

Problemes 2

- 2.1. **(Revisió)** Un professor rep n sol·licituds de revisions d'examen. Abans de començar, el professor mira la llista dels n estudiants que han sol·licitat revisió i pot calcular, per a cada estudiant i , el temps t_i que utilitzarà per atendre l' i -èsim estudiant. Per a estudiant i , el temps d'espera e_i és el temps que el professor triga a revisar els exàmens dels estudiants que fan la revisió abans que i .

Dissenyeu un algorisme per a computar l'ordre en que s'han de revisar els exàmens dels n estudiants de manera que es minimitzi el temps total d'espera: $T = \sum_{i=1}^n e_i$.

Solució

L'algorisme ordena els estudiants en ordre creixent de t_i .

Per veure que aquesta ordenació és òptima farem servir un argument d'intercanvi. Suposem que els estudiants estan ordenats de forma òptima y que aquesta ordenació no és la nostre. Llavors hem de tenir un estudiant i pel què $t_i > t_{i+1}$. Si calculem el temps de espera de i i de $i+1$ tenim

$$e_i = \sum_{j < i} t_j \text{ i } e_{i+1} = \sum_{j < i+1} t_j = e_i + t_{i+1}.$$

Si intercanvien la posició de l'estudiant i amb la del $i+1$ els temps d'espera dels estudiants amb $j < i$ o $j > i+1$ no canviam (d'acord amb la definició). Després del intercanvi el temps d'espera del estudiant $i+1$ és e_i i el del estudiant $i+1$ és $e_i + t_{i+1}$. Si sumem aquest temps tenim

$$e_i + e_i + t_{i+1} < e_i + e_i + t_i = e_i + e_{i+1}.$$

Per tant el temps d'espera total ha millorat i en conseqüència arribem a que l'ordenació no és òptima.

2.2. Tenim un graf no dirigit $G = (V, E)$. Donat un subconjunt $V' \subseteq V$ el *subgraph induït* per V' és el graf $G[V'] = (V', E')$ on $E' = E \cap (V' \times V')$, és a dir, conté totes les arestes que tenen els dos extrems a V' . El grau d'un vèrtex a un graf és el nombre d'arestes incidents al vèrtex. Doneu un algorisme eficient per al següent problema: donat G i un enter positiu k , trobar el subconjunt (si hi ha algun) més gran V' de V , tal que cada vèrtex a V' té grau $\geq k$ a $G[V']$.

mientras tenga un vértice con grado < k $G = G - \{v\}$

G lista ady

Q cola

N marca para borrar

$O(m)$ { para $v \in V$
 si $d(v) < k$ añade v a Q
 marca v

$O(m)$ { mientras $Q \neq \emptyset$
 extraer v de Q
 para $w \in N(v)$ y no esté marcado
 $d(w) = d(w) - 1$
 si $d(w) < 1$ añade w a la cola y lo marco

Si todos marcados \rightarrow NO , si no tiene marcados son V'

2.3. El problema de la *partició interval* (*Interval Partitioning Problem*) és similar al problema de la selecció d'activitats vist a classe però, en lloc de tenir un únic recurs, tenim molts recursos (és a dir, diverses còpies del mateix recurs). Doneu un algorisme que permeti programar totes les activitats fent servir el menor número possible de recursos.

També si s'ordenen intervals pel final no?

Interval - Partitioning (activitats) {

ordenar_per_tempo_inici (activitats) ; // $O(m \lg(m))$
 vector < vector < activitat > > Recursos ;
 int d = 1 ; \rightarrow nombre_recursos

$Q = \text{Priority-Queue}()$ \rightarrow ordena recursos per temps si més petit

$Q.\text{insert}(d, \text{activitats}[0].\text{final})$;

Recursos.push-back(activitat[0]);

Recursos[0].push-back(activitat[0]);

for ((i ; i < activitats.length ; i + +) { } \rightarrow no overlapping

if (activitats[i].inici \geq Q.top().key()) { }

Recursos[Q.top().index() - 1].push-back(activitats[i])

Q.update-key(Q.top().index(), activitats[i].final)

}

else { } \rightarrow overlapping

$d++;$

Recurv . push - back (vector < activitat> l) ;

Recurv [d - 1] . push - back (activitat[i]) ;

Q . insert (d , activitat[i] . final) ;

}

}

L' algoritme és optim, ja que:

ascendent

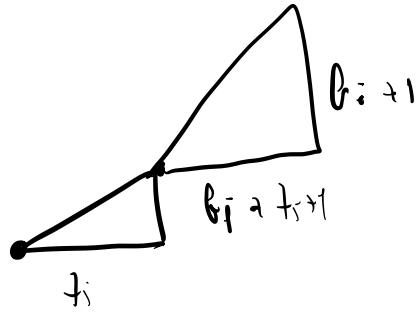
- Com totum les activitats ordenades per temps d'inici i els recursos ordenats per temps final ascendent, si el temps d'inici d'una activitat és anterior al temps final del recurs que acaba abans, llavors aquella activitat es desplaça amb tots els recursos existents, de forma que s'obrirà un recurs quan una activitat es desplaça amb tots els recursos existents.

2.4. Streaming. Tenim n paquets de vídeo que s'han d'enviar seqüencialment per una única línia de comunicació (això s'anomena *streaming*). El paquet i -èsim té una grandària de b_i bits i triga t_i segons a travessar, on t_i i b_i són enters positius (no es poden enviar dos paquets al mateix temps). Hem de decidir una planificació de l'ordre en què hem d'enviar els paquets de manera que, un cop tenim un ordre, no pot haver-hi retard entre la fi d'un paquet i el començament del següent. Assumirem que comencem a l'instant 0 i finalitzem al $\sum_{i=1}^n t_i$. A més, el proveïdor de la connexió vol utilitzar una amplada de banda no massa gran, per tant s'afegeix la restricció següent: Per a cada enter $t > 0$, el nombre total de bits que enviem de temps 0 a t ha de ser $\leq rt$, per a una $r > 0$ fixada (noteu que aquesta restricció no diu res per a períodes de temps que no comencen a l'instant 0). Direm que una planificació és *vàlida* si satisfa la restricció prèvia.

