Algoritmoaren gorputza jarraian agertzen diren bi bigiztez osaturik dago:

1.- Lehenengo begiztak *p-1* aldiz bi asignazio egikaritzen ditu: $\Theta((p-1)\ 2) = \Theta(p)$,

2.- Bigarren begiztak ere *p-1* aldiz errepikatzen ditu bere barneko eragiketak:
– asignazioek behar duten denboraren ordena: $\Theta(1)$
– ondoren dauden "for" bakoitzak behar duen denboraren ordena: $\Theta(p)$-koa da.
– => baturaren erregela erabiliz => bigarren "for" aginduaren gorputzak behar duen denboraren ordena: $\Theta(p)$ da.

=> bigarren begiztaren ordena: $\Theta((p-1)\ p) = \Theta(p^2)$

3.- Berriz ere, baturaren erregela erabiliz, Prim algoritmoaren ordena: $\Theta(p^2)$ da.

Ondorioak: Prim\Kruskal

Kruskalen algoritmoak hartzen duen denboraren ordena kasu txarrenean $\Theta(a \lg a)$ da. Grafoak ertz asko baditu, a-ren balioa $(p\ (p-1)/2)$ baliotik gertu egongo da, kasu horretan kruskalen algoritmoaren ordena $\Theta(p^2 \lg p)$ izango da ($\Theta(\lg p)=\Theta(\lg a)$) eta Prim-en algoritmoa hobea litzateke. Grafoak ertz gutxi baditu, a-ren balioa n-tik gertu egongo litzateke, kasu horretan, Kruskalen algoritmoaren ordena $\Theta(p \lg p)$ izango litzateke, eta Prim-en algoritmoa ez litzateke hain eraginkorra izango.

4.2.4. Dijkstra-ren algoritmoa

Oraingo honetan grafo zuzenduekin lan egingo dugu. Har dezagun G grafo zuzendua eta pisuduna: G = <Erpinak, Arkuak, Pisuak>. Erpin bat abiapuntutzat hartuko dugu. Gure helburua, abiapuntu erpinetik grafoko beste erpinetara doazen bide motzenak mugatzea izango da.

Arazo hau Dijkstraren algoritmo jaleak ebazten du.

Prozesua:

Izan bitez H eta E hautagai erpinen eta erpin hautatuen multzoak hurrenez hurren. Hasieran, E multzoan abiapuntu-erpina dago soilik. Pauso bakoitzean X erpin bat aukeratzen da hautagaien artetik, H-tik, non abiapuntutik X-ra dagoen distantzia minimoa den. Orduan, X erpina E multzoari eransten zaio. Honela, abiapuntutik beste erpin batera doan bide motzena ezaguna bada, erpin hau E multzoan egongo da, eta bestela H-n. Azkenean, erpin guztiak E-n egongo dira.

Distantzia minimoak kalkulatzeko, algoritmoak *bide berezi* izeneko bideen distantziak kalkulatuko ditu. Honela, *bide* bat *berezia* dela esango dugu, baldin eta abiapuntu-erpinetik beste erpin batera doan bidea E-ko erpinez osatuta badago soilik.

Grafoko erpin bakoitzari D taula bateko sarrera bat egokituko zaio eta taulako posizio bakoitzean abiapuntu-erpinetik *D*-ko osagai horrek adierazten duen erpineraino doazen *bide berezi guztietatik motzenaren balioa*, distantzia, biltegiratuko da.

Erpin aukeratuen E multzoari beste erpin bat, X, eransten zaionean, X-erako bide berezi motzenaren luzera eta X-erako bide motzenaren luzera berdinak dira.

Algoritmoaren exekuzioa amaitzen denean, E-n erpin guztiak daude eta abiapuntutik beste erpinetarainoko bideak bereziak dira. Honela, algoritmoaren emaitza D-n dago.

Kalkuluak erraztearren hau suposatuko dugu:

– Grafoko erpinen multzoa zenbakituta dagoela suposatuko dugu: {1,2, .. , P},

– abiapuntu-erpina 1 dela,

– G pisuen matrizeak arku bakoitzaren pisua azalduko duela:

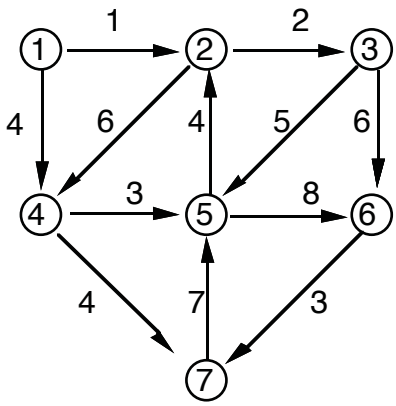
G(i,j) ≥ 0 ->
(i,j) arkuak grafoan existitzen du, eta G(i,j) bere pisua da

G(i,j) = ∞ ->
(i,j) arkua ez dago grafoan.

Adibide bat ikus dezagun.

Adibidea:

Demagun ondorengo grafoa:



3 grafoa

Pausoa	*X*	*H*	*D taula*
hasieraketa		{2,3,4,5,6,7}	[1, ∞ , 4 , ∞ , ∞ , ∞]
1	2	{3,4,5,6,7}	[1, 3 , 4 , ∞ , ∞ , ∞]
2	3	{4,5,6,7}	[1, 3 , 4 , 8 , 9 , ∞]
3	4	{5,6,7}	[1, 3 , 4 , 7 , 9 , 8]
4	5	{6,7}	[1, 3 , 4 , 7 , 9 , 8]
5	7	{6}	[1, 3 , 4 , 7 , 9 , 8]

D-k abiapuntutik erpin bakoitzerainoko bere distantzia motzenaren balioa biltegiratzen du.

Algoritmoaren inplementazioa:

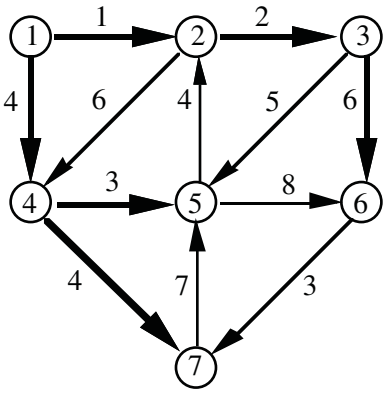
```
type GRAFOA is array (POSITIVE range <>, POSITIVE range <>)
                        of INTEGER;
type  TAULA is array  (POSITIVE range <>) of INTEGER;
procedure  DIJKSTRA ( G: in GRAFOA;  D: out TAULA) is
    H,HK: LISTA;
    P: NATURAL; Y:ERPINA;
    D_LAG: TAULA;
begin
    P:= G'LENGTH(1)       — Erpin kopurua
    HASIERATU2_P(G,H);  —2,3,4,...,P erpinekin hasieratzen da
                        —E inpliziturik egongo da
                        —E = Erpinak-H
    for I in  G'FIRST(1)+1..G.'LAST(1) loop  — Oro har: 2..P
          D_LAG(I):= G(1,I);
    end loop;
    — Begizta jalea:
    for  I in ERPIN_KOP(G)-2 loop                          — Oro har: P-2
          GERTUEN_DAGOEN_ERPINA(H,D_LAG,X);
                — D_LAG(X) balio txikiena duen  X Œ H erpina; hau da,
                — abiapuntutik eta E-ko erpinak bakarrik zeharkatuz X-rainoko
                — bide berezi motzenaren balioa.
          HAUTAGAIETATIK_KENDU(H,X);
          HK:= H;
          — H-ko erpinak banan bana aztertu ahal izateko;
          for J in 1..ERPIN_KOPURUA_L(H) loop
          Y:= LEHENENGOA_L(HK);
          HONDARRA_L(HK);— lehenengoa eta erakuslea aurreratu
          if D_LAG(Y) > D_LAG(X)+G(X,Y)  then
                D_LAG(Y):= D_LAG(X)+G(X,Y);
          end if;
          end loop;
    end loop;
    D:= D_LAG;
end DIJKSTRA ;
```

Algoritmo honek D taula itzultzen du eta bertan abiapuntutik erpin bakoitzeraino dauden distantzia motzenak daude. Baina, luzera ez ezik bide motzenak zein erpinetatik igarotzen diren jakitea interesa dakiguke. Horretarako, beste taula bat beharko genuke: izan bedi BIDEAK taula. Honela, BIDEAK(X)-k bide laburrenean X-en aurretik zeharkatu beharko litzatekeen erpina metatuko luke; hots, bide motzenean (BIDEAK(X),X) arkua azaldu beharko litzateke. Bidea eraikitzeko BIDEAK taula aztertuz abiapunturaino bidea egin beharko litzateke.

Algoritmoan zenbait aldaketa egin beharko genuke:

```
⇒ for I in G'FIRST(1)+1..G'LAST(1)loop BIDEAK(I):=G(1,I); end loop;
⇒ 2. begiztaren barneko gorputzaren ordez ondorengo hau ezarri:
          if D_LAG(Y)>D_LAG(X)+G(X,Y) then
                D_LAG(Y):= D_LAG(X)+G(X,Y);
                BIDEAK(Y):= X;
          end if;
```

Bide motzenen distantziak lortzeaz gain, bideak ere lortuko genituzke:



<u>*Algoritmoaren zuzentasuna*</u>:

Hautaketa jalea segurua eta induktiboa dela erraz froga daiteke Kruskalen frogaketaren eredua jarraituz. Irakurlea saia bedi.

<u>*Denbora-ordenaren analisia kasu txarrenean*</u>:

Demagun G-k a arku eta p erpin dituela, eta grafo honen distantziak edo pisuak G: GRAFOA(1..P,1..P) matrizearen bidez ematen direla.

1.- Erpinen kopurua mugatu: $\Theta(1)$ (edo $\Theta(p)$, programazio lengoaiaren funtziopean)

2.- H hautagaien multzoa hasieratu: $\Theta(p)$

3.- 1 erpinetik beste erpin bakoitzera dauden distantziekin D taula hasieratu: $\Theta(p)$

4.- Begizta jalea: (p-2) aldiz errepikatzen dela gogoan edukiz

4.1. X erpina E multzoko erpinen batetik hurbilen dagoen H-ko osagaia mugatzea: Hasieran |H|= p-1 da, eta p-1 erpinetik bat aukeratuko dugu. Begiztaren hurrengo biran |H|= p-2 izango da, eta orduan p-2 erpinetatik bat aukeratu beharko da. Honela: (p-2)+(p-3)+ ...+ 2 erpin aztertu beharko ditugu: $\Theta(p^2)$

4.2. H-tik erpin bat kendu: Aurrekoaren arrazonamendu bera: $\Theta(p^2)$
4.3. H hautagaien listari beste erakusle batek erakustaraztea $\Theta(1)$-koa (p-2) aldiz errepikatuko da: $\Theta(p)$
4.4. H-k dituen erpinen kopurua mugatzea: $\Theta(p^2)$
4.5. Barneko "for" begizta: 4 puntuko begizta lehenengo aldiz egikaritzen denean "for" honek (p-2) azterketa egingo ditu, bigarrengo txandan (p-3) azterketa,... . Kanpoko begiztaren (p-2) egikaritzapenetan zehar guztira (p-2) + (p-3)+ ... + 1 azterketa egin beharko ditu: $\Theta(p^2)$

5.- Azkeneko asignazioa Ada lengoaiak eskatzen du, baina beste programazio lengoaia askotan ez litzateke beharrezkoa izango. Hori dela eta ez dugu kontutan izango (derrigorrezkoa balitz: $\Theta(p^2)$).
⇒Azkenean, kanpoko begiztak behar duen denboraren ordena $\Theta(p^2)$-koa da.

Bestetik, 1,2,3 eta 4 jarraian daudenez, baturaren erregela aplikatuz, Dijkstraren algoritmoak behar duen denboraren ordena $\Theta(p^2)$-koa da.

Baina, arkuen kopurua $a<<p^2$ denean, G matrizean ∞ balioa maiz azalduko da eta balio honekin alderaketak egiten baditugu alferrikakoak direnez denbora galdu besterik ez dugu egingo. Zer egin orduan? Pisuen (edo distantzien) matrize bat eduki ordez, grafoa p listez osaturik dagoen taula baten bidez adieraz genezake. Honela, taulako i posiziotik i erpinaren auzokideak diren erpinak zintzilikatuko dira lista bat osatuz. Inplementazio honek barneko "for" aginduaren denbora asko aurreratuko du, X-en auzokideak diren erpinak bakarrik aztertuko baitira. Baina X erpina aukeratzen duen prozesua (p-2) aldiz errepikatzen da, eta prozesu osoak behar duen ordena $\Theta(p^2)$-koa da. Nola txikitu dezakegu ordena hau?

Soluzio bat: minimoen meta egitura erabili. Metak, H multzoak dituen erpin kopuruaren haina adabegi izango ditu eta edozein y erpin, $y \in H$ izanik, D(y) distantzien balioen arabera sailkatu eta metatuko dira. Honela, D(x) erpina D(x) minimoa duenez gero metaren gailurrean kokaturik egongo da.

• metaren hasieraketa lineala da: $\Theta(p)$

• X erpina aukeratu ondoren, ez da berriz ere aztertuko: metatik kendu beharko da hautagaien zerrenda eguneratzeko. Metaren egiturarekin lanean ari bagara, honek erroa kentzea eta metaren egitura berreskuratzea suposatuko du: $\Theta(\lg p)$

• Barneko "for" begiztan X-en auzokideak diren Y erpin bakoitzarentzat $D(Y) \geq D(X)+G(X,Y)$ gertatzen ote den aztertu behar da. Kasu txarrenean hau gertatzen da; hau da, X zeharkatzen duen Y-rako bide berria motzagoa da, eta D(Y) balio berriarekin eguneratu ondoren, hautagaien metan D(Y) balioa azaleratu beharko da. Honek, $\Theta(\lg p)$ ordenako denbora beharko du, kasu txarrenean. Azaleratzea gehienez behin gertatuko da arku bakoitzeko; beraz, *a* aldiz kasu

txarrenean. Azken batean, (*p-2*) erro kendu (hondoratu) eta *a* arku azaleratu behar badira: Θ((p-2)+a) lg p)= Θ((p+a) lg p).

Ondorioak:

Grafoa konexua bada a ≥ (p-1) eta metak badarabilzkigu, orduan algoritmoaren ordena Θ((p-2)+a) lg p) dela kontsidera dezakegu.

Grafoak arku asko baditu ($a \approx p^2$) -> lehenengo hurbilketa erabiltzea gomendatzen da.

Grafoak berriz arku gutxi baditu (a≈p-1) -> metaren egitura erabiltzea gomendatzen da.

4.3. EKINTZA-HAUTAKETA PROBLEMA

Demagun baliabide bera eskuratzearren lehiatzen duten ekintzen multzo bat dugula. Problema baliabidearen erabilpena ekintzen artean antolatzea da. Kasu honetan, teknika jalea erabiltzeak metodo dotore eta sinple bat eman diezaguke elkarren artean bateragarri eta kardinalitate handieneko ekintzen multzoa aukeratzeko.

Demagun, beraz, baliabide bera erabili nahi duten ekintzak N_EK = {1,...,N} direla. Une bakoitzean ekintza bakar batek gehienez erabil dezakeen baliabide bat inprimagailua izan daiteke, adibidez. *i* ekintza bakoitzak h_i hasiera-denbora eta b_i bukaera-denbora bana ditu, gainera $h_i \leq b_i$ beteko da beti. *i* ekintza aukeratzen bada, hau $[h_i,b_i)$ denbora tartean emango da. Bestalde, *i* eta *j* ekintzak bateragarriak izango dira, baldin eta $[h_i,b_i)$ eta $[h_j,b_j)$ denbora tarteak teilakatzen EZ badira (beraz, $h_i \geq b_j$ edo $h_j \geq b_i$ gertatzen denean).

Ekintza-hautatze problemaren ebazpenak elkar bateragarri (hots, teilakatzen ez) diren kardinalitate handieneko ekintzez osaturiko multzoa mugatzen du.

Algoritmoaren inplementazioa:

```
function EKINTZA_HAUTESLEA (N_EK: in EKINTZEN_L) return EKINTZEN_L is
     N_EK_LAG: EKINTZEN_L;
     S: EKINTZEN_L:= HASIERATU;
     I,J: EKINTZA;
begin
     BUKAERA_T_ORDENA_GORAKORRA_JARRAITUZ(N_EK, N_EK_LAG);
     LEHENENGO_L(N_EK_LAG, J);
     HONDARRA_L(N_EK_LAG);
     ERANTSI_L (S, J);
     while  not HUTSA_DA_L(N_EK_LAG)  loop
           LEHENENGO_L(N_EK_LAG, I);
           HONDARRA_L(N_EK_LAG);
```

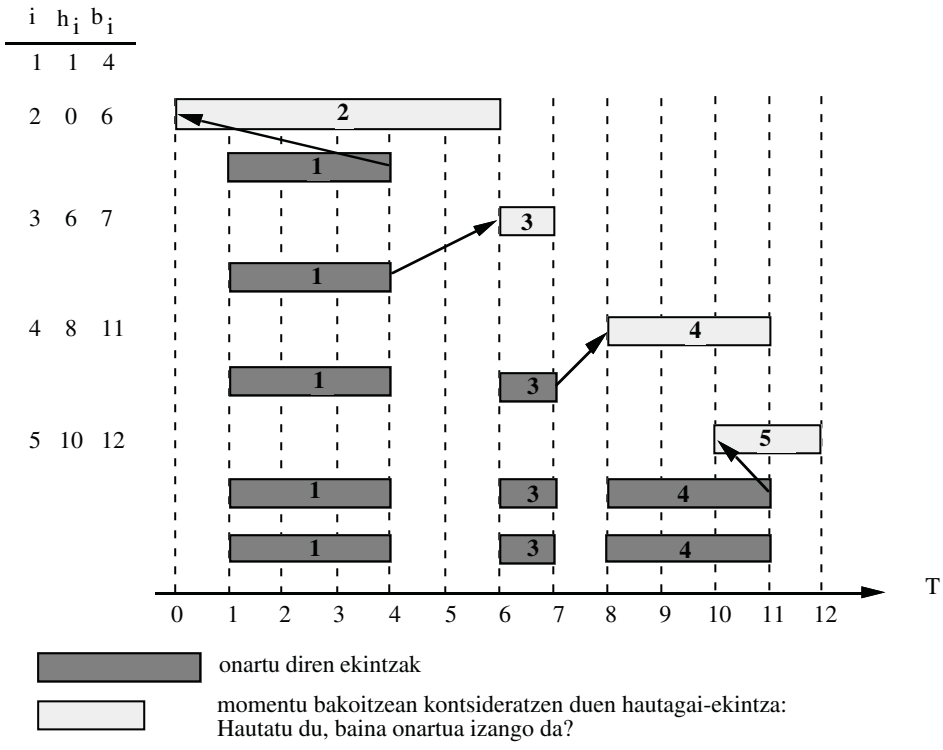
```
            if HASIERA_T(I)>= BUKAERA_T(J) then
                  J:= I;
                  ERANTSI_L (S, J);
            end if;
      end loop;
      return S;
end EKINTZA_HAUTESLEA;
```

J aldagaiak soluzio multzoari erantsitako azkeneko ekintza adierazten du. J-ren bukaera-denbora S-ko edozein ekintzen bukaera-denbora baino handiagoa da. Dakigun bezala, aztertzen diren N_EK_LAG-eko ekintzak bukaera-denborarekiko ordena gorakorra jarraituz sailkatuta daude.

$$b_j = max\ \{\ b_k \mid \exists p(\ POS(N\_EK\_LAG, LEHENENGOA) \le p \le POS(N\_EK\_LAG, I) \wedge k = BUKAERA\_T(ZENB\_GARREN\_EK(S,p))\ )\ \}$$

Algoritmoa jalea al da? Bai, une oro aukeratzen eta eransten den ekintza berriaren *bukaera-denbora* lehenen amaituko den zilezko ekintza da, eta ondorioz, hautaketa horrek gelditzen diren ekintzei ahalik eta posibilitate gehienak uzten dizkie aukeratuak izan daitezen. Hori dela eta, aukeratutako ekintza aukera onena da.

Adibidea:



Denbora-ordenaren analisia kasu txarrenean:

1) N ekintza izanik, hauen ordenazioak behar duen denboraren ordena $\Theta(n \lg n)$ da.

2) Lehenen amaitzen den ekintzaren eskuraketak (kasu hauetan, lehenengo osagaiaren eskuraketak) eta ondoren listan aurrera egiteak denbora konstantea behar dute.

3) Soluzio multzoari (lista baten bidez inplementa dezakeguna) ekintza berri bat eransteak ere denbora konstantea beharko du.

4) Alderaketa eta gorputzak denbora konstantea beharko dute, n-1 aldiz egikarituko dira eta: $\Theta(k\ (n-1))= \Theta(n)$, non *n* ekintza kopurua den.

5) Emaitzaren itzulketa multzoaren kopia ($\Theta(n)$) edo lista bidez inplementatutako kasuan erakusle baten kopia izan liteke ($\Theta(1)$).

$\Rightarrow$ Baturaren erregela aplikatuz, algoritmoaren ordena $\Theta(n \lg n)$ da. Baina, algoritmoak ez balitu ekintzak haien bukaera-denbora balioarekiko ordenatu behar, orduan lineala litzateke. Beraz, nahiko algoritmo eraginkorra dugu.

Ekintza-hauteslea algoritmo jalearen zuzentasuna:

Algoritmoak beti elkar bateragarri diren kardinalitate handieneko ekintza--multzoa mugatzen al du?

Frogaketa:

(Berriz ere, hautaketa jalea segurua eta induktiboa dela besterik ez da frogatuko hemen.)

Hautaketa jalea *segurua* al da? Demagun N_EK= {1,...,N} dagoeneko bukaera-denbora balioarekiko sailkatuta dauden N ekintza direla. Horrela, 1 ekintzaren bukaera-denbora N_EK beste edozein ekintzen bukaera-denbora baino txikiago (edo berdina) izango da. Baina 1 ekintza onartzea algoritmo hautesleak egin dezakeen lehenengo aukera onena da? Suposa dezagun OPT $\subset$ N_EK emandako problemaren soluzio optimo bat dela eta bertako ekintzak bukaera--denbora balioarekiko ordena gorakorra jarraituz sailkatuta daudela. Demagun bertako lehenengo ekintza *k* dela. k=1 bada, orduan algoritmoko S ekintzen soluzio-multzo optimoa soluzio onenarekin hasieratuta dago, S={1}. Baina $k \neq 1$ bada, orduan N_EK-rentzat 1 ekintza bere barnean duen beste OPT' ere soluzio optimoak existitzen duela frogatuko dugu. Izan bedi OPT' =(OPT-{k}) $\cup$ {1}. OPT' ere soluzio optimoa da, zeren: a) $b_1 \leq b_k$ gertatzen denez gero, OPT'-ko ekintzak ez dira denboran elkar teilakatzen eta b) OPT'-k OPT adina ekintza dituenez OPT' ere optimoa da. Horrela, OPT' 1 aukera barne duen N_EK-ko soluzio optimoa da. Ondorioz, beti 1 aukera barnean duen soluzio optimoak existitzen duela frogatu dugu.

