Programazio dinamikoa

R. Arruabarrena
LSI - UPV/EHU

Sarrera

- Goitik beherako diseinua naturala eta indartsua da
- Azpiproblema berdinen bidez soluzioa maiz lor liteke
- Ohiko inplementazioa: errekurtsioa

- errekurtsioaren murriztapenak
  - Azpiarazo bakoitza independenteki ebazten da
  - Azpiarazo bera, kalkulu bera, errepikatuz maiz
  - Alferrikako lana eta denbora
- Programazio dinamikoan
  - Azpiarazoen soluzioak metatu egiten dira
  - Aurrerago beharko balira, berrreskuratzeko

P. D.: Ezaugarriak

- Problema indukzio bidez definigarria izan behar du
- Azpiproblema desberdin bakoitza egitura batean posizio bat erlazionaturik izango du
- Azpiproblema bat ebazterakoan
  - 1.go aldian
    - azpiproblema indukzio bidezko ekuazioak dioen moduan kalkulatzen da
    - egituran hari dagokion posizioan metatzen da
  - Hurrengo alditan
    - Memoria egituran dagokion posiziotik berreskuratzen da balioa

P. D.: Ezaugarriak

- Egitura betetzeko ordenak

  Beti indukzio ekuazioak erabiliz

  - Iteratiboki gorantz: kasu nabarietatik hasiz
  - Errekurtsiboki beherantz ebatzitako kasuetaraino
- Optimizazio problemak ebazteko erabili ohi da
  - Bestelako problemak ebazteko ere balio dezake
  - **GAKOA**: **indukzio bidez definigarria izatea problema**

P. D. eta optimizazio problemen ezaugarriak

- Azpiegitura optimoa (Bellman-en optimotasun printzipioa)

```
Problema bat ebazten duen
erabakien sekuentzia optimo baten
edozein erabakien azpisekuentzia batek,
honek ebazten duen azpiproblemaren
sekuentzia optimoa ere izan behar du.
```

Problema batek azpiegitura optimo bat duela esango dugu baldin eta problemaren soluzioak azpiproblemen soluzio optimoak barne baditu.

(≡ Indukzio bidez definigarria)

Fibonacci

1. Indukzio ekuazio ezagunak:

- Fib(0)=1
- Fib(1)=1
- Fib(n)= Fib(n-1)+Fib(n-2)

Errekurtsio puruz

$$\Theta\left(\left(\frac{1+\sqrt{5}}{2}\right)^n\right)$$

P.D bidez:

2. Metatze egitura: TF(i)=x ≡ i. fibonacci zenbakia x da

TF

1	1			y	x	y+x	

Mee(n) ∈Θ (n)

3. Analisia: n. gelaxkako balio lortzeko (1..n-1) gelaxkak denbora konstantean beteko dira: **Θ(n)**

4. Iteratiboki: kasu nabarietatik interesatzen den kasu orokorra lortu arte, ekuazioek diotena inplementatuz

```
function Fib_PD (n: in Integer) is

 FT: array(1..n)of integer;          Tarteko emaitzak
                                     Metatzeko egitura
  i: integer=1;

begin
  FT(0)=1;                 Kasu nabariak.
  FT(1)=1;                 1 eta 2 ekuazioek markatzen duten balioak

  while (i<=n) do                    Kasu orokor bakarra.
     FT(i):=FT(i-2)+FT(i-1);         3 ekuazioak agintzen duen
     i=i+1;                          operazioak burutuz
  end loop;

  return FT(n);            Soluzioa
end;
```


