Algorísmia Q1 2020–2021

Examen Parcial **Una solució** **4 de novembre de 2020**

Durada: 1:30

Instruccions generals:

- Entregueu per separat les solucions de cada exercici (Ex 1, Ex 2, Ex 3 i Ex 4).
- Heu d'argumentar la correctesa i l'eficiència dels algorismes que proposeu. Per fer-ho podeu donar una descripció d'alt nivell de l'algorisme suficient per tal que, amb les explicacions i aclariments oportuns, justifiqueu que l'algorisme és correcte i té el cost indicat.
- Podeu fer crides a algorismes que s'han vist a classe, però si la solució és una variació, n'haureu de donar els detalls.
- Es valorarà especialment la claredat i concisió de la presentació.
- La puntuació total d'aquest examen és de **10 punts**.

Exercici 1 (3.5 punts). Ja sabeu que fer la fusió ordenada de dues seqüències ordenades d'm i n elements, respectivament, comporta fer $m + n$ moviments de dades (penseu, per exemple, en un *merge* durant l'ordenació amb *mergesort* d'un vector). Però si hem de fer la fusió d'N seqüències, dos a dos, l'ordre en què es facin les fusions és rellevant. Imagineu que tenim tres seqüències A, B i C amb 30, 50 i 10 elements, respectivament. Si fusionem A amb B i després el resultat el fusionem amb C, farem $30 + 50 = 80$ moviments per a la primera fusió i $80 + 10 = 90$ per a la segona, amb un total de 170 moviments. En canvi, si fusionem primer A i C i el resultat el fusionem amb B farem un total de 130 moviments.

Dissenyeu un algorisme golafre (*greedy*) per fer les fusions i obtenir la seqüència final ordenada amb mínim nombre total de moviments. Justifiqueu la seva correctesa i calculeu-ne el cost temporal del vostre algorisme en funció del nombre de seqüències N.

Solució:

El algoritmo voraz que propongo utilizará la misma regla voraz que el algoritmo de Huffman, *fusionar las dos listas con menor número de elementos*

Después de fusionar dos listas tenemos una nueva lista con los elementos de las dos listas ordenados.

Para aplicar el criterio voraz utilizaremos una cola de prioridad Q donde guaradaremos las listas a fusionar en orden creciente de tamaño.

```
Insert in Q the N lists
while Q has more than two elements do
  a= Q.extract-min()
  b= Q.extract-min()
  c= Merge(a,b)
  Q.insert(c,c.size())
return  Q.extract-min()
```

En el análisis del coste del algoritmo nos piden el coste en función del número de secuencias, por ello asumo que el merge tiene coste $O(1)$. En una ejecución real el coste de todos los merges realizados por el algoritmo será el mínimo posible.

En cada iteración del algoritmo el número de listas se reduce en una unidad, así tenemos $N-1$ iteraciones. En total realizamos $O(N)$ extract-min y $O(N)$ inserts en la cola. Utilizando una implementación de cola con un Heap el coste es $O(N \log N)$.

Para demostrar que el algoritmo voraz proporciona una solución con número mínimo de movimientos, observemos que la solución calculada por el algoritmo se puede representar (como en Huffmann) con un árbol enraizado. En este árbol las hojas son las secuencias a fusionar. Cada nodo interno representa una de las fusiones dos secuencias que realiza el algoritmo. Observemos que los elementos de una secuencia a participan en todas las fusiones que se realizan en el camino desde su hoja hasta la raíz del árbol. Así si h_a es la altura de la secuencia a, el número de movimientos que produce a es $|a|h_a$.

Supongamos que en una solución óptima las dos secuencias a, b, $|a| \leq |b|$) con menor número de elementos no son hojas contiguas a la mayor porfundidad posible ($h_{\max}$). Si las dos secuencias a profundidad $h_{\max}$ son c, d con $|c| \leq |d|$. Si $c \neq a$, intercambiando a con c, tenemos

$|a|h_{\max} + |c|h_a \leq |a|h_a + |c|h_{\max}$, ya que $|a| \leq |c|$ y $h_a \leq h_{\max}$. Después de intercambiar $a$con c en el árbol la nueva solución también es óptima.

