Problemes 2

2.1. Donat un conjunt $\{x_1, x_2, \ldots, x_n\}$ de punts de la recta real, doneu un algorisme, el més eficient que pogueu, per a determinar el conjunt més petit d'intervals tancats amb longitud unitat, que cobreixen tots els punts (cada punt ha d'aparèixer almenys a un interval).

Una solució: Para ver como resolver el problema voy a considerar que el conjunto de puntos X están ordenados en orden creciente de valor ($x_1 \leq x_2 \leq \cdots \leq x_n$). Si no lo estuviesen, se pueden ordenar en tiempo $O(n \log n)$ utilizando merge sort.

Una solución óptima Y al problema se puede representar por una secuencia creceinte $y_1 < y_2 < \cdots < y_k$ de valores, indicando los puntos de inicio de los k intervalos de longitud 1 que forman Y. La secuencia es estrictamente creciente ya que si no lo fuese tendríamos intervalos repetidos y la solución no sería óptima.

Si Y es óptima, cada intervalo $[y_j, y_j + 1]$ tiene que contener al menos un punto de X. Si no, podríamos eliminar el intervalo y podríamos cubrir todos los puntos con un intervalo menos.

Además, si desplazamos el inicio del intervalo al primer punto de X que cubre, seguimos teniendo una solución óptima. Ya que cada intervalo cubre al menos todos los puntos que ya cubría antes y sigue siendo una solución. Así tenemos que siempre hay una solución óptima en la que los puntos de inicio de los intervalos son valores en el conjunto de puntos dado.

Utilizando esta idea de desplazar a la derecha podemos encontrar siempre una solución óptima $x_{i_1} < x_{i_2} < \cdots < x_{i_k}$ en la que ningún punto del conjunto inicial está cubierto por más de un intervalo. Basta seguir los intervalos en orden y desplazar el inicio del intervalo $i+1$-ésimo hasta el primer punto cubierto por el intervalo $i+1$ que no esté cubieto por el intervalo i.

Si analizamos este último tipo de solución óptima, $x_{i_1} < x_{i_2} < \cdots < x_{i_k}$ en la que los conjuntos de puntos cubiertos por cada intervalo son disjuntos dos a dos, tenemos que, $x_{i_1} = x_1$, si no x_1 no estaría cubierto. Además, $x_{i_{j+1}}$ tiene que ser el primer punto en X con valor mayor que $x_{i_j} + 1$, ya que si no este valor no estaría cubierto en la solución.

Esta última solución óptima se puede obtener en tiempo $O(n)$ asumiendo que los puntos estén ordenados, y en tiempo $O(n \log n)$ si tenemos que ordenarlos.

2.2. 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, suposem 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 .

(a) 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 $B_{\min}$.

(b) Proporciona un algorisme golafre que determini si es pot o no comprar el regal i, en cas afirmatiu, retorni una aportació equitativa.

Una solució:

(a) 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.

Una demostració alternativa. D'una banda es pot comprar el regal i per a qualsevol solució ("aportació") $\mathbf{x}$ es verifica

$$\sum_{i=1}^{n} x_i = C.$$

Sigui $k < n$ el número d'amics amb pressupost mínim $B_{\min}$ ($k \neq n$ ja que $n \cdot B_{\min} < C$). Sense pèrdua de generalitat podem suposar que els amics amb pressupost mínim són els amics $1, 2, \ldots,$ k. Llavors, considerem el problema amb els $n' = n - k$ amics amb pressupost $> B_{\min}$ i cost del regal $C' = C - k \cdot B_{\min}$, i sigui $\mathbf{x}'$ una aportació equitativa per aquest nou problema. Tal aportació segur que existeix, ja que el els n' amics poden comprar un regal de cost C'.

Tornant al problema original, sigui $\bar{\mathbf{x}}$ definida de la seguent manera:

$$(B_{\min}, \ldots, B_{\min}, x'_{k+1}, \ldots, x'_n)$$

En aquesta aportació $\bar{\mathbf{x}}$ tots els amics amb pressupost mínim $B_{\min}$ aporten tots els seus diners. És vàlida:

$$\sum_{i=1}^{n} \bar{x}_i = k \cdot B_{\min} + \sum_{i=k+1}^{n} x'_i = k \cdot B_{\min} + C' = C.$$

I finalment, és equitativa. Com que $\sum_{i=k+1}^{n} x'_i = C'$, es dedueix que

$$c(\mathbf{x}') = \max_{k<i\leq n} \{x'_i\} \geq \left\lfloor \frac{C'}{n'} \right\rfloor,$$