Hautaketa jalea *induktiboa* al da? 1 aukera egin ondoren, problemaren ebazpena 1 ekintzarekin bateragarriak diren ekintzen artean soluzio optimoa bilatzera mugatzen da. Hori dela eta, S jatorrizko N_EK-ren soluzio optimoa bada, orduan S'=S-{1} multzoa N_EK'= $\{i \mid i \in N\_EK \wedge h_i \geq b_1\}$ multzoaren soluzio optimoa izango da. Zergatik? Ezin al da azpiproblema horren beste soluzio bat mugatu kardinalitate handiagoarekin eta aldi berean aurrez onartutako hautagaiak barnean dituen multzoarekin bateragarria litzatekeena? S' baino ekintza gehiago lituzkeen S'' beste soluzio bat balego, S''-i 1 ekintza erantsi ondoren hau N_EK-ren soluzioa izango litzateke |S''|>|S'| gertatuz, eta hori S optimoa izatearen aurkakoa litzateke. Honela, aukera bat egin ondoren ebatzi beharreko problema jatorrizko optimizazio problemaren berdina da, baina tamainaz kaxkarragoa.

Hautagaien kopuruan indukzioa egin beharko litzateke algoritmoak eraikitzen duen soluzio multzoa momentu oro osogarria dela frogatzeko, eta horrela, frogaketa guztiz amaitutzat emateko. Hori irakurleak egin beza.

4.4. HUFFMAN-EN KODEKETA

Huffman-en kodeketa datuak trinkotzen dituzten tekniken artean erabilienetako bat da. Trinkotzen diren fitxategien ezaugarrien funtziopean %20 eta %90 inguruko espazio aurrezkiak egiten ditu. Huffman-en algoritmo jaleak karaktere bakoitzaren agerraldien maiztasunak biltegiratzen dituen taula bat eraikitzen du. Taula honi esker, karaktere bakoitza kodetuko duen kate bitar optimoa zein den mugatu ahal izango du.

<u>*Adibidea*</u>:

Demagun 100.000 karakterez osatutako fitxategi bat trinkotu/konprimitu nahi dugula. Bertan, 6 karaktere ezberdin besterik ez daude eta haien maiztasuna ondorengo taulako lehenengo lerroan bildu da:

	a	b	c	d	e	f
Maiztasuna (milakotan)	45	13	12	16	9	5
Luzera finkoko kodeketa	000	001	010	011	100	101
Luzera aldakorreko kodeketa	0	101	100	111	1101	1100

Fitxategi horretako informazioa hamaika modu desberdin jarraituz gorde dezakegu. Karaktere bakoitza kate bitar bakar batez adierazi nahi bagenu ere, posibilitate hauek izango genituzke:

1) *Luzera finkoko kodeketa*: 6 karaktere desberdin adierazteko 3 bit beharko genituzke. Adib.: a=000, b=001,...,f=101. Baina metodo honek 300.000 bit beharko lituzke fitxategia kodetzeko. Kopuru hori hobe genezake?

2) *Luzera aldakorreko kodeketa*: Teknika honen bidez maizen agertzen diren karaktereak kodetzeko luzera txikieneko kodeketa-hitzak erabiliko dira, eta gutxitan agertutako karaktereak, berriz, luzera handiagoa duten kodeketa-hitzekin kodetuko dira. Gure kasuan, eta aurreko maiztasunak kontutan izanik, luzera aldakorreko kodeketa taulan 3. lerroan adierazten den moduan egiterakoan, lortuko genukeen trinkotutako fitxategiak

$$(45 \bullet 1 + 13 \bullet 3 + 12 \bullet 3 + 16 \bullet 3 + 9 \bullet 4 + 5 \bullet 4)\ 1000 = 224.000 \text{ bit}$$

izango lituzke.

Huffman-en kodeketaren definizioa

Huffmanek bere izena daraman kodeketa asmatu zuen; hain zuzen, kodetu nahi den informazioa emanik honen *aurrizki kodeketa* optimo bat eraikitzen duen algoritmo jalea asmatu zuen.

Aurrizki kodeketa

Aurrizki kodeketa batean informazio desberdin bakoitza kodeketa-hitz baten bidez kodetzen da. Gainera, hauetako kodeketa-hitz bakoitza ez da beste kodeketa-hitzen aurrizkia.

Hau gertatzen denean, bai kodeketa bai deskodeketa prozesuak oso sinpleak dira. Aurrizki kodeketako hasierako kodeketa-hitza zein den erraz mugatzearren, deskodeketa prozesuak datuen adierazpide egoki bat eskatzen du: zuhaitz bitarrak. Zuhaitz hauen hostoetan kodetutako datu desberdin guztiak eta haien agerraldien maiztasunak agertzen dira eta errotik hosto baterainoko bideko adarren etiketak kateatzerakoan lortzen den katea hosto horretako datua kodetzen duen kodeketa-hitza da. Zuhaitzeko barne adabegi bat eta bere ezkerreko umea konektatzen duen adarrak 0 etiketa izango du beti eta, eta eskuineko umea konektatzen duenak aldiz 1.

Aurrizki kodeketa optimoa

Zuhaitz bitarra betea denean, fitxategiaren kodeketa optimoa da.

Fitxategiak A alfabetoa definitzen duenean eta fitxategiaren kodeketa optimoa denean (trinkotasun maximoa ematen duenean) zuhaitz bitarrak |A| hosto izango ditu eta |A|-1 barne adabegi.

Z zuhaitzak A alfabetoko aurrizki kodeketa optimoa adierazten duenean, alfabetoko *k* karaktere bakoitzarentzat ondorengo hau beteko da:

– fitxategian *k* karakterearen agerraldien maiztasuna MAIZTASUNA(k) funtzioak definitzen du, eta

– $S_z(k)$ espresioak Z zuhaitzean *k* karaktereari dagokion hostoaren maila adierazten du. $S_z(k)$ espresioak, aldi berean, *k* karakterea kodetzen duen kodeketa-hitzaren luzera adierazten du; hots, bit kopurua.

Aurreko propietateak ikusi ondoren, fitxategi baten aurrizki kodeketa optimoa erabiliz fitxategi horren trinkoketak beharko duen biten kopurua:

$$B(Z) = \sum_{k \in A} MAIZTASUNA(k) \bullet S_z(k) \qquad (1\ Ek.)$$

Aurreko ekuazioak Z-ren *kostua* ematen digu; hau da, trinkotutako fitxategiak beharko duen bit kopurua.

Algoritmoa:

Huffmanen algoritmoak kodeketa optimo bati legokiokeen Z zuhaitza behetik gora eraikitzen du. |A| zuhaitz dituen multzo batekin hasten da eta |A|-1 bateraketa eragiketa egikaritu ondoren |A| hosto dituen zuhaitz bakar bat lortzen du.

Algoritmoak L_ILARA lehentasun-ilara erabiltzen du, non lehentasuna MAIZTASUNA() funtzioak definitzen duen. Funtzio honek A alfabetoko *k* karaktereen maiztasuna, MAIZTASUNA(*k*), ematen duela suposatuko dugu.

L_ILARA lehentasun txikieneko 2 objektu bateratzeko erabiltzen da. Bateratu ondoren lortutako objektu berriaren lehentasuna, bateratutako objektuen lehentasunen baturaren berdina da.

```
function HUFFMAN (A_M: ALFABETOA_MAIZTASUNA)
                  return Z_BITAR_BETEA is
   L_ILARA: LEHENTASUN_ILARA := HUTSA_LI;
   LAG_EZK,LAG_EsK,Z:Z_BITAR_BETEA;
   M:MAIZTASUNA;N:INTEGER;
begin
   N:= ZENBAT_KARAKTERE(A_M);
   HASIERATU(L_ILARA,A_M);
   for  J  in 1..N-1  loop
         LAG_EZK:= LEHENENGOA_LI(L_ILARA);
         LAG_ESK:= LEHENENGOA_LI(L_ILARA);
         M:= MAIZTASUNA(LAG_EZK)+MAIZTASUNA(LAG_ESK);
         Z_ERAIKI( Z, M, LAG_EZK, LAG_ESK);
```

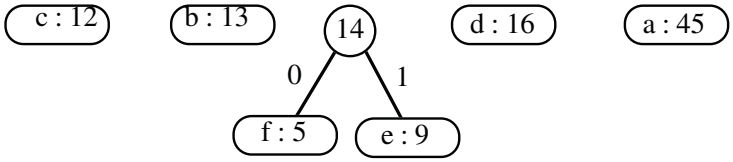
```
            TXERTATU(L_ILARA,Z);
        end loop;
        return LEHENENGOA_LI(L_ILARA);
    end  HUFFMAN;
```

Adibidea:

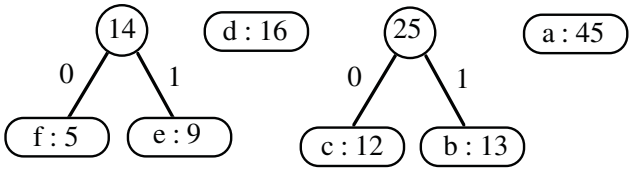
Ikusi bedi algoritmoaren egikaritzapena ondorengo irudian:

(a)

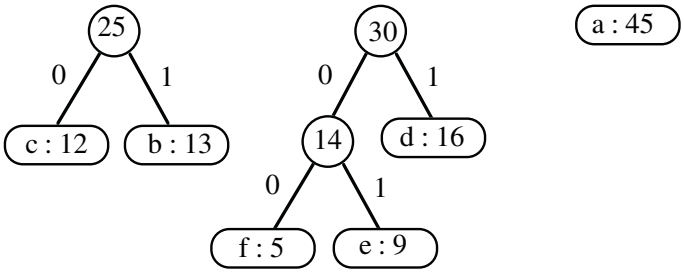
(b)



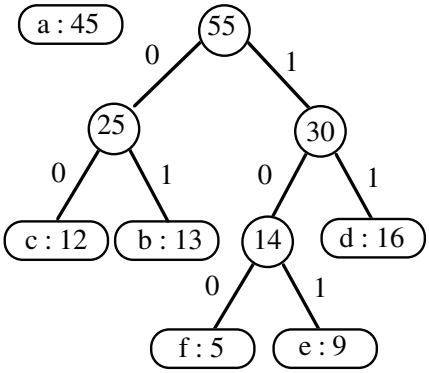
(c)



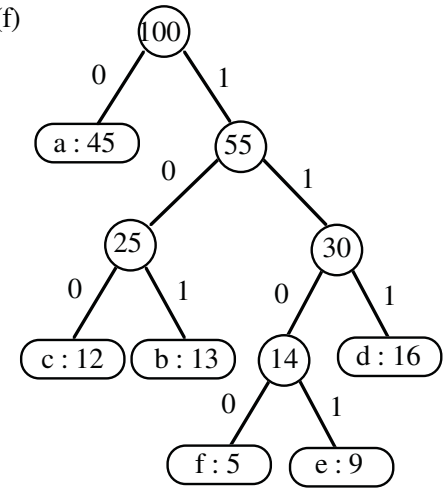
(d)



(e)



(f)



Alfabetoak 6 karaktere dituenez N=6 izango da eta begiztan 5 bira egin ondoren lortuko da zuhaitza, eta ondorioz Huffmanen aurrizki kodeketa optimoko kodeketa-hitzak. Karaktere bat kodetzen duen kodeketa-hitza zuhaitzaren errotik karaktere hori dagoen hostorainoko bideko adarretako etiketak kateatuz lortzen da.

Prozesua:

Lehendabiziko bi aginduak hasieratze aginduak dira. Lehentasun ilara, karaktereekin eta haien agerraldien maiztasun balioekin hasieratzen da, lehentasuna agerraldien maiztasunak mugatzen duelarik. Behin eta berriro, lehentasun gutxien duten bi azpizuhaitzek Z zuhaitz berri bat ematen dute. Zuhaitz berriaren erroan bere bi azpizuhaitzen erroek biltegiratzen duten maiztasunen batura gordetzen da: zuhaitz berriaren maiztasuna izango da. Bi azpizuhaitzetatik ezker azpizuhaitza zein eta eskuin azpizuhaitza zein izango den ez du garrantzirik, azken batean bai bateko bai besteko karaktereen kodeketen kostua bit batean gehitzen da eta. N-1 Z_ERAIKI ondoren ilaran osagai bakarra gelditzen da: alfabetoaren aurrizki kodeketa optimoa.

Ordenaren analisia kasu txarrenean:

Normalki, Huffmanen algoritmoko ilara meta baten bidez inplementatzen dela jakinik eta algoritmoaren egikaritzapena zein den azaldu ondoren, erraz ikus daiteke:

1) Zenbat karaktere ezberdin duen alfabetoak mugatzeko: konstantea izan daiteke Alfabetoa eta maiztasunak taula egitura batean emanik baletoz, eta, lista batean baletoz, aldiz, |A|-rekiko lineala.

2) Ilararen hasieraketa: N karaktere dituen alfabeto bat izanik, L_ILARAren hasieraketak hartuko duen denboraren ordena $\Theta(n)$ da, aurreko gaian ikusitako META_BILAKATU algoritmoa erabiliez gero.

3) Begizta N-1 aldiz egikarituko da, non N alfabetoko karaktereen kopurua den:

3.1) eta 3.2) aginduak: meta batetik tontorreko osagaia kentzea eta berriz ere meta egitura berreskuratzea izango da azken batean L_ILARAko LEHENENGOA lortzea. Ikusi genuen bezala, horren ordena O(lg n) da (O goi bornea, metaren tamaina txikituz doa eta).

3.3) Ilaratik lortutako bi osagaien maiztasunen atzipena konstantea dela suposatuz, objektu berriaren maiztasunaren kalkulua ere denbora konstantean egingo da.

3.4) Zuhaitz berriaren eraikuntzak denbora konstantea hartuko du

3.5) Objektu berri baten txertaketa, azken batean metan dagokion posizioraino azaleratuko da, eta horren ordena logaritmikoa da.

Beraz, $O((n-1) \lg n) \in O(n \lg n)$.

$\Rightarrow$ baturaren erregela erabiliz, $T_{HUFFMAN}(n) \in O(n, n, n \lg n) \in O(n \lg n)$

Huffmanen algoritmoaren zuzentasunaren frogaketa:

Aurreko algoritmo jalea zuzena dela frogatzeko, lehendabizi hautaketa jalea segurua eta induktiboa dela frogatuko dugu, baina hori frogatzea ondorengo bi lemak frogatzearen parekoa da:

Lehenengo lemak, aurrizki kodeketa optimoko problema bati aurre egiten diogunean beti Huffmanen algoritmoak egiten duen aukera egin daitekeela frogatzen du:

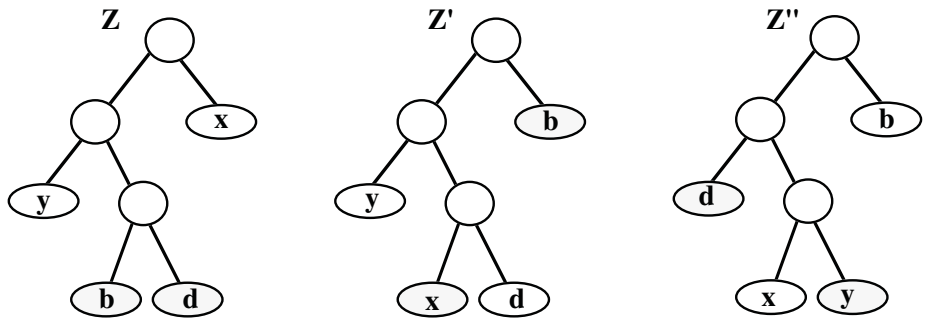
Lema (= hautaketa segurua da**)**

Demagun A alfabetoko $k \in A$ karaktere bakoitzaren maiztasuna MAIZTASUNA*(k)* funtzioaren bidez emana dela. Izan bitez *x* eta *y* A-ko maiztasun txikien duten bi karaktereak. Baldintza hauetan, A-ren aurrizki kodeketa optimo desberdinen artean, beti egongo da bat, non bertan:

– *x* eta *y* kodetzen dituzten bi kodeketa-hitzek luzera berdina izango duten eta azkeneko bitean bakarrik desberdinduko diren, eta

– x eta y kodetzen dituzten kodeketa-hitzen luzera baino kodeketa-hitz luzeagorik ez dagoen.

Frogaketa



Ideia: Z zuhaitzak edozein fitxategiko aurrizki kodeketa optimo bat adierazten badu, orduan beste kodeketa zuhaitz berri batera, Z''-ra, eraldatu behar da: zuhaitz berria ere fitxategi bereko beste aurrizki kodeketa optimo bat izango da, baina bertan *x* eta *y* karaktereak guraso beraren umeak izanik, sakonera mailan egongo dira. Hau beti egin daitekeela froga badezakegu, orduan *x* eta *y*-ren kodeketa-hitzek luzera berdina izango dute eta haien kodeketa-hitzak azkeneko bitean soilik izango dira ezberdin.

Izan bitez *b* eta *d* Z zuhaitzeko maila handieneko guraso beraren bi ume; hau da, sakonera mailan dauden guraso bereko bi umeak. Orokortasunik galdu gabe, *b*-ren maiztasuna *d*-rena baino txikiagoa edo berdina dela ontzat har dezakegu, MAIZTASUNA(b)≤MAIZTASUNA(d), bai eta *x* eta y-renak ere, MAIZTASUNA(x)≤MAIZTASUNA(y). Gainera *x* eta *y* maiztasun txikienak dituzten bi karaktereak direnez (ordena horretan gainera) eta *b*-k eta *d*-k berriz edozein maiztasun dutenez (ordena horretan gainera), orduan MAIZTASUNA(x) ≤MAIZTASUNA(b) eta MAIZTASUNA(y)≤MAIZTASUNA(d) inekuazioak betetzen dira. Orain, *b* eta *x* karaktereak posizioz elkarraldatuko bagenitu Z' zuhaitz berri bat lortuko genuke, eta ondoren, *d* eta *y* karaktereekin gauza bera egingo bagenu, Z'' beste zuhaitz berria lortuko genuke. (1 Ek.) ekuazioa erabiliz Z eta Z' zuhaitzen kostuen diferentzia lor dezakegu

$$B(Z) - B(Z') = \sum_{k \in A} \text{MAIZTASUNA}(k) \bullet S_z(k) - \sum_{k \in A} MAIZTASUNA(k) \bullet S_{z\copyright}(k)$$

$$= \text{MAIZTASUNA}(x) \bullet S_z(x) + \text{MAIZTASUNA}(b) \bullet S_z(b) - \text{MAIZTASUNA}(x) \bullet S_{z'}(x) - \text{MAIZTASUNA}(b) \bullet S_{z'}(b)$$

$$= \text{MAIZTASUNA}(x) \bullet S_z(x) + \text{MAIZTASUNA}(b) \bullet S_z(b) - \text{MAIZTASUNA}(x) \bullet S_z(b) - \text{MAIZTASUNA}(b) \bullet S_z(x)$$

$$= (\text{MAIZTASUNA}(b) - \text{MAIZTASUNA}(x))\ (S_z(b) - S_z(x))$$

$$\geq 0$$

$\Rightarrow$ B(Z) $\geq$B(Z')

ez MAIZTASUNA(b)-MAIZTASUNA(x) eta ez S_z(b)- S_z(x) ez baitira negatiboak. Konkretuki:

- MAIZTASUNA(b)–MAIZTASUNA(x) ez da negatiboa, *x* maiztasun txikien duen hostoa baita, eta
- bestea berriz, S_z(b)- S_z(x), ez da negatiboa, *b* hostoa Z zuhaitzaren sakonera mailan, maila handienean, baitago.

Arrazoi beragatik, *y* eta *d* hostoen elkarraldaketak ez du kostua gehitzen, beraz, B(Z')-B(Z'') ez da negatiboa. Horregatik, B(Z'')$\leq$B(Z) betetzen da, baina Z optimoa denez beste hau ere betetzen da B(Z)$\leq$B(Z''), eta hemendik B(Z'')=B(Z) halabehartzen da. Hori dela eta, Z'' zuhaitza ere kodeketa zuhaitz optimo bat da, eta bertan, *x* eta *y* sakonera mailan guraso bereko haurrak dira, lema deduzituz.

Zuhaitz optimo bat eraikitzen duen prozesua, orokortasunik galdu gabe, maiztasun txikieneko bi karaktereak bat egiten dituen aukera jalea eginez has daitekeela halabehartzen du lemak. Eta, zergatik da hori aukera jalea? Bateraketa ekintzaren kostua bat egiten diren bi objektuen maiztasunen batura dela kontsidera dezakegulako. Eta ondoren, erraz froga daiteke eraikitako zuhaitz osoaren kostua egin diren bateraketen batura dela. Honela, Huffmanen algoritmoak pauso bakoitzean, egin zitezkeen bateraketa guztietatik, kostu merkeena ematen duen bateraketa egitea erabakitzen du.

Ondorengo lemak hautaketa induktiboa dela azaltzen du.

Lema:(=Hautaketa induktiboa da)

Demagun Z zuhaitz bitar beteak A alfabetoarekiko aurrizki kodeketa optimo bat adierazten duela eta A alfabetoko karaktere bakoitzaren maiztasuna MAIZTASUNA*()* funtzioak ematen duela. Demagun *x* eta *y* karaktereak *w*-ren umeak direla. Orduan, *w* karakterearen maiztasuna MAIZTASUNA(w)= MAIZTASUNA(x)+MAIZTASUNA(y) dela suposatuz, Z'=Z-{x,y} zuhaitzak A' = (A- {x, y}) $\cup$ {xy} alfabetoarekiko aurrizki kodeketa optimo bat adierazten du.

