Diseño de algoritmos

Jesús Bermúdez de Andrés. UPV-EHU
Ejercicios: Divide y vencerás

Curso 2008-09

1. Para resolver cierto problema se dispone de un algoritmo trivial cuyo tiempo de ejecución $t(n)$ es cuadrático (i.e. $t(n) \in O(n^2)$). Se ha encontrado una estrategia *Divide y Vencerás* para resolver el mismo problema; dicha estrategia realiza $D(n) = n \lg n$ operaciones para dividir el problema en dos subproblemas de tamaño mitad que el original y $C(n) = n \lg n$ operaciones para componer una solución del original con la solución de dichos subproblemas. ¿Es la estrategia *Divide y Vencerás* más eficiente que la empleada en el algoritmo trivial?

2. Escriba un algoritmo para ordenar una lista enlazada de elementos utilizando la estrategia de ORDEN_POR_MEZCLA. Analice su algoritmo en tiempo y espacio y compárelo con la versión clásica utilizada para ordenar vectores.

3. Este ejercicio propone estudiar algunas variantes del algoritmo de ordenación por mezcla.

   a) Supóngase que queremos utilizar el algoritmo de ordenación por mezcla, pero dividiendo el vector en tres tercios, ordenando cada trozo recursivamente y mezclándolos después de ordenados. Escriba y analice esa versión del algoritmo, que denominaremos ORD_MEZCLA_3.

   b) Generalice el análisis para el caso de dividir el vector en $k \geq 3$ porciones de tamaño similar, ordenar recursivamente y mezclar las k porciones ordenadas.

   c) ¿Qué conclusiones puede extraer de lo anterior?

   d) Considere un algoritmo ORD_MEZCLA_RAÍZ que divida el vector en $\lfloor\sqrt{n}\rfloor$ trozos de tamaño $\lfloor\sqrt{n}\rfloor$ aproximadamente. Estudie con cuidado cómo debería ser el algoritmo de mezcla de los $\lfloor\sqrt{n}\rfloor$ trozos ordenados. Analice el algoritmo ORD_MEZCLA_RAÍZ y compárelo con los anteriores.

   e) Llevando al extremo la idea de dividir el vector cada vez en más porciones, para aplicar la idea de ordenación por mezcla. Considere un algoritmo que divida el vector en $n/2$ trozos de tamaño 2.

   Diseñe y analice un algoritmo siguiendo esta variante.

4. Sea $T[1 \ldots n]$ un vector de enteros distintos ordenado de menor a mayor, algunos de los cuales pueden ser negativos. Diseñe un algoritmo que determine un valor $i \in \{1 \ldots n\}$ tal que $T[i] = i$, siempre que dicho i exista. Debe emplearse un tiempo $o(n)$.

5. Sea $T[1..n]$ un vector de números naturales distintos. Una pareja de valores $(T[i], T[j])$ es una *inversión* si no respetan el orden, es decir si $i < j$ pero $T[i] > T[j]$.

   Escriba un algoritmo de $O(n \lg n)$ que calcule el número de inversiones de un vector $T[1..n]$.

6. Sea $F(x)$ una función monótona decreciente y sea n el mayor entero que cumple $F(n) \geq 0$. Asumiendo que n existe, un algoritmo para determinar dicho n es el siguiente:

   $x \leftarrow 0$
   while $F(x) = 0$ **loop**
     $x \leftarrow x + 1$
   end loop
   return $x - 1$

   Compruébese que esta solución tiene complejidad $O(n)$. Escríbase un algoritmo cuyo comportamiento asintótico sea mejor en función de n.

7. Considere el siguiente algoritmo de ordenación:

   proc OrdenSolapa($V,\ i,\ j$)
     $k \leftarrow \lfloor \frac{j-i+1}{3} \rfloor$
     if $V[i] > V[j]$ **then** Intercambiar($V[i],\ V[j]$) **end if**
     if $j - i + 1 > 2$ **then**
       OrdenSolapa($V,\ i,\ j - k$)
       OrdenSolapa($V,\ i + k,\ j$)
       OrdenSolapa($V,\ i,\ j - k$)

   - Determine el tiempo de ejecución y el espacio extra empleado por OrdenSolapa (en el peor caso) para ordenar un vector de tamaño n.
   - Compare el tiempo de ejecución de OrdenSolapa con el tiempo de ejecución de los algoritmos de ordenación Orden_Por_Inserción, Orden_Por_Mezcla y Orden_Por_Montículo, para un vector de tamaño n.

   (Nota: Para analizar el tiempo de ejecución de OrdenSolapa, considere únicamente valores de n de la forma $(\frac{3}{2})^k$. Obsérvese que $\lg_3 2 = 0{,}63092$.)

8. Un mínimo local de un array $A(a-1..b+1)$ es un elemento $A(k)$ que satisface $A(k-1) \geq A(k) \leq A(k+1)$. Suponemos que $a \leq b$ y $A(a-1) \geq A(a)$ y $A(b) \leq A(b+1)$; estas condiciones garantizan la existencia de algún mínimo local. Escriba un algoritmo que encuentre algún mínimo local de $A(a-1..b+1)$ y que sea sustancialmente más rápido que el evidente de $O(\mathrm{n})$ en el caso peor. ¿De qué orden es su algoritmo?

9. Un vector T contiene n números naturales distintos. Queremos un algoritmo que *imprima* los m números menores de T. Sabemos que m es mucho menor que n. ¿Cómo lo haría para que resultara más eficiente?

   - Ordenar T usando ORDEN_POR_MEZCLA o bien ORDENACIÓN_RÁPIDA y luego imprimir los m primeros.
   - Usar algún otro método.

   Estudie si cambiaría algo para los casos particulares: $m \in \Theta(\sqrt{n})$ y $m \in \Theta(\lg n)$.

10. Sea un vector $T(1 \ldots n)$ de números enteros, que representa n temperaturas tomadas en sucesivos intervalos de un minuto de tiempo. Diseñe un algoritmo que use la técnica *Divide y Vencerás* para encontrar el menor intervalo de tiempo en el que todas las temperaturas registradas son de valor bajo cero. Es decir, si hay temperaturas bajo cero, queremos los índices a y b del intervalo $T(a \ldots b)$ de menor número de elementos de $T(1 \ldots n)$ tal que $(a \leq x \leq b \rightarrow T(x) < 0) \wedge (a > 1 \rightarrow T(a-1) \geq 0) \wedge (b < n \rightarrow T(b+1) \geq 0)$. Si no hay temperaturas bajo cero en $T(1 \ldots n)$, basta una pareja de valores 0 para indicar esa circunstancia. ¿Cree que ese algoritmo es óptimo? Justifique su respuesta.

11. El *diámetro* de un árbol es la longitud del camino más largo entre cualquier par de nodos. Diseñe un algoritmo, lineal en el número de nodos, que calcule el *diámetro* de un árbol.