Observem que $c(\bar{\mathbf{x}}) = c(\mathbf{x}') \geq B_{\min}$, ja que $c(\mathbf{x}') \geq \lfloor C'/n' \rfloor \geq B_{\min}$:

$$C > n \cdot B_{\min} \Longrightarrow C - k \cdot B_{\min} > n \cdot B_{\min} - k \cdot B_{\min} \Rightarrow$$
$$\frac{C - k \cdot B_{\min}}{n - k} > B_{\min} \Rightarrow \left\lfloor \frac{C - k \cdot B_{\min}}{n - k} \right\rfloor \geq B_{\min}.$$

Suposem que $c(\bar{\mathbf{x}})$ **no** es mínima. És a dir, existeix una $\mathbf{x}^*$ pel problema amb cost C i n amics, equitativa amb $c(\mathbf{x}^*) < c(\bar{\mathbf{x}})$. La suma de les aportacions dels amics $k+1$ a n en $\mathbf{x}^*$ ha de ser $C^* \geq C - k \cdot B_{\min}$. Llavors considerant només $x^*[k+1..n]$ com a solució del problema amb cost C^* per a $n-k$ amics $c(\mathbf{x}^*) \geq c(\mathbf{x}')$ doncs els $n' = n-k$ ara han de pagar un regal més car i $\mathbf{x}'$ és equitativa. Però $c(\mathbf{x}') = c(\bar{\mathbf{x}})$ i arribem a una contradicció.

(b) 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. Una solució alternativa:

```
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]
   for (int i = 1; i <= n; ++i) {
       x[i] = min(B[i], C/(n-i+1));
       C = C - x[i];
   }
   return x;
```

```
}
```

2.3. **(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: $T = \sum_{i=1}^{n} e_i$.

$t = [3,2,1] \Rightarrow e = [0,3,5] \Rightarrow T = 8$
$t' = [1,2,3] \Rightarrow e = [0,1,3] \Rightarrow T = 4$

Algorisme:

1- Algorisme d'ordenació segons el temps de revisió $t \rightarrow O(n \log n)$

2.4. 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 $G[V']$.

2.5. 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.

```
procedure INTERVAL_PARTITIONING (A)
  Ordenar A en ordre ascendent de temps de inici
  n = A.size();
  for i := 1 to n do
      if A[i] és compatible amb una classe ja creada
          afegim i a la classe compatible
          cambiem l'hora final de la classe compatible
      else
          Creem una classe nova
          Afegim i a aquesta classe
          Actualizem l'hora de fi de la classe
```

Interval Scheduling → Para un único procesador el mayor número de tareas.

Interval Partitioning → Para un conjunto de tareas cual es el mínimo número de procesadores que debo usar

Una tarea tiene un tiempo de inicio y un tiempo de finalización que representan el tiempo de ejecución

Cost:

- Ordenació $O(n\log n)$
- Bucle for amb n elements $\Theta(n)$
- Per afegir les activitats a una classe utilitzem priority queues $O(\log n)$

Hem d'insertar i eliminar una classe cada vegada que afegim una activitat

$$O(n\log n) + O(n) * O(\log n) = O(n\log n) + O(n\log n) = O(n\log n)$$

Correcció:

Imaginem que n son el nombre de classes que hem obtingut al aplicar l'algoritme, llavors existeix una activitat en la classe n que no podia anar en la classe n-1, donat que l'activitat començaba abans de finalitzar l'última activitat de n-1.

Amb això i degut a que les activitats estan ordenades segons l'hora d'inici, sabem que l'última activitat de cada classe finalitza després que l'inici de l'activitat totes les classes superiors a aquesta.

2.6. 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).

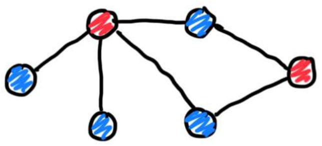
En aquest problema demostrareu que, a diferència del problema de trobar un recobriment de vèrtexs mínim, el problema de trobar un recobriment de vèrtexs minimal pot ser resolt en temps polinòmic.