El problema a resoldre és el següent: donat un conjunt de n paquets, cadascun especificat per la seva grandària b_i i la durada de la seva transmissió t_i , i donat el valor del paràmetre r , hem de determinar si existeix una planificació vàlida dels n paquets. Per exemple, si $n = 3$ amb $(b_1, t_1) = (2000, 1)$, $(b_2, t_2) = (6000, 2)$ i $(b_3, t_3) = (2000, 1)$ amb $r = 5000$, aleshores la planificació 1, 2, 3 és vàlida, donat que compleix la restricció.

Per a resoldre aquest problema, heu de resoldre els apartats següents:

- Demostreu o doneu un contraexemple a l'enunciat de: és veritat que existeix una planificació vàlida si, i únicament si, per a cada paquet i tenim $b_i \leq rt_i$.
- Doneu un algorisme que per a una entrada de n paquets (cadascun especificat per (b_i, t_i)) i el paràmetre r , determini si existeix una planificació vàlida. El vostre algorisme hauria de tenir complexitat polinòmica en n .



Ondas no pendiente

2.5. Sigui $G = (V, E)$ un graf no dirigit. Un subconjunt $C \subseteq V$ s'anomena *recobriment de vèrtexs* de G si

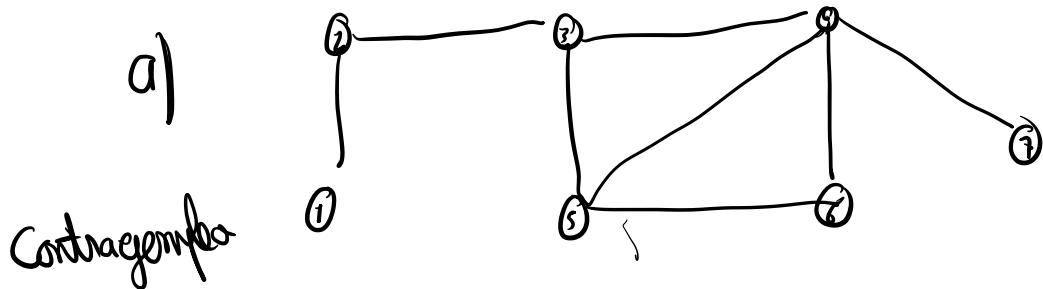
$$\forall \{u, v\} \in E : \{u, v\} \cap C \neq \emptyset.$$

Donat $G = (V, E)$, un recobriment de vèrtexs C és diu que és *minimal* si per qualsevol $C' \subseteq V$ a G , tal que $C' \subset C$ hem de tenir que C' no és un recobriment de vèrtexs.

Un conjunt $C \subset V$ és un *recobriment de vèrtexs mínim* si C és un recobriment de vèrtexs a G amb mínima cardinalitat. (Quan G és un arbre, hi ha un algorisme polinòmic per a trobar un recobriment de vèrtexs mínim a G , però per a G generals el problema és NP-hard).

En aquest problema demostrareu que, a diferència del problema de trobar un recobriment de vèrtexs mínim, el problema de trobar un recobriment de vèrtexs minimal pot ser resolt en temps polinòmic.

- (a) Demostreu que un recobriment de vèrtexs minimal no necessàriament ha de ser un recobriment de vèrtexs mínim.
- (b) Demostreu que tot recobriment de vèrtexs mínim també és minimal.
- (c) Doneu un algorisme polinòmic per trobar un recobriment de vèrtexs minimal a G .



$\{2, 3, 5, 6, 7\} \rightarrow$ minimal i no mínima

c) $S \neq \emptyset$

para $v \in V$

Si v tiene algun vecino que no está en S

$$S = S \cup \{v\}$$

$$\Rightarrow O(m + m)$$

→ Asegurar que no hay ningún vértice en S con todos vecinos en S . Si hay alguno, lo eliminamos de S .

2.6. COOPC és una companyia de lloguer de cotxes que vol diversificar el seu negoci per tal de competir en els desplaçaments curts. La companyia té oficines distribuïdes a Barcelona i un dipòsit central de cotxes al Prat. COOPC vol un mètode que li permeti processar les peticions de trajectes curts rebuts pel dia següent. Per això, vol determinar el nombre de cotxes que ha de deixar a cada oficina a l'inici del dia dedicats a trajectes curts. Aquesta assignació ha de permetre tenir suficients cotxes disponibles al llarg del dia (en cadascuna de les oficines) per tal de poder cobrir tots els trajectes curts del dia. A més, COOPC vol minimitzar el nombre de cotxes destinats durant el dia a desplaçaments curts.

La informació de la qual disposa COOPC per a cada sol·licitud de trajecte curt per al proper dia és la següent: una tupla (l, t, l', t') on el parell (l, t) indica l'oficina i l'hora de recollida del vehicle i el parell (l', t') indica l'oficina i l'hora de devolució.

A efecte d'aquesta aproximació COOPC assumeix que tots els cotxes són idèntics i que disposa d'una flota prou gran per a poder cobrir qualsevol nivell de demanda de trajectes curts. A més, assumeix també que els cotxes seran recollits i retornats puntualment a les hores i locals estipulats.