Frogaketa

Z zuhaitzaren kostua Z' zuhaitzaren kostuaren funtziopean defini daitekeela ikusiko dugu lehendabizi. Horretarako (1 Ek) ekuazioa erabiliko dugu. Honela, w $\in$ D=A-{x,y} karaktere bakoitzarentzat bai S_z(w)=$S_{z'}$(w) eta bai MAIZTASUNA(w) • S_z(w) = MAIZTASUNA(w) • $S_{z'}$(w) berdinketak beteko dira. Bestalde, S_z(x)=S_z(y)=$S_{z'}$(xy)+1 betetzen denez, ondorengo hau lortzen dugu:

$$
\begin{aligned}
B(Z) &= \sum_{k\in A} \text{MAIZTASUNA}(k) \bullet S_z(k) = \\
&= \sum_{k\in A} MAIZTASUNA(k) \bullet S_z(k) + \text{MAIZTASUNA(x)} + \\
&\quad + \text{MAIZTASUNA(y)} \cdot S_Z(y) \\
&= \sum_{k\in A} MAIZTASUNA(k) \bullet S_{z©}(k) + \left(\text{MAIZTASUNA(x)} + \right. \\
&\quad \left. + \text{MAIZTASUNA(y)}\right)\left(S_{Z'}(xy) + 1\right) \\
&= \sum_{k\in A} MAIZTASUNA(k) \bullet S_{z©}(k) + \text{MAIZTASUNA(xy)} \bullet S_{Z'}(xy) + \\
&\quad + \left(\text{MAIZTASUNA(x)} + \text{MAIZTASUNA(y)}\right) \\
&= \sum_{k\in A} MAIZTASUNA(k) \bullet S_{z©}(k) + \left(\text{MAIZTASUNA(x)} + \text{MAIZTASUNA(y)}\right) \\
&= B(Z') + \left(\text{MAIZTASUNA(x)} + \text{MAIZTASUNA(y)}\right)
\end{aligned}
$$

Z' zuhaitzak A' alfabetoaren aurrizki kodeketa optimo bat adieraziko ez balu, orduan Z'' beste zuhaitz batek existituko luke non A' alfabetuko karaktereak hostoetan izango lituzkeen eta B(Z'')<B(Z') beteko lukeen. *xy* karakterea A'-ko karakterea dela kontsideratu denez, orduan Z'' zuhaitzeko hostoren batean egongo litzateke. Z''-n *x* eta *y* *xy*-ren umeak bezala erantsiko bagenitu, A-rentzat lortuko genukeen aurrizki kodeketaren kostua B(Z'')+(MAIZTASUNA(x)+ MAIZTASUNA(y))<B(Z) litzateke, eta hau Z optimoa izatearen aurka doa. Ondorioz, Z'-k A' alfabetuarekiko optimoa izan behar du.

Teorema

Huffman algoritmoak aurrizki kodeketa optimoa ematen du beti.

Frogapena

3 lema eta 4 lema zuzenean aplikatu besterik ez dugu egin behar.

ARIKETAK: ALGORITMO JALEAK

1- Ondorengo aldarrikapenak egiazkoak ala faltsuak diren argudia bitez, frogaketak edo kontrako adibideak emanez:
   - Bakarra bada pisu txikieneko ertza, hedapen zuhaitz minimoaren barnean dago.
   - Ertz guztiek pisu desberdina badute, hedapen zuhaitza bakarra da.

2- Ertzen pisuak positiboak dituen grafo konexu bat emanik, bere hedapen zuhaitz pseudo-minimoa eraikitzea eskatzen da; hau da, ertz mugatu batzuk dituena.(= Ertz hauek barnean dituzten hedapen zuhaitz guztietatik pisu minimoa duen bat eraiki behar da).

3- Ertzen pisuak positiboak dituen grafo konexu ez-zuezendu baten hedapen zuhaitz minimoa lortzeko, pisurik handieneko ertza kentzearen estrategia *jalea* ona den ala ez aztertu. Estrategia ona balitz, hautaketa jalea segurua eta induktiboa dela frogatu beharko litzateke. Algoritmoa idatzi eta Kruskalen eta Primen soluzioekin aldara bedi. Ona ez balitz, ordea, kontrako adibide bat erabiliz argudia bedi, adibidez.

4- Demagun grafo zuzendu azikliko bat dugula, zeinaren arku bakoitzak pisu ez-negatibo bat erlazionaturik duen. Helburua, abiapuntu batetik beste erpinetaraino doazen bideen arteko bide luzeenen distantziak zenbatekoa den kalkulatzea da. Zuzena litzateke ondorengo hautaketa jalea erabiltzea algoritmo jale batez problema ebazteko?

   Izan bedi S onartutako erpinen multzoa (hasieran S= {Abiapuntua}). Pauso bakoitzean bide BEREZI luzeena duen eta e $\notin$ S den erpina onartuko du hautaketa jaleak.

   (Bide bereziaren kontzeptua Dijkstraren algoritmoan ikusitakoa da).

5- Metropolis hiriko Trafiko Sailak zenbait kaletan zehar konponketa lanak egitea erabaki du. Baina lan hauek aurrera emateko trafikoa etetea beharrezkoa da konponketak egiten ari diren kaleetan zehar. Trafikoa eten daitekeen kaleen zerrenda maximoa ematea eskatzen da, bidagurutze guztiek atxikigarriak izan behar dutela jakinik eta trafiko edukiera ahalik eta handien mantenduz.

6- *n* konputagailuz osaturiko sare internazional bateko X konputagailu bakoitza dobloi bat ordainduz bere Y konputagailu auzokidearekin komunika daiteke. Aldi berean, Y beste batzurekin komunikatuta dagoenez, X beste konputagailurekin komunika daiteke. Noski, komunikatzen den konputagailu bakoitzari dagokion dobloia ordaindu beharko dio.

   PREZIOA(1..N) taula kalkulatuko duen algoritmoa idatz bedi, non PREZIOA(J) 1 konputagailuak J konputagailuarekin konexioa eduki dezan, lehenengoak

ordaindu beharko dituen dobloi kopurua den. Derrigorrezkoa da algoritmoa lineala izatea, O(n).

7- G grafo ez-zuzendua emanik Kruskalen algoritmoak G-ren hedapen zuhaitz minimoa kalkulatzen du Bilatu-Bateratu (UNION-FIND) datu-egitura erabiliz. Metodoak lehendabizi grafoaren ertzez osatutako multzoa haien pisu gorakorrean ordenatzen du, ondoren ertz bakoitza banan bana aztertzeko. Kruskalen ondorengo bariante berria aztertzea eskatzen da. Aldaera berriak meta (HEAP) datu-egitura erabiliko du ertzak metatzeko, eta bertatik banan bana hartuko ditu ertzak ondoren iturri bertsioan bezala tratatzeko. Algoritmoa idaztea eta haren ordena kasu txarrenean kalkulatzea eskatzen da. Erantzun biak justifikatzea derrigorrezkoa da.

8- Dijkstraren algoritmoa (grafo zuzendu bat emanik, erpin batetik beste erpinetaraino doazen bide motzenen kostua kalkulatzen duena) ikasi ondoren, hura aldaraztea eskatzen da gainera ondorengoak kalkula eta itzul ditzan exekuzio denboraren ordena mantenduz:

a) ZENBAT bide minimo dagoen erpin bakoitzeraino

b) bide minimoak (hots, bideak) zein diren.

[Pista:
Dijkstraren benetako algoritmoak bide motzenen kostua kalkulatzeaz at, gainera bide propioak kalkula ditzan (erpinen sekuentziak) aski da bide horiek metatuko dituen BMD taula eranstea. Taula berriaren hasieraketa eta eguneratzea honela egin daiteke:

– BMD(x) posizioan justu X erpinaren aurretik bisitatu behar den erpina zein den metatuko du; beraz, (BMD (x),x) bide minimoen arku bat da.
– Taula berriaren hasieraketa eta eguneratzea D distantzia minimoen taularen hasieraketa eta eguneratzearekin batera egin behar da hurrenez hurren.

```
if D(Y)>D(X)+D(X,Y)  then        — X erantsi den azkeneko erpina da.
     D(Y):= D(X)+G(X,Y);         — Bide berezi berri bat topatu da.
     BMD(Y):= X;
end if;
```

]

9- Erpin batetik besteetaraino dauden distantzien arteko distantzia luzeena kalkulatzeko ondorengo ideia pentsatu da: izan bedi Z oso handia den zenbaki bat eta erpin auzokideen arteko distantzia berria D_Berria(u,v)= Z - D(u,v) formula bidez berkalkulatzen da (non u eta v edozein erpin auzokide diren). Distantzia berriak definitu ondoren, Dijkstraren algoritmoa grafo berriari aplikatuz, honek itzulitako bide motzenen distantzia, jatorrizko grafoan bide

luzeenen distantziak dira. Baina, non dago estrategia honen hanka sartzea? Zergatik ez da zuzena?

10- Kardinalitate handien duten eta elkar bateragarriak diren ekintzen multzoa ekoizteko edozein hautaketa ez da ona. Honela, gutxien irauten duen eta dagoeneko onartuak izan diren ekintzekin bateragarria den ekintza hautatzea eta onartzea, hautaketa jalea ona ez dela frogatzeko, adibide bat ematea eskatzen da. Gelditzen diren (eta dagoeneko onartuak izan diren haiekin bateragarriak diren) ekintzen artetik ekintza gutxienekin teilakatzen den ekintza hautatzea eta onartzea ere hurbilpen okerra dela baiezta bedi adibide batez.

11- Huffmanen kodeketa:

– Zuhaitz bitar ez-beteak aurre-kodeketa optimoei ez dagozkiela froga bedi.

– Aurre-kodeketa optimoen zuhaitzen kostu guztia ere barne adabegi bakoitzaren bi umeen maiztasunen konbinazioa eginez kalkula daitekeela froga bedi.

– Demagun 8 bit erabiliz kodetzen diren karakterez osaturik dauden datuen fitxategi bat dugula. 256 karaktere hauen maiztasunei dagokienez, ezaguna da maiztasun handiena maiztasun txikienaren bikoitza baino txikiagoa dela. Froga bedi Huffmanen kodeketa erabiltzea 8 biten luzera finkoko kodeketa arrunta baina eraginkorragoa ez dela.

12- Eskiko irakasle batek n pare eski bere ikasle berrien artean banatu behar ditu. Eskiko klubaren erregelak jarraituz gero, ikasle bakoitzak bere altuerarekin bat datorren eski parea jaso beharko luke. Irakaslearen problema n eski pareak bere n ikasle berrien artean ahalik eta hobekien banatzea da. Ikasle bakoitzaren altueraren eta honi esleitutako eski parearen luzeraren arteko diferentziaren balio absolutuen batura minimizatzea da irakaslearentzat asignaziorik onena.

Problema ebazteko nahikoa izango da ikasle baxuenari eski pare motzena esleitzea eta modu berean jarraitzea gainontzeko ikasleekin. Estrategia zuzena dela froga bedi.

13- Midas irakaslea bere autoa Iruñetik Tarifara ari da gidatzen X errepidetik. Bere gasolio depositua beterik dagoela habiatzen bada, Midasek n Km gida ditzake. Bestalde bere errepide mapa eguneratuan zerbitzuguneak markaturik datoz, gainera bere bideko zerbitzuguneen arteko distantzia kilometrotan azaltzen du. Irakasleak ahalik eta geldiune kopuru txikiena egin nahi duela eta, Midas irakasleak zein zerbitzugunetan gelditu beharko lukeen adieraziko duen algoritmo eraginkor bat idaztea eskatzen da. Erabilitako estrategiak beti soluzio onena emango duela froga bedi.

14- Demagun maiz kanbiorik gabe gelditzen den kafe banatzaile makina bat dagoela Joneren lantokian. Kanbiorik gabe gelditzen denean kaferik ez duela banatzen jakinik, egoera horretara hain maiz ez iristeko makina berprogramatzea pentsatu Jonek. Jonek honako estrategia inplementatzea pentsatu du:

Datuak: Hasieran N mota desberdineko txanponak daude; txanpon klase bakoitzeko behar adina txanpon kopuru dago (asko).

Algoritmoaren sarrera: KOP= eskatutako kafe motaren prezioa.

Algoritmoaren irteera: KOP kopurua itzultzeko behar den txanpon kopururik txikiena (eta noski, kopuru minimo hori ematen duten txanponak).

Algoritmoaren prozesua: N txanponen artetik, KOP bezain handia edo txikiagoa den txanponik handiena aukeratu, demagun txanpona(K) (erabilitako txanpon kopurua +1 egin eta txanpona itzulketa egiteko erregistratu), eta prozesu berdina KOP-Balioa txanpona(K)) kopuruarentzat egin.

Jonek estrategia hori erabiliko balu, algoritmo eta, ondorioz, programa zuzenak lortuko lituzke? Hau da, bere algoritmoarekin kafe makinak beti txanpon kopuru txikiena itzuliko luke? Erantzuna argudia bedi.

15- Izan bedi M1,...,Mn n matrizez osaturiko sekuentzia, non Mi matrizearen dimentsioa Di-1xDi den $\forall i$ $1 \leq i \leq n$. Ondoren, matrize-sekuentziako matrizeen arteko biderkaketa egiteko estrategia jale bat proposatzen da:
Pauso bakoitzean, gelditzen diren matrizeen dimentsio handiena hartu bedi eta dimentsio hori konpartitzen duten bi matrize auzokideren arteko biderkaketa egin bedi.
Estrategia honek faktorketa-ordena optimoa ematen duela ikus bedi.

• Algoritmoaren exekuzio-ordena zein da? (Faktorketa egiteko soilik, benetako biderkaketa egin gabe).

• Edo estrategiak beti faktorketa optimoa kalkulatzen duela froga bedi (hots, propietate seguru eta induktiboak frogatu) edo aurkako adibide bat eman bedi.

5. GAIA

ZATITU ETA IRABAZI ALGORITMOAK

5.1. SARRERA

5.1.1. Zenbait kontzeptu eta azalpen

'Zatitu eta irabazi' teknika pauso hauetan banatzen da.

– Problema tamaina txikiagoa duten problema bereko zenbait instantziatan banatzen da,

– txikiagoak diren instantziak errekurtsiboki ebazten dira metodoren bat erabiliz,

– azkenean, lortutako emaitza guztiak konbinatuz, jatorrizko sarreraren emaitza lortzen da.

Ordenazio algoritmoen barnean zatitu eta irabazi teknika jarraitzen duten *Quicksort* eta *Mergesort* algoritmoak aztertu ditugu. Ikusitako beste adibide bat *bilaketa dikotomikoa* dugu. Orain, gai honetan adibide berriak ikusiko ditugu.

5.2. BATERAKETA-ORDENAZIO BEREZI BAT

Ondorengo errekurtsio-ekuazioa: $T(n) = \begin{cases} d & \text{baldin } n \le a \\ \sqrt{n}\,T(\sqrt{n}) + bn & \text{baldin } n > a \end{cases}$

askatu nahi dugu. Horretarako n-ren balioak a^{2^j} modukoak direla eta $k \ge 0$ betetzen dela suposatuko dugu.

Bateraketa (*Mergesort*) algoritmoaren beste hurbilketa bat azter dezagun: ordenatu nahi den taula oraingoan $\sqrt{n}$ zatitan banatzen da, eta algoritmoa errekurtsiboki aplikatu ondoren lortutako taulak nahastuz (*Merge*) lortzen da emaitza. Aurreko errekurtsio-ekuazioak bateraketa algoritmoaren hurbilketa berriak hartuko lukeen exekuzio-denbora adierazten du? Zergatik? Errekurtsio--ekuazio zuzena idatzi eta argudiatuko dugu.

Ebazpena:

$$T(n) = \sqrt{n}\,T\left(\sqrt{n}\right) + bn = n^{\frac{1}{2}}\left(n^{\frac{1}{4}}T\left(n^{\frac{1}{4}}\right) + bn^{\frac{1}{2}}\right) + bn = n^{\frac{1}{2}+\frac{1}{2^2}}T\left(n^{\frac{1}{2^2}}\right) + 2bn$$

$$= n^{\frac{1}{2}+\frac{1}{2^2}+\frac{1}{2^3}}T\left(n^{\frac{1}{2^3}}\right) + 3bn, \text{ eta orokorrean i. pausoan}$$

$$= n^{\sum_{k=1}^{i}\frac{1}{2^k}}T\left(n^{\frac{1}{2^i}}\right) + ibn \quad \text{jakinik } n=a^{2j} \text{ modukoa dela, eta i=j eginaz}$$

$$= n^{1-\frac{1}{2^i}}T(k) + ibn \underset{\text{desegiten}}{\overset{\text{aldaketak}}{=}} d\frac{n}{\frac{1}{n^{\lg_a n}}} + bn\left(\lg\lg_a n\right)$$

$$T(n) \in \Theta(n \lg(\lg_a n))$$

Lortzen den emaitza arretaz aztertzen badugu, alderaketa bidezko algoritmoen behe bornea baino txikiagoa dela konturatzen gara. Ondorioz, ebatzitako errekurtsio-ekuazioa alderaketa bidezko ordenazio algoritmoren bati egokitzea ezinezkoa da.

Azaldutako metodo edo algoritmoari dagokion errekurtsio-ekuazioa honako hau da (sinplifikaturik):

$$T(n) = \begin{cases} d & \text{baldin } n \le a \\ \sqrt{n}\,T(\sqrt{n}) + n\sqrt{n} & \text{baldin } n > a \end{cases}$$

Ekuazioa *i* aldiz hedatuz:

$$T(n) \le n^{\sum_{k=1}^{i}\frac{1}{2^k}}T\left(n^{\frac{1}{2^i}}\right) + n\sum_{j=1}^{i} n^{\frac{1}{2^i}}$$

bertako bigarren batugaian ageri den batukaria honela borna dezakegu:

$$n^{1/2}\sum_{j=1}^{i} n^{\frac{1}{2^i}} \le 2n^{1/2}$$

$$T(n) \in O(n^{3/2})$$

Metodo berriak beharko lukeen denbora-ordena alderaketa bidezko algoritmoen behe borne minorantea baino okerragoa da.

5.3. ZENBAKI BATEN N. POTENTZIA

Demagun *a* zenbaki baten *n*. potentzia kalkulatu nahi dugula berreketa eragilerik erabili gabe, eta zatitu eta irabazi teknika aplikatuz.

Algoritmoa:

```
function  BER(A: in  INTEGER; N: in INTEGER) return INTEGER is
    X: INTEGER;
begin
    if  N=1 then return A
    else  X:= BER(A, N/2);
          if N rem 2 = 0  then    return X * X;
          else return A * X * X;
          end if;
    end if;
end  BER;
```

Ordenaren analisia:

Algoritmoa errekurtsiboa da, eta batez ere biderkaketak egiten ditu. Ondorioz, algoritmoak egiten dituen biderkaketa kopurua egindako lanaren adierazgarri ona izango da:

$$T(n)\begin{cases}0 & n=1\\ 1+T(n/2) & r \text{ rem } 2=0\\ 2+T(n/2) & r \text{ rem } 2=1\end{cases}$$

T(n) = k+T(n/2) errekurtsioa hedatuz algoritmoaren ordena $\Theta(\lg n)$ dela erraz ikusten da, non k= max (1,2) den.

5.4. MAXIMOA ETA MINIMOA BILATZEN

Bektore bateko osagai maximoa eta minimoa bilatzen dituen algoritmo sinple batek bere osagaien artean 2n-2 alderaketa egiten du. Zatitu eta irabazi teknika erabiliz *(3/2) n - 2* alderaketa egingo duen eta problema ebazten duen algoritmoren bat idaztea ez da lan zaila.

Algoritmo baten inplementazioa:

```
procedure MIN_MAX (B: in BEKTOREA; MIN, MAX: out INTEGER) is
    MIN1,MIN2,MAX1,MAX2: INTEGER;
begin
    if B'LENGTH=2
    then  if B(B'FIRST)>B(B'LAST)
          then MAX:= B(B'FIRST); MIN:= B(B'LAST);
          else MAX:= B(B'LAST); MIN:= B(B'FIRST);
          end if;
    else  MIN_MAX(B(B'FIRST..(B'FIRST+B'LAST)/2),  MIN1, MAX1);
          MIN_MAX(B(((B'FIRST+B'LAST)/2)+1)..B'LAST),  MIN2, MAX2);
          if MAX1>MAX2 then  MAX:= MAX1; else MAX:= MAX2;
          end if;
          if MIN1<MIN2 then MIN:= MIN1; else MIN:= MIN2;
          end if;
    end if;
end MIN_MAX ;
```

Ordenaren analisia:

Egiten diren osagaien arteko konparazioak neurtzen dituen errekurtsio--ekuazioak:

$$T(n)=\begin{cases}1 & n=2\\ 2T\left(\frac{n}{2}\right)+2 & n>1\end{cases}$$

Kasu nabarian alderaketa bakarra egiten da, eta orokorrean, berriz, 2 alderaketa gehi dei errekurtsibo bakoitzean egiten direnak. Sarreraren tamaina 2ren potentzia dela suposatuko dugu.

Errekurtsioa ebazteko haren hedapena kasu nabariraino eramango dugu:

$$T(n)=2T\left(\frac{n}{2}\right)+2=2\left(2T\left(\frac{n}{2^2}\right)+2\right)+2=2^2T\left(\frac{n}{2^2}\right)+2^2+2=$$

$$=2^3T\left(\frac{n}{2^3}\right)+2^3+2^2+2=\ldots=$$

$$=2^iT\left(\frac{n}{2^i}\right)+\sum_{j=1}^{i}2^j= \qquad \text{kasu nabaria:}\frac{n}{2^i}=2$$

$$=\frac{n}{2}T(2)+2\frac{n}{2}-2=\frac{3}{2}n-2\in\Theta(n)$$

5.5. SEKUENTZIA BATEKO BATURA MAXIMOA EMATEN DUEN SEGMENTUAREN MUGA

Zenbaki osokoen taula bat emanik, batura maximoa ematen duen segmentua mugatu nahi da zatitu eta irabazi teknika erabiliz.

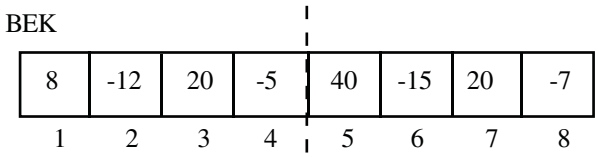
Ideia sinplea bada ere soluzioa ez da begibistakoa. Osokoen taula bitan banatuko bagenu, batura maximoa emango lukeen segmentua topatuko genuke: ezkerraldeko zatian, eskuineko zatian edo banaketa puntua barnean izango lukeen segmentu batean. Bai ezkerraldeko zatiko batura maximoa bai eskuinaldekoa ematen duten segmentuak kalkula ditzakegu errekutsio bidez. Banaketa-puntua barnean duten segmentu guztietatik batura maximoa ematen duenaren kalkulua aldiz modu ezberdinetan egin daiteke. Ondorengo azpiatalean posibilitate bat aurkezten dugu.