4. Memoriadun funtzioekin: soilik "Bete"-ri deia baldin kalkulatu gabe badago ekuazioak eskatzen duen azpiarazoaren balioa

```
function Fib_MemoriadunFun (n: in Integer) is

  FT:array(1..n):=(0,1=>1; others=> -1);     Azpiarazoa kalkulatu
                                             gabe dago adierazteko
  proc  Bete (i: in Integer) is              Baldin azpiarazoa ez
  begin                                      dagoen kalkulatua,
                                             kalkulatu
   if (FT(i-2)=-1) then Bete(i-2); end if;   Soberan
   if (FT(i-1)=-1) then Bete(i-1); end if;

   FT(i)= FT(i-2)+FT(i-1);
  end                                        Berreskuratu jada
                                             kalkulatutako
begin                                        (tarteko) balioak
  if (FT(n)=-1) then Bete(n); end if;
  return FT(n);
End.
```

Motxila 0/1

E edukiera duen motxilaren balioa maximizatu nahi dugu 1..n objektuak bertan osorik sartuz, jakinik p(i) eta b(i) i objektuaren pisua eta balioa direla h. h.. (non i, $1 \leq i \leq n$)

Zein da motxilaren irabazi maximoa?

Optimizazio problema: ekuazioak honekiko

Zeintzuk objektuk ematen dute irabazi hori?

Errekuperazio problema, aurrekoa osatu ostean

Garapeneko 4 pausoak:

Ekuazioak ⇒ Metatze egitura ⇒ Analisia+Kodea

Motxila 0/1: Ekuazioak.

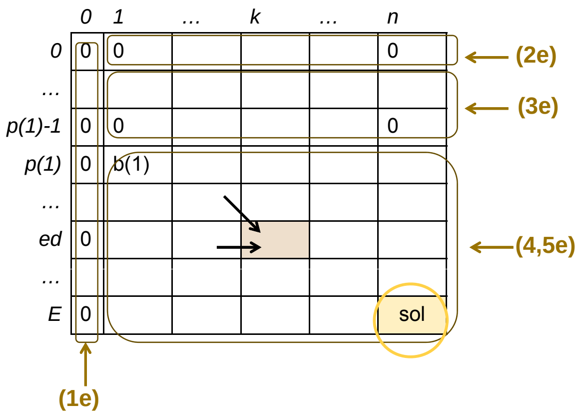
IrabaziaM (ed, k): motxilaren edukiera "ed" izanik eta 1,2,…,k objektuak eskuragarri digula, lor litekeen irabazi maximoa objektu horiek motxilan osorik sartuz

```
Kasu nabariak:
(1e) IrabaziaM (ed,0)= 0
(2e) IrabaziaM (0,k)= 0
(3e) IrabaziaM (ed,k)=0        if ed<p(1)   -- p(1)<....<p(N) baleude,
                                            -- bestela arinena

Kasu orokorrak
   IrabaziaM(ed,k)
(4e) = Max { b(k) + IrabaziaM (ed-p(k), k-1),
           IrabaziaM (ed-p(k), k-1) }        if  p(k)≤ed ∧ 1 ≤ K

(5e) = IrabaziaM (ed, k-1)                   if  p(k)>ed ∧ 1 ≤ K
```


Motxila 0/1: Metatze egitura.

O(n x E)

Mee(n,E) ∈Θ (n x E)



- Betetze ordena erabaki:

Iteratiboki:

egitura betetze ordena (zutabeka , lerroka, diagonalka, ezker->eskuin)

Memoriadun funtziokin:

balio lehenetsia eta dei errekurtsiboa(k) identifikatu

Memoriadun funtzioak

function Motxila01 (E, p(1..n),b(1..n))
TMotx(0..E,0..N)=(others= >(others => -1)) ***(kalkulatu gabea)***

```
  procedre Bete(Ed,K) is
  begin
    if TMotx(Ed,K-1)=-1 then Bete (Ed,K-1); end if;
    if p(K)<=Ed then
        if TMotx(Ed-p(K),K-1)=-1 then Bete (Ed-p(K),K-1); end if;
(4e)    TMotx(Ed,K):=MAX( b(K)+TMotx(Ed-p(K),K-1), TMotx(Ed,K-1));
(5e) else TMotx(Ed,K):= TMotx(Ed,K-1); end if;
  end;
begin
(1e) for Ed in 0..E loop TMotx(Ed,0):=0; end loop;
(2e) for K in 0..N loop TMotx(0,K):=0; end loop;
     for Ed in 1..p(1)-1 loop
(3e)     for K in 1..N loop TMotx(Ed,K):=0; end loop;
     end loop;
     if E>p(1) then Bete(E,N); end if;
     return TMotx(E,N);
end;
```

