Algorísmia QT 2018–2019

Examen parcial **Una solució** **14 de Novembre de 2018**

Exercici 1 (3 punts) (Vídeo blocs)
Tenim n blocs de vídeo que s'han d'enviar per una única línia de comunicació a mida que es filmen. El bloc i-èsim té una grandària de b_i bits i està disponible per iniciar la seva transmissió a temps s_i, on s_i i b_i són enters positius. Podeu assumir que transmetre un bit requereix una unitat de temps. A més, la transmissió del paquet i no pot començar abans de l'instant de temps s_i, però sí després. Encara que no es poden enviar dos paquets al mateix temps, la transmissió d'un bloc de vídeo es pot suspendre a qualsevol instant per completar-la més endavant. Per exemple, si tenim dos blocs de vídeo amb $b_1 = 5$, $s_1 = 2$, $b_2 = 3$ i $s_2 = 0$ podríem enviar a temps 0 els 3 bits del segon paquet i a temps 3 el 5 bits del bloc 1 o bé, enviar a temps 0, 2 bits del bloc 2, després els 5 bits del bloc 1 i finalitzar amb el bit restant del bloc 2. També seria possible enviar a temps 0, 2 bits del bloc 2, després 2 bits del bloc 1, finalitzar el bit restant del bloc 2, i finalitzar els bits del bloc 1. Observeu que no podem començar la transmissió de cap bit del bloc 1 fins al instant 2.
Una vegada fixada la planificació de l'ordre en què enviarem els bits dels diferents blocs, ens interessa el temps en què finalitza la transmissió de cada bloc. Anomenem f_i al temps en què finalitza la transmissió del bloc i. Observeu que, per a l'exemple anterior, la primera planificació dona $f_1 = 8$ i $f_2 = 3$, la segona planificació dona $f_1 = 7$ i $f_2 = 8$, mentre que la tercera dona $f_1 = 8$ i $f_2 = 5$.
Volem obtenir una planificació de la transmissió dels n blocs de vídeo que minimitzi la suma dels temps de finalització de la transmissió dels blocs, és a dir $\sum_{i=1}^{n} f_i$. Doneu un algorisme que, per a una entrada de n blocs, cadascun especificat per (b_i, s_i), determini una planificació (començant a temps 0) resolgui el problema proposat.

Solució: Voy a utilizar un algoritmo voraz siguiendo la siguiente regla: En cada instante de tiempo programamos el envío de un bit del paquete disponible al que le quedan menos bits por finalizar la transmisión.
Observemos que si una solución óptima no sigue esta regla buscamos el primer instante de tiempo t, en esta planificación, en que se envía un bit de un paquete i al que le faltan m_i bits por transmitir mientras que hay un paquete j disponibles a tiempo t al que le le faltan m_j bits por transmitir a tiempo t con $m_j < m_i$. Consideramos ahora los instantes de tiempo, $\geq t$, en que se envian los bits de i y de j. Podemos iniciar la transmisión de j a tiempo t, retrasando todos sus bits, al tiempo del bit de j previo. Compensar, retrasando la transmisión de los bits de i hasta llegar a utilizar el espacio dejado por el último buiit de j. La nueva planificación es válida y proporciona nuevos tiempos de finalización f'_i, f'_j. Si $f_j < f_i$, al replanificar los paquetes tenemos $f'_j < f_j$ y $f'_i = f'_j$. En este caso la planificación no seria óptima. Si $f_j > f_i$, $f'_j = f_i$ y $f'_i = f'_j$, la suma no cambia, por lo que la nueva planificación sigue siendo óptima. Realizando los intercambios correspondientes podemos conseguir una planificación óptima con el criterio de nuestro algoritmo voraz.
Para implementar el algoritmo voraz tenemos que ir con cuidado para que el número de pasos sea polinómico en el número de bloques y no en función del tiempo de finalización de la planificación que es función de los valores de los números s_i i t_i.

Para determinar eficientemente los paquetes activos ordenamos los paquetes en orden creciente de s_i, en caso de empate por orden creciente de b_i. Mantendremos una cola de prioridad, con clave el número de bits que faltan por enviar, de los paquetes disponibles y cuya transmisión no ha terminado. Por otra parte mantendremos dos índices de vídeos v_a y v_s (actual y siguiente), tres contadores de tiempo, t tiempo actual, t_a tiempo previsto finalización v_a, t_s tiempo en que v_s estará disponible.
Inicialmente $t = t_a = t_s = s_1$, $v_s = 1$, $v_a = 0$. Iremos avanzando el tiempo de acuerdo con diferentes eventos:

- Si $t_a = t_s$, introducimos en la cola los paquetes i con $s_i = t_s$ con clave b_i. v_s será el primer paquete no introducido en la cola y $t_s = s_{v_s}$.