Algoritmoaren inplementazio bat:

```
procedure   BATURA_MAX   (BEK: in BEKTOREA; B: out INTEGER;
                          LEHENA,AZKENA: out INDIZEA) is
    — Ondoren eta ordenaren kalkulua erraztearren BEK-ren luzera 2ren
    — potentzia izatea eskatuko dugu.
    ...          — Aldagai lokalen erazagupena
begin
    if  BEK'LENGTH=2
    then
          3_ARTEAN_MAX(BEK(BEK'FIRST..BEK'FIRST),
                       BEK'FIRST,BEK'FIRST,
                       BEK(BEK'FIRST..BEK'LAST), BEK'FIRST,BEK'LAST,
                       BEK(BEK'LAST..BEK'LAST), BEK'LAST,BEK'LAST,
                       B,LEHENA,AZKENA);
    else
          — Ezkerreko zatiko Batura maximoa zein segmentuk ematen duen
          — kalkulatzen du.
          BATURA_MAX  (BEK(BEK'FIRST..(BEK'FIRST+BEK'LAST)/2),
                       B_EZ,LEHENA_EZ,AZKENA_EZ);
          — Berdin, baina eskuineko zatian.
          BATURA_MAX  (BEK(((BEK'FIRST+BEK'LAST)/2)+1..BEK'LAST),
                       B_ES,LEHENA_ES,AZKENA_ES);
          — Batura maximoa ematen duen segmentuak ezkerreko zatiko osagai
          — batzuk izan ditzake eta eskuineko beste batzuk; hau da, n/2 puntua tartean
          — duten segmentuetatik batura maximoa ematen duena kalkulatuko dugu orain.
          — n/2 barne duen segmentu handienaren aurrizki handienaren kalkulua.
          LAG(BEK'FIRST+BEK'LAST)/2):= BEK(BEK'FIRST+BEK'LAST)/2);
          for I in reverse BEK'FIRST..((BEK'FIRST+BEK'LAST)/2)-1
                   loop
                LAG(I):= BEK(I)+LAG(I+1);
          end loop;
```

```
        BEKTOREAN_MAXIMOA(LAG(BEK'FIRST..(BEK'FIRST+BEK'LAST)/2),
                          B_ERD_EZ, POSI_ERD_EZ);
        — n/2 barne duen segmentu handienaren atzizki handienaren kalkulua
        LAG((BEK'FIRST+BEK'LAST)/2)+1):=
              BEK((BEK'FIRST+BEK'LAST)/2)+1);
        for I in ((BEK'FIRST+BEK'LAST)/2)+2..BEK'LAST  loop
              LAG(I):= LAG(I-1)+BEK(I);
        end loop;
        BEKTOREAN_MAXIMOA(LAG((BEK'FIRST+BEK'LAST)/2)+1..BEK'LAST),
                          B_ERD_ES, POSI_ERD_ES);
        — batura maximoa eta n/2. osagaia barne duen segmentua BEK bektoreko
        — POSI_ERD_EZ eta POSI_ERD_ES posizio artekoa da eta baturaren balioa berriz
        — BAT_ERD_EZ+BAT_ERD_ES
        3_ARTEAN_MAX(B_EZ,LEHENA_EZ,AZKENA_EZ,
                     B_ERD_EZ+B_ERD_ES, POSI_ERD_EZ ,POSI_ERD_ES,
                     B_ES,LEHENA_ES,AZKENA_ES,
                     B,LEHENA,AZKENA);
end BATURA_MAX;
```

Adibide bat:



Eta aldagaiak honela geldituko lirateke:

B_EZ= 20	LEHENA_EZ=3	AZKENA_EZ=3
B_ERD_EZ+B_ERD_ES=15+45	POSI_ERD_EZ=3	POSI_ERD_ES=7
B_ES=45	LEHENA_ES=5	AZKENA_ES=7

Denbora-ordenaren analisia:

Algoritmoa (eta ondorioz bere inplementazio zuzena) errekurtsiboa denez gero, denbora-ordena kalkulatzeko lehendabizi algoritmoaren errekurtsio-ekuazioa idatzi behar dugu. Hau idazteko, aurrez 3_ARTEAN_MAX prozeduraren ordena jakin behar da. Honek, hiru balioen arteko balio maximoa kalkulatzeko 2 alderaketa egin eta balio hori itzultzeaz gain, balio maximoa den azpitartearen hasiera eta amaiera indizeak itzultzen ditu. Lan horren kalkuluak denbora konstantea behar du. Demagun konstante hori *a* dela. Hori mugatu ondoren, errekurtsio-ekuazioa honela gelditu litzateke:

Kasu orokorrean:

– n/2 tamainako bi dei errekurtsibo egiten dira: 2 T(n/2).

– n/2 tamainako bi bektore osatzen dira eta bi bektore horietako balio handienak zein bi posiziotan dauden mugatzen da; beraz, n/2 tamainako bi bektoreen osatzea eta haien korritzea ondokoa litzateke: b (n/2)=b' n.

– Zenbait oinarrizko eragiketaren denbora (beti eragiketa kopuru berdina) : d

$$T(n)=\begin{cases} a & n=2 \\ 2T\left(\frac{n}{2}\right)+b\frac{n}{2}+d & n>2 \end{cases}$$

Errekurtsioa kasu nabariraino hedatzen badugu:

$$T(n)=2T\left(\frac{n}{2}\right)+b\frac{n}{2}+d=2\left(2T\left(\frac{n}{2}\right)+b\frac{n}{2^2}+d\right)+b\frac{n}{2}+d=$$

$$=2^2T\left(\frac{n}{2^2}\right)+2b\frac{n}{2}+2d+2=$$

$$=2^3T\left(\frac{n}{2^3}\right)+3b\frac{n}{2}+4d+2d+d=\ldots=$$

$$=2^iT\left(\frac{n}{2^i}\right)+ib\frac{n}{2}+d\sum_{j=0}^{i-1}2^j= \quad \text{i. pausuan}$$

Kasu nabaria zein den kontutan izanik, hau da, $\frac{n}{2^i}=2$, ondoko ordezkapena lortzen dugu: $\lg\frac{n}{2}=i$ edo $\lg n-\lg 2=i$.

$$=\frac{a}{2}n+(\lg n-1)b\frac{n}{2}+d\left(\frac{n}{2}-1\right)\in\Theta(n\lg n)$$

(Irakurlea ohar liteke T(n) ekuazioa 2T(n/2)+n ekuazioan sinplifika daitekeela, biak denbora-ordena berekoak baitira. Ekuazio berria, aldiz, MergeSort algoritmoari dagokionaren berdina denez eta MergeSorta $\Theta(n \lg n)$ denez, zuzenean BATURA_MAXren ordena $\Theta(n \lg n)$ dela halabehar zezakeen irakurleak).

Azpisekuentzia guztiak kalkulatuko lituzkeen algoritmoaren ordena *koadratikoa* litzatekeenez, eman den soluzioaren *n lg n* goi bornea nahiko ona da. Halere, posiblea litzateke soluzio lineal bat ematea.

(Ideia: jatorrizko sekuentziaren igarotze bakar batean:

– metagailu batean itzultzea espero den segmentuko osagaien batura metatuko da,

– metagailua negatiboa (edo 0) ez den bitartean: itzuli beharreko segmentuari osagai bat gehiago eransten zaio eta haren balioa, berriz, metagailuari gehitu.

– bestela, metagailua 0 edo negatiboa bihurtzen bada, itzuli beharreko sekuentziaren eragiketa berriz hasi eta metagailua zerora hasieratu.)

5.6. HAUTAKETA ETA MEDIANAREN BILAKETA

m T bektore bateko osagai bat izanik, T-n m hau baino txikiagoak diren osagaien kopurua eta *m* hau baino handiagoak diren osagaien kopurua berdinak direnean m T bektorearen *mediana* dela esango dugu. Definizioa oso zehatza ez bada ere, nahi dugun ideia argi azaltzen du. Benetako definizio formalean T bektorearen osagai kopurua bikoitia izan daiteke eta osagai guztiek ez dute desberdinak izan beharrik. Definizioa hau da:

$$m \text{ T-ko mediana } + m = T(k) \wedge N i \{ 1 \le i \le n \wedge T(i) < m \} < (n/2) \wedge N i \{ 1 \le i \le n \wedge T(i) > m \} \le (n/2)$$

37	16	3	15	11	12		8	30
1	2	3	4	5	6	7	8	9

Taula ordenatu gabe dago: m = 12

2	3	8	11	12	15	16	30	37
1	2	3	4	5	6	7	8	9

Taula odenatuta dago: T($\lfloor n / 2 \rfloor$)

Zein algoritmok kalkula dezake bektore baten mediana?

1. hurbilketa:

Bektorea ordenatu ondoren $\lceil n/2 \rceil$. osagaia aukeratuko bagenu, hura bektorearen mediana litzatekeela argi dago. Baina hurbilketa horrek bai kasu txarrenean bai eta batezbesteko kasuan ere $\Theta(n \lg n)$ ordenako denbora beharko luke.

Hau baino eraginkorragoa den beste algoritmo bat badago:

2. hurbilketa:

Gure problema orokortuko bagenu, *k*.aren hautaketa problema definituko genuke. T bektorea emanik *k*-k ondokoa bete behar du 1=T'FIRST ≤ *k* ≤ T'LAST=n. T-ren *k*. osagai txikienak ondorengo baldintza hauek betetzen dituen *m* osagaia da:

$$m \text{ T-ko } k. \text{ osagai txikiena da} \Leftrightarrow \begin{array}{l} \mathcal{N}i\ \{\ 1 \le i \le n \wedge T(i) < m\} < k\ \wedge \\ \mathcal{N}i\ \{\ 1 \le i \le n \wedge T(i) \le m\} \ge k \end{array}$$

Hau da, T ordena gorakorra jarraituz ordenatuta balego, *m* osagaia T-ko *k*. posizioan legoke.

Gure hasierako *medianaren bilaketaren problema*, *k.aren hautaketa problema*ren kasu konkretu bat da: Medianaren bilaketa eta $\lfloor (n/2) \rfloor$. . txikien osagaia hautatzea problema bera da.

Ondorengo algoritmoak Quicksort (Hoare-ren) algoritmoaren ideia bera jarraitzen du:

Algoritmoa:

```
procedure  K_HAUTAKETA  (T: in BEKTOREA: K: in INTEGER,
                         TK: out T_ELEM) is
    T1: BEKTOREA:=T; B: INDIZEA;
— T bektoreko K. osagai txikiena TK-n itzultzen du, non 1 ≤ K ≤ n (=T'Last-T'First+1).
begin
    if  Txikia_da?(T) then
          Sailkatu_ordena_gorakorra_jarraituz(T,T1);
          TK:=T1(K)
    else
          BANAKETA(T1,B);      — B = Banaketa-puntua
          if Banaketa_Puntua_K_da?(T1,B,K)
          then  TK:= T(B);
          else
                if K<B
                then  K_HAUTAKETA( T1(T1'FIRST..B-1), K, TK);
                else   — K>B
                      K_HAUTAKETA( T1(B+1..T1'LAST), K-B, TK);
          end if;
    end if;
end K_HAUTAKETA;
```

Banaketa_Puntua_K_da?(T,B,K)=T'FIRST+K-1. Hau da, logikoki, banaketa puntua taulako K.go posizioa den galdetzea da.

Hautaketaren ordenaren analisia:

Kasu txarrena:

Kasu txarrenean, Banaketa(1,n,B)-ren exekuzioak banaketa-puntua, B, T bektorearen ertz batean kokatzen du eta ondorengo dei errekurtsiborako n-1 tamaina duen bektore bat uzten du.

(Begira bedi BANATU algoritmoak egiten duena bai eta hartzen duen denbora-ordena ere: 2.4.1 puntuan $\rightarrow \Theta(n)$)

Ondorioz, algoritmoaren exekuzioak behar duen denbora honako errekurtsio--ekuazioaren bidez adieraz dezakegu: $T_t(n) = a\ n + T_t(n-1)$. Errekurtsibitatearen hedapen metodoa aplikatuz erraz $T_t(n) = (a/2)\ (n^2+n) \in \Theta(n^2)$

Batezbesteko kasua:

Batezbesteko kasuaren analisia egiteko garaian, n osagai dituen T bektorearen n! permutazioak ekiprobableak direla suposatuko dugu. Honek banaketa-puntuaren banaketa-balioaren kokapenaren probabilitatea mugatzen du: 1/n.

K. osagai txikiena hautatzeko n osagai dituen T bektore batek beharko duen batezbesteko denbora, $T_b(T, n, K)= t^K(n)$, honela goi-borna dezakegu:

$$t^K(n) \le an + \frac{1}{n}\left(\sum_{B=1}^{K-1} t^{K-B}(n-B) + \sum_{B=K+1}^{n} t^K(B-1)\right) \quad n \ge 2$$

$$t^K(1) = d$$

$$t^K(0) = 0$$

Izan bedi R(n):

$$R(n) = \max_{1\le K\le n}\left\{t^K(n)\right\} \qquad \forall n R(n) \le R(n+1)$$

$t^K(n)$ bere goi-bornaketarekin ordezkatuz:

$$R(n) = \max_{1\le K\le n}\left\{an + \frac{1}{n}\left(\sum_{B=1}^{K-1} t^{K-B}(n-B) + \sum_{B=K+1}^{n} t^K(B-1)\right)\right\}$$

$$= an + \frac{1}{n}\max_{1\le K\le n}\left\{\sum_{B=1}^{K-1} t^{K-B}(n-B) + \sum_{B=K+1}^{n} t^K(B-1)\right\}$$

$$\leq an + \frac{1}{n}\max_{1\leq K\leq n}\left\{\sum_{B=1}^{K-1} R(n-B) + \sum_{B=K+1}^{n} R(B-1)\right\}$$

$$\leq an + \frac{1}{n}\max_{1\leq K\leq n}\left\{\sum_{i=n-K+1}^{n-1} R(i) + \sum_{i=K}^{n-1} R(i)\right\}$$

Sarrerako tamainaren gaineko indukzio bidez $R(n) \leq 4\,an$ betetzen dela frogatuko dugu; hau da, R(n) lineala den funtzio batekin goi-borna dezakegula.

• Kasu nabaria: n=2

Definizioz:

$$R(2) \leq 2a + \frac{1}{2}R(1) = 2a + \frac{1}{2}d \overset{(*)}{\leq} 4an = 8a \qquad (*)\forall a \geq \frac{1}{12}d = \frac{1}{12}R(1)$$

• Indukzio hipotesia: .$n(2 \leq n < m \rightarrow R(n) \leq 4an)$,

• Kasu orokorra:

$$R(m) \leq am = \frac{1}{m}\max_{1\leq K\leq n}\left\{\sum_{i=m-K+1}^{m-1} R(i) + \sum_{i=K}^{m-1} R(i)\right\}$$

Baina, badakigu R(n) ez dela beherakorra, eta ondorioz bi batukariek

$$\sum_{i=m-K+1}^{m-1} R(i) + \sum_{i=K}^{m-1} R(i)$$

balio maximoa K= m/2 *m* bikoitia denean, edo
K= (m+1)/2 *m* bakoitia denean emango dute.

<u>m bikoitia bada ondorengo hau daukagu</u>:

$$\begin{aligned} R(m) &\leq am + \frac{2}{m}\sum_{i=m/2}^{m-1} R(i) \\ &\leq am + \frac{2}{m}4a\sum_{i=m/2}^{m-1} i \\ &= am + \frac{8a}{m}\frac{3m^2 - 2m}{8} \\ &= am + 3am - 2a \\ &= 4am - 2a < 4am \end{aligned}$$

m bakoitia bada, berriz, ondorengo hau daukagu:

$$
\begin{aligned}
R(m) &\le am + \frac{2}{m} \sum_{i=(m+1)/2}^{m-1} R(i) \\
&\le am + \frac{2}{m} 4a \sum_{i=(m+1)/2}^{m-1} i \\
&= am + \frac{8a}{m} \frac{3m^2 - 4m + 1}{8} \\
&= am + 3am - 4a + \frac{a}{m} \\
&< 4am \qquad (m \ge 1 / 4\text{-rentzako, baina m osoa da})
\end{aligned}
$$

K edozein izanik $t^K(n) \le R(n)$ betetzen denez, $\forall K$, $1 \le K \le n$, $t^K(n) \in O(n)$ ordenakoa da.

5.7. OSO HANDIAK DIREN ZENBAKI OSOKOEN ARITMETIKA

Orain arte egin ditugun analisietan batuketa eta biderkaketa oinarrizko eragiketak zirela suposatu dugu; hau da, eragiketa hauen egikaritzapena erabiltzen ari garen konputagailuaren funtziopean dagoen konstante batez goi-borna dezakegu. Baina suposaketa hau zuzena da, baldin eta eragigaiak hardware bidez maneiagarriak badira. Bestetik, badago zenbait aplikazio oso handiak diren zenbakiak erabili behar dituena. Koma mugikorrezko idazkera erabiltzea ez da gomendagarria kasu hauetan, zenbakiaren ordena mantentzen bada ere mantisak digitu garrantzizkoenak bakarrik gordetzen baititu. Azken batean, konputagailuaren hitzaren tamainak zenbakia adierazteko zehaztasuna mugatzen du. Honela, digitu guztiak gordetzea guretzat beharrezkoa bada, irtenbide bakarra zenbakien errepresentazioa eta eragiketa aritmetikoak software bidez inplementatzea da.

Oso handiak diren zenbaki osokoek kriptografiaketan ezinbesteko garrantzia dute.

Gure helburua beraz, N digitu dituen zenbaki baten adierazpide bat aurkitzea da. Posibleak diren adierazpide guztietatik, zein aukeratu? Adierazpideak baldintza hauek betetzea nahiko genuke:

– bai zenbaki negatiboak bai eta zenbaki positiboak ere kontutan izan ditzala,

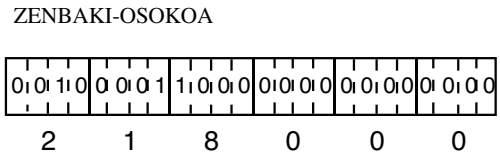
– batuketak, kenketak eta hamarren (edo beste oinarriren bateko) multiploak

diren biderkaketen eta zatiketa osoen eragiketen exekuzio-denborak linealak izan daitezela.

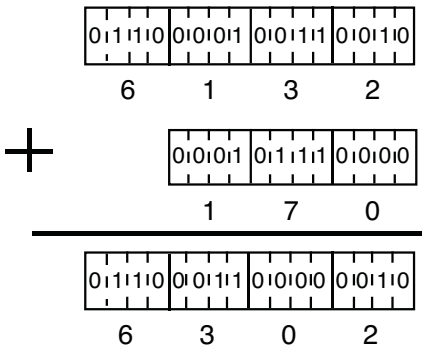
Posiblea al da orduan N digitu dituen zenbaki bat adierazteko biten konbinazio lineal bat erabiltzea? (hau da, = O(N) ordenako bit kopurua soilik behar izatea).

Digitu hamartar baten balio posibleak 0tik 9raino bitarteko balioak dira, biok barne. Beraz, digitu hamartar bat adierazteko behar dugun bit kopurua $\lceil \lg 10 \rceil = 4$ da: (gure konputagailuarentzat 21.800 handia den zenbaki osoko bat dela suposatzen badugu)

```
subtype DIGITUA is INTEGER range 0..9;
type ZNBK_OSO_HANDIA is array (NATURAL range <>) of DIGITUA;
ZENBAKIA: ZNBK_OSO_HANDIA:= HASIERATU;
```



Denborari dagokionez, konputagailu gehienetan oinarrizko biderkaketak ez dira batuketak baino askoz ere garestiagoak. Baina, eragigaien tamainak oso handiak direnean, ez da gauza bera gertatzen. Honela, N eta M digitu dituzten bi zenbakien arteko batura kalkulatzen duen algoritmo bat bilatzean, honen exekuzio--denbora $\Theta(N+M)$ ordenakoa den bat aurkitzea ez da zaila. Adibidez:



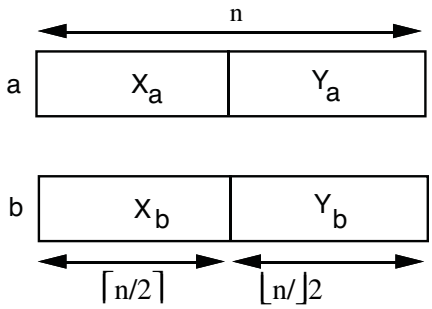
Aurreko adibidean ikus dezakegunez, N eta M zifra dituzten (gure kasuan digitu hamartar) bi zenbakien batura kalkulatzeko, hau digituz digitu egiten badugu, baturaren exekuzioaren ordena da:

Θ((N+M) (4_bit_bikote_batzeko_denbora)) =

Θ((N+M) (kte.)) = O(N+M)

Beraz, batuketaren exekuzio-denboraren ordena lineala da (zifren kopuruan). Baina biderkaketarekin berriz, zer gertatzen da?

Biderkaketa errusiarraren eta klasikoaren exekuzio-denbora koadratikoa da (bi zenbakien tamaina berdina denean). Ezin al da denbora hori hobetu? Izan bitez *a* eta *b* N digitu dituen bi zenbaki. Beraien arteko biderkadura kalkulatu nahi dugu. Zatitu eta irabazi teknikari jarraituz egin nahi badugu, eragigai bakoitza ahalik eta antzekoen diren bi zatitan banatzea gomendatzen dugu:



$a = X_a\ 10^s + Y_a$ eta $b = X_b\ 10^s + Y_b$ non $0 \leq Y_a, Y_b < 10^s$ eta $s = \lfloor \lfloor n/ \rfloor 2 \rfloor$. Honela, X_a eta X_b osokoek $\lceil n/2 \rceil$ digitu izango dituzte. Berez, gehienez $\lceil n/2 \rceil$ digitu izango lituzke. Baina hemendik aurrera eta hizkera sinplifikatzearren, 10^j baino txikiagoa den zenbaki batek *j* digitu dituela esango dugu, nahiz eta berez, zenbakia 10^{j-1} baino txikiagoa izatea gerta daitekeen.