(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.

a)



recobriment minimal
recobriment mínim

recobriment minimal → $\forall v \in C$ si $C - \{v\}$ no es recobriment

recobriment mínim → compleix la propietat de recobriment minimal i $|C|$ és el mínim

b) C mínim $\Rightarrow$ C minimal

Sup C mínim $\underbrace{\text{però no minimal}}$

$\Downarrow$

$\exists C' \subseteq C$ tq és recobriment i té menys nodes $(|C'| < |C|) \Rightarrow$

$\Rightarrow$ no pot ser C mínim

c) $C \subseteq V$ és recobriment minimal $\Leftrightarrow$ $V - C$ és subconjunt independent màxima l

$\begin{cases} I \leftarrow \text{maximal IS}(G) \\ \text{retorna } V/I \end{cases}$

maximal IS(G):

$O(n+m)$

```
I ← ∅
mentre V ≠ ∅ fer
    v ← escollir node de V
    I ← I ∪ {v}
    V ← V - {v}
    V ← V - N(v)
retorna I
```

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 golafre 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:

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

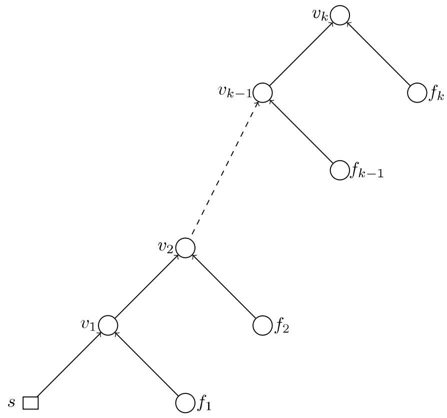
Suposem que hem determinat que s'han necessitat k^* colors per la coloració dels intervals 1 a $i-1$. Més encara, hi ha $k \leq k^*$ colors aparentment en ús (per cobrir el punt $L[i] - \epsilon$). Una estructura de dades conté la informació dels k intervals (específicament els seus extrems drets) que 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 podriem esborrar tots els intervals del min-heap amb R més petita o igual que $L[i]$ de manera que el min-heap no contingués intervals "acabats" i que ja no necessiten tenir color assignat; però és innecesari i molt més senzill esborrar a cada iteració només un interval (o cap si no és possible).

2.8. ✎ 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 produeix 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.)

Solució: Sigui T un arbre prefixe corresponent a un codi Huffman òptim, i sigui s la fulla que conte a.
Sigui $P = s, w_1, w_2, \ldots, v_k$ la seqüències de nusos que van de s fins a l'arrel v_k, i siguin f_i els fills de v_i que no son a P.
Sabem que $p(v_i) = p(v_{i-1}) + p(f_j)$. Però $p(f_j) \geq p(f_{j-1})$ i $p(f_j) \geq p(v_{j-2})$, per tant $p(f_j) \geq (p(f_{j-1}) + p(v_{j-2}))/2$, tenim $p(f_j) \geq p(v_{j-1})/2$.
Posant tot junt, $p(v_j) = p(v_{j-1}) + p(f_j) \geq 1.5p(v_{j-1})$.
Utilitzant inducció podem demostrar que $p(v_k) \geq 1.5^{k-1}p(v_1) \geq 1.5^{k-1}p(a)$ i a mes, $p(f_k) = 1$. Això implica $1.5^{k-1}p_a \leq 1$ i per tant la profunditat de a es $k+1 = O(\lg 1/p_a)$.



2.9. 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.

2.10. Si utilitzem l'algorisme de Huffmann per a comprimir un text format per n, símbols que apareixen amb freqüències $f_1, f_2, f_3, \ldots, f_n$ quina és la màxima longitud de compressió d'un símbol que podem obtenir? Doneu un exemple de les freqüències on es compleix aquesta condició.

2.11. La companyia Googol vol detectar a la web en quin idioma està escrita cada pàgina a la web. Per això, ha dissenyat un programa que quan descarrega una pagina, pot demanar si la pàgina està escrita en idioma L_i $(1 \leq i \leq 8)$ i el programa respon SI o NO. Googol sap que de totes les pàgines a internet, el 40% estan escrites en L_1, el 17% en L_2, el 15% en L_3 , el 11% en L_4, el 9% L_5, el 5% L_6, el 2% L_7, i el 1% L_8.

(a) Googol, ha dissenyat un programa al que, quan descarrega una pagina, pot demanar si la pàgina esta escrita en idioma L_i $(1 \leq i \leq 8)$ i el programa respon SI o NO. En quin ordre s'han a de fer les preguntes per que el nombre mitjà de preguntes necessaries per a identificar l'idioma sigui mínim?