  Extraemos el min de la cola en v_a incrementamos t_a con el número de bits que faltan de transmitir de v_a.

- Si $t_a < t_s$, planificamos los bits que faltan v_a entre t y t_a, $t = t_a$.

  Si la cola no está vacía extraemos el min de la cola en v_a, actualizamos t_a con t más el número de bits que faltan de transmitir de v_a.

  Si la cola está vacía, actualizamos t y t_a con el valor de t_s.

- Si $t_a > t_s$, transmitimos bits de v_a en los tiempos $t, \ldots, t_s - 1$, introducimos v_a en la cola con clave el número de bits que falten por transmitir.

  Actualizamos los contadores de tiempo, t y t_a para que coincidan con t_s.

La implementación es correcta ya que la prioridad en la cola solo se puede alterar cuando tenemos un nuevo vídeo disponible. Observemos que un cada paquete de vídeo entra al menos una vez en la cola. Cuando reintroducimos de nuevo un paquete, lo hacemos coincidiendo con la introducción de un paquete nuevo, pero en este caso solo reintroducimos un paquete. Esto nos da un total de $O(n)$ inserciones en la cola de prioridad.
El coste total del algoritmo es $O(n \log n)$: la ordenación inicial $O(n \log n)$y las n insercciones en la cola de prioridad con coste $O(n \log n)$.

Comentaris

1. La implementación de la regla voraz de la solución una tabla con dimensión la longitud total de la planificación. Aún siendo correcta requiere espacio y tiempo exponencial en el tamaño de la entrada.

2. Algunas de las soluciones propuesta utilizan un criterio de ordenación de los paquetes de vídeo. A partir de esta ordenación, el algoritmi planifica los bits del paquete lo antes posible en los espacios libres que quedan a partir de su tiempo de disponibilidad. Hay dos criterios que no funcionan:

   (a) Ordenar por b_i. Aplicando esta regla a $s_1 = 1, b_1 = 5, s_2 = 3, b_2 = 3$, obtenemos la planificación (una entrada por tiempo) 11122211 con coste 15. Mientras que la planificación 11111222 tiene coste 13.

   (b) Ordenar por $s_i + b_i$. Aplicando esta regla voraz a $s_1 = 1, b_2 = 2, s_2 = 1, b_2 = 4, s_3 = 3, b_2 = 3$ tenemos 112222333 con coste 17, mientras que 113332222 tiene coste 16.

Exercici 2 (2.5 punts) (Tallant ADN).
En termes computacionals, podem pensar que una cadena d'ADN S és un mot sobre l'alfabet $\{A, C, G, T\}$. El cost d'una cadena de ADN el definim com $\prod_{X\in\{A,C,G,T\}}(\text{freq}[X]+1)$ on freq$[X]$ és el nombre d'ocurrències del símbol X a S. Per exemple, si $S = ACGAC$ el seu cost es $3\times3\times2\times1 = 18$. Tenim una cadena d'ADN de llargada n i volem dividir-la en k trossos (subcadenes no buides), de manera que es minimitzi el cost màxim d'entre les k portions. Per exemple per $S = ACGAC$ i $k = 2$ tenim 4 formes de trencar la cadena S en dos i una de les què ens dona el millor cost es AC–GAC (amb cost de trencament 8). Proporcioneu un algorisme, el més eficient possible, per a resoldre aquest problema, donats S i k.

Solució: El problema es similar al de la Partició Lineal que hem fet a classe. Té la mateixa estructura de suboptimalitat però la funció de cost és diferent.
Observem que si dividim la seqüència S en k trossos (amb $1 \le k \le n$) no buits de forma òptima, tindrem un símbol s_i (amb $k-1 \le i < n$) que marcarà el final dels primers $k-1$ trossos i l'inici del darrer tros. Els primers $k-1$ trossos composen el prefix $s_1 \cdots s_i$ de S, i el darrer tros serà el sufix $s_{i+1} \cdots s_n$. La forma de tallar el prefix té que ser una solució òptima del problema de tallar en $k-1$ trossos el prefix. El cost del sufix serà cost$(s_{i+1} \cdots s_n)$.
Sigui $M(i, j)$ el cost òptim (mínim cost màxim) de particionar la seqüència $s_1 \cdots s_i$ en j trossos no buits (notem que $j \le i$. D'acord amb l'estructura de suboptimalitat el seu valor de la relació entre el mínim cost màxim de les $j-1$ particions (subproblema) i el cost del sufix $s_{\ell+1} \cdots s_i$. El mínim sobre totes les possibles seleccions de ℓ ens donarà l'òptim per aquest subproblema. Obtenim la següent recurrència, per $1 \le j, i \le n$:

$$M(i,j) = \begin{cases} -\infty & \text{si } i < j \quad \text{(cas base 1)} \\ 2 & \text{si } i = j \quad \text{(cas base 1)} \\ \text{cost}(s_1 \cdots s_i) & \text{si } j = 1 \quad \text{(cas base 2)} \\ \min_{\ell=1}^{i-1}\{\max\{M(\ell, j-1), \text{cost}(s_{\ell+1} \cdots s_j)\}\} & \text{altrament} \end{cases}$$