Aukeratutako adierazpidean aukeratutako oinarria 10 dela eta, kalkulatu nahi dugun biderkaketa ondorengoa da:

$$a\ b = X_a\ X_b\ 10^{2s} + (X_a Y_b + X_b\ Y_a)\ 10^s + Y_a\ Y_b$$

Zatitu eta irabaziren ideia berari jarraituz, beste oinarri bat erabiliko bagenu adierazpidean, adibidez:

```
subtype DIGITUA is INTEGER range 0..OINARRIA-1;
type ZNBK_OSO_HANDIA is array (NATURAL range <>) of DIGITUA;
```

beste ondorengo biderkaketa egin beharko genuke:

$$a\ b = X_a\ X_b\ \text{OINARRIA}^{2s} + (X_a Y_b + X_b\ Y_a)\ \text{OINARRIA}^s + Y_a\ Y_b$$

Algoritmoaren lehenengo hurbilpena:

```
function BIDER (A,B: in ZNBK_OSO_HANDIA) return ZNBK_OSO_HANDIA is
   N,XA,YA,XB,YB,S: ZNBK_OSO_HANDIA;
begin
   N:= DIGITU_KOPURU_MAX(A,B);
   if  TXIKIA_DA?(N)  then  return(BIDERKAKETA_KLASIKOA(A,B))
   else  S:= N / 2;
         XA:= A / (10**S);
         YA:= A rem (10**S);
         XB:= B / (10**S);
         YB:= B rem (10**S);
         return  ( (BIDER (XA,XB) * (10 ** (2*S))) +
                   ((BIDER(XA,YB) + BIDER (XB,YA)) * (10**S) ) +
                     BIDER (YA,YB) );
   end if;
end BIDER;
```

Algoritmoaren hurbilketaren analisia:

$T_t(N)$ N digitu dituen bi zenbakien arteko biderkadura kalkulatzen duen algoritmoaren exekuzio-denbora kasu txarrenean honakoa da:

⇨ Atal honen hasieran oso handiak diren zenbaki osokoak adierazteko azaldutako datu-egituraren adierazpidearen ordena O(N) da, non N zenbakiaren digitu kopurua den. Datu-mota horretako bi eragigaien arteko batuketa eta kenketa algoritmoen ordena $\Theta(N+M)$ da, non N eta M bi zenbakien digitu kopuruak diren. Bestetik aukeratu den adierazpenean hamarren multiploak diren zatiketak eta biderkaketak ere linealak dira.

Emaitza hauek kontutan izanik, BIDER algoritmoan agertzen diren:

– 10^{2s} eta 10^{s} zenbakiekin eginiko biderkaketen eta zatiketen kalkulua (korritze bidez egin daitezkeenak oinarriaren multiploak direnez) lineala da,

– batuketak ere denbora linealean egikari daitezke,

– **rem** eragiketarekin beste hainbeste gertatzen da (*rem* eragiketa zatiketa osoa, biderkaketa eta kenketa eragiketak konbinatuz egin baitezakegu).

⇨ Algoritmoan agertzen den azkeneko aginduan 4 dei errekurtsibo agertzen dira. Dei hauek, gutxi gora-behera N/2 tamaina duten zenbaki osokoen biderkaketa kalkulatuko dute. Ondorioz:

$$T_t(N) = \Theta(N) + 3T_t(\lceil N/2 \rceil) + T_t(\lfloor N/2 \rfloor)$$

Errekurtsio-ekuazio hau 2ren berredura bada ($N= 2^k$)

$T_t(N) = \Theta(N) + 4\ T_t\ (N/2)$

Errekurtsio-ekuazio hori ebatziz (aldagai aldaketaren metodoa erabil beza irakurleak) $T_t(N) \in \Theta(N^2)$.

Algoritmo klasikoa bezala, hau ere koadratikoa da. Ezin al dugu orduan algoritmo hoberik lortu? Bai. Biderkaketen kopurua murriztea komenigarria da batez ere eragigaiak oso handiak direnean, bere egikaritzapenak denbora asko behar baitu. Beraz, BIDER algoritmoan egiten ditugun 4 biderkaketen ordez (X_aX_b, X_aY_b, X_bY_a eta Y_aY_b) saia gaitezen biderkaketa kopuru gutxiago egiten nahiz eta horretarako batuketa eta kenketa kopuru gehiago egin behar.

$$a\ b = X_a\ X_b\ 10^{2s} + (X_aY_b + X_b\ Y_a)\ 10^s + Y_a\ Y_b$$

$$r = (X_a+Y_a) * (X_b+Y_b) = X_a\ X_b + (X_aY_b+X_b\ Y_a) + Y_a\ Y_b$$

r biderkaketa egiten badugu, bertan agertzen diren 3 gaiak *a b* biderkaketan ere agertzen dira. Orain termino bakoitza banandu besterik ez dugu egin behar. Horretarako batuketa eta kenketak aplikatuko ditugu, non hauen kostua ez den biderkaketarena bezain handia.

Algoritmoa:

```
function BIDER (A,B: in ZNBK_OSO_HANDIA) return ZNBK_OSO_HANDIA is
   N, LAG_S:ZNBK_OSO_HANDIA;
begin
   N:= MAX(A,B);
   if  TXIKIA_DA?(N)  then  return(BIDERKAKETA_KLASIKOA(A,B));
   else  S:= N / 2; LAG_S:= (10**S)
         XA:= A / LAG_S;
         YA:= A rem LAG_S;
         XB:= B / LAG_S;
         YB:= B rem LAG_S;
         E:= BIDER(XA+YA, XB+YB);
         F:= BIDER(XA, XB);
         G:= BIDER(YA, YB);
         return( F * (10 ** (2*S))) + ( (E-F-G) * (10**S) ) + G );
   end if;
end BIDER;
```

Algoritmoaren analisia:

Aurreko kasuko arrazonamendua jarraituz gero, eta jakinik X_a+Y_a eta X_b+Y_b batuketek $1+\lceil N/2 \rceil$ digitu behar lituzketela

$$T_t(N) = \Theta(N) + T_t(1+\lceil N/2 \rceil) + T_t(\lceil N/2 \rceil) + T_t(\lfloor N/2 \rfloor)$$

Aurreko errekurtsioa honen parekoa da ordena aldetik:

$$T_p(N) = 3T_p(\lceil N/2 \rceil) + \Theta(N)$$

eta n bikoitia bada:

$$T_p(N) = 3\ T_p(\ N/2\ ) + \Theta(N) = \ 3\ T_p(\ N/2\ ) + N$$

$N=2^k>1$ aldagai ordezkapena eginez ($k= \lg N$):

$$T_p(2^k) = 3\ T_p(\ 2^{k-1}\ ) + 2^k$$

Ekuazioa ebatziz:

$$T_k - 3\ T_{k-1} = 2^k$$

lortzen dugun ekuazio karakteristikoa $(x-3)\ (x-2) = 0$

$$T_k= k_1\ 3^k+ k_2\ 2^k$$

Ordezkapena deseginez eta jakinik $a^{\lg_b c} = c^{\lg_b a}$ berdinketa betetzen dela

$$T_p(N) = k_1\ 3^{\lg N}+ k_2\ 2^{\lg N} + k_3 = \ k_1\ N^{\lg 3}+ k_2\ N^{\lg 2} \in \Theta(N^{\lg 3})$$

Bestetik eta logaritmo hori $1{,}58 \le \lg 3 \le 1{,}59$ tarte honetan dagoenez:

$$T_p(N) \in O\ (N^{1{,}59})$$

N digitu dituzten 2 zenbaki biderkatzen baditugu, bere kalkuluak beharko duen denbora $\Theta(N^{\lg 3})$ ordenakoa izango da, eta hori aldi berean $O(N^{1{,}59})$-ren ordenakoa izango da. Zein da benetan azkarragoa algoritmo klasikoa ala BIDER algoritmoa? Zenbakiak oso handiak direnean soilik izango da azkeneko algoritmoa azkarragoa (exekuzio-denboraren ekuazioan inplizituki dagoen konstantearen ondorioz).

ARIKETAK: ZATITU ETA IRABAZI

1- Taula bateko osagai maximoa eta minimoa bilatzen dituen algoritmo sinple batek bere osagaien artean 2n-2 alderaketa egiten ditu. Aurkitu bedi ((3/2) n-2) alderaketa egingo dituen eta zatitu eta irabazi teknika jarraitzen duen algoritmoa.

2- n zenbaki oso desberdin ordenaturik dituen T taula bat emanik, T(k)=k gertatzen bada k indizea itzuliko duen algoritmo eraginkor bat idaztea eskatzen da (lineala baino hobea). (Baldintza hori betetzen duen posiziorik egongo ez balitz, algoritmoak 0 itzuli beharko luke).

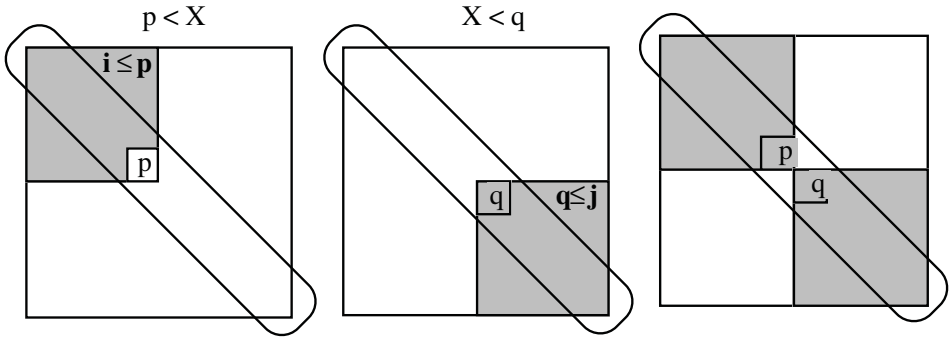
3- Zenbaki osoak ordenaturik metatzen dituzten bi taula emanik, k. osagai txikiena zein den mugatuko duen algoritmoa idaztea eskatzen da. Algoritmoaren ordenak min(k,n-k) kopuruan logaritmikoa izan behar du.

4- $a_1,a_2,...,a_n$ osokoen sekuentzia taula batean metaturik dago. Gainera, taulako edozein bi posizio jarraitan metaturik dauden sekuentziako osagaien balioak gehienez unitate batean desberdinak direla ezaguna da. Sekuentzian osagai jakin bat, X, ote dagoen eraginkorki mugatuko duen algoritmoa idaztea eskatzen da.

5- Osoen taula bat emanik, diseina bedi taulako batura maximoa ematen duen segmentua kalkulatzen duen algoritmoa. Bere denbora-ordenak $o(n^2)$-koa izan beharko du.

6- Izan bedi T(1..n,1..n) errenkadetako eta zutabeetako osagaiak ordena gorakorra jarraituz ordenaturik dituen matrize bat (ezkerretik eskuinera eta goitik behera). X osagaia T matrize horretan dagoen ala ez mugatzeko honako algoritmoa jarraitzen da:

   – Bilaketa dikotomiko bidez X diagonalean bilatzen da.
   – Baldin eta X aurkitzen badugu, ordena amaituko dugu.
   – Bestela, (p eta q aurkitzen ditugu non $p<X<q$) X egon ezin daitekeen bi matrize zatiak baztertzen ditugu eta errekurtsiboki prozesu bera gelditzen diren beste bi zatitan aplikatzen da.



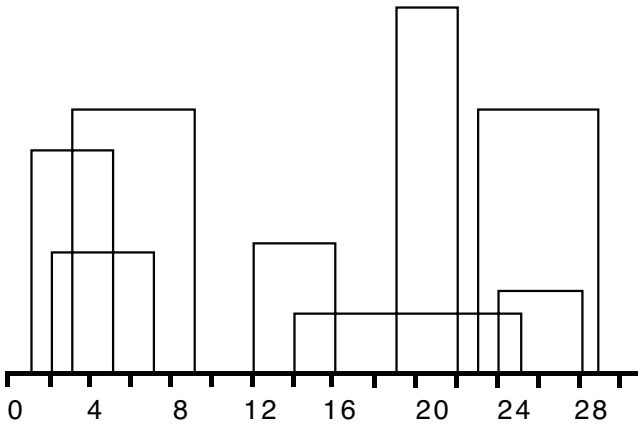
Zein da algoritmoaren ordena kasu txarrenean?

7- Skyline (poligonoen batura).

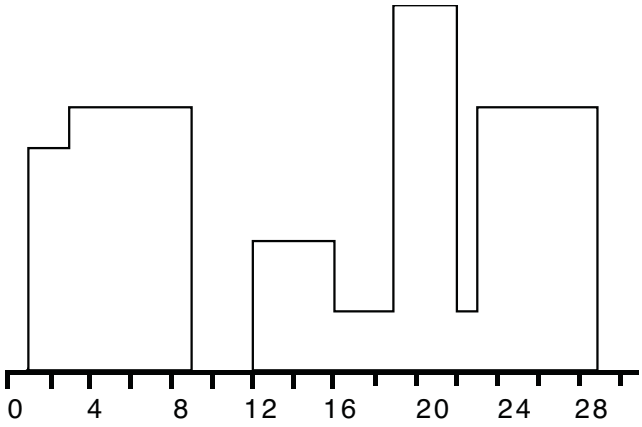
Hiri bateko eraikuntzen posizio eta figura zehatzak emanik, ezkutuan gelditzen diren marrak ezabatuz zeruko perfila bi dimentsiotan kalkulatuko duen algoritmoa idaztea eskatzen da

Ad.:

Sarrera: (1,**11**,5),(2,**6**,7),(3,**13**,9),(12,**7**,16),(14,**3**,25), (19,**18**,22), (23,**13**,29) y (24,**4**,28).



Irteera: (1,**11**,3,**13**,9,**0**,12,**7**,16,**3**,19,**18**,22,**3**,23,**13**,29,**0**)



8- Izan bedi T(1..N) zenbakien bektore bat. Kard({i| T(i)=X})> n/2 propietatea betetzen duen X osagaia mugatuko duen algoritmoa idaztea eskatzen da; hau da, n/2 aldiz edota maizago agertzen den X osagaia T-n. Halere T-ko N osagaien artean propietate hori betetzen duen X osagairik ez egotea gerta liteke.

6. GAIA

“PROGRAMAZIO DINAMIKOA” TEKNIKAREN BIDEZKO ALGORITMOAK

6.1. SARRERA

Goitik beherako diseinua naturala eta indartsua da. Hasieran, problema bere osotasunean aztertzen da eta, ondoren, hau zehazten joan behar da; hau da, konplexutasun handiko arazoa, azpiarazotan banatzen da. Errekurtsioa erabiliz, maiz mota bateko arazo handi bat mota bereko baina tamaina txikiagoa duten azpiarazoak (instantziak) ebazten garatu dezakegu. Adibidez, zatitu eta irabazi teknikak estrategia hau erabiltzen du. Ikusi den bezala teknika honi esker sailkapen algoritmo azkarrak lor ditzakegu. Baina, errekurtsioak murriztapenak ditu: zatitu eta irabazi teknikak azpiarazo bakoitza independenteki ebazten du eta honek askotan alferrikako lana eginarazten du.

Programazio dinamikoaren estrategia jarraitzen duen algoritmo batek bitarteko emaitzak biltegiratzen ditu: azpiproblemen emaitzak. Ondoren, tamaina handiagoko instantziak ebazteko aurrez kalkulatutako eta biltegiratutako emaitzak bilatzen eta konbinatzen ditu *berriz ere kalkulatu ordez*. Honela, azpiproblemen emaitzak konbinatuz problema beraren baina tamaina handiagoa duten instantzien emaitzak lortzen ditu. Prozesu bera jatorrizko instantziaren emaitza lortu arte errepikatzen du. Hau dela eta, problema bat ebazteko problema beraren azpiproblema asko behin baino gehiagotan ebatzi behar den kasurako programazio dinamikoaren teknika guztiz aproposa gertatzen da. Hartaz, programazio dinamikoa goranzko metodoa da, eta algoritmo jaleen teknika berriz, beheranzko metodo bat. Halere, programazio dinamikoa memoriadun funtzioekin inplementatzen denean, beheranzko diseinua erabiliko dugu. Hau gaiaren amaieran azalduko dugu.

Programazio dinamikoaren teknika maiz optimizazio problemetan erabiltzen da. Problema horiek soluzio bat baino gehiago eduki dezakete. Soluzio bakoitzari balio bat dagokio eta kalkulatu nahi dena soluzio guztien artetik soluzio onena optimoa da: balio maximoa ala balio minimoa. Soluzio hori, soluzio optimo *bat* da, soluzio optimo bat baino gehiago izan baititzake problemak. Teknika honen bidez ebatz daitezkeen [optimizazio] problemen ezaugarri komunak ondorengo ataletako adibideetan ikusi ahal izango dira.

Programazio dinamikoaren ezaugarri nagusien zerrenda:

– Problemak indukzio bidez definigarria izan behar du.

– Azpiproblema desberdin bakoitzari egitura batean posizio bat egokitzen zaio.

– Azpiproblema bat aztertzean:

• Hau aztertzen den lehenengo aldia bada: azpiproblemaren soluzioa indukzio bidezko ekuazioak dien moduan kalkulatzen da eta soluzioa datu-egituran azpiproblemari dagokion posizioan metatzen da.
• Hau lehenago aztertua izan bada: datu-egituran erlazionaturik duen posizioan dagoeneko azpiproblemaren soluzioa metatuta dagoela eta, handik jaso besterik ez da egiten.

– Ondorioz, azpiproblema desberdin bakoitza BEHIN bakarrik kalkulatzen da.

-Programazio dinamikoko teknika desberdinak datu-egitura betetzeko ordenagatik ezaugarritzen dira.

Programazio dinamikoa eta optimizazio problemak: ezaugarri komunak:

⇨ *Azpiegitura optimoa (= indukzio bidez definigarria)*

Problema batek azpiegitura optimoa duela esango dugu baldin eta problemaren soluzioak azpiproblemen soluzio optimoak jasotzen baditu.

Problema batek propietate hori betetzen duenean problema programazio dinamikoaren teknika erabiliz ebatzi liteke.

⇨ *Azpiproblemen teilakapena*

Algoritmo errekurtsibo batek behin eta berriro azpiproblema bera ebazten duenean, optimizazio problemak teilakatzen diren azpiproblemak dituela esan ohi da.

Planteamendu horri etekina ateratzen diote programazio dinamikoa jarraituz diseinaturiko algoritmoek: teilakatzen diren azpiproblema desberdin bakoitza *behin* bakarrik ebatziko dute, ondoren emaitza hori nonbait gordez. Aurrerantzean, azpiproblema baten soluzioa behar dutenean, aurrez kalkulatua izan ote den aztertzen dute. Hala balitz, soluzioaren emaitza denbora konstante batean atzituko lukete (eta bestela lehenengo aldiz kalkulatuko lukete.)

6.2. FIBONACCI-ZENBAKIAK

Fibonacci-zenbakiak aurreko puntuan azaldutako ideiaren eredu sinple eta argia ditugu.

Fibonacciren *n*. zenbakia kalkulatzeko aurreko bi Fibonacci-zenbakien batura kalkulatu behar da (indukzio bidezko definizioa):

$$F(n) = \begin{cases} 0 & n = 0 \\ 1 & n = 1 \\ F(n-1) + F(n-2) & n > 1 \end{cases}$$

6.2.1. Fibonacci-zenbakien lehenengo hurbilketa

Fibonacciren zenbakiak errekurtsiboki definitzen direnez gero, modu naturalena funtzio errekurtsiboen bidez kalkulatzea da. Baina algoritmo errekurtsiboa oso ez-eraginkorra da alferrikako lan asko egiten baitu; eragiketa asko birkalkulatzen du. F_n kalkulatzeko egin behar den lana (egiten diren dei errekurtsiboen kopurua neurtzen bada, adibidez) esponentziala da:

$$\Theta\left(\left(1+\sqrt{5}\right)/2\right)^n\Big)$$

Ez al dago Fibonacci-zenbakiak kalkulatzen dituen algoritmo azkarragorik?

6.2.2. Fibonacci-zenbakien bigarren hurbilketa

Estrategiaz alda dezagun. Fibonacciren *n*. zenbakiaren kalkulua, F_n, "bukaeratik" hasita eta *n-1* bira eginez kalkula dezakegu: 0 eta 1 balioetatik hasi eta aurreko bi Fibonacci-zenbakien batura eginez ondorengo Fibonacci-zenbakia lortzen da. Prozesu bera *n-1* aldiz errepikatuz *n*. zenbakia lor dezakegu:

– Taula bat erabiliz aurrez kalkulatutako balioak metatzeko: MEE(n) $\in\Theta$(n)

```
function FIBONACCI2 (N: in NATURAL) return NATURAL is
    F: TAULA(0..N);
begin
    F(0):= 0; F(1):= 1;
    for K in 2..N loop
          F(K):= F(K-2)+F(K-1);
    end loop;
    return F(N);
end FIBONACCI2;
```

– Taula bat erabili ordez, nahikoa dira bi aldagai erabiltzea: MEE(n) $\in \Theta(1)$

```
function FIBONACCI2 (N: in NATURAL) return NATURAL is
    I,J: NATURAL;
begin
    I:= 1; J:= 0;     – F0=J
    for K in 1..N loop
          J:= I+J;
          I:= J-I;    – Fk=J
    end loop;
    return J;         – Fn=J
end FIBONACCI2;
```

Batuketa oinarrizko eragiketa dela kontsideratzen badugu, orduan algoritmo honen egikaritzapenak behar duen denboraren ordena lineala da. Eta honek, algoritmo errekurtsiboak behar duen denboraren ordenarekin alderatuz, denbora irabazi handia suposatzen du.

Baina, ba al dago beste algoritmorik Fibonacci-zenbakiak kalkulatzen dituena eta oraindik azkarragoa dena?

6.2.3. Fibonacciren beste hurbilketa bat

Matrize karratu jakin baten *n*. berredura kalkulatuz eta ondoren F_0 eta F_1 balioak matrize honekin konbinatuz, *n*. zenbakia lortzeko behar den denboraren ordena logaritmikoa dela froga dezakegu. Azter dezagun beraz beste teknika hau:

$$\begin{pmatrix} F_n \\ F_{n+1} \end{pmatrix} = A \begin{pmatrix} F_{n-1} \\ F_n \end{pmatrix}$$

Jakinik: $F_{n+1} = F_{n-1} + F_n$ eta $F_n = F_n$ baldintzak bete behar direla, A = ?

$$A = \begin{pmatrix} 0 & 1 \\ 1 & 1 \end{pmatrix}$$

$$\begin{pmatrix} F_n \\ F_{n+1} \end{pmatrix} = \begin{pmatrix} 0 & 1 \\ 1 & 1 \end{pmatrix} \begin{pmatrix} F_{n-1} \\ F_n \end{pmatrix} = \begin{pmatrix} 0 & 1 \\ 1 & 1 \end{pmatrix}^2 \begin{pmatrix} F_{n-2} \\ F_{n-1} \end{pmatrix} = \quad = \begin{pmatrix} 0 & 1 \\ 1 & 1 \end{pmatrix}^n \begin{pmatrix} F_0 \\ F_1 \end{pmatrix}$$

Programazio dinamikoaren teknika non aplika dezakegu hemen? Matrizearen *n*. berredura kalkulatu nahi dugu, eta horretarako ez dugu kalkulurik errepikatu nahi. Aldiz, aurrez lortutako emaitzak berrerabili nahi ditugu. Adibide bat ipiniz azalduko dugu.