(b) Un dels informàtics a Googol pensa que potser seria millor fer servir un programa que, donats dos conjunts disjunts d'idiomes, diu si la pàgina està escrita amb un idioma del primer conjunt o amb un idioma del segon. Dissenya un algorisme, que fent servir aquest tipus de qüestions, trobi l'idioma en el que està escrit la pàgina. Quin serà el nombre esperat de qüestions utilitzant el teu algorisme? (Ajut: Penseu en el codis de Huffman)

2.12. **(Planificació)**. Ens donen un conjunt de treballs $S = \{a_1, a_2, \ldots, a_n\}$, 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 processem a_2 primer, aleshores el temps mitja per a completar els dos treballs és $(5+8)/2 = 6.5$, però si processem 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 port partir, 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 $a_1, \ldots, a_n$ 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 $c_i = p_1 + \cdots + p_i$, por lo tanto

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

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 vacia
  - $(j, p) = pop()$, 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 implementicón 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. ✎ 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 $O(n + m)$.

2.14. 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 - Bottleneck Spanning Tree

Un *bottleneck spanning tree* T d'un graf no dirigit i ponderat $G = (V, E, w)$, on $w : E \to \mathbb{R}^+$, 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.

1. Demostreu la correctesa o trobeu un contraexemple pels enunciats següents:

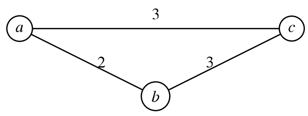
   - *Un bottleneck spanning tree és també un arbre d'expansió mínim.*
   - *Un arbre d'expansió mínim és també un bottleneck spanning tree.*

2. 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 $\leq b$.

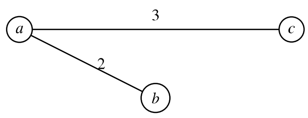
Una solució (prof. Maria J. Serna):

La diferencia fundamental entre un bottleneck spanning tree y un minimum spaning tree es que la medida que tomamos sobre un árbol de expansión (ST) es, en el primer caso, el peso máximo de las aristas en el árbol y, en el segundo la suma de los pesos de todas las aristas. Las propiedades de un MST ya las habeis estudiado recordar las reglas rojas y azul. Para los bottleneckST, por su definición, todos tienen la arista de peso máximo con el mismo valor.

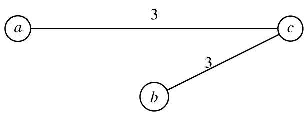
1. - No todo bottleneckSTes un MST. Un contraejemplo es el siguiente



G



MST



BottleneckST

   El BottleneckST tiene peso total 6 y no es un MST.

   - Sí, un MST es un bottleneck spanning tree. La demostración la haremos por reducción al absurdo. Supongamos que un MST T no es un BottleneckST . Consideremos un BottleneckST T' y sea w el peso de la arista con peso mayor en T'. Esto quiere decir que:

     (a) T tiene una arista e con peso mayor que w, si no sería BottelneckST.

     (b) Todas las aristas en T' tienen peso menor o igual que w.

     Si eliminamos e de T dividimos los vértices de G en dos partes. Pero al ser T' un ST algunas de las aristas de T' tienen que conectar estas dos partes. e no es la arista de peso mínimo en este corte por lo que deducimos que T no puede ser un MST, aplicado la regla azul.

2. Dados G y b, el algoritmo considera solo aquellas aristas con peso $\leq b$ y con un BFS comprueba si el grafo es conexo o no. En el primer caso la respuesta es sí y en el segundo no.

   Para que exista un BottleneckST en las condiciones que se pide, tiene que existir un ST en el que todas las aristas tengan peso $\leq b$, el algoritmo comprueba esta propiedad.

   Podemos modificar la implementación del BFS para que trate solo las aristas con peso $\leq b$ sin tener que modificar el grafo con coste $O(n + m)$.

2.15. 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 reciproc 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

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

2.17. 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.

a) No, donat un graf $G = (V, E)$ obtenim una puntuació màxima P, on l'últim node marcat ha sigut $v \in V$, per tant, $P = \sum_{u \in (V-v)} \ell(u)$, si ara afegim un nou node w la màxima puntuació sera $P + \ell(v)$ si $\ell(v) \geq \ell(w)$ o $P + \ell(w)$ altrament i en el pitjor cas $\ell(v) = \ell(w) = 0$ donat que $\ell(v) \geq 0$ $\forall v \in V$ i per tant, $P + \ell(v) = P + \ell(w) = P$, per tant, mai podem obtenir una puntuació pitjor en cas de no marcar un node.

b) Anem recorrent els nodes en ordre descendent, tal que $\forall v \in V$ $\ell(v_i) \geq \ell(v_{i+1})$

Correcció:

Donat $G = (V, E)$ una puntuació màxima P i l'últim node visitat ha sigut $v \in V$ si afegim un node u de manera que $G = (V \cup u, E)$ i $\ell(u) > \ell(v) \Leftrightarrow P + \ell(u) > P + \ell(v)$ no obtenim la puntuació màxima.