Procedure Motxila01PD (E, p(1..n),b(1..n)) **Elem(1..N),** Irabazia)
TMotx(0..E,0..N): float; **TObj(0..E,0..N) := (others=> (others => 0))** *(Ez sartu)*

Iteratiboa + errekuperazioa

```
begin
     for Ed in 0..E loop TMotx(Ed,0):=0; end loop;
(1e) for K in 0..N loop TMotx(0,K):=0; end loop;
(2e) for Ed in 1..p(1)-1 loop
          for K in 1..N loop TMotx(Ed,K):=0; end loop;
(3e) end loop;
   for K in 1..N loop
     for Ed in p(1)..E loop
(5e)    TMotx(Ed,K):= TMotx(Ed,K-1);
        if p(K)<=Ed and (Tmotx(Ed,K-1)< (b(K)+TMotx(Ed-p(K),K-1))
        then  TMotx(Ed,K):= b(K)+TMotx(Ed-p(K),K-1);
(4e)          Tobj(Ed,K):= 1;                              (Sartu)
         end if;
     Ed:=E;
     for K in 1..N loop  Elem(K):= TObj(Ed,K);
                         if (Elem(K)=1)then Ed:= Ed - p(K); end if;
     end loop;
     Irabazia:=TMotx(E,N);
end;
```


E=10 eta 4 objektu eskuragarri izanik, zein da motxilaren irabazi maximoa? Zeintzuk objektu sartu behar ditugu? Eman zaizun ekuazio sistema erabiliz osatu ondorengo taula

	1	2	3	4
p(i)	4	2	6	3
b(i)	6	10	12	8

K\Ed	0	1	2	3	4	5	6	7	8	9	10
0											
1											
2											
3											
4											

E=10 eta 4 objektu eskuragarri izanik, zein da motxilaren irabazi maximoa? Zeintzuk objektu sartu behar ditugu? Eman zaizun ekuazio sistema erabiliz osatu ondorengo taula

	1	2	3	4
p(i)	4	2	6	3
b(i)	6	10	12	8

K\Ed	0	1	2	3	4	5	6	7	8	9	10
0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	**0**	0	**6**	**6**	6	**6**	**6**	6	**6**
2	0	**0**	0	0	**10**	10	16	**16**	16	16	**16**
3	0	0	0	0	10	10	16	**16**	22	22	**22**
4	0	0	0	0	10	18	18	28	22	24	**24**

Txanponak. Konbinazio desberdin kopurua

$b_1, b_2, \ldots, b_n$ N txanponen klaseak emanik eta jakinik txanpona klase bakoitzeko behar hainbat txapona eskuragarri ditugula,
L kopuru finkoa batzen duten txanpon konbinazio desberdinen kopurua kalkulatu.

- Konbi(L,i)=L kopurua itzultzeko [1..i] txanpon klaseak erabiliz dagoen txanpon konbinazio kopurua

```
Kasu nabariak:
  konbi(0,i)= ?  -- kopururik ez bada itzuli behar,

  konbi(L,1)= ?
```

Konbinazioak.

- Konbi(L,i)=L kopurua itzultzeko [1..i] txanpon klaseak erabiliz dagoen txanpon konbinazio kopurua

```
Kasu nabariak:
  konbi(0,i)= 1  -- kopururik ez bada itzuli behar, konbinazio
                 -- bakarra dago: txanponik ez erabiltzea

 konbi(L,1)= 1   if L=b1, L>0
           = 0   if L≠b1, L>0  ( Deskonposa ezina(L))

Kasu orokorrak:
(A aukera)
   konbi(L,i)= konbi(L,i-1)+ konbi(L-bi,i)       if 0<bi≤L, 1<i
   konbi(L,i)= konbi(L,i-1)                      if 0<L<bi, 1<i

(B aukera)
   konbi(L,i)= konbi(L,i-1) + konbi(L-bi,i-1) + konbi(L-2bi,i-1)+…
               + konbi(L-p*bi,i-1)               if 1<i, p = L div bi
```


Konbinazioak.