Dissenyeu un algorisme voraç que permeti obtenir el nombre de cotxes que s'han d'assignar a cada oficina, de manera que, al llarg del dia, sempre hi hagi almenys un cotxe disponible en una oficina a l'hora en què algun client l'ha de recollir d'allà. Tenint en compte que mantenir immobilitzat un cotxe té un cost alt, l'assignació obtinguda ha de garantir el requisit de disponibilitat i ha d'assignar el menor nombre possible de cotxes a cada oficina.

2.7. Sigui X un conjunt de n intervals a la recta real. Una coloració pròpia de X assigna un color a cada interval, de manera que dos intervals que se superposen tenen assignats colors diferents. Descriu i analitza un algorisme golafré eficient per obtenir el mínim nombre de colors necessaris per acolorir (amb una coloració pròpia) un conjunt d'intervals X . Podeu assumir que l'entrada està formada per dos vectors $L[1..n]$ i $R[1..n]$, representant els extrems esquerres (L) i drets (R) dels intervals a X .

Una solució: Ordenem els vectors L i R de manera que estiguin en ordre creixent d'extrem esquerre:

$$L[1] \leq L[2] \leq \dots \leq L[n]$$

Suposem que hem determinat que s'han necessitat k^* colors per la coloració dels intervals 1 a $i - 1$. Més encara, hi ha $k \leq k^*$ colors aparentment en ús (per cobrir el punt $L[i] - \epsilon$). Una estructura de dades conté la informació dels k intervals (específicament els seus extrems drets) que ténen un dels k colors assignats i suposarem que podem accedir i eliminar eficientment d'aquesta estructura (un min-heap) l'interval que té R mínima. Sigui R_{\min} aquest valor. Llavors si $R_{\min} \leq L[i]$ llavors el color assignat al interval corresponent es pot assignar al interval $X[i] = (L[i], R[i])$. S'elimina de l'estructura l'interval amb R_{\min} , que tenia assignat el color j , $1 \leq j \leq k$, i s'afegeix $X[i]$, assignat-li el color j i la seva prioritat seria $R[i]$. Si $R_{\min} > L[i]$ tots els k colors estàn en ús i s'haurà d'assignar un nou color, el $k + 1$, a l'interval $X[i]$ i s'afegirà a l'estructura amb prioritat $R[i]$. Si $k^* = k$ llavors actualitzem $k^* := k + 1$. En acabar, k^* és el mínim nombre de colors necessari que s'ens demana.

El cost de l'algorisme és $O(n \log n)$ per a ordenar els n intervals per L creixent i $O(n \log n)$ per a processar els n intervals: a cada una de les n iteracions s'ha d'actualitzar el min-heap amb una inserció (la de l'interval $X[i]$ en curs) i ocasionalment amb una eliminació del mínim quan ens consta que un color queda alliberat.

A cada iteració podríem esborrar tots els intervals del min-heap amb R més petita o igual que $L[i]$ de manera que el min-heap no contingüés intervals “acabats” i que ja no necessiten tenir color assignat; però és innecessari i molt més senzill esborrar a cada iteració només un interval (o cap si no és possible).

2.8. **⌚(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 els 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.

Una 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 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 buit 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 sería 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 $t, \dots, t_s - 1$, 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 inserciones en la cola de prioridad con coste $O(n \log n)$.

2.9. La cadena de sota és el títol d'una cançó codificat amb codis de Huffman.

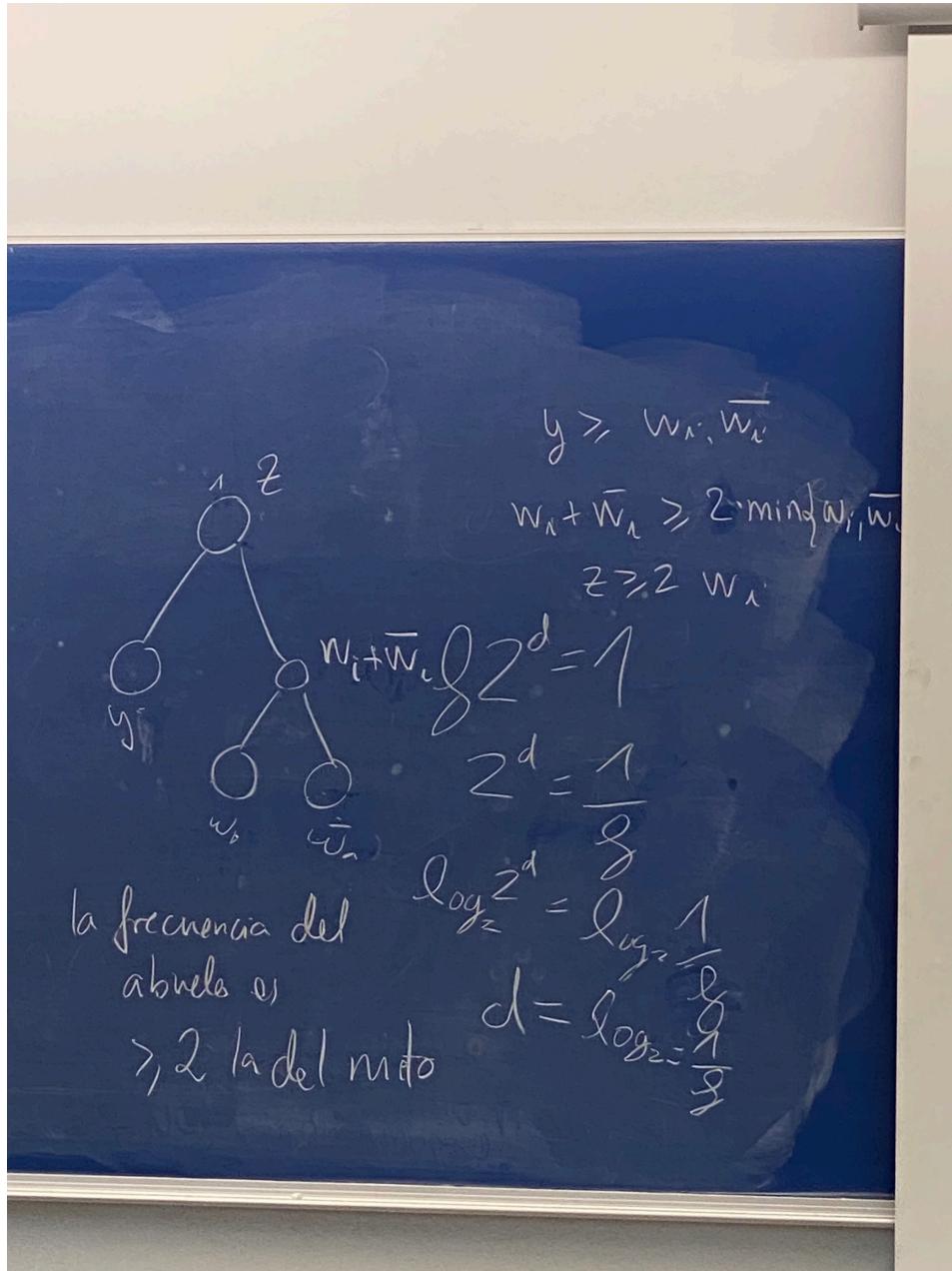
0011000101111101100111011101100000100111010010101