A^n-ren kalkulua programazio dinamikoaren teknika erabiliz:

Suposa dezagun n=23 dela eta A^n kalkulatu nahi dugula programazio dinamikoaren bidez, horretarako egin beharko liratekeen kalkuluak hauek dira:

• 23 zenbakia 2 oinarrian: $23 = 1\,2^4 + 0\,2^3 + 1\,2^2 + 1\,2^1 + 1\,2^0 \rightarrow 23_2 = 10111$

• Ondorengo berreketak: A^1, A^2, A^4, A^8, $A^{16} = A^8 * A^8$
A^{2^0}, A^{2^1}, A^{2^2}, A^{2^3}, $A^{2^4} = A^{\lfloor \lg n \rfloor}$

• A-ren berreketak aurreko berreketak berrerabiliz lortu dira eta hauek konbinatuz lortzen da emaitza:

$$A^n = 1 * A^1 + 1 * A^2 + 1 * A^4 + 0 * A^8 + 1 * A^{16}$$

Fibonacciren zenbakiak kalkulatzen dituen hirugarren hurbilketak ideia bera azaltzen du. Algoritmoak darabiltzan aldagai nagusienak:

– MAT matrizean A matrizearen berreketak kalkulatzen dira,

– SOL aldagaiak erdibideko soluzioa metatuko du: begizta bakoitzean kalkulatutako A matrizearen berreketa, *i*.a, SOLi erantsiko dio baldin eta *n* zenbakiaren 2 oinarriko adierazpeneko *i*. bita =1 den (2 oinarriko bitak lortzeko **rem** eta **/** eragileak konbinatu besterik ez da egin behar.)

```
function FIBONACCI3 (N: in NATURAL) return NATURAL is
   MAT,SOL: MATRIZEA(1..2,1..2); X:
begin
   MAT:= HASIERATU_MAT_A_BEZALA;
   SOL:= HASIERATU_MAT_0_RA; X:= N;
   for K in 0..⌊lg N⌋ loop
         if  X rem 2 /= 0  then  SOL:= BATU(SOL, MAT);
         X := X / 2
         MAT:= KARRATU(MAT);
   end loop;
   if  X rem 2 /= 0  then  SOL:= BATU(SOL, MAT);
   return(SOL(1,2));
end FIBONACCI3;
```

Aurreko algoritmo honen ordena logaritmikoa dela begi-bistakoa da: $\Theta(\lg N)$. Memoria-espazio estra aldetik, aldiz, MAT eta SOL matrize karratuak erabiltzen ditu,(2x2). Ondorioz, $MEE(n) \in \Theta(1)$.

6.3. KONBINAZIO-ZENBAKIEN KALKULUA

n zenbaki izanik, hauek *k*-naka hartuko bagenitu lortuko genituzkeen konbinazio desberdinen kopurua kalkulatzen duen funtzioa egin nahi dugu. Hau da, konbinazio-zenbakien kalkulua egin nahi dugu. Indukzio-bidezko definizioa hau da:

$$\binom{n}{k} = \begin{cases} \binom{n-1}{k-1} + \binom{n-1}{k} & 0 < k < n \\ 1 & k = 0 \vee k = n \end{cases}$$

6.3.1. Konbinazio-zenbakien kalkuluaren lehenengo hurbilketa

Konbinazio-zenbakien kalkulua egiten duen lehenengo hurbilketa, besterik gabe definizioa zuzenean aplikatuta ateratzen da:

```
function KONBINAZIO_ZNBK1(N,K: in NATURAL) return NATURAL is
begin
    if  K=0  or  N=K  then  return 1;
    else
          return
                KONBINAZIO_ZNBK1(N-1,K-1)+
                KONBINAZIO_ZNBK1(N-1,K));
    end if;

end KONBINAZIO_ZNBK1;
```

Gogora beza irakurleak konbinazio-zenbakiak definiturik dauden moduagatik hauek kalkulatzeko funtzioaren parametro formalen balioek ondorengo baldintza hauek bete behar dituztela: $(0 < N)$ eta $(0 \le K \le N)$.

Algoritmo honekin Fibonacciren lehenengo hurbilketan gertatzen zen gauza bera gertatzen da. KONBINAZIO_ZNBK1(i,j) balioak behin baino gehiagotan kalkulatzen dira i<N eta j<K balioentzat. Emaitza 1 zenbakia zenbait aldiz batuz lortzen da. Egiten diren deien kopurua oso handia da. Kopuru zehatza ondorengo ekuazioak ematen digu:

$$T_d(n,k) = \begin{cases} 2 + T_d(n-1,k-1) + T_d(n-1,k) & 0 < k < n \\ 0 & n = k \text{ edo } k = 0 \end{cases}$$

Algoritmo honek egiten dituen dei errekurtsiboen kopurua $\Omega\left(\binom{n}{k}\right)$ ordenakoa izan daitekeelako tankera eman badiezaiokegu ere, benetan bere ordena

zehatza $\Theta\left(2\binom{n}{k}-2\right)$ dela indukzio-bidez froga dezakegu. Beraz, frogatuko duguna honako hau izango da:

$$\forall n \forall k \left(0 \le k \le n \Rightarrow T_d(n,k) = 2\binom{n}{k} - 2\right)$$

Frogaketa *n*-ren eta *k*-ren gaineko indukzio-bidez egingo dugu, (lehendabizi, *n*-ren gain).

• Kasu nabaria:

$$n = 0 \rightarrow T_d(0,0) \overset{'\Delta':T_d}{=} 0 = 2\binom{0}{0} - 2$$

• Indukzio-hipotesi hedatua:

$$\forall t \forall n \forall k \left(0 \le k \le t \le n \Rightarrow T_d(n,k) = 2\binom{n}{k} - 2\right)$$

• Kasu orokorra:

$$\forall k \left(0 \le k \le n+1 \Rightarrow T_d(n+1,k) \overset{?}{=} 2\binom{n+1}{k} - 2\right)$$

Kasu desberdinak aztertu behar dira. Hori k-ren gainean indukzioa aplikatuz frogatuko dugu:

1- Kasu nabaria k=0 :

$$T_d(n+1,0) = 0 = 2\binom{n+1}{0} - 2$$

2- Indukzio-hipotesia:

$$\forall r \left(0 \le r \le k \Rightarrow T_d(n+1,r) = 2\binom{n+1}{r} - 2\right)$$

3- Kasu orokorra:

$$T_d(n+1,k+1) \overset{?}{=} 2\binom{n+1}{k+1} - 2$$

• Baldin k+1=n+1:

$$T_d(n+1,n+1) = 0 = 2\binom{n+1}{n+1} - 2$$

• Bestela, (k+1)<(n+1) (eta beraz, k+1≤n):

$$T_d(n+1,k+1) = 2 + T_d(n,k) + T_d(n,k+1) \overset{i.h.}{=} 2 + 2\binom{n}{k} - 2 + 2\binom{n}{k+1} - 2$$

$$= 2\left(\binom{n}{k} + \binom{n}{k+1}\right) - 2 = 2\binom{n+1}{k+1} - 2$$

Beraz, $T_d(n,k) = 2\binom{n}{k} - 2$ f.n.g.b

6.3.2. Konbinazio-zenbakien kalkuluaren bigarren hurbilketa

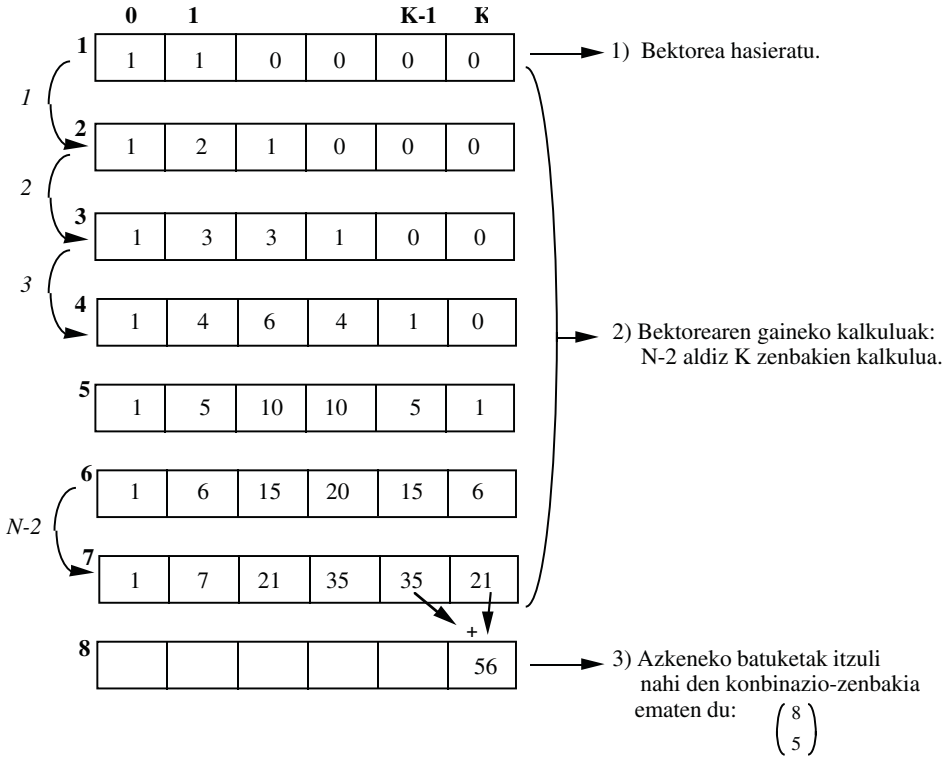
Prozesua:

Konbinazio-zenbakien kalkulu-denboran hobekuntzarik lortu nahi badugu, lehenengo pausoa erabiltzen ditugun zenbakiak aztertzea da:

	0	1	2	3	...	K-1	K
0	1						
1	1	1					
2	1	2	1				
3	1	3	3	1			
...					...		
N-1					T(N-1,K-1)	T(N-1,K)	
N						+ T(N,K)	

Aurreko irudiak matrize itxura du. Bi dimentsioko matrize batean biltegiratu ditzakegu, baina nola lortu zenbakiok? Azter ditzagun polikiago zenbakiok beraien artean erlazioren bat aurkitzeko asmoz. Alde batetik, matrizean triangelu bat ikus daiteke eta bestetik M(i,j) posizioko zenbakiaren balioa M(i-1,j-1), M(i-1,j) posizioetako balioen batura da: M(i,j) = M(i-1,j-1) + M(i-1,j). Beraz, triangelua *Pascalen triangelua* dugu. Inplementa dezagun orduan Pascalen triangelua matrize batean kalkulatzen duen algoritmo bat eta, ondoren, M(n,k) posizioko balioa itzul dezagun. Are gehiago, berez ez dugu matrize baten beharrik; nahikoa izango da *k* luzera duen taula bat erabiltzea eta taulako osagaiak eskuinetik ezkerrerantz eguneratzen joatea.

Adibide bat: $\binom{8}{5} = \frac{8!}{5!(8-5)!} =$ *56 ren kalkulua.*



Algoritmoa:

```
function  KONBINAZIO_ZNBK2  (N, K: in NATURAL) return NATURAL;
   T: BEKTOREA;
begin
   if  K=0 or K = N  then  return 1;
   else
         HASIERATU_TAULA(T,K);                    — 0 < K ≤ N
```

```
            for I in 1..N-2  loop
                  for  J  in reverse 1..K  loop
                        T(J):= T(J-1)+T(J);
                  end loop;
            end loop;
            return T(K-1)+T(K);
      end if;
end KONBINAZIO_ZNBK2;
```

Algoritmo honek kalkulatu nahi genuen konbinazio-zenbakia itzultzen du eta aurreko puntukoa baino eraginkorragoa da.

Algoritmoaren exekuzio- eta memoria-ordena:

Algoritmo honek k luzera duen taula bat erabiltzen duenez: $\Theta(K)$ ordenako memoria-espazioa behar du. Bestetik, batuketa oinarrizko eragiketa dela kontsideratzen bada, orduan gure algoritmoaren ordena zehatza NK-koa da: $\Theta(NK)$.

"Programazio dinamikoa" teknika, azpiegitura optimoa propietatea betetzen duten optimizazio problemak ebazteko erabiltzen da. Hona hemen adibide batzuk:

6.4. MATRIZE SEKUENTZIAK BIDERKATZEKO FAKTORKETA-ORDENA OPTIMOAREN KALKULUA

Demagun matrizez osaturiko sekuentzia bat izanik, bere osagaien arteko biderkadura kalkulatu nahi dugula matrize bakar bat lortzeko eta matrizeko osagaien arteko biderkaketa kopuru txikiena eginaz.

Matrizeen arteko biderkadura kalkulatzeko, bi matrize biderkatzen dituen ohizko algoritmoa erabiliko da. Honela, P_*Q eta Q_*R dimentsioak dituzten bi matrize biderkatzerakoan, egiten diren osagaien arteko biderkaketen kopurua P_*R da.

```
procedure BI_MATRIZE_X (M1,M2: in MATRIZEA; M3: in out MATRIZEA) is
begin
    for  I  in  1..P  loop
          for  J  in  1..R  loop
                M3(I,J):= 0;
                for  K  in  1..Q  loop
                      M3(I,J):= M3 (I,J) + M1 (I,K) * M2 (K,J);
                end loop;
          end loop;
    end loop;
end BI_MATRIZE_X;
```

Egiten diren biderkaketen kopurua neurtzea, algoritmoak egiten duen lanaren adierazgarri ona denez, algoritmoaren ordena zehatza $\Theta(PQR)$ da.

Baina bi matrize baino gehiago biderkatu nahi ditugunean, matrizeen biderkaketak elkartze-propietatea betetzen duenez gero, honen kalkulua faktorketa-ordena desberdinak jarraituz kalkula dezakegu. Baina biderkaketa bera kalkulatzeko faktorketa-ordena bat edo beste jarraitzeak sekulako garrantzia du, egin beharko diren osagaien arteko biderkaketa guztien kopurua nabarmenki aldatzen da eta.

Adibidea:

Demagun bada A, B, F eta G matrizez osaturiko sekuentzia bat non bere dimentsioak (30,1), (1,40), (40,10) eta (10,15) diren hurrenez hurren. Gure helburua matrize hauen arteko biderkadura kalkulatzea da:

$$\begin{array}{cccccccc} A & * & B & * & F & * & G \\ 30*1 & & 1*40 & & 40*10 & & 10*25 \end{array}$$
$$\Downarrow$$
$$30 * 25$$

Matrizeen biderkaketak elkartze-propietatea betetzen duenez, biderkaketa hainbat faktorketa-ordena desberdin jarraituz kalkula dezakegu. Ondorengo kalkuluek, jarrai dezakegun faktorketa-ordena bakoitzarentzat egin beharko liratekeen osagaien arteko biderkaketen kopuruak ematen dizkigute:

(((AB) F) G)	$\Rightarrow$	30 1 40	+	30 40 10	+	30 10 25	=	20.700
(A (B (FD)))	$\Rightarrow$	40 10 25	+	1 40 25	+	30 1 25	=	11.750
((AB) (FG))	$\Rightarrow$	30 1 40	+	40 10 25	+	30 40 25	=	41.200
(A ((BF) G))	$\Rightarrow$	1 40 10	+	1 10 25	+	30 1 25	=	<u>1.400</u>
((A (BF)) G)	$\Rightarrow$	1 40 10	+	30 1 10	+	30 10 25	=	8200

Atal honetan, matrize sekuentzia bat biderkatzeko faktorketa-ordena optimoa (biderkaketa kopuru minimoa emango duena) zein den mugatuko dugu. Ondoren, bere ordenaren analisia egingo dugu.

Suposa dezagun M_1, M_2,...,M_n matrize-sekuentzia dugula, non M_i matrizearen dimentsioa $D_{i-1}*D_i$ den. Zein faktorketa-ordena jarraituz biderkatu beharko genituzke matrizeak matrizeen osagaien arteko biderkaketa kopurua ahalik eta txikiena izan dadin?

$$M_1 * M_2 * \ldots * M_n$$

Lehendabizi, biderkaketa kopuru minimoaren aurkikuntzari ekingo diogu, FAKTORKETA_OPTIMOA eta, ondoren, kopuru minimo hori lortzen duen algoritmoa eraikiko dugu, SAILKAPENAREN_ERAKUSKETA.

Zein teknika jarraituko du gure algoritmoak? Biderkaketa asko egin behar ditugu; zergatik ez, hortaz, gure helburua azpiproblematan zatitu? Hau da, zatitu eta irabazi teknika jarrai dezagun, honela errekurtsioa erabiliz, tamaina txikiagoa duten instantziak ebatziz kalkulatuko dugu soluzioa.

Problemaren (i,j) azpiproblemak, jarraian dauden Mi,...,Mj matrize sekuentzia biderkatzeko egin litezkeen biderkaketa kopuru txikiena zein den kalkulatuko du. Defini dezagun MBM funtzio errekurtsiboa:

MBM(i,j)= $1 \le i \le j \le n$ -rentzat $M_i * M_{i+1} * \ldots * M_j$ matrizeen biderkaketa egitean egin beharko liratekeen osagaien arteko biderkaketen kopuru *minimoa* kalkulatzen duena.

Suposa dezagun matrizeak honela faktoratu ditugula:

$$(M_i * M_{i+1} * \cdots * M_k) * (M_{k+1} * \cdots * M_j)$$

Faktoratzean lortzen ditugun bi matrizeen dimentsioak $D_{i-1}*D_k$ eta D_k*D_j dira. MBM-ri dei errekurtsiboak eginez, faktore bakoitzerako beharko liratekeen biderkaketen kopuruak lortuko genituzke. Baina zein da *k* indizeak har dezakeen balio guztien artetik aukerarik onena? Matrize-sekuentzia banatzeko punturik egokiena zein den ez dakigunez gero, posibleak diren aukera guztietatik txikiena aukeratu beharko dugu. Hori, MBM-ren gainean definiturik dagoen ondorengo errekurtsio honen bidez egin dezakegu.

- $MBM(i,j) = \min_{i \le k \le j-1} \{MBM(i,k) + MBM(k+1,j) + D_{i-1}D_kD_j\}$
  non $1 \le i < j \le n$

- $MBM(i,i) = 0$

Formula honetan, MBM(i,k)-k ($M_i * M_{i+1} * \ldots * M_k$) kalkulatzeko egin beharko liratekeen matrize-osagaien arteko biderkaketen kopuru minimoa adierazten du (faktorketa posible guztiak kontutan izanik) eta MBM(k+1,j)-k ($M_{k+1} * \ldots * M_j$) kalkulatzeko egin beharko liratekeen matrize-osagaien arteko biderkaketen kopuru minimoa adierazten du (faktorketa posible guztiak ere kontutan edukiz). Azkeneko bi matrizeen arteko biderkaketa egiteko, adibidez, ($M_i * M_{i+1} * \ldots * M_k$) + ($M_{k+1} * \ldots * M_j$) + $D_{i-1}*D_k*D_j$ biderkaketa egin beharko dira. *k*-ren balio posible guztiak kontsideratuz (i baliotik, j-1 baliora arte) balio txikiena ematen duena jaso behar da MBM(i,j)-n. MBM(i,i)-k kalkulatzeko egin behar diren biderkaketen kopuru minimoa kalkulatuko du, eta kopuru hori 0 da.

MBM errekurtsioa funtzio baten bidez adieraztea erraza litzateke. Baina MBM(1,n) deiaren egikaritzapenak denbora asko hartuko luke, guztiz ez-eraginkorra litzateke eta. Ikus dezagun adibide sinple batez, errepikatzen den lan kopuruaren magnitudea:

$$(M_i * \cdots * M_k) * [\ (M_{k+1} * \cdots * M_{k'}) * (M_{k'+1} * \cdots * M_j)\ ]$$
$$[\ (M_i * \cdots * M_k) * (M_{k+1} * \cdots * M_{k'})\ ] * (M_{k'+1} * \cdots * M_j)$$

MBM(k+1,k')-ren kalkulua MBM(k+1,j) eta MBM(i,k') kalkuluen egikaritzapen bietan ebatzi beharko den azpriproblema da. Aldi berean, beste problema askoren egikaritzapenetan ebatzi beharko den azpiproblema da. Ondoren, MBM-ren funtzioa zuzenean inplementatuko bagenu, egin beharko liratekeen dei errekurtsiboen kopurua esponentziala izango litzateke.

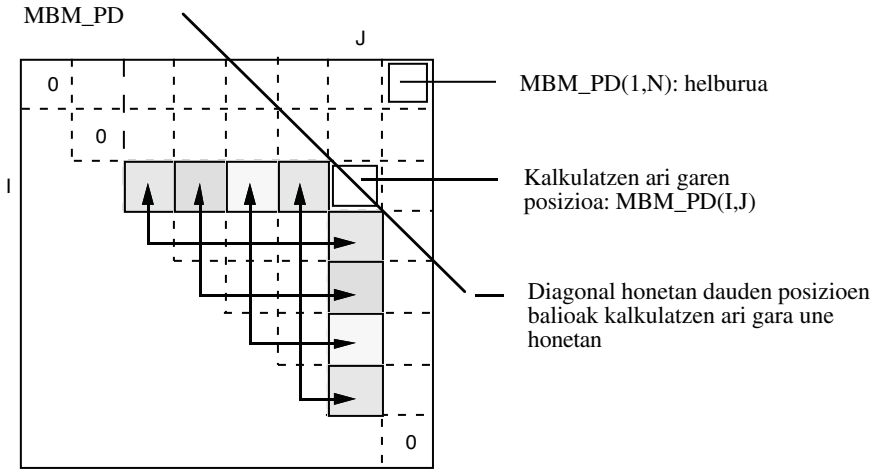
Zatitu eta irabazi teknikak emaitza onak ematen ditu jatorrizko problema ebazten duten azpiproblemak ezagunak direnean eta hauen kalkulua errepikatzen ez denean.

Gure kasuan MBM(k+1,k') azpiproblema maiz birkalkulatzen da. Alferrikako lana ez egitearren, gorde ditzagun azpiproblemen emaitzak. Baina, zenbat matrize--biderkatze azpiproblema dugu? MBM(i,j) $1 \le i \le j \le n$ balioen konbinazio desberdin guztientzat ‘bakarrik’ egikarituko dugu. Beraz, n^2 baino egikaritzapen gutxiago egingo dugu.

Hemendik aurrera, MBM errekurtsiboa den funtzioa bezala ikusi ordez, $N*N$ dimentsioa duen taula bezala ikusiko dugu. MBM-k osatzen duen matrizearen eskuineko goi-triangelua aztertuko dugu (ezkerreko behe-triangelua ez dugu ezertarako erabiliko). Eskuineko goi-triangeluko posizio bakoitzak honako hau adieraziko du:

$1 \le i \le j \le n$ -rentzat MBM_PD(i,j)-k $M_i * M_{i+1} * \ldots * M_j$ matrizeen biderkaketa egiterakoan egin beharko liratekeen osagaien arteko biderkaketen kopuru *minimoa* metatzen du. Balio hori minimoa denez, optimoa da.