Per tant, si visitem $\ell(v)$ abans que $\ell(u)$

c) És evident que no. Exemple: -5 Si visita el node té puntuació negativa, en canvi, si no el visita té puntuació 0

d)

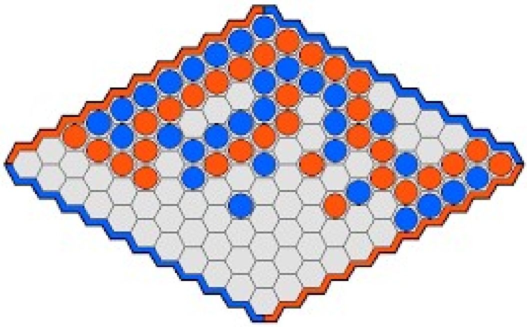
ordre visita

Puntuació = 0

Ordre visita

Puntuació = 4

2.18. 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. Descriviu un esquema eficient que determini, després de cada jugada, si el jugador que acaba de jugar ha guanyat el joc d'HEX.



Red won

2.19. (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.

2.20. **(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 tos 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) Descriviu 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.

2.21. ✎ 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.

(a) Demostreu que quan S és una selecció vàlida de cost mínim, $G_s = (V, S)$ no té cicles.

(b) Proporcioneu un algorisme eficient per a resoldre el problema. Justifiqueu la seva correctesa i el seu cost.

Solució:

(a) 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.

(b) Por el aparatdo 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 $O(n \log m)$

2.22. Tenim un tauler de dimensions $n \times n$, amb n fitxes col·locades a certes posicions $(x_1, y_1), \ldots, (x_n, y_n)$ 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.

2.23. 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

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

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

$$\boxed{c = \underset{v \in V}{\text{argmin}} \left\{ \underset{u \in V}{\max}\, d(u,v) \right\}} \equiv$$ És el vèrtex que fa que entre tots els vèrtex el valor del màxim sigui el mínim de tots

argument (v) que minimitza

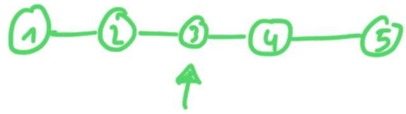
1- BFS des de node a l'atzar → trobem fulla v

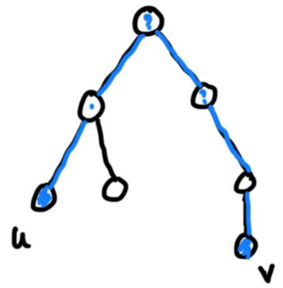
2- BFS des de v i trobarem camí més llarg

3- El centre és el node del mig del camí

cost $O(\underbrace{n+m}_{\leq n-1}) = O(n)$

ARBRE!!





$\forall v \in V, \max(d(1,v)) = 4$

$\forall v \in V, \max(d(2,v)) = 3$

$\forall v \in V, \max(d(3,v)) = 2$

$\forall v \in V, \max(d(4,v)) = 3$

$\forall v \in V, \max(d(5,v)) = 4$

min → $v = 3$ argmin

Altra opció:

<u>mentre</u> $|V| > 1$

| $F \leftarrow$ fulles $T(V)$ → arbre induït per V

| $V \leftarrow V - F$

$O(n)$ + Càlcul grau $O(m)$

```
Q ← fulles T(v)
  mentre Q no buida
    v ← pop()
    --grau[pare(v)]
    si grau[pare(v)] == Ø
      Q.push(pare(v))
  return centre (únic)
```

2.24. 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.

Exercici 4 (3 punts) 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.

Solució: Tal y como dice el enunciado cada anuncio dura exactamente un minuto. Teniendo esta restricción en cuenta, en cualquier solución podemos asumir que los tiempos de inicio son enteros, si no es así podemos retrasar los tiempos de inicio al entero anterior sin perder beneficios. Además puesto que los anuncios solo se pueden proyectar una vez, es suficente con considerar los tiempos de inicio de proyección de anuncios en los valores 0 a $n-1$. Si en una solución óptima hay algún anuncio emitido a tiempo $t \geq n$ por fuerza queda algún hueco en la planificación en alguno de los instantes 0 a $n-1$. En ese caso podemos retrasar el tiempo de emisión de los anuncios después de hueco. Observemos que el beneficio solo puede aumentar.