Tenim dos casos base:

- El cas base 1 reflexa el cas en què s'intenten fer més trossos que símbols té la seqüència.
- El cas base 2 tracta el cas que es volen fer tants trossos com símbols té la seqüència. En aquest cas cada tros tindrà exactament un símbol i, independentment del símbol que sigui, el seu cost serà 2. Per tant, tots els trossos unitaris tindran el mateix cost i per tant el mínim serà també 2.
- D'altra banda, el cas base 3 tracta el cas que només s'ha de fer un tros i, per tant, el cost de la seqüència $s_{\ell+1} \cdots s_j$ serà el què indica l'enunciat.

Observem que el cas recursiu va provant tots els possibles punts de tall a la seqüència $s_1 \cdots s_i$ on establir la separació entre els primers $k-1$ trossos i l'últim tros no buit. Observem que el nombre de subproblemes es $O(n^2)$ per tant implementarem la recurrència fent servir PD.

Donada una seqüència S amb $|S| = n$ símbols i una determinada k, la solució òptima ens la proporciona el terme $M(n, k)$ de la recurrència. Implementant-ho de la manera clàssica amb una taula T on cada cel·la $T[i, j]$ conté el valor $M(i, j)$ corresponent. Tindrem kn valors a calcular

(cel·les de la taula) i, donat que cada cel·la depén linialment de $O(n)$ altres, el cost total és $O(kn^2)$ consultes a la taula ($O(1)$) i càlcul del cost de una subcadena, en principi lineal en la seva llargada. Observem que podem precalcular el cost de totes les possibles subcadenes de S, de manera que qualsevol consulta a $\text{cost}(s_i \cdots s_j)$ tingui després temps constant. Això es pot fer en temps $O(n^2)$, computant la funció de cost per a tots els prefixos $s_i \cdots s_j$ de S (amb $1 \leq i \leq j \leq n$) a partir del càlcul del prefix anterior $s_i \cdots s_{j-1}$. Sigui P el vector on precomputem aquests valors, de manera que $P[i,j] =< \text{freq}[A]_{s_i \cdots s_j}, \text{freq}[C]_{s_i \cdots s_j}, \text{freq}[T]_{s_i \cdots s_j}, \text{freq}[G]_{s_i \cdots s_j} >$, és a dir les freqüències de les quatre diferents bases al prefix $s_i \cdots s_j$. Amb aquests quatre valors guardats a $P[i,j]$, podem implementar el càlcul de $\text{cost}(s_i \cdots s_j)$ y $P[i,j]$ a partir dels corresponents valors per al parell $(i, j-1)$ en temps constant. Essent el cas base quan $i = j$.
Amb aquest precalcul el cost del nostre algorisme esdevé $O(kn^2)$

Per a reconstruir la solució òptima afegirem l'estructura d'apuntadors necessària per a enmagatzemar, per a cada $M(i,j)$, el valor p tal que

$$p = \text{argmin}_{\ell=i}^{j-1}\{\max\{M(\ell, j-1), \text{cost}(s_{\ell+1} \cdots s_j)\}\}.$$

Exercici 3 (4.5 punts)

(a) (0.5 punts) Donats n enters amb rang entre $\{0, 1, 2, \ldots n^5 - 1\}$, quin és el temps pitjor de computació per ordenar els n enters **fent servir RADIX**, quan :

i. utilitzem una representació base-10?,

Solució: Fent servir base-10, cada enter té $d = \log_{10} n^5 = 5 \log n$ dígits. Un Counting sort triga $\Theta(n+10) = \Theta(n)$. Així el temps de Radix es $\Theta(dn) = \Theta(n \log n)$.

ii. utilitzem una representació base-n?.

Solució: Fent servir base-n, cada enter té $d = \log_n n^5 = 5$ dígits. Un Counting sort triga $\Theta(n + n) = \Theta(n)$. Així el temps de Radix es $\Theta(dn) = \Theta(n)$.

(b) (0.5 punts) En un codi de Huffman, si tots els caràcters tenen freqüència $< 2/5$, aleshores no hi ha caràcters que es pugui codificar amb longitud 1.

