Algorísmia Q1 2023–2024

Examen Parcial **27 d'octubre de 2023**

Durada: 1h 50min

---

1. **[Exercici 1] (3 punts)**:

   Tenim el següent problema:

   L'entrada és una seqüència de nombres diferents $S = \{x_1, x_2, \ldots, x_n\}$, tal que cada element $x_i \in S$ té associat un pes $w(x_i) > 0$. Sigui $W = \sum_{x_i \in S} w(x_i)$ la suma dels pesos de tots els elements de la seqüència.

   Donat un valor X $(0 \leq X \leq W)$, el problema consisteix a trobar l'element x_j de la seqüència tal que

   $$\sum_{x_i < x_j} w(x_i) < X \qquad \text{i} \qquad w(x_j) + \sum_{x_i < x_j} w(x_i) \geq X$$

   Dissenyeu un algorisme eficient $(o(n \log n))$ per a resoldre aquest problema.

   Una solució:

   Este problema se denomina problema de la selección ponderada (weighted selection) y es una generalización del problema de la selección. Vamos a plantear una solución basada en el algoritmo QuickSelect para encontrar ese elemento x_j. En cada paso (o llamada recursiva) del algoritmo realizaremos lo siguiente.

   1. Utilizaremos el algoritmo QUICK_SELECT(S,$n/2$) para particionar los elementos y calcular la posición (pos) de la mediana.
   2. Sumaremos los pesos $w(x_i)$ de los elementos a la izquierda de la mediana (sum).
   3. Dependiendo de la suma de pesos ($X \leq sum$) o ($X > sum$), procederemos recursivamente a seleccionar/buscar el elemento x_j en la mitad inferior o superior, respectivamente.

   Una codificación posible de nuestro algoritmo se muestra en el siguiente pseudocódigo.

   function WEIGHTED_SELECT(S, low, high, X)
   ▷ 1. QuickSelect to partition S[low..high] and compute position of the median
   $pos \leftarrow$ QUICK_SELECT(S, (low + high)/2)
   ▷ 2. Sum the lower elements
   sum $\leftarrow \sum_{i=low}^{pos} w(x_i)$
   ▷ 3. Recursively call weighted select on one half
   if $X \leq$ sum **then**
   if $pos ==$ low **return** pos
   return WEIGHTED_SELECT(S, low, $pos - 1$, X)
   else
   if $pos ==$ high **return** $pos + 1$
   return WEIGHTED_SELECT(S, $pos + 1$, high, $X -$ sum)

```
    end if
  end function
```

El coste de nuestro algoritmo dependerá del coste de QuickSelect y de cuan "bueno"sea escoger la mediana como pivote. Por un lado, sabemos que una buena implementación de QuickSelect tendrá coste $\Theta(n)$. Por otro, escoger la mediana como pivote en cada paso divide el vector en dos subvectores cada uno de los cuales contiene un fracción del tamaño original del vector $(n/2)$. Por tanto, el coste de nuestro algoritmo será $C(n) = \Theta(n)$.

Comentario:

Conceptualmente, podríamos haber planteado una sencilla solución basada en (1) ordenar los elementos de S y (2) sumar sus pesos $w(x_i)$ hasta llegar a un elemento x_j que, sumando su $w(x_j)$, superamos el valor X (todo en coste $O(n \log n)$).

```
function WEIGHTED_SELECT_USING_SORT(S, X)
    ▷ Sort using O(n log n) algorithm
    S ← MERGESORT(S)
    ▷ Sum O(n) until ≥ W
    sum ← 0
    for i ← 0 to |S| do
        sum ← sum + w(x_i)
        if sum ≥ X return i ▷ Found x_j
    end for
    return |S| − 1
end function
```

No obstante, el enunciado indica que nuestra solución ha de ser $o(n \log n)$; es decir, crecimiento estrictamente más lento que $n \log(n)$. Además, dado que los elementos de S no parecen estar acotados (o dentro de un rango de valores concreto), descartamos el poder realizar una ordenación linear. Pese a no ser una respuesta correcta, se ha valorado ün poco"aquellas soluciones basadas en ordenación $O(n \log n)$.

2. **[Exercici 2] (3.5 punts)**:

A l'empresa *WebSocial* li han donat un premi molt important per la seva contribució tecnològica a l'estudi de les relacions interpersonals a xarxes socials. Per celebrar-ho han decidit organitzar una gran festa amb els seus clients en què volen fer gala dels bons resultats que obtenen.