Donades les freqüències de les lletres que es donen a la taula de sota, obtingueu els codis de Huffmann i feu-los servir per decodificar el títol.

lletra	a	h	v	w	'	'	e	t	l	o
freqüència	1	1	1	1	2	2	2	2	3	3

Tingueu en compte que, quan hi ha múltiples eleccions, cal un criteri de desempat. Per això podeu suposar que quan es fusionan dos arbres es posa a la esquerra el que té, la fulla de més a la esquerra, més a l'esquerra a l'ordre inicial donat a la taula. A més s'assigna a la branca esquerra el símbol 0 i a la dreta el símbol 1.

- 2.10. Tenim un alfabet Σ on per a cada símbol $a \in \Sigma$, p_a es la probabilitat que aparegui el caràcter a . Demostreu que, per a qualsevol símbol $a \in \Sigma$, la seva profunditat en un arbre prefix que produueix un codi de Huffman òptim és $O(\lg \frac{1}{p_a})$. (Ajuts: en un arbre prefix que s'utilitzi per a dissenyar el codi Huffman, la probabilitat d'un nus és la suma de les probabilitats dels fills. La probabilitat de l'arrel és, doncs, 1.)



2.11. Una tribu de matemàtics ha decidit utilitzar un alfabet molt concís de 3 símbols (més el blanc \flat) per a comunicar-se entre ells. Utilitzant cadenes dels símbols $\{+, -, \star\}$ els matemàtics codifiquen totes els paraules que necessiten per a comunicar-se. Donada la seqüència $+ - + \star \flat \star + \star - \star \flat \star$, trobeu la freqüència de cada un dels símbols $\{+, -, \star, \flat\}$. Dibuixeu l'arbre de Huffman. Quina és la compressió màxima que podeu obtenir utilitzant Huffman? Quin és el guany de compressió respecte a utilitzar una compressió de longitud fixada? (i.e. utilitzant $\{0, 1\}^2$). Tingueu en compte que els matemàtics es comuniquen entre si utilitzant Internet, per tant, abans d'enviar el missatges, l'ordinador converteix els símbols del seu alfabet en cadenes binàries via ASCII.

2.12. **(Planificació).** Ens donen un conjunt de treballs $S = \{a_1, a_2, \dots, a_n\}$, a on per a completar el treball a_i es necessiten p_i unitats de temps de processador. Únicament tenim un ordinador amb un sol processador, per tant a cada instant únicament podem processar una treball. Sigui c_i el temps on el processador finalitza de processar a_i , que dependrà dels temps del treballs processats prèviament. Volem minimitzar el temps "mitja" necessari per a processar tots els treballs (el temps amortitzat per treball), es a dir volem minimitzar $\frac{\sum_{i=1}^n c_i}{n}$. Per exemple, si tenim dos treballs a_1 i a_2 amb $p_1 = 3, p_2 = 5$, i processsem a_2 primer, aleshores el temps mitja per a completar els dos treballs és $(5 + 8)/2 = 6.5$, però si processsem primer el treball a_1 i després a_2 el temps mitja per processar els dos treballs serà $(3 + 8)/2 = 2.2$

- (a) Considerem que la computació de cada treball no es porti partit, es a dir quan comença la computació de a_i les properes p_i unitats de temps s'ha de processar a_i . Doneu un algorisme que planifique la computació dels treballs a S de manera que minimitze el temps mitja per a completar tots els treballs. Doneu la complexitat del vostre algorisme i demostreu la seva correctesa.
- (b) Considereu ara el cas de que no tots els treballs a S estan disponibles des de el començament, es a dir cada a_i porta associat un temps r_i fins al que l'ordinador no pot començar a processar a_i . A més, podem suspendre a mitges el processament d'un treball per a finalitzar més tard. Per exemple si tenim a_i amb $p_i = 6$ i $r_i = 1$, pot començar a temps 1, el processador aturar la seva computació a temps 3 i tornar a computar a temps 10, aturar a temps 11 i finalitzar a partir del temps 15. Doneu un algorisme que planifique la computació dels treballs a S de manera que es minimitze el temps mitja per a completar tots els treballs.