Si $b \neq d$, intercambiamos además b con d y por el mismo motivo la solución después de intarcambiar b con c es una solución óptima. Como siempre tenemos una solución óptima en la que el mínimo y el segundo mínimo se fusionan a la máxima profundidad el algoritmo es correcto.

Exercici 2 (2 punts). Hi ha un concurs de TV amb n participants on cada participant escull un enter entre 0 i 1000000. El premi és per als dos concursants que escullen els enters més propers. Dissenyeu un algorisme que, en temps lineal, li digui al presentador quins són els dos concursants guanyadors.

Solució:

Assumim que els n enters triats pels participants es troben emmagatzemats en un vector (si no fos així, els copiaríem en temps lineal). Per saber quins són els dos concursants guanyadors podríem:

1. Ordenar els n números triats pels concursants fent servir RADIX-SORT.
2. Recórrer la llista ordenada del punt anterior, guardant quin parell (x, y) de números consecutius té menor diferència entre ells (un qualsevol en cas d'empat).
3. El concursant que havia triat x (un qualsevol d'ells si més d'un ho havia fet) i el concursant que havia triat y (ídem) són els guanyadors.

Considerant la base (radix) $k = 10$, els $10^6 + 1$ números entre els quals han de triar els concursants es poden representar amb $d = \lceil \log_{10}(10^6 + 1) \rceil = 7$ dígits. Aleshores, el RADIX-SORT del pas 1 tindria un temps d'execució de $O(d(n + k)) = O(7(n + 10)) = O(n)$. El recorregut del pas 2 també té cost $O(n)$. Finalment, recuperar quins eren els concursants que havien proposat els números del parell guanyador també es pot fer en temps $O(n)$, ja sigui fent un recorregut addicional pel vector d'entrada, o fent que el vector inicial ja inclogui les tuples d'informació {concursant, número}. Per tant, en total, l'algorisme proposat requereix temps d'execució lineal, tal i com demanava l'enunciat.

Podríem optar per fer servir COUNTING-SORT en comptes de RADIX-SORT però observeu que aleshores el cost del punt 1 seria $O(n+k)$, essent $k = 10^6$. Continua essent lineal (perquè $k = O(1)$) però és una constant que pot ser significativament gran. Tanmateix, els requeriments d'espai per a fer l'ordenació també serien molt majors.

Per demostrar la correctesa, considerem els següents casos:

- La seqüència d'entrada no té repeticions:
  L'algoritme és correcte perquè el parell de números més propers de la seqüència d'entrada sempre es trobarà en posicions consecutives si considerem la seqüència ordenada: Sigui S la seqüència d'entrada, i suposem que $d = |a - b|$ és la mínima distància entre tot parell de números de S. Sigui S' la seqüència S ordenada, i siguin i i j les posicions dels elements a i b a S', respectivament, (és a dir, $S'[i] = a$ i $S'[j] = b$). Si $|i - j| > 1$, aleshores $\exists k : i < k < j : a < S'[k] < b$ i, per tant, $|a - S'[k]| < d$ i $|b - S'[k]| < d$, la qual cosa contradiu la consideració de d com a distància mínima.
- La seqüència d'entrada té repeticions:
  L'algoritme és correcte perquè sempre existirà com a mínim un parell de concursants guanyadors que es trobaran en posicions consecutives a la seqüència ordenada. Noteu que el parell guanyador serà un format per concursants que han triat el mateix número i que tots els jugadors que trien el mateix número estaran consecutius a la seqüència ordenada. L'algoritme, doncs, també donarà una resposta correcta.