Els CEOs de WebSocial han determinat un conjunt d'n tasques que s'han de portar a terme per a la correcta organització de la festa, i disposen d'un conjunt S d'n treballadors per encarregar-se'n. Totes les tasques requereixen un encarregat i totes menys una requereixen que una altra persona del grup de treballadors seleccionats actuï com a supervisor. Un treballador pot supervisar cap, una o més d'una tasca.

Fent ús de la seva pròpia tecnologia l'equip directiu de WebSocial ha mesurat, per a cada parell de treballadors (a, b), un coeficient que indica la insatisfacció derivada del fet que un d'ells hagi de supervisar una tasca encarregada a l'altre. També ha determinat un subconjunt de treballadors $I \subseteq S$ que no poden ser supervisors de cap tasca (tot i ells sí que poden ser supervisats), i també un conjunt de treballadors $J \subseteq S$, $I \cap J = \emptyset$, que podrien fer-se càrrec de tasques sense necessitat de cap supervisió.

També han fixat un llindar màxim d'incompatibilitat u.

(a) Doneu un algorisme amb cost polinòmic per determinar si és possible trobar una estructura jeràrquica compatible amb les restriccions de WebSocial de manera que, entre tot parell de supervisor i supervisat, el nivell d'insatisfacció sigui com a màxim u.

(b) Assumint que hi ha una estructura jeràrquica compatible amb les restriccions de l'apartat (a), proporcioneu un algorisme eficient per obtenir una estructura jeràrquica en la qual es minimitzi la suma dels coeficients d'insatisfacció entre supervisors i supervisats.

Una solució:

De todos los datos de entrada, los únicos que son relevantes son las parejas de trabajadores (a, b) con nivel de insatisfacción w_{ab} menor o igual que u. Este conjunto de parejas permiten definir un grafo G no dirigido y ponderado cuyos vértices son el conjunto de trabajadores.

Nos piden encontrar una jerarquía, es decir un árbol de expansión enraizado (cada padre supervisará a sus hijos). Podría ser un bosque de expansión, pero como solo hay una tarea que puede no tener supervisión, la jerarquía solo puede tener una raíz.

La solución buscada es un árbol de expansión T, con las propiedades adicionales, la raíz tiene que pertenecer a J, y todos los trabajadores en I tienen que ser hojas en T.

(a) El algoritmo que propongo es el siguiente:

0. Obtener el grafo $G = (S, E)$ con aristas los pares (a, b) con nivel de incompatibilidad menor o igual que u
1. Si $J = \emptyset$, return NO
2. Si existe $a \in I$ que no tinen ningún vecino en $V \setminus I$, return NO
3. Si $G[V \setminus I]$ es conexo, return SI, si no return NO.

Notemos que el algoritmo es correcto. Si la respuesta es SI, al árbol de expansión de $G[V \setminus I]$ le podemos añadir los vértices de I conectando cada uno de ellos con un único vecino en $V \setminus I$. Este árbol lo podemos enraízar en cualquier vértice de J y así obtenemos una solución válida. Si la respuesta es NO. En el primer caso no podemos asignar la tarea que no necesita supervisión a nadie. En el segundo caso, al menos a un trabajador de I no se le puede asignar ningún supervisor fuera de I, y no tenemos solución. En el último caso, si la rspuesta es NO y el problema tiene solución tendríamos un árbol de expansión T' con las características requeridas, eliminado de T' las hojas que pertenezcan a I, obtendríamos un árbol de expansión de $G[V \setminus I]$, contradiciendo que no sea conexo.

El paso 0 tiene coste $O(n^2)$, ya que para obtener G necesito recorrer toda la entrada. El paso 1 tiene coste $O(n)$ y el paso 2 $O(|I|n) = O(n^2)$. En el paso 3 puedo utilizar un BFS con coste $O(|S| + |E|)$. El coste del algoritmo es $O(n^2)$.

(b) En una jerarquía que minimiza la suma de coeficientes, des pués d eeliminar las hojas tengo que tener un árbol de expansión de coste mínimo en $G[S \setminus I]$, si no podría reemplazarlo por otro árbol de expansión con menor suma. Además, el supervisor de un trabajador $a \in I$ tiene que un trabajador $b \in V \setminus I$, con el menor valor posible de coeficient de insatisfación de (a, b). Si no fuese así podríamos cambir de supervisor y mejorar la suma total.