Solució.

Notemos que en la función a optimizar, $\frac{\sum_i c_i}{n}$, el denominador no depende de la planificación. Por lo tanto la planificación con coste mínimo es la del coste medio mínimo y viceversa. Los algoritmos que propondré resuelven el problema de buscar una planificación con coste mínimo.

- (a) El algoritmo ordena los trabajos en orden creciente de p_i , y los planifica en ese orden. El coste es el de la ordenación, $O(n \log n)$.

Para ver que es correcto utilizo un argumento de intercambio. Supongamos que la planificación con coste mínimo no sigue el orden creciente de tiempo de procesado. Para simplificar asumo que el orden a_1, \dots, a_n es el que proporciona coste óptimo y que en él se produce una inversión, es decir $p_i > p_{i+1}$, para algún i .

Tenemos que $c_i = p_1 + \dots + p_i$, por lo tanto

$$\sum_i c_i = np_1 + (n-1)p_2 + \dots + (n-i)p_i + \dots + 1p_n.$$

Si intercambiamos a_i con a_{i+1} solo cambia la contribución al coste de estos dos elementos que pasa de ser $(n-i)p_i + (n-i-1)p_{i+1}$ a ser $(n-i)p_{i+1} + (n-i-1)p_i$. El incremento en coste debido al intercambio es

$$(n-i)p_{i+1} + (n-i-1)p_i - [(n-i)p_i + (n-i-1)p_{i+1}] = p_{i+1} - p_i < 0.$$

Por tanto, la ordenación no es óptima y tenemos una contradicción.

- (b) En este segundo apartado tendremos que seguir el criterio del apartado anterior, pero teniendo en cuenta que se incorporarán a lo largo del tiempo nuevos trabajos. La regla voraz del algoritmo es: procesar en cada instante de tiempo el proceso disponible al que le quede menos tiempo por finalizar. Utilizando el mismo argumento de intercambio que en el apartado (a) la regla voraz es correcta.

Tenemos que ir con cuidado en la implementación ya que el número total de instantes de tiempo es $\sum_i t_i$ y este valor puede ser exponencial en el tamaño de la entrada. Sin embargo, los tiempos

en los que se para la ejecución de un proceso coinciden con los de disponibilidad de un nuevo proceso. Necesitamos controlar solo los instantes de tiempo en los que finaliza la ejecución de un proceso o en los que un proceso está disponible, un número polinómico.

El algoritmo ordena en orden creciente de r_i los procesos y mantiene una cola de prioridad con los procesos disponibles y no finalizados, utilizando como clave lo que le falta al proceso para finalizar su ejecución.

- Ordenar por r_i ;
- Insertar en la cola todos los procesos con $r_k = r_1$ (clave p_k), $i =$ primer proceso no introducido en la cola, $t = r_1$.
- mientras cola no vacía
 - $(j, p) = pop()$, si $t + p \leq r_i$ procesamos lo que queda de a_j , $t = t + p$, y repetimos hasta que la cola quede vacía o $t + p > r_i$.
 - Si $t + p > r_i$, insertamos $(j, t + p - r_i)$, $t = r_i$.
 - Insertamos en la cola todos los procesos con $r_k = r_i$ (clave p_k), $i =$ primer proceso no introducido en la cola.

La implementación es correcta ya que el conjunto de trabajos disponibles y no finalizados solo se modifican cuando hay un nuevo trabajo disponible o cuando iniciamos el procesamiento de uno de ellos. En el primer caso ese caso actualizamos la cola y el posible trabajo que se estaba ejecutando se interrumpe, y se vuelve a insertar en la cola con el tiempo restante. En el segundo, sacamos al proceso con menor tiempo para finalizar y iniciamos o reiniciamos su ejecución.

El coste de la ordenación es $O(n \log n)$ y el coste de cada inserción en la cola es $O(\log n)$. Para contabilizar el número total de inserciones, notemos que cada proceso se inserta en la cola cuando está disponible, lo que nos da n inserciones. Un proceso puede volver a reinsertarse en la cola varias veces, sin embargo, por cada tiempo de disponibilidad se reinserta un proceso como mucho, esto nos da $\leq n$ inserciones debido a paradas en la ejecución. Sumando todo, el coste del algoritmo es $O(n \log n)$.

2.13. Un *bottleneck spanning tree* T d'un graf no dirigit i ponderat $G = (V, E, w)$, on $w : E \rightarrow \mathbb{R}^+$, és un arbre d'expansió de G on el pes més gran és mínim sobre tots els arbres d'expansió de G . Diem que el valor d'un bottleneck spanning tree és el pes de la aresta de pes màxim a T .

(a) Demostreu la correctesa o trobeu un contraexemple pels enunciats següents:

- Un bottleneck spanning tree és també un arbre d'expansió mínim.
- Un arbre d'expansió mínim és també un bottleneck spanning tree.

(b) Doneu un algorisme amb cost $O(|V| + |E|)$ que donat un graf G i un enter b , determini si el valor d'un bottleneck spanning tree és $\leq b$.

