Algorísmia QT 2021–2022

Examen Parcial **Una solució** **4 de novembre de 2021**

Durada: 1h 25mn

Instruccions generals:

- Entregueu per separat les solucions de cada exercici (Ex 1, Ex 2, Ex 3 i Ex 4).
- Heu de donar i 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 (2.5 punts). Tenim un vector $A[1,\ldots,n]$ no ordenat amb claus no necessàriament numèriques, però que pertanyen a un conjunt totalment ordenat de claus. Sigui x_i el 2^i-èsim element més petit en A. Doneu un algorisme per calcular la suma dels valors x_i, per $1 \leq 2^i \leq n$, en $\Theta(n)$ passos.

Una solución

Sea k el valor tal que $2^k \leq n$ y $2^{k+1} > n$. $k = O(\lg n)$ y se puede calcular junto con el valor 2^k en tiempo $O(\lg n)$.

Nos piden calcular la suma de los elementos en posiciones, $1, 2, 4, \ldots, 2^k$ de un vector con n elementos.

Si utilizamos el algoritmo de selección para cada uno de los valores $1, 2, \ldots, 2^k$, el coste del algoritmo será $O(n \lg n)$. Para reducir el coste necesitamos reducir el tamaño del vector en cada iteración.

El elemento x_i, $i < k$, ocupa la posición 2^i en A pero también ocupa la posición 2^i entre los elementos que son menores que x_{i+1}. Además el número de elementos $\leq x_{i+1}$ es 2^{i+1}.

- Compute k, coste $O(\lg n)$.
- Usando el algoritmo de selección encontrar x_k, el elemento 2^k-ésimo de A con coste $O(n)$.
- Sea B el conjunto de elementos menores que x_k, B tiene $\leq 2^k$ elementos. Coste $\Theta(n)$.
- For $i = k - 1$ to 0
  - Usando el algoritmo de selección encontrar x_i, el elemento 2^i-ésimo de B con coste $O(2^{i+1})$.
  - Sea B el conjunto de elementos menores que x_i, B tiene $\leq 2^i$ elementos. Coste $\Theta(2^{i+1})$.
- Calcular la suma de los elementos en x, coste $O(\lg n)$

De acuerdo con la propiedad anterior el algoritmo calcula correctamente los valores x_i pedidos. El coste del bucle es $\leq \sum_{i=0}^{k} 2^i = O(2^{k+1}) = O(n)$. El coste del algoritmo es por lo tanto $\Theta(n)$.

Exercici 2 (2.5 punts). Et donen un immens graf $G = (V, E)$ amb pesos w a les arestes i has de calcular el MST. Quan finalitzes el càlcul te n'adones que has fet un error copiant el pes d'una aresta $e \in E$. Li has donat un pes $w'(e)$ i havia de ser $w(e)$. Dona un algorisme que trobi el MST correcte en temps lineal.

Una solución

Sea T el MST calculado a partir de G. Voy a analizar los 4 casos posibles, y ver que tenemos que hacer en cada caso.

- $e \in T$ y $w'(e) \leq w(e)$.

  En este caso T continua siendo un MST del grafo correcto ya que su coste ha bajado el máximo posible con relación al cambio.

- $e \in T$ y $w'(e) > w(e)$.

  En este caso tenemos que examinar las aristas en el corte obtenido al eliminar e de T, si hay una arista e' en el corte con $w(e') < w'(e)$, por la regla azul, tenemos que reemplazar e por e' para obtener el MST.

  Recorrer las aristas de un corte implica acceder a las listas de vecinos de los vértices en un lado del corte, el coste total es $O(m)$.

- $e \notin T$ y $w'(e) \leq w(e)$.

  En este caso tenemos que examinar el ciclo formado al añadir e to T, si e ahora es la arista de peso mínimo en el ciclo, de acuerdo con la regla roja, tenemos que reemplazar la arista de peso máximo en el ciclo por e.

  Como en T solo hay un camino entre los dos extremos de E, tenemos que extraer esta parte del ciclo y examinarla. Lo podemos hacer en $O(n)$

- $e \notin T$ y $w'(e) > w(e)$.

  En este caso T continúa siendo un MST del grafo corregido ya que e fue descartada y ahora tiene un peso mayor.