- Konbi(L,i)=L kopurua itzultzeko [1..i] txanpon klaseak erabiliz dagoen txanpon konbinazio kopurua

```
Kasu nabariak:
  konbi(0,i)= 1  -- kopururik ez bada itzuli behar, konbinazio
                 -- bakarra dago: txanponik ez erabiltzea

 konbi(L,1)= 1   if L=b1, L>0
           = 0   if L≠b1, L>0  ( Deskonposa ezina(L))

Kasu orokorrak:
(A aukera)
   konbi(L,i)= konbi(L,i-1)+ konbi(L-bi,i)       if 0<bi≤L, 1<i
   konbi(L,i)= konbi(L,i-1)                      if 0<L<bi, 1<i

(B aukera)
   konbi(L,i)= konbi(L,i-1) + konbi(L-bi,i-1) + konbi(L-2bi,i-1)+…
               + konbi(L-p*bi,i-1)               if 1<i, p = L div bi
```


Konbinazioak. Metatze egitura eta kostua

C

Mee(n,E) $\in\Theta$ (n x E)

	1	*…*	*i*	*…*	*n*
0	1		1		1
…	0				
b1	1				
…					
2(b1)*	1				
…					
L					sol

Analisia

- i garren zutabea kalkulatzeko (i-1)garrena bakarrik erabiltzen denez, memoria espazio estra bi zutabeetara mugagarria da iteratiboki inplementatzen denean.
- Denbora ordena: Gelaxka garestiena x Gelaxka kopurua

(a aukera) **Θ (n x E)**

(b aukera) **O(n x E x m)** m= L div b1 b1<…<bn

Iteratiboa

(b aukera)

```
procedure  KonbDestPD (L: in Kopurua; B: in Txanponak;  S: out
   Natural) is
   M: Matrizea (0..L,1..N);    -- Sup. B(1)<...<B(N)
begin
   for  i in  1.. N  loop  M(0,i) := 1; end loop;
   for  C in 1 .. L loop
     if (C mod B(1)) =0 then M(C, 1):= 1; else M(C, 1):= 0;  end if;
   end loop;

   for  i  in  2 .. N  loop     -- Kasu orokorra
        for  C  in  1 .. L  loop
        KonbKop:= M(C, i-1);
        ZenbatAldiz:= C div B(K);
        for  P  in 1.. ZenbatAldiz  loop
                 KonbKop := KonbKop + M(C- (P* B(K)), K-1);
        end loop;
        M(C,i) := KonbKop;
       end loop;
   end loop;
   S:= M(L,N);
end KonbDestPD;
```

Diru itzulketa. Txanpon kopuru minimoa (TKM)

$b_1, b_2, \ldots, b_n$ N txanponen klaseak emanik eta jakinik txanpona klase bakoitzeko behar hainbat txapona eskuragarri ditugula,

L kopuru finkoa emango duten txanpon kopuru minimoa kalkulatuko duen algoritmo bat idaztea eskatzen da.

- L deskonposa ezina izan liteke: ***kasu horretan 0 txanpona itzultzea erabakitzen bada,*** **ez** da desberdinduko deskonposa ezina eta 0 kopurua itzultzeko 0 txanpon behar direla
- Suposatuz: b1<b2<...<bn y 10^d<=L< 10^{d+1}

TKM.