- 2.14. Demostreu que un graf G té un únic MST si, per a tot tall C de G , existeix una única aresta $e \in C$ amb valor mínim. Demostreu que el el reciprocal no és cert, i.e. pot ser el cas de que per un o més talls C tinguem més d'una aresta mínim pes, però que el MST sigui únic

2.15. Tenim un graf no dirigit i connex $G = (V, E)$ i una coloració de les arestes amb dos colors, roig i blau ($c : E \rightarrow \{R, B\}$). Doneu un algorisme per a obtenir un arbre d'expansió amb el mínim nombre d'arestes blaves.

for all $e_i \in E$ do

if $c_{e_i} = \text{roig}$ $\text{weight}(e_i) = 1$;

else $\text{weight}(e_i) = 2$

end for

$\text{Prim}(G, w, \text{random } v \in V) \circ \text{Kruskal}(G, w)$

- Com Prim amb Fibonacci Heap $\rightarrow \cancel{O(|E|)} + O(|E|) + |V| \log(|V|)$

- Com Kruskal $\rightarrow \cancel{O(|E|)} + O(|V| + |E| \log(|V|))$

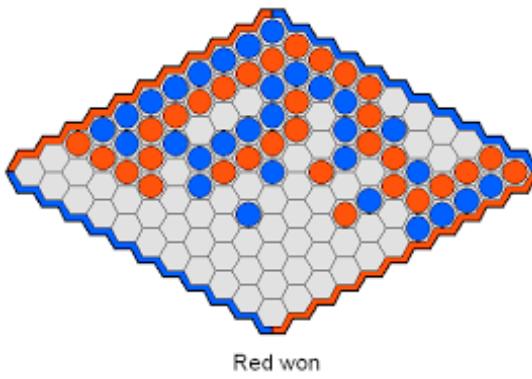
2.16. Tenim un graf connex no dirigit $G = (V, E)$ on cada node $v \in V$ té associat un valor $\ell(v) \geq 0$; considerem el següent joc unipersonal:

- (a) Els nodes inicialment no estan marcats i la puntuació del jugador és 0.
- (b) El jugador selecciona un node $u \in V$ no marcat. Sigui $M(u)$ el conjunt de veïns de u a G que ja han sigut marcats. Aleshores, s'afegeix a la puntuació del jugador el valor $\sum_{v \in M(u)} \ell(v)$ i marquem u .
- (c) El joc es repeteix fins que tots els nodes siguin marcats o el jugador decideixi finalitzar la partida, deixant possiblement alguns nodes sense marcar.

Per exemple, suposeu que el graf té tres nodes A, B, C on A està connectat a B i B amb C , amb $\ell(A) = 3, \ell(B) = 2, \ell(C) = 3$. En aquest cas, una estratègia òptima seria marcar primer A , després C i finalment B . Aquest ordre dona al jugador una puntuació total de 6.

- (a) És possible obtenir una puntuació millor deixant algun dels nodes sense marcar? Justifiqueu la vostra resposta.
- (b) Dissenyeu un algorisme voraç per tal d'obtenir la millor puntuació possible. Justifiqueu la seva correctesa i doneu-ne el cost.
- (c) Suposeu ara que $\ell(v)$ pugui ser negatiu. Continua el vostre algorisme proporcionant la puntuació màxima possible?
- (d) Considereu la següent modificació del joc per aquest cas en què els nodes poguessin tenir valors negatius: eliminem primerament de G tots els nodes $v \in V$ amb $\ell(v) < 0$, per tal de seguidament executar el vostre algorisme sobre el graf resultant d'aquesta eliminació. Doneu un exemple on aquesta variació no proporcioni la màxima puntuació possible.

2.17. El joc d'HEX té com un dels seus inventors, al matemàtic John Nash. En aquest joc, dos jugadors, un amb color negre i l'altre amb color e blanc, fan torns on a cada torn el jugador que li toca col·loca una pedra del seu colors a una posició encara buida, a una xarxa $n \times n$ de cel·les hexagonals. Un cop col·locada una pedra, no es pot moure. L'objectiu de cada jugador és connectar els costats del mateix color a la graella, amb un camí continuo fet amb les seves pedres. Dues cel·les es consideren connectades si comparteixen una vora les dues tenen la pedres amb el mateix color. Descriuvi un esquema eficient que determini, després de cada jugada, si el jugador que acaba de jugar ha guanyat el joc d'HEX.



2.18. (SOS Rural). Considerad un camino rural en el Pirineo con casas muy dispersas a lo largo de él. Por motivos de seguridad se quieren colocar estaciones SOS en algunos puntos de la carretera. Los expertos indican que, teniendo en cuenta las condiciones climáticas de la zona, se debería garantizar que cada casa se encuentre como máximo a una distancia de 15km, siguiendo la carretera, de una de las estaciones SOS.

Proporcionad un algoritmo eficiente que consiga este objetivo utilizando el mínimo número de estaciones SOS posibles. Justificad la corrección y el coste de vuestro algoritmo e indicad la complejidad en tiempo de la solución propuesta.

2.19. (**RHEX**). El joc de RHEX és una variació del joc de l'HEX inspirada en el Reversi. En aquest joc, hi han $2n$ fitxes, les fitxes tenen dos cares, una de color blanc i l'altre de color negre. El tauler es una xarxa $n \times n$ de cel·les hexagonals, com la de l'HEX. Hi han dos jugadors, un amb color negre i l'altre amb color blanc, que fan torns. A cada torn el jugador que li toca col·loca una fitxa mostrant el seu color a una posició encara buida. Un cop col·locada una fitxa al tauler, no es pot moure.

Quan es col·loca una fitxa al tauler es considera la *zona dominada*: totes les celles ocupades i connectades amb la nova cella amb un camí continuo de fitxes, normalment hi hauran fitxes de tots dos colors en aquests camins. Dues cel·les es consideren connectades si comparteixen una vora i les dues tenen una fitxa, independentment del color de les fitxes. Si a la zona dominada hi ha menys fitxes del color del jugador que té el torn que del color de l'altre jugador, es giren totes les fitxes a la zona dominada.

La partida acaba quan un dels dos jugadors guanya (aconsegueix tenir més de n fitxes del seu color) o quan s'hagin col·locat les $2n$ fitxes.