Solució: FALS, considereu $(0.39, 0.39, 0.22)$ el codi Huffman te longitud $(1, 2, 2)$

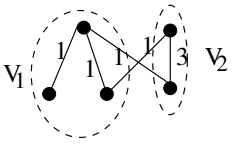
(c) (1 punt) Donat un vector A amb n elements, és possible posar en ordre creixent els $\sqrt{n}$ elements més petits i fer-ho en $O(n)$ passos?

Solució: Seleccionar l'element $\sqrt{n}$-èsim i particionar al voltant, d'aquest element (cost $O(n)$). Ordenar la part esquerra en $O(\sqrt{n}^2)$.
Alternativament, construir un min-heap en $O(n)$ i extreure el mínim element $\sqrt{n}$ cops, el nombre de passos és $O(n + \sqrt{n} \lg n)$.

(d) (0.5 punts) Donat un graf no dirigit $G = (A, E)$ amb pesos $w : E \to \mathbb{Z}$, raoneu com trobaríeu *l'arbre d'expansió amb màxim pes* de G.

Solució: Ordenar les arestes de max pes a minim i utilitzar Kruskal, escollint les arestes en aquest ordre. Complexitat: $O(m \lg n)$.

(e) (0.5 punts) Sigui $G = (V, E, w)$ un graf no-dirigit, connectat i amb pesos $w : E \to \mathbb{Z}^+$. Si fem una partició de V en V_1 i V_2, calculem el MST de cada subgraf induït, i unim amb l'aresta de pes més petit entre V_1 i V_2, és el resultat un MST per G?



Solució: FALS Considereu el graf de la figura:
Combinant els MST de V_1 i V_2 tenim un arbre amb pes total 6. Mentre que el MST del graf G té pes 4.

(f) (0.5 punts) Suposem que comencem un procés dinàmic per formar un graf G amb un conjunt de n vèrtexs i m arestes. Sigui $G_0 = (V, \emptyset)$. Considerem el procés per crear G partint de G_0 on a cada pas s'afegeix una aresta. Per tant tindrem $G_0, G_1, \ldots, G_m$, on G_m serà el graf que volem G. Al pas t, per crear el graf G_t afegim l'aresta e_t a les que ja havíem afegit prèviament $\{e_1, e_2, \ldots, e_{t-1}\}$. Doneu un algorisme $o((m + n) \log n)$ per a calcular el nombre de components connexes del graf obtingut a cada pas del procés. Observeu que G_0 té n components connexes.

Solució: Fent servir Union-Find i tractant les arestes a l'ordre donat, com en l'algorisme de Kruskal. Inicialment tenim n components connexes. Quan afegim l'aresta (u, v), si $Find(u) \neq Find(v)$ fem l'unió dels dos sets i decrementem en 1 el nombre de components connexes. En total fem com a molt $3m$ operacions union and find. Llavors el cost de l'algorisme és $O(m \log^* n) = o((m + n) \log n)$.

(g) (1 punt) Recordeu que una formula booleana està donada en *forma normal conjuntiva (CNF)* com una conjunció de clàusules on cada clàusula és una disjunció de literals. En el problema **MaxSat** donada una fórmula booleana en CNF F (amb n variables i m clàusules), volem trobar una assignació de valors booleans a les variables que facin certes el major nombre possible de clàusules a F. Considereu l'algorisme següent:

- Obtenir el nombre c_0 de clàusules a F que són cert quan assignem a totes les variables el valor fals.
- Obtenir el nombre c_1 de clàusules a F que són cert quan assignem a totes les variables el valor cert.
- Si $c_0 > c_1$ retornem l'assignació $x_i = 0$, $1 \leq i \leq n$. En cas contrari retornem l'assignació $x_i = 1$, $1 \leq i \leq n$.

Demostreu que aquest algorisme és una 2-aproximació per a **MaxSat**.

Solució: Observem que l'assignació amb tots 1s (tots 0s) satisfà totes les clàusules que en tenen un literal que és una variable afirmada (negada). Només cal recorre totes les clàusules, si en tenen un variable afirmada incrementem c_1 i si en tenen una de negada incrementem c_0. , L'algorisme triga temps lineal.

Observem que una clàusula que no es certa sota la assignació tots 1s es certa sota la assignació de tos 0s. Llavors tenim que $c_0 + c_1 \geq m$. Per tant $\max c_0, c_1 \geq m/2$.

Per un altre part el nombre màxim de clàusules opt que es poden satisfer és com a molt m. Tenim $m/2 \leq \max\{c_0, c_1\} \leq \text{opt} \leq m$. Per tant l'algorisme és una 2-aproximació.