# Algorísmia QP 2022–2023

**Examen Final** **19 de juny de 2023**

Una solució

---

**Exercici 1 (2 punts) (Ratolins i caus)**

A un experiment sociològic es disposen $n$ ratolins a les posicions $R = \{r_1, \ldots, r_n\}$ determinades sobre una línia recta. A la mateixa línia hi ha $n$ caus localitzats a les posicions $C = \{c_1, \ldots, c_n\}$. A cada cau només es podrà allotjar un ratolí.

Un ratolí pot romandre a la seva posició, moure's un pas cap a la dreta de la línia (és a dir, de la posició $x$ a la posició $x + 1$), o moure's un pas cap a l'esquerra de la línia (és a dir, de la posició $x$ a la posició $x - 1$). Qualsevol d'aquests moviments consumeix un minut de temps.

Dissenyeu un algorisme, el més eficient possible, per a assignar els ratolins als caus de manera que es minimitzi el temps en què l'últim ratolí entra dins d'un cau. Raoneu el cost de l'algorisme proposat.

**Una solució:** Aquest problema es pot resoldre mitjançant una estratègia *greedy*. Podem posar cada ratolí al cau que li pertoca segons la seva posició relativa dins de la recta (és a dir, el ratolí de més a l'esquerra al cau de més a l'esquerra, el segon ratolí al segon cau, etc.). Això es pot fer ordenant les $n$ posicions dels ratolins i les $n$ posicions dels caus, per aleshores poder fer les assignacions un a un (l'$i$-èsim ratolí a l'$i$-èsim forat). En fer aquestes assignacions, podem calcular també la diferència màxima entre les posicions dels ratolins i les posicions dels forats assignats. Aquesta diferència màxima serà el temps que triga a entrar al cau el ratolí que triga més.

Un possible pseudocodi per aquest algorisme seria:

**function** RATONIS-A-CAUS $(R = \{r_1, \ldots, r_n\}, C = \{c_1, \ldots, c_n\})$
  $R' \leftarrow sort(R)$ ▷ p.ex. creixentment
  $C' \leftarrow sort(C)$ ▷ De la mateixa manera que s'ha ordenat $R$
  $L \leftarrow \{\}$ ▷ Llista amb les assignacions
  $t_{\max} \leftarrow 0$ ▷ Temps que triga l'últim ratolí
  **for** $i = 1$ **to** $n$ **do**
    $L \leftarrow L \cup \{r'_i, c'_i\}$ ▷ Assignar l'$i$-èsim ratolí a l'$i$-èsim cau
    $t_{\max} \leftarrow \max\{t_{\max}, \mid r'_i - c'_i \mid\}$
  **return** $L, t_{\max}$

L'algorisme retorna una assignació correcta perquè tots els ratolins acaben tenint un cau assignat, i a cada cau no s'hi allotja més d'un ratolí. L'algorisme és també òptim: retorna l'assignació que minimitza el temps en què l'últim ratolí entra dins del cau. Noteu que, una vegada les posicions dels ratolins i les posicions dels caus s'han ordenat, tenim que $r'_1 \leq r'_2 \leq \cdots \leq r'_n$ i $c'_1 \leq c'_2 \leq \cdots \leq c'_n$. Aleshores el primer ratolí $r'_1$ necessàriament ha d'anar al primer forat $c'_1$ i, recursivament, els restants $n-1$ ratolins s'han de posar òptimament en els restants $n-1$ forats (subestructura òptima).

Per justificar això, fixem-nos en els dos primers ratolins i caus, és a dir, en $r'_1 \leq r'_2$ i en $c'_1 \leq c'_2$; en qualsevol dels casos que es poden plantejar[1] podem veure que

$$\max(|r'_1 - c'_1|, |r'_2 - c'_2|) \leq \max(|r'_1 - c'_2|, |r'_2 - c'_1|).$$

I s'aplicaria el mateix raonament pel subproblema restant.

El cost de l'algorisme ve donat per les ordenacions essent, per tant, $O(n \log n)$.

**Observacions:**

- L'objectiu del problema és minimitzar el temps que triga a arribar al cau l'últim ratolí o, dit d'una altra manera, el temps que es triga en tenir tots els ratolins als caus. L'objectiu **no** és, doncs, minimitzar la suma dels temps que triguen els ratolins individualment.
- Aquest algorisme **no** està assignant cada ratolí al cau que té més proper (és a dir, $c'_i$ no necessàriament és el cau més proper a $r'_i$).
- Escollir com a criteri greedy l'assignació d'un ratolí al seu cau més proper **no** porta sempre a una solució òptima (independentment de si els assignem seguint l'ordre d'entrada o primer ordenem els ratolins segons la seva distància mínima a un cau).