- (A ekuazio sistema)
  Konbi(L)=txanpon klase guztiak erabilgarri izanik, L kopurua itzultzeko behar diren txanpona kopuru minimoa

```
Kasu Nabariak:
TKM(0)  = 0
TKM(bi) = 1          ∃i, 1≤i<n, L= bi
TKM(L)  = ∞          1≤L<b1

Bestelako baldintzetan, kasu orokorra,:
TKM(L)= 1+ min {TKM(L-bi)  |bi ≤ L, 1≤i≤n}
```

- Metatze egitura
  Egitura: T(1..L); T(j) = j kopurua itzultzeko txanpon kopuru txikiena.
- Denbora-ordena: **O(nL) = O(n 10^d);**
  gelaxka bakoitza betetzeko gehienez ***n*** txanpona klase erabiliko dira

TKM. (B ekuazio sistema)

Konbi(L,i)= [1..i] txanpon klaseak erabilgarri izanik, L kopurua itzultzeko behar den txanpona kopuru minimoa

```
TKM(L,i) = ∞          not(Deskonposagarri (L)) - barne 0≤L<b1
TKM(0,i) = 0
TKM(bj,i) = 1          if L=bj, 0≤j≤i
TKM(L,1) = L div b1    if (L mod b1)=0, L>0

TKM(L,i)=min{TKM(L,i-1), 1+TKM(L-bi,i)} bi≤L,Deskonposagarri(L-bi)
        = konbi(L,i-1)                    bi>L,Deskonposagarri(L-bi)
```

- Metatze egitura: M(1..n, 0..L)
  M(i,j)= j kopurua itzultzeko txanpon kopuru txikiena jakinik soilik 1..i klaseko txanponak erabil litezkeela.
- Denbora-ordena: **O(2nL) = O(n 10^d)** M(i,j) betetzeko $\in$ O(1)

Ariketa:

L=21 kopurua itzultzeko b={2, 5, 10} txanpona klaseak edukiz, erabaki taulak osatuz, TKM eta existitzen duten Konbinazio kopurua.

- Konbi (c aukera)

	0	*1*	*2*	*3*	*4*	*5*	*6*	*7*	*8*	*9*	*10*	*11*	*12*	*13*	*14*	*15*	*16*	*17*	*18*	*19*	*20*	*21*
1																						
2																						
3																						

- TKM (b aukera)

	0	*1*	*2*	*3*	*4*	*5*	*6*	*7*	*8*	*9*	*10*	*11*	*12*	*13*	*14*	*15*	*16*	*17*	*18*	*19*	*20*	*21*
1																						
2																						
3																						

Ariketa:

L=21 kopurua itzultzeko b={2, 5, 10} txanpona klaseak edukiz, erabaki taulak osatuz, TKM eta existitzen duten Konbinazio kopurua.

- Konbi (c aukera)

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21
1	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0
2	1	0										1										2
3	1																					3

- TKM (b aukera)

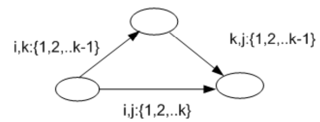
	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21
1	0	∞	1	∞	2	∞	3	∞	4	∞	5	∞	6	∞	7	∞	8	∞	9	∞	10	∞
2	0	∞					2					4					5					6
3	0	∞										4										5

Bide motzenak. Floyd.

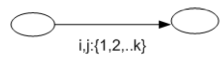
Izan bedi G grafo zuzendua eta pisuduna, non grafoko arku bakoitzak negatiboa ez den pisu bat erlazionaturik duen.

Grafoko erpin bikote bakoitzaren arteko distantzia minimoak mugatu.

- b=<e1,e2,...,ek> bideko tarteko erpinak:e2,...,ek-1
  i-tik, j-ra iristen eta {1,...,k} tarteko erpinak soilik erabiliz osa daitezkeen bide guztien artetik **b** distantzia motzena duena izan bedi.
  Grafoen teoriaren arabera, **b** bidean bi gauza gerta daitezke:



k ∈b



k ∉b

Floyd

- Indukzio ekuazioak

$d_{i,j}^{k}$ i-tik hasi, j-ra iristen eta {1,...,k} tarteko erpinak soilik erabiliz osa daitezkeen bide guztien artetik distantzia motzena duena jasotzeko ekuazio sistema honakoa da :

$$d_{i,j}^{k} = \begin{cases} pisua(i,j) & if\ k = 0 \\ \min\left(d_{i,j}^{k-1}, d_{i,k}^{k-1} + d_{k,j}^{k-1}\right) & if\ k \geq 1 \end{cases}$$