- (a) Descriu un algorisme eficient per al seguiment d'una partida. L'algorisme, després de cada jugada, ha de permetre mantenir el nombre total de fitxes de cada color al tauler i a més ha d'indicar si el jugador que acaba de jugar ha guanyat el joc de RHEX.
- (b) Expliqueu com modificar el vostre algorisme per tal de que a més de mantenir la puntuació podeu actualitzar el tauler.

- 2.20. El centre de documentació de la UE gestiona el procés de traducció de documents pels membres del parlament europeu. En total han de treballar amb un conjunt de n idiomes. El centre ha de gestionar la traducció de documents escrits en un idioma a tota la resta d'idiomes.

Per fer les traduccions poden contractar traductors. Cada traductor està especialitzat en dos idiomes diferents; és a dir, cada traductor pot traduir un text en un dels dos idiomes que domina a l'altre, i viceversa. Cada traductor té un cost de contractació no negatiu (alguns poden treballar gratis).

Malauradament, el pressupost per a traduccions és massa petit per contractar un traductor per a cada parell d'idiomes. Per tal d'optimitzar la despesa, n'hi hauria prou en establir cadenes de traductors; per exemple: un traductor anglès \leftrightarrow català i un català \leftrightarrow francès, permetria traduir un text de l'anglès al francès, i del francès a l'anglès. Així, l'objectiu és contractar un conjunt de traductors que permetessin la traducció entre tots els parells dels n idiomes de la UE, amb cost total de contractació mínim.

El matemàtic del centre els hi ha suggerit que ho poden modelitzar com un problema en un graf amb pesos $G = (V, E, w)$. G té un node $v \in V$ per a cada idioma i una aresta $(u, v) \in E$ per a cada traductor (entre els idiomes u i v de la seva especialització); el pes de cada aresta seria el cost de contractació del traductor en qüestió. En aquest model, un subconjunt de traductors $S \subseteq E$ permet portar a terme la feina si al subgraf $G_s = (V, S)$ hi ha un camí entre tot parell de vèrtexs $u, v \in V$; en aquest cas direm que S és una *selecció vàlida*. Aleshores, d'entre totes les seleccions vàlides han de triar una amb cost mínim.

- Demostreu que quan S és una selecció vàlida de cost mínim, $G_s = (V, S)$ no té cicles.
- Proporcioneu un algorisme eficient per a resoldre el problema. Justifiqueu la seva correctesa i el seu cost.

Solució:

- Supongamos que $G_s = (V, S)$ es una selección válida de coste mínimo que tiene ciclos. Si eliminamos una arista (u, v) de un ciclo en $G_s = (V, S)$ seguimos teniendo caminos entre todos los vértices, ya que podemos ir de u a v a través de lo que queda del ciclo.
Como la selección tiene coste mínimo, y eliminando una arista de un ciclo también es solución. Tenemos que todas las aristas de un ciclo tienen coste 0. Así, mientras tengamos ciclos vamos eliminando una arista de peso 0 del ciclo. Hasta que tengamos una selección válida con coste mínimo sin ciclos.
- Por el apartado a) nos basta con buscar un árbol con peso mínimo que cubra todos los idiomas. Es decir tenemos que obtener un MST del grafo. Utilizando el algoritmo de Prim, podemos encontrarlo en tiempo $O(n \log m)$

2.21. Tenim un tauler de dimensions $n \times n$, amb n fitxes col·locades a certes posicions $(x_1, y_1), \dots, (x_n, y_n)$ i una fila i . Volem determinar el mínim nombre de moviments necessaris per a posar les n fitxes a la fila i (una a cada casella). Els moviments permesos són: cap a la dreta, esquerra, amunt i avall. Durant aquests moviments es poden apilar tantes fitxes a la mateixa posició com calgui. Pero en finalitzar ha de quedar una fitxa per casella.

Pista: El nombre de moviments verticals (amunt/avall) necessaris es pot calcular fàcilment.

2.22. Sigui $T = (V, E)$ un arbre no dirigit amb n vèrtexs. Per a $u, v \in V$, sigui $d(u, v)$ el nombre d'arestes del camí de u a v en T . Definim el *centre* de T com el vèrtex

$$c = \operatorname{argmin}_{v \in V} \{ \max_{u \in V} d(u, v) \}.$$

Describiu un algorisme de cost $O(n)$ per a obtenir un centre de T .

2.23. CinemaVis ha de programar l'aparició d'un seguit d'anuncis a una pantalla gegant a la Plaça del Mig la diada de Sant Jordi. La tirada d'anuncis es pot iniciar a temps 0 (l'inici programat) però mai abans. A més, CinemaVis disposa d'un conjunt de n anuncis per fer la selecció. L'anunci i té una durada de 1 minut i té associat dos valors reals no negatius t_i i b_i . L'anunciat pagarà b_i euros a CinemaVis si l'anunci i s'emet a l'interval $[0, t_i]$ i 0 euros si s'emet després. Cap dels n anuncis es pot mostrar més d'una vegada. CinemaVis vol projectar la selecció d'anuncis que li proporcioni màxim benefici. Dissenyeu un algorisme, el més eficient que podeu, per a resoldre aquest problema.

2.24. (Agenda) A la vostra agenda teniu una llista L de totes les tasques que heu de completar en el dia de avui. Per a cada tasca $i \in L$ s'especifica la durada $d_i \in \mathbb{N}$ que indica el temps necessari per a completar-la i un factor de penalització $p_i \in \mathbb{Z}^+$ que n'agreuja el retard. Heu de determinar en quin ordre realitzar totes les tasques per obtenir el resultat que menys penalització total acumuli.