MBM_PD(i,j) kalkulatzeko, k-ren balio posible bakoitzeko, MBM_PD(i,k)-k eta MBM_PD(k+1,j)-k zenbat balio duen ezaguna izan behar du. Hauek, *i*. errenkadan *j*. zutabearen ezkerraldean eta *j*. zutabean *i*.go errenkadatik beherantz (*i*.a baino handiagoak diren errenkadak) dauden matrize-posizioak dira (ikus bedi irudia).



MBM_PD(i,j)-ren kalkulua. Kalkulu hori egiteko, posible den *k*-ren balio bakoitzeko behar diren MBM_PD(i,k)-ren eta MBM_PD(k+1,j)-ren balioak, bi punta dituen gezi baten bidez adierazirik datoz. Bikote bakoitzak faktorketa posible bat adierazten du.

Taulako balioak, MBM_PD(a,b), diagonalez-diagonal kalkulatzen dira, kalkulatuko den lehenengo diagonala, hain zuzen, matrizearen diagonal nagusiaren gainean dagoena izanik. Jarraian, banan-bana goiko eskuin ertzeraino kalkulatuko dira.

Gure jatorrizko helburua, matrize-sekuentzia bat biderkatzeko faktorketa--ordena optimoaren muga zen, ez faktorketa-ordena horretan egingo liratekeen biderkaketen kopurua. Hori dela eta, matrize-azpisekuentzia faktoratzeko modurik egokiena zein den mugatzen dugunean, faktorketa hori lortzen duen K-ren balioa gordeko dugu. Honela, MBM_PD(i,k) + MBM_PD(k+1,j) + $D_{i-1}D_kDj$ minimoa egiten duen K-ren balioa FAKTORKETA izeneko beste bi dimentsioko taularen (i,j) posizioan gordeko dugu: hots, FAKTORKETA(i,j) = k.

Adibidearen jarraipena:

Ikus dezagun adibide baten bidez faktorketa-ordena optimoa nola kalkulatzen duen algoritmoak. Atal honen hasieran ikusi dugun A, B, F eta G matrizeen adibidean, dimentsioen taula ondorengo hau litzateke

D

30	1	40	10	25

eta FAKTORKETA_OPTIMOA-k berriz, beste bi taula sortuko lituzke, hots, BIDERKETAK eta FAKTORKETA.

Faktorketa-ordena optimoaren kalkulua egiterakoan kalkulatu beharko genituzkeen matrize-osagaien arteko biderkaketen kopurua 1.400 izango litzateke: MBM(1,4)=1.400.

Algoritmoa:

```
procedure  FAKTORKETA_OPTIMOA (D: in OSOEN_TAULA;
                               BIDERKAKETAK out INTEGER;
                               FAKTORKETA: out MATRIZEA)  is
    MBM_PD: MATRIZEA; J,K: INTEGER;
    N: INTEGER:= D'LAST;
    J: INTEGER;
begin
    for I  in  1..N  loop
         MBM(I,I):= 0;
    end loop;
    for DIAGONALA  in  1..N-1  loop
         — lehenengo diagonala, 1, matrizearen diagonal nagusiaren
         — goi aldeko alboan dagoena da.
         for I  in  1..N-DIAGONALA  loop
               J:= I+DIAGONALA;
               MBM_PD(I,J):= min I≤K≤J-1 { MBM_PD(I,K)+
                                           MBM_PD(K+1,J)+
                                           D(I-1)*D(K)*D(J) };
               FAKTORKETA(I,J):= K;
         end loop;
    end loop;
    BIDERKAKETAK:=MBM_PD(1,N);
end FAKTORKETA_OPTIMOA;
```

Azpiegitura optimoaren propietatea betetzen al du? Benetan, soluzio optimoa itzultzen du? Indukzio bidezko definizioa zuzena da, zeren:

• MBM_PD(I,K)-k eta MBM_PD(K+1,J)-n $M_i * ... * M_k$ eta $M_{k+1} * ... * M_j$ biderkaketak egiteko egin beharko liratekeen osagaien arteko biderkaketa kopuru *minimoak* adierazten dute.

• I-1*K eta K*J dimentsioko bi matrize biderkatzen ditugunean egiten diren biderkaketen kopurua, D(I-1)*D(K)*D(J), beti finkoa da eta ezin da hobetu (bi matrize biderkatzeko algoritmoa aldatu gabe behintzat).

• K-k, egin daitezkeen faktorketa-ordena posible guztiak ematen dizkigu. K-ren balio guztien artetik espresioa minimizatzen duena aukeratzeak, faktorketa-ordena onena hartzea suposatuko du.

• MBM_PD(I,J)-k $M_i * ... * M_j$ biderkaketa egiteko egin beharko liratekeen osagaien arteko biderkaketa kopuru *minimoa* biltegiratzen du.

⇒ Aurreko arrazonamenduak algoritmoak beti soluzio onena aurkitzen duela ziurtatzen du. Hau da, azpiegitura optimoaren propietatea betetzen da.

Ondorengo prozedurak, SAILKAPENAREN_ERAKUSKETA(1,N), matrizeak biderkatu beharko genituzkeen faktorketa-ordenaren adierazpena, idazkera parentesidunaren bidez azalduko digu. $M_i * ... * M_j$ matrizeen biderkaketa egiteko garaian, *K*-k (FAKTORKETA(I,J) = K) azkeneko eta ez lehenengo faktorketa puntua adierazten duela jabetzerakoan guztiz sinplea den algoritmo bat berehala lor dezakegu:

```
procedure  SAILKAPENAREN_ERAKUSKETA (I,J: in INTEGER) is
   K: INTEGER;
begin
   if  I=J  then  PUT("M" & INTEGER'IMAGE(I) );
   else  K:= FAKTORKETA(I,J);
         PUT("(");
         SAILKAPENAREN_ERAKUSKETA(I,K);
         PUT("*");
         SAILKAPENAREN_ERAKUSKETA(K+1,J);
         PUT(")");
   end if;
end SAILKAPENAREN_ERAKUSKETA;
```

Adibidearekin jarraituz, SAILKAPENAREN_ERAKUSKETA(1,4) "(M1 * ((M2 * M3) * M4))" idatziko luke.

<u>*Algoritmoaren denbora-ordenaren goi-bornaketaren analisia*</u>:

FAKTORKETA_OPTIMOA-k MBM matrizearen $n^2/2$ posizio betetzen ditu gutxi gora-behera. Posizio bakoitzeko balioa lortzeko *n* espresio baino gutxiago kalkulatu behar ditu, non espresio bakoitzeko eragiketa matematiko gutxi batzuk egiten dituen; hau da, posizio bakoitza kalkulatzeko denbora O(n) da. Beraz, egiten diren pausoen kopurua $O(n^3)$ da, eta honek, zatitu eta irabazi teknikarekin lortuko genukeen algoritmo esponentzialarekin aldaratuz gero, denbora-irabazi nabarmena dakar. SAILKAPENAREN_ERAKUSKETAri eginiko dei bakoitzeko, konkretuki matrize baten izena edo '*', '(' eta ')' ikurrak idazten ditu, beraz, guztira 2n-1 dei egiten dira. SAILKAPENAREN_ERAKUSKETA algoritmoak behar duen exekuzio-denboraren ordena zehatza n da: $\Theta(n)$.

<u>*Denbora-ordenaren kalkulu zehatzaren analisia*</u>:

Sekuentziak *n* matrize dituela suposatuz:

• MBM_PD matrizea hasieratzeko $\Theta(n^2)$ denbora behar da.

• 0 diagonala *n* 0 duen diagonala izanik, diagonal horretako gelaskei 0 balioa esleitzeko $\Theta(n)$ denbora beharko da.

• Goiko eskuin triangeluko gelasken balioak kalkulatzeko egin behar den lana:

– Guztira (n-1) diagonal kalkulatu beharko dira.

– Diagonal bakoitzeko: diagonaleko gelaxka guztien balioak kalkulatu behar dira. Zehazki, d diagonalean (n-d) gelaxka daude.

– *d* diagonaleko edozein gelaxkarentzat *d* balio desberdin har ditzake *k*-k; hau da, azpisekuentzia d modu desberdinetan zati daiteke.

– Gelaxka bakoitzeko: *k*-ren balio desberdin bakoitzeko 2 biderkaketa eta 2 batuketa egin behar dira.

– Demagun *b* dela 2 biderkaketa eta 2 batuketa kalkulatzeko behar den denbora konstantea.

Hori dela eta, *d* diagonaleko gelaxka bat kalkulatzeko beharko den denbora $b * d$ da. Eta beraz, diagonal osoa kalkulatzeko beharko den denbora $b * d\ (n - d)$. Kalkuluak diagonal guztientzat egin behar dira:

$$T(n) = \sum_{d=1}^{n} \left(b^{*}\ d(n-d) \right) = b\left(n\sum_{d=1}^{n} d - \sum_{d=1}^{n} d^{2} \right) = b\left(n\frac{1+n}{2}n - \frac{2n^{3}+3n^{2}+n}{6} \right) \in \Theta(n^{3})$$

Hiru puntuetako denbora-ordenak ikusiz, algoritmoaren ordena zehatza kubikoa da.

Memoria-espazio estraren ordenaren kalkulua

Baina ez dugu ahantzi behar algoritmoak MBM_PD eta FAKTORKETA taulak erabiltzen dituela. Hauentzat behar duen memoria-espazio estra $\Theta(n^2)$-koa da.

FAKTORKETA_OPTIMOA-ren algoritmo errekurtsiboak $\Theta(n)$ memoria--espazio estra behar du pilarentzat soilik.

6.5. BIDE MINIMOAK: FLOYD-EN ALGORITMOA

Izan bedi G grafo zuzendua eta pisuduna: G=<ERPINAK, ARKUAK, PISUAK>, non grafoko arku bakoitzari negatiboa ez den pisu bat dagokion.

Grafoko erpin bikote bakoitzaren arteko distantzia minimoak mugatu nahi ditugu.

Orain arte ikusi duguna gogoratuz, problema erraz ebazteko ideia berehala etor dakiguke: Dijkstra-k erpin batetik besteetarako distantzia minimoak

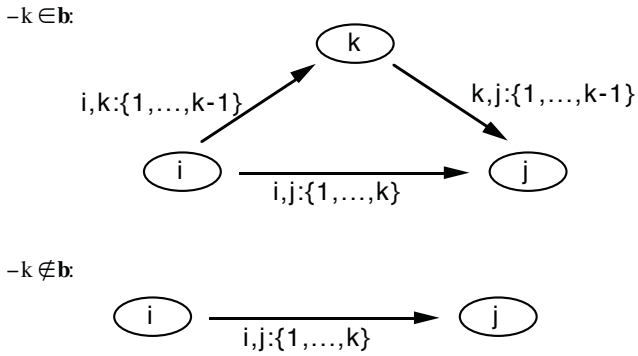
kalkulatzen dituenez, zergatik ez hau P aldiz aplikatu aldi bakoitzean abiapuntu bezala beste erpin bat hartuz? Horrela egingo bagenu, Dijkstraren algoritmoaren ordena (metarik erabili gabe eta matrize pisudunen adierazpidea erabiliz) kuadratikoa denez P erpinen kopuruan, gure lehengo hurbilketa kubikoa litzateke baita ere grafoko erpinen kopuruan (atal honen amaieran zerbait gehiago esango dugu hurbilketa honi buruz).

Orain aurkeztuko dugun soluzioa ordena berdinekoa bada ere, oraingo soluzioaren konstante biderkatzailea oso txikia da. Soluzio hau FLOYD--WARSHALL-en algoritmoa izenez ezagutzen da.

Hemen emango den soluzioa *bide bateko tarteko erpin*en definizioan eta grafoetako bideen propietate batean oinarritzen da:

• $b=<e_1,e_2,...,e_k>$ *bideko tarteko erpinak* $e_2,...,e_{k-1}$ dira.

• i-tik hasi, j-ra iristen eta {1,...,k} tarteko erpinak soilik erabiliz osa daitezkeen bide guztien artetik izan bedi **b** distantzia motzena duena. Grafoen teorian ezaguna da **b** bidean bi gauza gerta daitezkeela:



Ondorioz, eta {1,...,k} tarteko erpinak soilik erabiliz i-tik j-raino bide motzenaren distantzia $\min\left(d_{i,j}^{k-1}, d_{i,k}^{k-1} + d_{k,j}^{k-1}\right)$ da.

Propietate hau kontutan izanik, problema indukzio-bidez honela defini dezakegu:

$$d_{i,j}^{k} = \begin{cases} \text{pisua}(i,j) & \text{baldin } k = 0 \\ \min\left(d_{i,j}^{k-1}, d_{i,k}^{k-1} + d_{k,j}^{k-1}\right) & \text{baldin } k \geq 1 \end{cases}$$

$d^k_{i,j}$ -k, i-tik hasi, j-ra iristen eta {1,...,k} tarteko erpinak soilik erabiliz osa daitezkeen bide guztien artetik distantzia motzena duenaren balioa jasoko du.

Datuak eta hauen adierazpidea:

Demagun berriz ere G grafoko erpinak 1-etik P-ra zenbakitu ditugula ERPINAK= {1,2,... P}, eta LUZERA(1..P,1..P) matrizean arkuen distantziak (pisuak) ditugula:

– *i* erpinetik *i* erpineraino distantziarik ez dagoela eta LUZERA(i,i) = 0,

– (i,j) arkuaren distantzia (*i* erpinetik *j* erpineraino dagoen distantzia ($i \neq j$)) LUZERA matrizearen (i,j) posizioan biltegiratuko dugu: LUZERA(i,j) ≥ 0

– (i,j) arkuak existitzen ez duenean, orduan matrizean LUZERA(i,j) = SYSTEM.MAX_INT jasoko dugu.

Algoritmoa:

Ikusitako propietatea erabiliz, lehendabizi tarteko erpinik erabili gabe, $d^0_{i,j}$ balioak kalkulatuko lituzke algoritmoak, ondoren, eta balio hauek erabiliz, gehienez erpin bat zeharkatzea onartuko balu soilik, adib.{1}, $d^1_{i,j}$ balioak kalkulatuko lituzke, gehienez tarteko bi erpin zeharkatzea onartuko balu, adib.{1,2}, $d^2_{i,j}$ balioak kalkulatuko lituzke. Prozesu berdina P aldiz errepikatu ondoren $d^p_{i,j}$ balioak kalkulatuak egongo lirateke; hots, i-tik hasi, edozein erpin zehakatuz eta j-raino iristen diren guztien artetik laburrenaren distantziak kalkulatuta leudeke.

```
procedure   FLOYD  (LUZERA: in MAT_ERREAL;
                        D: in out MAT_ERREAL) is
begin
    D:= LUZERA;
    for K  in  D'RANGE(1)  loop
          for I  in  D'RANGE(1)  loop
                for  J  in  D'RANGE(1)  loop
                      D(i,j) = min (D(i,j), D(i,k)+D(k,j));
                end loop;
          end loop;
    end loop;
end;
```

D matrizeak erpin bikote bakoitzeko beraien arteko distantzia minimoa emango digu. Hasieran D matrizea LUZERA matrizea da. *k* iterazio ondoren, D matrizeko (i,j) posizioak *i* erpinetik *j* erpineraino eta {1,2,...,K} erpinak soilik

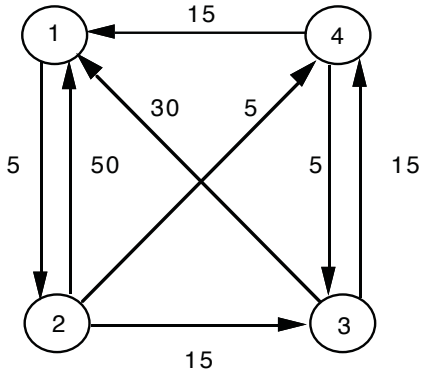
zeharka ditzaketen bide guztietatik minimoaren distantzia metatuta dago eta 1≤i,j≤P. P iterazioen ondoren, D matrizeko (i,j) posizioak *i* erpinetik abiatzen den eta grafoko edozein erpin zeharkatuz *j* erpineraino iristen diren bide guztietatik minimoaren distantzia metatuko du eta 1≤i,j≤P.

Bide optimoek beste propietate bat betetzen dute: sekulan ez dute erpin bera bi aldiz zeharkatzen. Gure kasuan, propietate bera betetzea inplizituki lortu da.

k. biran *k* errenkadako eta zutabeko balioak ez dira aldatzen, D(k,k) beti zero baita. D-ren eguneratzea egiten ari den bitartean ez dugu zergatik balio hauek aparte gorde behar. Hau dela eta, P$_*$P dimentsioak dituen matrize bakar batekin molda gaitezke eta ez dugu beste D' matrize laguntzaile baten beharrik izango.

Azaldu dugun ideia, FLOYD-en algoritmoak biltzen du.

<u>*Adibidea*</u>:



$$D_0 = \text{LUZERA} = \begin{bmatrix} 0 & 5 & \infty & \infty \\ 50 & 0 & 15 & 5 \\ 30 & \infty & 0 & 15 \\ 15 & \infty & 5 & 0 \end{bmatrix}$$

$$D_1 = \begin{bmatrix} 0 & 5 & \infty & \infty \\ 50 & 0 & 15 & 5 \\ 30 & \mathbf{35} & 0 & 15 \\ 15 & \mathbf{20} & 5 & 0 \end{bmatrix} \qquad D_2 = \begin{bmatrix} 0 & 5 & \mathbf{20} & \mathbf{10} \\ 50 & 0 & 15 & 5 \\ 30 & \infty & 0 & 15 \\ 15 & \infty & 5 & 0 \end{bmatrix}$$

$$D_3 = \begin{bmatrix} 0 & 5 & 20 & 10 \\ \mathbf{45} & 0 & 15 & 5 \\ 30 & 35 & 0 & 15 \\ 15 & 20 & 5 & 0 \end{bmatrix} \qquad D_4 = \begin{bmatrix} 0 & 5 & \mathbf{15} & 10 \\ \mathbf{20} & 0 & 15 & 5 \\ 30 & 35 & 0 & 15 \\ 15 & 20 & 5 & 0 \end{bmatrix}$$

<u>*Ordenaren analisia*</u>:

Begi-bistakoa da algoritmo honen egikaritzapenak hartzen duen denboraren ordena zehatza $\Theta(P^3)$ dela.

FLOYDen algoritmoa v.s. DIJKSTRAren algoritmoa:

Dijkstraren algoritmo jalea ikusita dugu. Bertan erpin bat abiapuntutzat harturik grafoko beste erpinetara doazen bide motzenak mugatzen dira. |ERPINAK|= P bada, Dijkstraren algoritmoa P aldiz egikarituz aldi bakoitzeko erpin desberdin bat abiapuntutzat aukeratuta, Floydek ebazten duen problema bera ebatz dezakegu. Baina, exekuzio-ordenen aldetik, desberdintasunik ba al dago?

Bi algoritmo horiek erabiliz, grafo baten bide minimoak kalkulatzeko hiru aukera desberdin ditugu. Ikus dezagun noiz zein erabili:

Dijkstraren algoritmoa P aldiz egikaritu erpin bakoitzeko (ikus Dijkstraren algoritmoa). Aldi berean, Dijkstraren algoritmoan *grafo* datu-egitura eta *distantzien* taula inplementatzeko bi aukera aztertu ditugu eta aukeratutakoaren arabera:

1- *Grafoa* auzokide matrize baten bidez definituta dagoenean eta *distantzien* taula arrunta denean, Dijkstraren soluzioa erpinen kopuruan koadratikoa da. Hau *p* aldiz errepikatuz: $\Theta(P^3)$.
Floyden hazkuntza-tasa berdina izanik Floyden algoritmoa Dijkstrarena baino sinpleagoa denez, badirudi praktikan Floyden algoritmoa azkarragoa izango dela; hots, konstante biderkatzailea txikiagoa izango du. Hala ere, formalki algoritmo bakoitzean inplizituki gelditzen diren konstanteak aztertu beharko genituzke.

2- Distantzien taula meta egituraz inplementatzen denean (gailurrean balio txikiena) eta grafoa auzokide-listen bidez definiturik datorrenean, Dijkstraren soluzioaren ordena $\Theta((A+P) \lg P)$ da. Hau *p* aldiz errepikatuz: $\Theta((AP+P^2) \lg P)$
Grafoa dentsoa balitz, arku ugari izango luke; orduan, $A \approx P^2$ eta algoritmo honen ordena zehatza $\Theta(P^3 \lg P)$ litzateke. Floydena erabiltzea gomendagarriagoa litzateke.
Grafoa dentsoa ez balitz, arku gutxi izango luke; orduan $A << P^2$. Gainera, kasu txarrenean $A \approx P-1$ gerta liteke. Baldintza horietan Dijkstrarena p aldiz errepikatzea gomendagarriagoa litzateke, ordena $\Theta(P^2 \lg P)$ litzateke eta.

6.6. FUNTZIO MEMORIADUNAK

Programazio dinamikoaren aldaera bat da. Goitik beherako estrategia mantentzen da. Honela, naturala den algoritmo errekurtsiboa mantentzen da. Programazio dinamikoa denez, azpiproblemen emaitzak ohiko taulan gordetzen

dira, baina hori betetzearen gaineko kontrola algoritmo errekurtsiboak darama. Algoritmo errekurtsibo hori *funtzio memoriadun* izenez ezagutzen da.

Azpiproblema berdinak ebatziko dituzten memoriadunek algoritmo errekurtsibo, taulako posizio bera dute erlazionaturik. Taulak azpiproblema desberdin bakoitzerako sarrera desberdin bat izango du. Hasieran, taulako posizioak balio berezi batez hasieratzen dira. Taulako posizio batean balio berezi hori egoteak taulako posizio horrek erlazionaturik duen azpiproblema oraindik ebatzi gabe dagoela adieraziko du. Azpiproblema jakin bat ebatzi behar denean lehendabizi azpiproblema honek erlazionaturik duen posizioa aztertuko da, eta ondorioz, ondoko bi kasuak gerta daitezke:

– Taulako posizioan balio berezia dago, beraz, azpiproblema hori oraindik ez da sekulan ebatzi. Ondorioz, lehendabiziko aldiz ebatzi eta emaitza posizio horretan biltegiratuko da.

– Taulan balio bereziaren desberdina den beste balio bat dago: azpiproblema hori aurrez, noizbait aztertu eta ebatzi da. Ez da birkalkulatzen, baizik eta taulako balioa zuzenean itzultzen da. Taulako posizio bateko balioaren itzulketa denbora konstantean egiten dela suposatuko dugu.