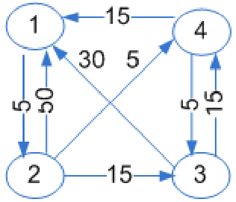
Adibidez: $d_{i,j}^{0}$ i-tik j-ra zuzenean, $d_{i,j}^{1}$ i-tik j-ra zuzenean ala 1 erpinak zehakatuz,…

- Metatze egitura
  Ez da metatze egitura estrarik behar
  k-ra hedatzerakoan matrizeko k lerroa eta k zutbeak ez dira eguneratzen.

Floyd.

```
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;
```

$\Theta(P^3)$



$D^0=$

	1	2	3	4
1	0	5	∞	∞
2	50	0	15	5
3	30	∞	0	15
4	15	∞	5	0

$D^1=$

	1	2	3	4
1				
2				
3				
4				

$D^2=$

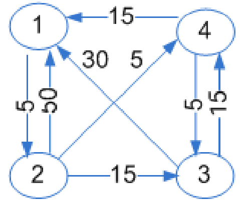
	1	2	3	4
1				
2				
3				
4				

$D^3=$

	1	2	3	4
1				
2				
3				
4				

$D^4=$

	1	2	3	4
1				
2				
3				
4				



$D^0=$

	1	2	3	4
1	0	5	∞	∞
2	50	0	15	5
3	30	∞	0	15
4	15	∞	5	0

$D^1=$

	1	2	3	4
1	0	5	∞	∞
2	50	0	15	5
3	30	**35**	0	15
4	15	**20**	5	0

$D^2=$

	1	2	3	4
1	0	5	**20**	**10**
2	50	0	15	5
3	30	35	0	15
4	15	20	5	0

$D^3=$

	1	2	3	4
1	0	5	20	10
2	**45**	0	15	5
3	30	35	0	15
4	15	20	5	0

$D^4=$

	1	2	3	4
1	0	5	20	10
2	20	0	**10**	5
3	30	35	0	15
4	15	20	5	0

Nilo.

Nilo ibaian N kai daude, bakoitzean Y ontzi aloka daitezkeelarik ibaian behera dauden beste kaietara joateko (ibaian gora ezin da joan, korronte handia du eta ibaiak). Edozein A abiapuntu-kaitik ibaian behera dagoen edozein H kaira iristeko ontzi bakar bat alokatzea garestiagoa gerta liteke Atik A+1ra, A+1etik A+6ra eta A+ 6tik H-ra joatea baino, adibidez.

A abiapuntu-kai guztietatik H helmuga-kai guztietara joateko bidai kostu posible guztien artetik bidai kostu 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.

- Datuak

Zuzenean (i,j)= i kaitik ibaian beheran dagoen j kaira zuzenean iristeko ontzi baten alogera kostua.

```
Zuzenean (i,j)=?      j<i (j ibaian goran dagoenean)
Zuzenean (i,i)= 0     kaitik bertara joateko alogera kostua 0
```


Nilo. Ekuazio sistema

Alogera(i,j)= i kaitik j ibaian beheran dagoen kaira iristeko ontzien alogeraren kostu minimoa (zuzenean ala ez)

```
Alogera(i,j)
  = 0                                 i=j
  = Zuzenean(i,j)                     i+1=j
  =min i<k<j {Zuzenean (i,j), Alogera(i,k)+Alogera(k,j)}      i+1<j   (A)

  =min i<k<j {Zuzenean (i,j), Zuzenean (i,k)+Alogera(k,j)}    i+1<j   (B)
```