El algoritmo primero preprocesará los tiempos de entrada, asignado al anuncio i el entero $\lfloor t_i \rfloor$ ($O(n)$). Luego ordenará en orden decreciente ($O(n \log n)$). Utilizando el orden podemos asignar a cada anuncio el tiempo $y_i \in \{0, \ldots, n-1\}$ mayor en el intervalo $[0, \lfloor t_i \rfloor]$ con coste $O(n)$, recorrido de mayor a menor.

Notemos que a efectos de nuestro algoritmo tenemos construida una entrada donde cada anuncio tiene asignado b_i y y_i y se obtiene beneficio b_i si el anuncio se emite en $\{0, 1, \ldots, y_i\}$. Además sabemos que $0 \leq y_i < n$.

Ahora buscamos la planificación en $\{0, \ldots, n-1\}$ que nos proporcione mayor beneficio. Para ello primero ordenamos los anuncios en orden decreciente de beneficio b_i ($O(n \log n)$). Tratamos los anuncios en este orden. Inicialmente no hay anuncio planificado para ningún instante de tiempo. Asigno el anuncio i a el instante $y_i - 1$, si no hay ningún anuncio ya planificado para este tiempo. En caso contrario busco el mayor tiempo anterior j que esté libre, si $j \leq 0$ planifico anuncio i a tiempo j. En caso contrario el anuncio no se mostrará.

La segunda parte del algoritmo tiene coste $O(n^2)$ ya que para cada anuncio tengo que mirar en caso peor n posibles tiempos. Así, el coste total del algoritmo es $O(n^2)$.

Porqué es correcto? Supongamos que una asignación óptima s no sigue este criterio y sea s' la solución del algoritmo voraz propuesto. Tratando los elementos en orden decreciente de b_i's.

Sea j el primer anuncio en el que las dos planificaciones difieren.

Si s' no muestra j, de acuerdo con el algoritmo, en todos los valores $\{0, \dots, y_j$ se ha planificado un anuncio anterior. Como para valores mayores de j las dos planificaciones coinciden, s tampoco muestra j. Nos quedan dos casos por considerar:

- s no muestra j y s' muestra i a tiempo $t = s'(j)$. A tiempo t, s puede no emitir un anuncio o emitir un anuncio k ($k > j$). En el primer caso podemos emitir j a tiempo t en s y aumentar el beneficio (s no sería óptimo). En el segundo caso emitiendo j en vez de k a tiempo t no podemos disminuir el beneficio ya que $b_j \geq b_k$.
- s no muestra j a tiempo $t = s(j)$ y s' muestra i a tiempo $t' = s'(j)$. Como las planificaciones coinciden hasta j, y el algoritmo lo emite en el instante mas tardío en el que es posible emitirlo, $t' < t$. De nuevo a tiempo t', s puede no emitir un anuncio o emitir un anuncio k ($k > j$). En el primer caso podemos emitir j a tiempo t' en s y aumentar el beneficio. En el segundo caso emitiendo j a tiempo t' y k a tiempo t, seguimos dentro de los intervalos con beneficio y obtenemos exactamente el mismo beneficio.

Aplicando reiteradamente los cambios mencionados o bien llegamos a lo conclusión que s no es solución óptima o, manteniendo el beneficio, transformamos s en s'. Por lo que concluimos que el algoritmo es correcto.