Teknika berriaren aplikazioaren zenbait adibide dator segidan.

6.6.1. Fibonacciren zenbakiak

<u>*Problemaren definizio errekurtsiboa*</u>:

Lehendabizi, beti, problemaren definizio errekurtsiboa eman behar da.

$$fib(n)\begin{cases}1 & n=0 \wedge n=1\\ fib(n-1)+fib(n-2) & n>1\end{cases}$$

Zenbait kasu nabari memoriaren (taularen edo matrizearen) hasieraketa egoki bidez inplementatzen da.

<u>*Algoritmo berria*</u>:
1 inplementazioa:

```
function FIBONACCI (N: in NATURAL) return INTEGER is
BEK: TAULA(0,N):= (0=>1, 1=>1, others=>0);
function  FIB (N: in NATURAL) return INTEGER is
begin
    if not (BEK(N)=0 ) then return BEK(N);
    else
      if BEK(N-1)=0  then  BEK(N-1):= FIB(N-1); end if;
      if BEK(N-2)=0  then  BEK(N-2):= FIB(N-2); end if;
```

```
            BEK(N):= BEK(N-1)+BEK(N-2);
            return BEK(N);
        end FIB;
    begin
        return FIB(N);
    end FIBONACCI;
```

2 inplementazioa:

```
    function FIBONACCI (N: in NATURAL) return INTEGER is
    BEK: TAULA(0,N):= (0=>1, 1=>1, others=>0);
    procedure  FIB (N: in NATURAL) is
    begin
        if BEK(N-1)=0 then FIB(N-1); end if;
        --BEK(N-2) ziur kalkulaturik dagoela une honetan
        BEK(N):= BEK(N-1)+BEK(N-2);
    end FIB;
begin
    -- N>=0
    --Dagoeneko balio bereziekin eta nabariekin hasieraturik dago taula
    if BEK(N)=0
    then  --kasu orokorra da eta oraindik ez dago kalkulaturik
          FIB(N);
    end if;
    return BEK(N)
end FIBONACCI;
```

Ordenaren analisia:

• Fibonacciren *n.* zenbakia kalkulatzeko aurreko *n-1* zenbakiak kalkulatu beharko ditugu.

• Ez da alferrikako dei errekurtsiborik egiten.

• Fibonacci-zenbaki baten kalkulua behin bakarrik egiten da.

• Fibonacci-zenbaki baten kalkulua egiteko aurreko bi Fibonacci-zenbakiak kalkulatuta badaude, kalkulu berriak denbora konstantea hartzen du.

Beraz, ordena zehatza lineala da, $\Theta(n)$.

Bi inplementazioen denbora-ordena berdina bada ere, bigarrenaren konstante-biderkatzailea txikiagoa da. Ziur faltsuak diren baldintzen galdeketa bai eta alperrikako esleipenak ekiditen ditu.

Memoria-espazio estra lineala da *n*-n bi inplementazioetan.

6.6.2. Matrize-sekuentzia bat biderkatzeko faktorketa optimoa

Problemaren definizio errekurtsiboa:

$\forall I,J$ non $1 \le I \le J \le N$

$$X\_MS(I,J)\begin{cases} 0 & I = J \\ \min_{1 \le K \le J-1}\{X\_MS(I,K) + X\_MS(K+1,J) + D(I-1) * D(K) * D(J)\} & I < J \end{cases}$$

Algoritmo berria:

```
procedure  MS_BIDERKAKETA  (D: in DIMENTSIOAK; BIDER: out INTEGER;
                                 FAKT:out MATRIZEA) is

  N: INTEGER:= D'LAST;      -- D matrizeko indizeak (0..N) direla suposatuko dugu
  X_MAT: MATRIZEA (1.. N,1..N):= (others=>(others=>SYSTEM.MAX_INT));
  procedure  FAKTORKETA_OPT (I,J: in INTEGER) is
          Q: INTEGER;
  begin
          -- Kalkulatu gabeko kasu orokorra
          for K in  I .. J-1  loop
               if  X_MAT(I,K)=SYSTEM.MAX_INT
               then  FAKTORKETA_OPT(I,K);
               end if;

               if  X_MAT(K+1, J)=SYSTEM.MAX_INT
               then  FAKTORKETA_OPT(K+1, J);
               end if;

               Q:= X_MAT(I,K)+ X_MAT(K+1,J)+D(I-1)*D(K)*D(J);
               if  Q<X_MAT(I,J)
               then  X_MAT(I,J):=Q; FAKT(I,J):=K;
               end if;
          end loop;
  end FAKTORKETA_OPT ;
begin
  -- kasu nabarien hasieraketa matrizean
  for  I  in 1..N loop  X_MAT(I,I) := 0;  end loop;
  -- Dagoeneko X_MAT  bai balio bereziekin bai kasu nabarietako balioekin hasieraturik dago
  if  X_MAT(1,N) = SYSTEM.MAX_INT
  then
          FAKTORKETA_OPT(1,N);
  end if;
  BIDER:= X_MAT(1, N);
end MS_BIDERKAKETA;
```

Ordenaren analisia:

• X_MAT hasieratzeko n^2 matrize-gelaxka hasieratu behar direla eta, hasieraketa egiteko denbora koadratikoa beharko da.

• Berez, ez dira gelaxka guztiak eguneratuko, baizik eta matrizearen eskuineko goi triangelua: $(n^2)/2$ gelaxka.

• Ez da alferrikako deirik egiten.

• Gelaxka bateko balioa behin bakarrik kalkulatzen da.

• Gelaxka konkretu baten balioa kalkulatzeko, adib. (I,J) gelaxka, beste zenbait gelaxka bikoteren balioak jakin behar dira. Konkretuki, matrize-sekuentzia bi faktoretan banatzeko K posibilitate desberdin daude non $I \leq K \leq J-1$ betetzen den. Beraz, J-I bikote bakoitzeko bi biderkaketa eta bi batuketa egin beharko ditugu beti, eta ondoren J-I balio desberdinetatik txikiena zein den erabaki. Azken batean, biderkatu behar diren matrize kopuruaren arabera egongo da. Kasu txarrenean $1 \leq K \leq N-1$ gertatuko da, (1,N) gelaxkaren balioaren kalkuluan. Kasu horretan, gelaxkako balioa mugatzeko (n-1) balio artetik txikienarekin gelditu gara.

Ordena beraz, $\Theta((n^2)/2)_*O(n)=O(n^3)$

6.6.3. Txanponen problema

Problemaren enuntziatua:

K kopuru finkoa emango duten txanpon kopuru minimoa kalkulatzen duen eta programazio dinamikoko teknika jarraitzen duen algoritmoa egin nahi da. Horretarako, N txanpon-mota ezberdin ditugula suposatuko dugu, non $b_1,b_2,...b_N$ haien balioak diren. Txanpon-mota bakoitzeko behar bezainbeste txanpon dugula suposatzen da. Ondoren, algoritmoaren *t(N)* kalkulatuko dugu (K konstantea dela suposatuko dugu).

Problemaren definizio errekurtsiboa:

$$\text{Txanponak}(K) = \begin{cases} 0 & \text{K deskonposaezina edota} \\ & \forall i(1 \leq i \leq n \Rightarrow b_i > K) \\ 1 & \exists i(1 \leq i \leq n \wedge b_i = K) \\ 1 + \min_{1 \leq i \leq n} \left\{ \text{Txanponak}(K - b_i) \mid (b_i < K) \wedge \text{deskonponsagarria}(K - b_i) \right\} & \text{Bestela} \end{cases}$$

non deskonponsagarria (Z) honela defini dezakegun:

$$\exists A_1,...,A_n(0 \leq A_1,...,A_n \wedge A_1b_1+...+A_nb_n=K)$$

Algoritmo berria:

```
function TXANPONAK_ITZULI (BALIOA: in TXANPONAK; KOP: in INTEGER)
                           return INTEGER is
    N: INTEGER:= BALIOA'LAST;
    —BALIOA taularen indizeak 1..N direla suposatuko dugu. Gainera ondorengoa betzen
    — dela suposatuz eginik dago kodea:
    — BALIOA(N) ≤ KOP ∧ ∀i (1≤i<N ⇒ BALIOA(i)≤ BALIOA(i+1) )

    T: TAULA(1,KOP):= (others=>SYSTEM.MAX_INT);

    procedure  TXANPONAK (K: in INTEGER) is
          Q: INTEGER;
    begin
          for R in 1 .. N loop
                if  0< K-BALIOA(TX,R) then
                   — Txanpona kopurua baino txikiagoa da if  T(K-BALIOA(R))=
                   SYSTEM.MAX_INT then
                               TXANPONAK (K-BALIOA(R));
                        end if;
                        Q:= 1+ T(K-BALIOA(TX,R));
                        if  1<Q<T(K)  then  T(K):=Q; end if;
                     end if;
                   end loop;
                   if T(K)=SYSTEM.MAX_INT  then
                      — K kopurua ez da deskonposagarria eta horregatik
                      — ez da oraindik K posizioko balioa aldatu.
                      — Deskonposaezina: 0 balioaz adierazten dugu.
                      T(K):=0;
                end if;
          end if;
          return TAULA(K);
    end TXANPONAK ;
begin
    — 1..BALIOA(1)-1   kopuruak deskonopasaezinak dira.
    for  K  in 1..BALIOA(1)-1  loop  T(K) := 0;  end loop;

    — Txanpon bakarra behar duten kopuruen hasieraketa
    for  K  in 1.. N  loop  T(BALIOA(K)) := 1;  end loop;

    — Balio bereziak nahiz kasu nabarietako balioak dauzka T taulak.
    if T(KOP) = SYSTEM.MAX_INT then  TXANPONAK(KOP); end if;
    — T(KOP) aldagaian jaso ditugu KOP kopurua itzultzeko behar
    — diren txanpon kopuru minimoa.
    return T(KOP);
end TXANPONAK_ITZULI;
```

Ordenaren analisia:

(Txanpon-mota desberdinen kopurua, *n*, problema desberdinetarako desberdina izan daitekeela onartuko dugu. Hartaz, *n* ezin dugu konstante dela kontsideratu. Honek ordena kalkulatzeko garaian eragina izango du.)

Aurreko bi adibideetan jarraitutako arrazonamendu berdina ere erabil dezakegu hemen:

- Ez da alferrikako dei errekurtsiborik egiten.

- Taulako posizio bakoitzeko balioa behin bakarrik kalkulatuko da eta KOP gelaxka ditugu.

- I gelaxka bateko balioa kalkulatzeko gehienez *n* gelasken balioak beharko ditugu; hain zuzen, Z gelasken balioak beharko ditugu non Z honen balioa honela defini dezakegun:

$$Z= Nj(1 \leq j \leq n \wedge KOP\text{-}b_j > 0)$$

Z kasu txarrenean *n* izango da (non *n* dauden txanpon-mota desberdinen kopurua izango den).

Orduan, I gelaxkako balioa mugatzeko gehienez *n* gelasketako balioetatik txikiena aukeratuko du eta balio horri 1 gehituko dio. Honen kalkulua *n*-ren ordenakoa izango da.

Beraz, algoritmoaren ordena: $T(n,KOP)= O(n \cdot KOP)$

Memoria-espazio estraren erabilera lineala da KOP-en.

ARIKETAK: PROGRAMAZIO DINAMIKOA

1- Demagun PARTIKETA funtzioa honela definituta dagoela:

```
type BEKTORE  is array(INTEGER range <>) of POSITIVE;
function PARTIKETA (B: BEKTORE ) return INTEGER  is
begin
    for I in  B'RANGE  loop
          if   sum_{j=B©FIRST}^{I-1} B(j) = sum_{j=I}^{B©LAST} B(j)  then  return  I end if;
    end loop;
    return 0;
end PARTIKETA;
```

Zenbaki osoko positiboen B bektorea emanik, PARTIKETA funtzioak ondorengo baldintza betetzen duen I (B bektoreko) posizioa itzultzen du:

• I posizioaren ezkerraldean dauden B-ko osagaien batura (B(I) balioa kanpo) I posizioaren eskuinaldean dauden B-ko osagaien (B(I) balioa barne) baturaren berdina da.

Aurreko baldintza betetzen duen posiziorik ez badago bektorean, funtzioak 0 itzuli beharko du.

– Frogatu azaldutako ezaugarriak betetzen dituen posizio bakar bat egon daitekeela gehienez.
– Programaren exekuzio-denbora eta denbora-ordena kalkula bitez.
– O(n) ordena hartuko duen algoritmo bat idatz bedi.
– O(lg n) denbora hartuko lukeen algoritmorik aurkitzea posiblea ote litzatekeen laburki eztabaidatu.

2- (Xj,Yj) jatorri-puntutik (Xn,Yn) helmuga-puntura zenbat bide posible dauden mugatzen duen algoritmoa idatz bedi. Puntuen koordenatuak zenbaki arruntak direla suposatu. Bestalde, (X,Y) puntu batetik beste puntu batera joateko eman daitezkeen pausoak (X+1,Y) eta (X,Y+1) bakarrik izango dira. Algoritmoaren exekuzio-denbora aztertu. Algoritmoak programazio dinamikoaren teknika jarraitu behar du.

3- Nilo ibaian N ontziratoki daude, bakoitzean Y ontzi aloka daitezkeelarik ibaian behera dauden beste ontziratokietara joateko (ibaian gora ezin da joan, korronte handia du eta ibaiak). Edozein A abiapuntu-ontziratokitik ibaian behera dagoen edozein H ontziratokira iristeko ontzi bakar bat alokatzea garestiagoa gerta liteke Atik A+1ra, A+1etik A+6ra eta A+6tik Hra joatea baino, adibidez.

A abiapuntu-ontziratoki guztietatik H helmuga-ontziratoki guztietara joateko bidai koste posible guztien artetik bidai koste minimoak kalkulatzea eskatzen da, bai funtzio memoriadunak erabiliz, bai hauek erabili gabe, baina bi kasuetan programazio dinamikoa teknika erabiliz. Ondoren, exekuzio--denboraren eta memoria-espazio estraren ordenak kalkula bitez.

4- K kopuru finkoa emango duten txanpon kopuru minimoa kalkulatzen eta programazio dinamikoko teknika jarraitzen duen algoritmo bat egin bedi. Horretarako N txanpon-mota ezberdin ditugula suposatuko dugu, non $b_1,b_2,...b_N$ haien balioak diren. Txanpon-mota bakoitzeko behar bezain beste txanpon kopuru dugula suposatzen da. Algoritmoaren *t(N)* kalkulatu (K konstante bat dela suposatuko da).

(Oharra: izan bedi Z(K) K batzen duten txanpon zenbakirik txikiena. Aztertu bedi zer erlazioa dagoen Z(K) eta Z(K-b_i)-ren artean, non $1 \le i \le n$ eta $b_i \le K$.)

5- Izan bedi G grafo zuzendua: G=(Erp, Ark). G Grafoaren adierazpena Auzokideen matrize bidez ematen da. Problema Dij matrize logikoa definitzea da, non Dij egiazkoa izango den *i*-tik *j*-ra gutxienez bideren bat badago, eta faltsua bestela. Egokitu bedi Floyd-en algoritmoa problema hau ebatz dezan. Bere exekuzio-denbora azter bedi.

6 $[h_i,b_i)$ $1 \le i \le n$ ekintzak izanik, ekintza-hautaketa problema programazio dinamikoa teknikaren bidez ebazten duen algoritmoa idaztea eskatzen da: algoritmoa $ZENB_i$ i=1,2,..,n zenbakiaren kalkulu iteratiboan oinarrituko da, non $ZENB_i$ zenbaki horrek {1,2,..i} ekintzen multzoko kardinalitate handieneko eta elkar bateragarri diren ekintzen kopurua metatuko duen. Ekintzak bukaera-denbora balioekiko gorantz ordenatuta daudela suposatuko dugu: $b_1 \le b_2 \le ... \le b_n$.

Aldera bitez soluzio honen eta problema beraren teknika jaleko soluzioaren exekuzio-denbora eta exekuzio-ordena.

7- $b_1,b_2,...,b_n$ *n* txanponen klaseak emanik eta jakinik txanpon klase bakoitzeko behar hainbat txanpon eskuragarri dugula, K kopuru jakina itzultzeko zenbat txanponen konbinazio desberdin dauden kalkulatuko duen algoritmoa idaztea eskatzen da.

Ad.: Txanponen klaseak honakoak balira: $b_1= 25$, $b_2 = 50$, $b_3 = 75$ eta $b_4 =100$.

Kopurua	Konbinazio desberdinak
100	4 modu: 25+75, 2*50, 100, 4*25, 50+2*25
116	0, ez dago konbinaziorik

Iradokizuna: 25+75 konbinazioa 75+25 konbinazioaren berdina da, eta ondorioz, konbinazio bakarra kontsideratu behar da.

8- Unibertsitate bateko sail batek *IK* ikastordu eman behar ditu. Horretarako, *N* klase desberdinetako irakasleak kontrata ditzake. *I* klaseko irakasle bakoitzak O_i ordu irakats ditzake gehienez, lan horregatik beti P_i pezeta eskuratuz. *IK* klaseak emateko eta ordaindu beharreko pezeta kopurua minimoa izan dadin, zenbat irakasle kontratatu beharko liratekeen kalkulatuko duen daudela direla jakinik, algoritmoaren exekuzio-denboraren ordena kalkulatu.

9- *Lagunak lanean* enpresako zuzendariak Marigorriri kontsultoria bat egin dezala eskatu diola eta, honi, enpresa afari bat antolatzea okurritu zaio. Gonbidatuen zerrenda egiteko, "gainbegirale" funtzio bat erabil dezake. Enpresako langileriaren egitura hierarkikoa denez, funtzio honen heinak piramide bat osatzen du, haren gailurrean gaur egungo zuzendaria dagoelarik. Bestalde, administrazioko langileek, langile guztiek erlazionaturik duten soziabilitate balioarekiko, lantegiko langile guztiak ordenatu dituzte zerrenda bat osatuz (soziabilitate balioa zenbaki erreala da). Afarira joango diren partaideak, hura goza dezaten, zuzendariak langile bat eta honen gainbegirale zuzena, biak, afarian ez egoteko murriztapena ezarri du:

   • Zerrenda kalkulatuko duen algoritmoa idatz bedi. Helburua, partaideen soziabilitate balioa maximizatzea izango da. Exekuzio denbora ordena kalkula bedi.
   • Marigorrik nola ziurta dezake partaideen artean enpresako zuzendaria egongo dela?

10- A alfabetoa A={a,b,c} izanik, ondorengo taula, alfabetoko bi karaktereen konbinazioak zertara berridatz daitezkeen jasotzen du:

a a→b	a b→b	a c→a
b a→c	b b→b	b c→a
c a→a	c b→c	c c→c

Berridazketa eragiketak, nahiz alfabeto itxian izan, ez du trukaketa ez eta elkartze propietatea ere betetzen.

X=x1 ... xn karakterez osaturiko karaktere katean parentesiak zein posiziotan idatzi beharko liratekeen katea a-ra berridazteko mugatuko du idatzi behar den algoritmoak. Horrela, X=bbbba bada, algoritmoak bai itzuli beharko luke eta, adibidez, (b(bb))(ba)→a (berridazketa ordena batek baino gehiagok existi baitezake: (b(b(b(ba))))→a. N berridatzi behar den karaktere katearen luzera bada, algoritmoaren denbora-ordena zein da?

11- Izan bedi T N zenbaki oso desberdinez osatutako taula. T-ko osagaiak (eta bertan agertzen diren ordena mantenduz) ordena gorakorra jarraituz ordenaturik dauzkaten sekuentzia desberdinen artetik sekuentzia luzeena kalkulatzea eskatzen da. Adibidez, sekuentzia 11,17,5,8,6,4,7,12,3 denean, algoritmoak 5,6,7,12 sekuentzia itzuliko du. Algoritmoaren denbora-ordena zenbatekoa da?

12- Bi karaktere-kateek azpikate komun ugari eduki dezakete. X=x1 ... xn eta Y=y1 ... ym bi karaktere kate emanik, bi kateek dituzten karaktere-kate komunen arteko azpikateen arteko luzeena mugatuko duen algoritmoa idatzi. Algoritmoaren denbora-ordena ere kalkulatu (problema programazio dinamikoa erabiliz ebazten bada, $\Theta(nm)$ ordenako soluzio bat lortzea posiblea da. Teknika hori erabili gabe ordena berdina duten beste soluzio batzuk ere badaude. Bi eratako soluzioak bila bitez.)

BIBLIOGRAFIA

Oinarrizkoa:

Cormen, Thomas H.; Leiserson, Charles E. ; Rivest, Ronald L.
Introduction To Algorithms
Ed.: The MIT Press, 1.992.

Baase, Sara.
Computer Algorithms. Introduction To Design And Analysis. 2nd edition.
Ed.: Addison-Wesley Publishing Company, 1.988.

Brassard, Gilles; Bratley, Paul.
Algorítmica. Concepción Y Análisis.
Ed.: Masson, 1.990.

Moret, B.M.E.; Shapiro, H.D.
Algorithms From P To Np. Vol.1. Design And Efficiency.
Ed.: The Benjamin/Cummings Publihing Company, Inc. 1.991.

Rawlins, J. E. Gregory
Compared To What? An Introduction To The Analysis Of Algorithms.
Ed.: Computer Science Press, 1.992.

Osagarria:

Aho, Alfred V.; Hopcroft, John E.; Ullman, Jeffrey D.
Data structures and algorithms.
Ad.: Addison-Wesley Iberoamericana, 1.983.

Banachowski, Lech; Kreczmar, Antoni; Rytter, Wojcoech.
Analysis Of Algorithms And Data Stuctures.
Ed.: Addison-Wesley Publishing Company, 1.991.

Galve, Javier; González, Juan C.; Sánchez, Angel; Velázquez, J. Angel
Algorítmica. Diseño Y Análisis De Algoritmos Funcionales E Imperativos.
Ed.: Rama, 1.993.

Lerman, Steven R.
Problem Solving And Computation For Scientists And Engineers: An Introduction Using C
Ed.: Prentice-Hall Inc., 1.993.

Manber, Udi
Introduction To Algorithms: A Creative Approach.
Ad.: Addison-Wesley 1.989.

Sedgewick, Robert.
Algorithms. Second edition
Ed.: Addison-Wesley Publishing Company, 1.988.

Sommerville, Ian; Morrison, Ron
Software Development With Ada.
Ed.: International Computer Science Press, 1.987.

Weis, Mark Allen
Estructuras De Datos Y Algoritmos.
Ed.: Addison-Wesley Iberoamericana, 1.995.

Wilf, Herbert S; Bratley, Paul.
Algorithms And Complexity.
Ed.: Prentice-Hall International Editions, 1.986.