Exercici 3 (2 punts) Suposeu que comencem un procés dinàmic per formar un graf G a partir d'un conjunt V de n vèrtexs i una seqüència d'arestes S donats. A cada pas del procés se'ns proporciona una nova aresta del graf, fins a introduir les m arestes de la seqüència $S = \{e_1, e_2, \ldots, e_m\}$. Així doncs, al llarg del procés tenim una seqüència de grafs $G_0, G_1, \ldots, G_m$, on $G_0 = (V, \emptyset)$ i al pas t, obtenim G_t afegint l'aresta e_t a G_{t-1}.

Doneu un algorisme, tan eficient com pugueu, per a poder obtenir el nombre de components connexes a cada pas de procés.

Solució:

Para resolver este problema tengo en cuenta las siguientes propiedades de las componentes conexas (cc):

- Las componentes conexas de un grafo forman una partición de sus vértices.
- Al añadir una arista (u, v) en el paso i:
  - si u y v están en la misma cc en G_{i-1}, las cc de G_i y de G_{i-1} son las mismas.
  - si u y v están en diferentes cc, $C_u \neq C_v$, G_i tiene una cc menos que G_{i-1} ya que se forma la cc $C_u \cup C_v$, y se mantienen el resto de cc.
- G_0 no tiene arista, así tiene n cc cada una de ellas formada por un solo vértice.

Teniendo en cuenta estas propiedades podemos utilizar una ED de Union-Find, ya que nos permite mantener particiones dinámicas. Utilizaremos los métodos de la ED para implementar las operaciones que hemos descrito arriba. Garantizando que al finalizar la iteración i la ED mantiene la partición correspondiente a las cc de G_i.

El algoritmo mantiene un contador CC con el número de cc de cada grafo.

- Par inicialiazar la ED hacemos un MakeSet(u), para $u \in V$. $CC = n$.
- Al añadir $e = (u, v)$
  $a = Find(u)$; $b = Find(v)$
  if $a \neq b$ **then**
  $Union(a, b)$
  $--CC$

Las operaciones en la ED se corresponden con las operaciones sobre las cc, por lo que el algoritmo es correcto.

En cuanto al coste, realizamos n Make-Sets y un total de $O(n+m)$ Find y Union. Utilizando la implementación con mejor coste amortizado, tenemos coste $O(n + \alpha(n)m)$. α es la inversa de la función de Ackermann, que como se ha comentado en clase se puede considerar constante en casos prácticos.

Exercici 4 (2.5 punts). Tenim un vector $A = (a_1, \ldots, a_n)$ d'elements d'un conjunt sobre els quals s'ha definit una relació d'ordre, i un vector $R = (r_1, \ldots, r_p)$ d'enters i ordenat, amb $1 \leq r_1 < r_2 < \ldots < r_p \leq n$.

Proporcioneu un algorisme que, donats A i R, i amb cost $o(pn)$, trobi l'r_1-èsim, r_2-èsim, ..., r_p-èsim del vector A. Justifiqueu la correctesa de l'algorisme proposat i el seu cost en funció de n i de p.

Solució:

El problema es pot resoldre trivialment ordenant tot el vector A (cost d'aquesta solució: $\Theta(n \log n + p)$) o bé invocant p vegades un algorisme de selecció amb cost lineal en cas pitjor, per a obtenir l'r_1-ésim, r_2-ésim, etc. successivament (cost d'aquesta solució: $\Theta(n \cdot p)$). Cap d'aquestes solucions es considera vàlida ja que no satisfà el requeriment de ser $o(p \cdot n)$.