Por tanto un algoritmo que, bajo la hipóteiss de existencia, resuelve el apartado b es:

1. $E_1 = \emptyset$
2. para todo $a \in I$, añadir a E_1 el par (a, b) con $b \notin I$ con menor coeficiente.
3. Obtener un MST $T = (V \setminus I, E_2)$ de $G[V \setminus I]$.
4. return $(S, E_1 \cup E_2)$

El paso más costoso es el paso 3, par el que podemos usar Prim o Kruskal con coste $O(m \log n)$, que en nuestro caso solo podemos acotar por $O(n^2 \log n)$.

Una altra solució:

Una solución alternativa. Para cada trabajador $a \in I$ seleccionar un trabajador r_a de entre el conjunto $R(a) = \{b \mid b \notin I, w_{ab} \geq u\}$, si el conjunto es vacío el problema no tiene solución ya que nadie se puede responsabilizar de la tarea que realice a. Después, construir el grafo no dirigido $G = (S, E)$, dónde en E dejamos todas las aristas (a, b) con $a, b \not\in I$ y nivel de insatisfacción $\leq u$ y añadimos las aristas (a, r_a) para cada vértice $a \in I$.

Es facil ver que el problema tiene solución si y solo si G es conexo. Ya que en un árbol de expansión de G, $a \in I$ tienen que tener grado 1, y al enraizar el árbol en un vértice de J será una raíz.

Para resolver (b), hay que seleccionar r_a como el elemnto del conjunto $R(A)$ con menor nivel de incompatibilidad. Con esta selección un MST de G enraizado en un nodo de J nos proporciona un árbol con minima suma de incompatibilidades que cumple todas las restricciones.

Una altra solució:

Una alternativa para el apartado (a).

Para $a, b \in S$, uso $w_{a,b}$ para representar el valor del coeficiente para el par (a, b). A partir de los datos de entrada voy a construir un grafo dirigido $G = (V, E)$ como sigue. Para $a \in I$, elijo un trabajador $b \notin I$ tal que el $w_{ab} \leq u$. Si no hay ninguno el problema no tiene solución, ya que nadie puede supervisar a a. Defino $E_a = \{(b, a)\}$. Para $a \notin I$, considero el conjunto $E_a = \{(a, b) \mid b \in S \text{ y } w_{ab} \leq v\}$. $E = \cup_{a \in S} E_a$.

El algoritmo comprobará, para cada trabajador $a \in J$, si es hay caminos de a a todos los vérices de S. Si esta condición se da para algún $b \in J$, tenemos solución, en caso contrario no la hay.

Para comprobar la condición es suficiente con hacer un BFS desde b y comprobar que alcanzamos todo S.

El recorrido en BFS desde un vértice de F nos permite obtener un árbol de expansión, con las aristas orientadas de padre a hijo en las que los vértices de I son hojas. Que se corresponde con la estructura jerárquica que nos pide el enunciado.

El coste de un BFS es $O(|S|+|E|) = O(n^2)$. Y el coste total del agoritmo es $O(|J|n^2) = O(n^3)$.

Comentari:

La solución que, para cada $a \in S$, considera el conjunto $R(a) = \{b \mid b \notin I, w_{ab} \geq u\}$, y asigna como supervisor de a al trabajador en $R(a)$ con menor coeficiente de incompatibilidad con a, suponiendo que todos los conjuntos sean no vacíos, no proporciona la estructura jerárquica que se pide. En particular, es fácil encontrar entradas para la s que esta asignación tiene ciclos.

3. **[Exercici 3] (3.5 punts)**:

Sigui $X = \{(a_1, b_1), \ldots, (a_n, b_n)\}$ un conjunt d'n intervals no buits. Direm que un conjunt de punts $P = \{p_1, \ldots, p_k\}$ *intercepta* un conjunt d'intervals X si tot interval a X conté com a mínim un punt de P, és a dir,

$$\forall i : 1 \leq i \leq n : (\exists j : 1 \leq j \leq k : a_i \leq p_j \leq b_i)\,.$$

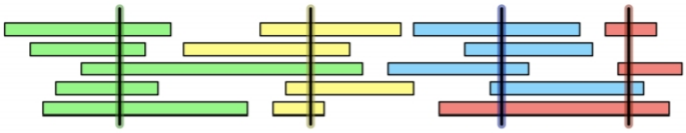

Figura 1: Exemple d'un conjunt d'intervals interceptats per 4 punts (representats com a segments verticals).