Comentarios sobre soluciones entregadas

- No se puede usar t_i como índice de una vector, es un valor real.
- El algoritmo propuesto aplicado sobre tiempos t_i no enteros puede eliminar soluciones óptimas. Supongamos que tenemos la entrada [pares $a_i = (t_i, b_i)$]: $a_1 = (1.5, 10)$, $a_2 = (2, 3)$. El algoritmo propuesto planifica a_1 a tiempo 0.5. Esta decisión no permite emitir a_2 con beneficio no nulo, ni antes ni después de a_1.
- Ordenar por orden decreciente de b_i/t_i y asignarlo a tiempo $t_i - 1$ si no está ocupado no resuelve el problema. Considera la entrada $a_1 = \cdots = a_{20} = (20.3, 2)$, $a_{21} = \cdots = a_{82} = (59.5, 5)$ que está ordenada por b_i/t_i. El algoritmo emitiría 20 del primer tipo (tiempos 0 a 19) y 38 del segundo tipo (tiempos 20 a 58) beneficio menor que si emitiésemos 59 del segundo tipo.
- Considerar el tiempo creciendo ($0 \rightarrow$) y emitir el anuncio posible que proporciona mayor beneficio no es correcto. Si la entrada es $a_1 = (1, 3)$, $a_2 = (2, 6)$, $a_3 = (2, 2)$. Se emitiría a_2 a tiempo 0 y a_3 a tiempo 1 que no es óptimo.
- Considerar el tiempo decreciendo ($\max t_i \rightarrow 0$) y emitir el anuncio posible que proporciona mayor beneficio proporciona el criterio correcto. Dependiendo de como se implemente el iterador sobre el tiempo, el coste del algoritmo puede ser o no polinómico. Si se itera desde $t = \max t_i$ hasta 0 con $t - -$ el número de iteraciones es exponencial.

45. 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.

Una solució:
Teniendo en cuenta que cada anuncio dura 1 minuto y que hay n anuncios podemos programar un anuncio para inciarse en el minuto 0, otro para iniciarse en el minuto 1, y asi hasta programar el anuncio a emitir entre los instantes $n-1$ y n. Sin pérdida de generalidad, supongamos que indexamos los anuncios de mayor a menor beneficio, esto es, $b_1 \geq b_2 \geq \cdots \geq b_n$. Para cada uno de ellos tenemos que asignarle un slot $[e_i - 1, e_i]$, el minuto en que se emitirá. Entonce nuestro objetivo es hallar una permutación $e = (e_1, \ldots, e_n)$ de los números del 1 al n que maximiza el valor

$$\mathrm{val}(e) = \sum_{i=1}^{n} b_i \cdot [\![e_i \leq t_i]\!],$$

donde $[\![P]\!] = 1$ si P es cierta y $[\![P]\!] = 0$ en otro caso.

Nuestra estrategia voraz asigna a cada anuncio, sucesivamente del 1 al n, el slot más tardío no asignado todavía en el periodo $[0, t_i]$, y si no hay ninguno entoces el slot más tardío no asignado en $[t_i, n]$. Esta estrategia puede implementarse trivialmente con coste $O(n^2)$ (y con un esfuerzo considerable y estructuras de datos apropiadas con coste asintóticamente menor).

En lo que resta vamos a demostrar que este algoritmo voraz devuelve una solución óptima. Sea $e^* = (e_1^*, \ldots, e_n^*)$ una solución óptima. Sea i el menor índice para el cual e y e^* difieren; si no existe entonces no hay nada que demostrar. En caso contrario supongamos primero que $e_i < e_i^*$. Entonces el valor b_i contribuye a $\mathrm{val}(e)$ pero no $\mathrm{val}(e^*)$, pues por definición no hay ningún slot libre en $[e_i - 1, t_i]$ una vez asignados $e_1, \ldots, e_{i-1}$; por lo tanto tendremos $e_i^* > t_i$ y $[\![e_i^* \leq t_i]\!] = 0$. Entonces podemos construir una nueva asignación como sigue. Habrá algún anuncio, digamos el j-ésimo, con $j > i$ que tiene asignado el slot $[e_i - 1, e_i]$, esto es, $e_j^* = e_i$. En la nueva asignación de slots $\hat{e}$ todos los anuncios reciben la misma asignación que en e^*, excepto que intercambiamos las asignaciones de los anuncios i y j, es decir, $\hat{e}_i = e_j^* = e_i$, $\hat{e}_j = e_i^*$ y $\hat{e}_\ell = e_\ell^*$. En primer lugar $[\![\hat{e}_i \leq t_i]\!] = [\![e_i \leq t_i]\!] \geq [\![e_i^* \leq t_i]\!] = 0$. Por otro lado $[\![\hat{e}_j \leq t_j]\!] = [\![e_i^* \leq t_j]\!]$

2.25. *(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.