Tingueu en compte que:

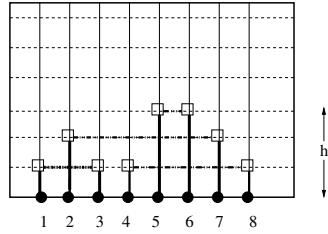
- en un instant de temps només podeu realitzar una única tasca,
- una vegada comenceu a fer una tasca, heu de continuar-la fins a finalitzar-la, i
- s'han de completar totes les tasques.

El criteri d'optimització és la penalització total que s'acumula. La penalització real associada a una tasca $i \in L$ és el temps de finalització t_i de la seva realització, multiplicat per la seva penalització p_i . El temps de finalització t_i es correspon al temps transcorregut des de l'inici de la jornada laboral (és a dir, des de l'instant de temps 0) fins al moment en que s'ha finalitzat la tasca.¹

Considereu l'algorisme voraç que programa les tasques en ordre decreixent de factor de penalització p_i . Determineu si aquest algoritme resol el problema. En cas que no ho faci, proporcioneu un algorisme (tant eficient com pogueu) per resoldre'l.

¹Observeu que, segons les restriccions del problema, la realització d'una tasca $i \in L$ amb temps de finalització t_i haurà començat a l'instant $t_i - d_i$.

2.25. Als circuits VLSI, s'utilitza un encaminament Manhattan sobre la placa aïllant on va muntat el circuit. Les connexions horitzontals van per la cara de sota i les connexions verticals per la cara de sobre. Quan es necessiten connectar les connexions horitzontals amb les verticals, es perfora la placa amb el que s'anomena una *via*. Les connexions del circuit amb l'exterior es realitzen amb *pins* (veure la figura, on les connexions de sobre estan dibuixades en sòlid i les que van per sota amb línia discontinua). Tant les connexions horitzontals com verticals segueixen unes pistes dibuixades sobre la placa en forma d'una graella, i els pins estan alineats a un extrem de la placa. Sigui h el nombre de pistes horitzontals utilitzades. Si $L = \{(p_1, q_1), (p_2, q_2), \dots, (p_n, q_n)\}$ són una seqüència de parells de pins a connectar (dos a dos), volem dissenyar un algorisme que connecti els parells de pins utilitzant el mínim nombre de pistes horitzontals h . Per exemple, considereu la següent figura:



Tenim com a entrada $L = \{(1, 3), (2, 7), (4, 8), (5, 6)\}$, el nombre de h és 3, i no es pot fer amb $h = 2$. En particular: dissenyeu un algorisme eficient, que donats n parells de pins, resolgui el problema de l'encaminament, de manera que es minimitzi h . Doneu-ne la complexitat i demostreu-ne la correctesa.

2.26. Doneu un algorisme que resolgui el següent problema i doneu la seva complexitat en funció de n . Justifiqueu-ne la correctesa:

Volem anar de Barcelona a Paris seguint l'autopista a través de Lyon, i tenim n benzineres, B_1, \dots, B_n , al llarg d'aquesta ruta. Amb el diposít ple, el vostre cotxe pot funcionar K km. La benzinera B_1 és a Barcelona, i cada B_i , $2 \leq i \leq n$ és a $d_i < K$ km. de la benzinera B_{i-1} . La benzinera B_n és a Paris. Quina és l'estratègia per aturar-se el mínim nombre de cops al llarg del viatge?

Una solució: L'estratègia que farem servir és esperar el màxim per a carregar benzina, però sense quedar-nos amb el dipòsit buit. Aquest criteri es pot implementar com el següent algorisme voraç:

```

function GREEDY( $d_1, \dots, d_n, K$ )
     $d = 0; R = \emptyset; j = 1$ 
    for  $i = 1 \dots n - 1$  do
        if  $d + d_{i+1} > K$  then
             $j := j + 1;$ 
             $R = R \cup \{B_j\};$ 
             $d = 0;$ 
             $d := d + d_{i+1};$ 
    return ( $j$ );

```

Per demostrar la correctesa de l'algorisme compararem la solució obtinguda pel nostre algorisme amb una possible solució òptima amb menys aturades. Sigui $R = \{b_1, \dots, b_j\}$ les benzineres seleccionades per GREEDY. Suposem que la solució òptima requereix $m < j$ aturades, sigui $S = \{c_1, \dots, c_m\}$ el conjunt de les benzineres a una solució òptima.

Primer demostrarem que, per $j = 1, \dots, m$, $d(B_1, b_i) \geq d(B_1, c_i)$. L'enunciat és cert per $j = 1$, si no ens quedariem sense benzina abans d'arribar a c_1 . Suposem que l'enunciat és cert per $i < j$. Llavors tenim

$$d(B_1, c_j) - d(B_1, c_{j-1}) \leq K,$$

ja que S és una solució, i

$$d(B_1, c_j) - d(B_1, b_{j-1}) \leq d(B_1, c_j) - d(B_1, c_{j-1}),$$

per hipòtesi d'inducció. Combinant les dues desigualtats tenim

$$d(B_1, c_j) - d(B_1, b_{j-1}) \leq K.$$

Llavors, c_j ha de ser abans de b_j .

Com a conseqüència del resultat tenim que $d(b_m, B_n) \geq d(c_m, B_n) \leq K$, llavors l'algorisme voraç no s'aturaria a cap benzinera després de b_m i tenim $j = m$.

Per tant GREEDY és òptim i la seva complexitat és $O(n)$.

2.27. 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 algoritme 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 .

Una solución

El algoritmo voráz 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 este criterio utilizaremos una cola de prioridad Q donde guardaremos 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, para 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 en Heaps 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 a una fusión entre dos secuencias. 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 profundidad posible (h_{\max}). Consideremos 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 la solución es óptima. Si $b \neq d$, intercambiamos además b con d y por el mismo motivo la solución 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.