(a) Descriviu i analitzeu un algorisme eficient que donats el conjunt d'intervals X i el conjunt de punts P, calculi el subconjunt més petit de P que intercepta X, o bé que determini que P **no** intercepta a X.

Una solució:

Per proposar una solució a aquest problema ens podem inspirar en el problema de l'*Interval Scheduling* que ja coneixem (vegeu les transparències de teoria i/o el capítol 5 de *"Algorithm Design"* de Kleinberg & Tardos). L'algorisme consisteix en ordenar creixentment els intervals segons el temps de finalització i anar considerant, per cadascun d'ells, els punts que els poden anar cobrint. Primer es mirarà si el punt que cobria interval anterior també intercepta l'interval que s'està considerant (i, per tant, l'interval ja queda interceptat); si no l'intercepta aleshores s'agafarà, d'entre tots els punts que ho fan, aquell que està més a la dreta. El següent pseudocodi en fa una descripció més precisa:

PRE: $X = \{(a_1, b_1), \ldots, (a_n, b_n)\}$ conjunt d'n intervals, i P conjunt de punts.
funció MINIM_SUBCONJUNT_INTERCEPTADOR (X, P)
  $P' \leftarrow \emptyset$
  $X' \leftarrow$ Ordenar X creixentment segons els temps de finalització dels intervals (b_i)
  mentre $X' \neq \emptyset$ **fer**
    $x_i = (a_i, b_i) \leftarrow$ SEGÜENT(X') ▷ Següent interval a tractar
    si ($\exists p \in P'$ tal que $a_i \leq p \leq b_i$) **aleshores** ▷ Ja interceptat
      no cal fer res, ens oblidem d'x perquè ja queda interceptat
    altrament
      si ($\nexists p \in P - P'$ tal que $a_i \leq p \leq b_i$) **aleshores** ▷ Cap punt l'intercepta
        retornar P no intercepta X
      altrament

```
                p ← argmax_{p∈P−P'} {p | a_i ≤ p ≤ b_i}   ▷ Punt que intercepta x més a la dreta
                P' ← P' ∪ {p}
            fi si
        fi si
    fi mentre
    retornar P'
fi funció
```

L'algorisme és correcte perquè si algun interval no pot ser interceptat per cap punt, aleshores retorn que P no intercepta X. Si això no passa, aleshores retorna un subconjunt $P' \subseteq P$ de punts que intercepta tots els intervals del conjunt X, i cap $x_i \in X$ queda sense interceptar.

El pas inicial d'ordenar els intervals té cost $O(n \log n)$.[1] El posterior tractament del bucle té cost $O(n)$ perquè observeu que:

- per comprovar si ja hi ha un punt a P' que intercepti x_i, n'hi ha prou amb mirar el darrer punt afegit a P', i
- tant per comprovar si hi ha més punts (dels no considerats) que interceptin x_i com per, en cas que n'hi hagi, escollir-ne el de més a la dreta, només caldrà mirar, *en total*, una vegada els k punts si aquests els tenim ordenats.

Per tant, la complexitat temporal total de l'algorisme és $O(n \log n + k \log k)$, condicionada per les ordenacions.

L'algorisme és òptim perquè el subconjunt $P' \subseteq P$ que retorna és un dels mínims (és a dir, uns dels de menor cardinalitat) d'entre tots els possibles subconjunts de P que intercepten tots els intervals d'X. Sigui $p'_1 \in P'$ el primer punt de la solució proporcionada per l'algorisme. Aquest punt p'_1 és el punt de P que intercepta el primer interval d'X' (és a dir, l'interval acaba abans que cap altre). Suposem que hi ha una solució òptima P'' que no inclou p'_1. Aleshores, el primer punt de P'' es troba abans o després de p'_1. Si es troba després, no podria interceptar el primer interval d'X' i, per tant, no seria solució. Així que ha de ser abans. En aquest cas, si el primer punt de P'' és abans que p'_1, resulta que la solució que inclou p'_1 també és una solució òptima, ja que cap altre interval acaba abans i, per tant, no es deixarien interval per interceptar. L'aplicació reiterada d'aquest argument en justifica l'optimalitat.