Contraexemple ordenació per penalització:

$d_1 = 4 \quad d_2 = 1$

$p_1 = 2 \quad p_2 = 1$

Ordre per penalització (①): $4 \cdot 2 = 8$, $5 \cdot 1 = 5$, total 13 (durada 4)

Millor ordre (②): $1 \cdot 1 = 1$, $5 \cdot 2 = 10$, total 11

Algorisme més eficient: Ordenem amb un ratio
- penalització/duració decreixent
- duració/penalització en ordre creixent

Cost: $O(n \log n)$

Correctesa: demostració per intercanvi

sup: Sopt en el què dues tasques $i, i+1$ consecutives que no compleixen el criteri d'ordenació, alesshores:

$$p_i / d_i < p_{i+1} / d_{i+1}$$

Sigui T el temps d'inici de la tasca i (des de Ø)

Sigui M el cost que no és degut a les tasques $i, i+1$

$$cost(Sopt) = M + (T + d_i) p_i + (T + d_i + d_{i+1}) p_{i+1}$$

[1] Observeu 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 $t_i - d_i$.

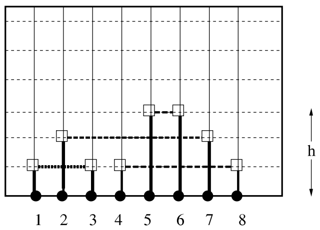
Intercanviem $i, i+1 \rightarrow$ Sopt' (les altres tasques no es veuen afectades)

$$\Rightarrow cost(Sopt') = M + (T + d_{i+1}) p_{i+1} + (T + d_{i+1} + d_i) p_i$$

veurem $cost(Sopt') - cost(Sopt) < Ø$

$$= d_i p_i + (d_i + d_{i+1}) p_{i+1} - d_{i+1} p_{i+1} - (d_{i+1} + d_i) p_i = d_i p_{i+1} - d_{i+1} p_i < 0$$

2.26. 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 discontínua). Tant les connexions horitzontals com verticals segueixen unes pistes dibuixades sobre la placa en forma d'una graella, i els pins estat alineats a un extrem de la placa. Sigui h el nombre de pistes horitzontals utilitzades. Si $L = \{(p_1, q_1), (p_2, q_2), \ldots, (p_n, q_n)\}$ 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.

2.27. 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, $B_1 \ldots, B_n$, al llarg d'aquesta ruta. Amb el diposit 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?

2.27 (Benzineres)

Volem anar de Barcelona a Paris seguint l'autopista a través de Lyon, i tenim n benzineres $B_1 \ldots, B_n$ al llarg d'aquesta autopista. Quan teniu el dipòsit ple, el vostre cotxe pot recórrer K km. La benzinera B_1 és a Barcelona i la benzinera B_n és a Paris. La distància entre benzineres consecutives és sempre menor que K, és a dir $d(B_i, B_i - 1) < K$,

Doneu un algorisme que ens proporcioni una estratègia per fer el mínim nombre d'aturades a benzineres al llarg del viatge Barcelona-Paris.

Solució: L'estratègia correcta és esperar el màxim per a carregar benzina. Fem el següent algorisme voraç:

$d := 0; j := 1$
for $i = 1$ **to** $n - 1$ **do**
  if $d + d_{i+1} > k$ **then** $d := 0,\ j := j + 1$ **fi**
  $d := d + d_{i+1}$
  od
return j

Si aquest algorisme torna j aturades, però l'òptim és aturar-se a $S_{i_1}, \ldots, S_{i_m}$ per $m < j$ i sigui S_{i_l} la primera benzinera després que la j-èsima aturada de l'algorisme, aleshores el cotxe es quedaria sense benzina entre $S_{i_{j-1}}$ i S_{i_j}, contradicció. La complexitat de l'algorisme és $O(n)$.

2.28. 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.

$|S| = 3 = N$

$S = \{A, B, C\}$

A B C

30 50 10

S[0] S[1]

$\left.\begin{array}{l} 1)\ \underbrace{A+B}=30+50=80 \\ 2)\ \downarrow\ +C = 80+10=90 \end{array}\right\} + = 170$ ordre: A,B,C

$\left.\begin{array}{l} 1)\ \underbrace{A+C} = 30+10=40 \\ 2)\ \downarrow\ +B = 40+50=90 \end{array}\right\} + = 140$ Ordre: A,C,B

$S = \{S[0], S[1], \ldots\}$

Ordenar S en nombre d'elements creixent $O(n \log n)$

mentre $S > 1$

elem = S[0] ∪ S[1]

S.erase (S[0], S[1])

S.insert (elem) $O(\log n)$

Sol. insert (S[0], S[1]) $O(1)$

Correctesa:

$A \leq B \leq C$

$A+B \leq C+B$

Una altra solució seria aplicant Huffman