El coste total del algoritmo propuesto es $O(n + m)$.

Exercici 3 (3 punts) Un grup de n amics ha de comprar un regal que val C euros, on C és un enter no negatiu. Tenim una llista amb els pressupostos B_i de cadascun dels amics, és a dir, una llista $\mathbf{B}$ de n enters positius $\mathbf{B} = (B_1, \ldots, B_n)$.

Per fer la compra hem de determinar (si és possible) una *aportació*, una llista de quantitats $X = (x_1, \ldots, x_n)$, essent x_i la quantitat que aporta l'amic i. L'aportació ha de cobrir el cost del regal, és a dir, $\sum_{i=1}^{n} x_i = C$. A més, l'aportació particular de cap amic no pot superar mai el seu pressupost, és a dir, per $1 \leq i \leq n$, $x_i \leq B_i$.

El cost d'una aportació X és $c(X) = \max\{x_i \mid 1 \leq i \leq n\}$. Diem que una aportació $\mathbf{x}^*$ es *equitativa* si el seu cost és mínim amb relació al conjunt de totes les possibles aportacions.

Per exemple, supossem que $C = 100$, $n = 3$ i $\mathbf{B} = (3, 45, 100)$. Llavors és possible comprar el regal i una aportació equitativa és $\mathbf{x}^* = (3, 45, 52)$. Si els pressupostos foren $\mathbf{B} = (3, 100, 100)$, una aportació equitativa seria $\mathbf{x}^* = (3, 48, 49)$, però en canvi $\mathbf{x}^* = (3, 45, 52)$ no ho seria .

1. (1 punt) Sigui $B_{\min}$ el pressupost més baix. Demostra que si el regal es pot comprar i $nB_{\min} < C$ hi ha una aportació equitativa en la qual tots els amics amb pressupost $B_{\min}$ aporten $B_{\min}$.
2. (2 punts) Proporciona un algorisme golafre que determini si es pot o no comprar el regal i, en cas afirmatiu, retorni una aportació equitativa.

Una solució:

1. Donat que el regal es pot comprar existeix al menys una solució equitativa. D'altra banda, com que $nB_{\min} < C$ existeix al menys un pressupost $B_j > B_{\min}$. Demostrarem aquest apartat per reducció a l'absurd. Es a dir, suposem que per a tota aportació equitativa $\mathbf{x}^*$ existeix un amic i amb pressupost $B_{\min}$ però $x_i^* < B_{\min}$; sense pèrdua de generalitat podem suposar que aquest amic és l'amic $i = 1$, $B_1 = B_{\min}$. Sigui $\mathbf{x}^*$ una aportació equitativa i $\Delta(\mathbf{x}^*) = B_1 - x_1^* > 0$. Sigui B_j el pressupost de l'amic que més diners aporta (x_j^* és màxim, $c(\mathbf{x}^*) = x_j^*$) i $x_j^* > x_1^*$ (altrament el regal no podria ser comprat). Llavors podem obtenir una nova aportació $\mathbf{x}'$ tal que $x_1' = x_1^* + 1$, $\Delta(x_1') = \Delta(x_1^*) - 1$, i $x_j' = x_j^* - 1$. Per tant, $c(\mathbf{x}') \leq c(\mathbf{x}^*)$, i tenim una contradicció si $c(\mathbf{x}') < c(\mathbf{x}^*)$. Així que $c(\mathbf{x}') = c(\mathbf{x}^*)$ i $\mathbf{x}'$ és també equitativa, doncs té el mateix cost que $\mathbf{x}^*$. Per a que això passi, hem de tenir al menys un altre amic j' que fa aportació màxima $x_{j'}^* = x_j^*$. I d'altra banda o bé $\Delta(x_1') > 0$ o bé $\Delta(x_i') > 0$ per un cert i amb $B_i = B_{\min}$, doncs la nostra hipòtesi (per fer a la reducció a l'absurd) és que per a tota aportació equitativa hi ha al menys un amic amb pressupost mínim que no aporta tot el seu pressupost. Així podriem obtenir una nova aportació $\mathbf{x}''$ que també és equitativa però j' aporta una unitat menys al regal i l'amic 1 (o i) aporta una unitat més al regal, i iterar el mateix raonament fins a concloure que existeix una aportació equitativa per a la qual tots els amics de pressupost mínim aporten tots els seus diners, en contradicció amb la nostra hipòtesi de partida.