(b) Per a la Festa FIB d'aquest any s'han plantejat un conjunt d'activitats per tot el Campus Nord. Cada activitat té una hora i minut d'inici i finalització. Donat que es vol ver una bona difusió a xarxes d'aquell dia, els organitzadors han contractat un servei de fotografia per tal de recollir instantànies de totes les activitats. L'empresa contractada pot enviar fins a n fotògrafs per tal que fotografiïn activitats simultàniament. Donat que és un servei car, es vol minimitzar les vegades que l'equip de fotògrafs ha de desplaçar-se al campus.

[1] N'obtindríem el mateix cost mantenint una cua de prioritat (minheap) que guardés els elements d'X segons el b_i.

Els organitzadors de la Festa FIB s'han posat en contacte amb estudiants d'Algorísmia per tal que els ajudin a decidir en quins moments requerir el servei dels fotògrafs per minimitzar-ne el cost. Descriviu com resoldríeu i implementaríeu aquest problema de la manera més eficient que se us acudeixi.

Una altra solució:

Observem que el problema que es planteja des de la FIB és una versió del problema tractat a l'apartat a). El conjunt X d'intervals està format per les activitats que s'han plantejat per a la Festa FIB i els punts interceptadors d'aquests intervals seran els moments en què s'ha de desplegar l'equip de fotògrafs per a aconseguir instantànies de les activitats. En aquest cas, però, no tenim inicialment cap conjunt P de punts. És justament el que el nostre algorisme ha de proposar: un conjunt mínim de punts que intercepti tots els intervals de temps que defineixen les activitats.

Considereu la següent estratègia greedy: ordeneu els intervals en ordre creixent del seu temps de finalització b_i. Ara agafeu el primer interval d'aquesta llista ordenada, afegiu el seu temps de finalització al conjunt de punts interceptadors, i elimineu tots els intervals que continguin aquest punt. Repetiu aquestes darreres accions mentre quedin intervals per tractar a la llista ordenada.

El següent pseudocodi en fa una descripció més precisa:

PRE: $X = \{(a_1, b_1), \ldots, (a_n, b_n)\}$ conjunt d'n intervals.
  funció MINIM_SUBCONJUNT_INTERCEPTADOR (X)
    $P' \leftarrow \emptyset$
    $X' \leftarrow$ Ordenar X creixentment segons els temps de finalització dels intervals (b_i)
    mentre $X' \neq \emptyset$ **fer**
      $x_i = (a_i, b_i) \leftarrow$ SEGÜENT(X') ▷ Següent interval a tractar
      $P' \leftarrow P' \cup \{b_i\}$ ▷ Nou punt al final de l'interval
      $X' \leftarrow X' \setminus \{x = (a, b) \mid a \leq b_i \leq b\}$ ▷ Traiem intervals interceptats pel punt
    fi mentre
    retornar P'
  fi funció

El cost temporal d'aquest algorisme és $O(n \log n)$ ja que, en considerar un nou punt, treure tots els intervals que intercepten amb ell es pot fer aprofitant el mateix recorregut sobre X (gràcies al fet que X està ordenat).

L'algorisme és correcte perquè retorna un subconjunt $P' \subseteq P$ de punts que intercepta tots els intervals del conjunt X, i cap $x_i \in X$ queda sense interceptar. Observeu que, en aquesta versió del problema, sempre existeix un subconjunt P' que cobreix X.

L'optimalitat d'aquest algorisme es dedueix de la de l'algorisme proposat a l'apartat a) ja que, el conjunt de punts solució P' que es construeix són els punts més a la dreta possibles dins dels intervals que els han definit. Nogensmenys, aquesta versió del problema es podria veure com un cas particular del presentat a l'apartat a), considerant com a conjunt inicial de punts un $P = \{b_i \mid (a_i, b_i) \in X\}$.[2]

[2]Observeu que en aplicar l'algorisme de l'apartat a) s'obtindria el mateix subconjunt $P' \subseteq P$ calculat aquí.

Notes:

Els següents criteris per ordenar el conjunt d'intervals X i tractar-los no solucionen el problema de manera òptima:

- Ordenar X creixentment per a_i i agafar el primer punt que intercepta l'interval
- Ordenar X creixentment per a_i i agafar el darrer punt que intercepta l'interval
- Ordenar X decreixentment per nombre de solapaments
- Ordenar X creixentment per b_i i agafar el primer punt que intercepta l'interval
- Ordenar P decreixentment per nombre d'intervals que intercepten

És fàcil trobar contraexemples que ho demostrin.