- Metatze egitura: Alogera(1..n, 1..n) eta emaitza da
- Analisia
  - *MEE*: Ez du espazio estrarik erabiltzen, “Alogerak” matrizea itzuli behar baita osorik.
  - *Denbora ordena*: ($n^2/2$) gelaxka bete behar dira. Denbora gehien kalkulatzea kostatzen den gelaxka Alogerak(1,N) da, Q(n) izanik bere denbora ordena. Ondorioz, **$O(n^* (n^2/2))=O(n^3)$.**

Iteratiboki: (B) beheko lerroetatik goiko lerroetara; lerro barruan, ezkerretik eskuinera

```
procedure Nilo (Zuzen: in Matrizea; Alogerak: in out Matrizea) is
begin
for Ontzi in 1..n-1 loop            -- kasu nabariak
   Alogerak(Ontzi,Ontzi):=0;
   Alogerak(Ontzi,Ontzi+1):= Zuzen(Ontzi,Ontzi+1);
end loop;
Alogerak(n,n):=0;

for Irten in reverse 1..n-2 loop     -- kasu orokorrak
   for Iritsi in Irten+2..n loop
      Merkeena:= Zuzen(Irten,Iritsi);
      for K in Irten+1..Iritsi-1 loop
           if Zuzena(Irten,K)+Alogerak(K,Iritsi)<Merkeena
           then Merkeena:= Zuzena(Irten,K)+Alogerak(K,Iritsi);
           end if;
           Alogerak(Irten,Iritsi):=Merkeena;
      end loop;
   end loop;
end.
```

(18,20)
(17,19),(17,20)
(16,18)(16,19)(16,20)
…
(1,3),(1,4),(1,5),…,(1,20)

Errekurtsiboki: (B ekuazio sistema)

```
procedure Nilo (Zuzen: in Matrizea; Alogerak: in out Matrizea) is
...
   prodedure  Bete (I,J: in Indizea) is
   begin
      Merkeena:= Zuzen(I,J);
      for K in I+1..J-1 loop
        if  Alogerak(K,J)=-1 then Bete(K,J); end if;
        if Zuzen(I,K)+Alogerak(K,J)<Merkeena
        then  Merkeena:= Zuzen(I,K)+Alogerak(K,J);
        end if;
      Alogerak(I,J):=Merkeena;
   end;

begin
```



```
begin
   -- balio lehenetsia, matrize osoa, sinpleagoa baita
   for Irten in 1..n loop
        for Iritsi in 1..n loop
                Alogerak(Irten,Iritsi):=-1;
        end loop;
   end loop;

   -- kasu nabariak
   for Ontzi in 1..n-1 loop
        Alogerak(Ontzi,Ontzi):=0;
        Alogerak(Ontzi,Ontzi+1):= Zuzen(Ontzi,Ontzi+1);
   end loop;
   Alogerak(n,n):=0;

   -- kasu orokorrak
   for Iritsi in 3..n loop           ←KONTUZ: Denak behar dira
        Bete(1,Iritsi);              -- in edo in reverse
   end loop;
end.
```


Unibertsitateko irakasleen kontratazioa.

Unibertsitate bateko sail batek IK ikastordu eman behar ditu. Horretarako, N klaseko desberdinetako irakasleak kontrata ditzake. I klaseko irakasle bakoitzak Hi ordu irakatsi ditzake gehienez lan horregatik beti Pi euro eskuratuz. IK klaseak emateko eta ordaindu beharreko euro kopurua minimoa izan dadin, zenbat irakasle kontratatu beharko liratekeen klase bakoitzeko kalkulatuko duen algoritmo bat idatzi eta soluzioaren denbora ordena eta memoria espazio estra kalkulatu.

Ekuazioak:

Ordaindu(k,j)= K ordu irakasteko eta [1..j] klaseetako irakasleak kontratagarriak izanik, unibertsitateak gutxienez ordaindu beharko lukeena.

1 aukera

Ordaindu(k,j)= K ordu irakasteko eta [1..j] klaseetako irakasleak kontratagarriak izanik, unibertsitateak gutxienez ordaindu beharko lukeena.