Un refinament és aplicar l'algorisme de selecció iterativament, però per trobar l'r_{i+1}-ésim en el subvector $A[r_i + 1..n]$ donat que el pas previ ha particionat A. Això és una petita millora, però insuficient. Per exemple si $r_i \approx i \cdot (n/p)$ llavors el cost de l'algorisme és

$$\begin{aligned} S(n,p) &= \sum_{i=1}^{p} \Theta\left(n - i \cdot n/p\right) \\ &= \Theta(n/p) \sum_{i=1}^{p} \Theta(p-i) = \Theta\left((n/p) \cdot p^2/2\right) = \Theta(p \cdot n). \end{aligned}$$

Un possible solució és fer un algorisme MULTISELECT recursiu tal que si $p > 1$ agafa el rang central $r_{p/2}$, el selecciona fent servir un algorisme de selecció en temps lineal i a continuació es fan dues crides recursives a MULTISELECT: una per buscar els rangs $(r_1, \ldots, r_{p/2-1})$ en el subvector $A[1..r_{p/2} - 1]$ i l'altra per buscar els rangs $(r_{p/2+1}, \ldots, r_p)$ en el subvector $A[r_{p/2} + 1..n]$. La recurrència del cost per a quest algorisme seria

$$M(n,p) = \Theta(n) + M(n', p/2) + M(n'', p/2),$$

ón $n' = r_{p/2} - 1$ i $n'' = n - r_{p/2}$. No cal resoldre-la. Considerem l'arbre de crides recursives. És un arbre quasi-complet, a l'arrel comencem amb p rangs, a les arrels dels fills esquerre i dret tenim $\lfloor p/2 \rfloor - 1$ rangs i $\lceil p/2 \rceil$ rangs, respectivament, etc. A cada node X_i ($0 \leq i < 2^k$) del nivell k d'aquest arbre es resol un problema de selecció amb un subvector de talla n_i i tenim $\sum_i n_i = n$. Llavors el cost de resoldre tots els problemes de selecció al nivell k és $\sum_i \Theta(n_i) = \Theta(n)$, sigui quin sigui k. Com que l'arbre té $\lg p$ nivells el cost de l'algorisme és $\Theta(n \cdot \log p)$. Aquest algorisme sí satisfà els requisits de l'enunciat.

Una altra psosible solució consisteinx en un algorisme recursiu MQUICKSELECT que a cada crida recursiva fa el següent:

1. Escull un pivot x i particiona el vector A respecte a x. Sigui k la posició final del pivot.
2. Busquem (amb cerca dicotòmica i cost $O(\log p)$) el rang r_i tal que $r_i \leq k < r_{i+1}$. Adoptem el conveni $r_0 = 0$.
3. Cridem recursivament MQUICKSELECT en el subvector $A[1..k]$ per trobar els rangs $(r_1, \ldots, r_i)$ (si $r_i < k$) o $(r_1, \ldots, r_{i-1})$ (si $r_i = k$).

4. Cridem recursivament MQuickSelect en el subvector $A[k+1..n]$ per trobar els rangs $(r_{i+1}, \ldots, r_p)$.

L'anàlisi del cost d'aquest algorisme segueix el mateix raonament que abans. Però l'arbre de crides recursives no és ara quasi-complet, és un arbre binari qualsevol. LLavors el nombre de nivells en cas pitjor és p i per tant el cost és $\Theta(p \cdot (n + \log p)) = \Theta(p \cdot n)$. Però en promig anirem divint més o menys per la meitat el vector de rangs, de manera semblant al que fa per exemple quicksort i es pot demostrar que en promig hi hauràn $\Theta(\log p)$ nivells i per tant el cost en cas promig d'aquest algorisme serà $\Theta(\log p(n+\log p)) = \Theta(n \cdot \log p)$. Tot i que aquesta solució no satisfà el requisit en cas pitjor, sí ho fa en cas promig, i es considera una solució raonable. De fet a la pràctica pot funcionar millor que l'anterior doncs el factor amagat en el cost $\Theta(n)$ d'aquesta solució és molt més petit que el correponent de la solució anterior. Dit d'una altra manera: el cost $\Theta(n)$ per nivell de recursió de la primera solució és molt més elevat que el cost $\Theta(n)$ per nivell de la segona solució. Això sí, la segona solució tindrà més (p.e. $5 \log p$) o molts més (p.e. p) nivells.