2. Aquest és l'algorisme golafre que proposem, amb cost $\Theta(n \log n)$:

```
if (B[1]+B[2]+...+B[n] < C) {
   cout << "el regal no es pot comprar" << endl;
   return false;
else {
   ordenar els amics de menor a major pressupost
   // B[1] <= B[2] <= ... <= B[n]
   i = 1;
   while ((n+1-i) * B[i] < C) {
       x[i] = B[i];
       C = C - B[i];
       ++i;
   }
   // el remanent C es distribueix equitativament entre els (n-i+1)
   // amics que encara no han aportat, els seus pressupostos són
   // tots >= B[i] i B[i] * (n-i+1) >= C; als últims r = C mod (n+1-i)
   // amics els fem aportar una unitat més cadascú---al menys
   // hi ha r amics amb pressupost >= q+1
   q = C / (n+1-i); r = C % (n+1-i)
   for (j = i; j <= n; ++j) {
       x[j] = q;
       if (i + r > n) ++x[j];
   }
   return x;
}
```

L'apartat previ demostra que si $nB_{\min} < C$ llavors existeix una aportació equitativa en la qual tots els amics amb pressupost mínim aporten tots els seus diners. Aquest criteri s'aplica iterativament: l'amic amb pressupost mínim aporta tots els seus diners i recursivament s'ha de fer una distribució equitativa dels diners pendents entre els $n-1$ amics restants. Es pot fer iterativament fins que només queden n' amics per aportar, tots amb pressupost $\geq B'_{\min} =$ "el pressupost més petit dels n' amics", i el import pendent de pagar és $C' \leq n'B'_{\min}$. En aquest cas és evident que la aportació equitativa és aquella en la que tots els n' amics paguen $q = \lfloor C'/n' \rfloor$ o $q+1$ (alguns d'ells, no tots, paguen $q+1$). I aquesta és exactament l'aportació calculada pel nostre algorisme.

Exercici 4 (2 punts) Tenim un graf no dirigit $G = (V, E)$. Com és habitual, d_u denota el grau del vèrtex u. Diem que una partició dels vèrtexs en V_1 i $\overline{V}_1 = V \setminus V_1$ és *equilibrada* quan $\sum_{u \in V_1} d_u = \sum_{v \notin V_1} d_v$.

Doneu un algorisme de programació dinàmica per a determinar si un graf donat té o no té una partició equilibrada.

Una solución

Sabemos que en un grafo $\sum_{u \in V} d_u = m$. El enunciado nos pide decir si es posible encontrar un conjunto $V_1 \subseteq V$ para el que $\sum_{u \in V_1} d_u = m$.

Supongamos que $V = \{v_1, \ldots, v_n\}$ y que tenemos una solución V_1 al problema. Vamos a analizar la estructura de suboptimalidad de esta solución.

Con relación al vértice v_n, tenemos dos casos

- $v_n \in V_1$, en este caso tenemos que $\sum_{v \in V_1 - \{v_n\}} d_v = m - d_{v_n}$
- $v_n \notin V_1$, en este caso tenemos que $\sum_{v \in V_1 - \{v_n\}} d_v = m$

Usando esta caracterización podemos identificar un conjunto de subproblemas, $P[i, x]$ determinar si en $V_i = \{v_1, \ldots, v_i\}$ se puede encontrar un subconjunto de vértices $V' \subseteq V_i$ tal que $\sum_{v \in V'} d_v = x$.

De acuerdo con el estudio anterior, tenemos caractarizadas la posibilidad de tener una solución que incluya el último vértice o no. Esto nos da la recurrencia

For, $1 \leq i \leq n$ and $0 \leq x \leq m$

$$P[i, x] = \begin{cases} d_{v_1} = x & i = 1 \\ P[i-1, x] & i > 1 \text{ and } x - d_{v_i} < 0 \\ P[i-1, x - d_{v_i}] \text{ or } P[i-1, x] & \textit{otherwise} \end{cases}$$

El número total de subproblemas es nm y el coste por elemento es $O(1)$. Por lo tanto implementando la recurrencia con memoización o mediante un cálculo en tabla tendremos un algoritmo con coste $O(nm)$.