Kasu nabariak:

```
Ordaindu(0,j)=0        -- Ez da klaserik eman behar
K>0 kasua:
Ordaindu(K,1) = P(1) * (K div H(1))          if K mod H(1)=0
        -- 1 klaseko irakasle guztiek H1 ordu irakatsiko dute
              = P(1) * ((K div H(1))+1)      if K mod H(1) ≠0
        -- 1 klaseko azkeneko irakasle kontratatuak ez
        -- ditu Ord(1) ordu irakatsiko, gutxiago baizik

Kasu orokorra: K>0, J>1
Ordaindu(K,j)
= min {Ordaindu (K,j-1),
      P(j)+Ordaindu (K-H(j),j-1),
      2*P(j) + Ordaindu (K-2*H(j),J-1),...,
      d*P(j) +Ordaindu (K-d*H(j),J-1),
      (d+1)* P(j)}     d=K div H(j) eta (d+1) kasua soilik 0≠K mod H(j) bada
```


2 aukera

Ordaindu(k)= K ordu irakasteko eta n klaseetako irakasleak kontratagarriak izanik, unibertsitateak gutxienez ordain dezakeena.

```
Ordaindu(K) = 0                                   if K≤0
        =min 1≤j≤n {P(j)+Ordaindu (K-H(j))}       if 0<K
```


Azoka.

K euro sakelan izanik azokara goaz. Honekin batera eros ditzakegun **m** produktuen zerrenda daramagu. *i* produktu bakoitzeko ($1 \le i \le n$) p_i prezioa eta e_i erabilgarritasun balioa (biak osoko positiboak) ezagunak ditugu. Produktu bakoitzeko gehienez 3 produktu eros ditzakegu. Gainera, eguneko eskaintzari esker, produktu beraren bigarren unitatearen salneurria 1 euro gutxiago kostatuko zaigu eta hirugarren unitatea, aldiz, 2 euro gutxiago. Erosketaren erabilgarritasuna erositako produktu bakoitzaren erabilgarritasunaren batura dela jakinik, *Programazio Dinamikoaren* teknika aplikatuz algoritmo bat idatz ezazu gehienez K euro azokako produktuetan gastatuz lor dezakegun erabilgarritasun maximoa mugatzeko bai eta erosi behar den **elementuen zerrenda** ere. Proposatutako soluzioaren denbora-ordena eta erabilitako memoria espazio estra kalkula ezazu.

Ekuazioak:

- MerkaAzoka (E,i) = lor dezakegun erabilgarritasun maximoa E euro gastatuz soilik [1...i] produktuetan (eta gehienez produktu bakoitzeko 3 unitate erosiz).

MerkaAzoka (E,i) = lor dezakegun erabilgarritasun maximoa E euro gastatuz soilik [1...i] produktuetan (eta gehienez produktu bakoitzeko 3 unitate erosiz).

MerkaAzoka (0,i) =0

MerkaAzoka (d,1) =0 if $d < p_1$

= e_1 if $p_1 \le d < 2p_1$

= $2e_1$ if $2p_1 \le d < 3p_1$

= $3e_1$ if $3p_1 \le d$

MerkaAzoka (d,i) = MerkaAzoka(d,i-1) if $d < p_1$

=max{ MerkaAzoka(d,i-1),
e_1 + MerkaAzoka(d-p_i, i-1)} if $p_i \le d < 2p_i$

=max{ MerkaAzoka(d,i-1),
e_1 + MerkaAzoka(d-p_i, i-1)
$2e_i$ + MerkaAzoka(d-$2p_i$-1, i-1)} if $2p_i \le d < 3p_i$

=max{ MerkaAzoka(d,i-1),
e_1 + MerkaAzoka(d-p_i, i-1)
$2e_i$ + MerkaAzoka(d-$2p_i$-1, i-1)
$3e_i$ + MerkaAzoka(d-$3p_i$-2, i-1)} if $3p_i < d$