  Considereu, per exemple, $R = \{4, 9, 11\}$ i $C = \{1, 5, 10\}$. Si s'assigna cada ratolí al cau més proper s'assignaria $r_1 \to c_2$, $r_2 \to c_3$ i $r_3 \to c_1$. Això donaria un temps total de 10 minuts, però l'òptim ho fa en 4 minuts.

  Aquest mateix exemple també serveix de contraexemple en cas d'haver interpretat la funció objectiu com la suma dels temps individuals dels ratolins. En aquest cas

[1] Es poden plantejar 6 situacions diferents:

1. $r'_1 \leq r'_2 \leq c'_1 \leq c'_2$
2. $r'_1 \leq c'_1 \leq r'_2 \leq c'_2$
3. $r'_1 \leq c'_1 \leq c'_2 \leq r'_2$
4. $c'_1 \leq r'_1 \leq c'_2 \leq r'_2$
5. $c'_1 \leq c'_2 \leq r'_1 \leq r'_2$
6. $c'_1 \leq r'_1 \leq r'_2 \leq c'_2$

donaria un temps total de $1 + 1 + 10 = 12$, en canvi la solució proposada trigaria $3 + 4 + 1 = 8$ minuts.

**Exercici 2 (2 punts) (Equipant un avatar)**

En un joc en línia un jugador ha d'equipar un avatar amb peces de certes categories: per exemple, un vestit, un casc, una armilla o cuirassa, una arma blanca curta, un arc de fletxes, una arma de foc curt i una arma de foc llarg. Per a cadascuna de les categories, el jugador ha de seleccionar un ítem de la corresponent categoria. Cada ítem té un preu i proporciona una puntuació.

Hi ha $k$ categories, $C_i$ on, per $1 \leq i \leq k$, $C_i = \{e_{i,0}, e_{i,1}, e_{i,2}, \ldots, e_{i,n_i}\}$, amb $n_i \geq 0$ per a tota $i$, i associat a cada possible ítem $e_{i,j}$ un preu $v_{i,j} > 0$ (excepte l'ítem $e_{i,0}$, que és de franc, és a dir, $v_{i,0} = 0$) i una puntuació $p_{i,j} \geq 0$. Per exemple, $C_1$ podria ser la categoria dels arcs de fletxes i podria haver-hi $n_1 + 1 = 4$ arcs de fletxes diferents, sent l'arc $e_{1,0}$ l'arc més bàsic, amb cost $v_{1,0} = 0$, i després hi hauria tres arcs més, amb diferents prestacions i costos. El nostre jugador haurà d'equipar el seu avatar amb un dels 4 tipus d'arc, de la mateixa manera que l'haurà d'equipar de vestit, proteccions corporals, etc. Fixeu-vos que pot haver-hi ítems amb puntuació 0 a més dels ítems bàsics $e_{i,0}$ de cada categoria.

Donades totes les dades del problema i la quantitat de diners $0 < Q \leq 1000$ disponibles, volem trobar una solució en la qual s'esculli exactament un ítem de cada categoria de manera que la suma dels costos és $\leq Q$ i que la puntuació combinada (suma de puntuacions) és màxima. Com que cada categoria conté un ítem $e_{i,0}$ gratis sempre serà possible trobar una solució respectant el pressupost $Q$. Proporcioneu un algorisme amb cost polinòmic (respecte de la mida de l'entrada) per resoldre aquest problema.

**Una solució:** Si tenemos un solución óptima $S = (e_{1,s_1}, \ldots e_{k,s_k})$, entonces $S = (e_{2,s_2}, \ldots e_{k,s_k})$ es una solución óptima del subproblema en el que consideramos las categorías $C_2$ a $C_k$ pero con presupuesto $Q - v_{1,s_1}$. Basándonos en esta estructura de recursividad podemos utilizar programación dinámica para resolver el problema.

El enunciado nos proporciona una cantidad de dinero, pero no nos indica las unidades en las que está expresado, asumo que $Q$ y las valoraciones son enteros expresados en función del valor de la moneda más pequeña en el sistema.

Teniendo en cuenta estas consideraciones, mi algoritmo rellenará una matriz $T[i, q]$, con $1 \leq i \leq k$ i $0 \leq q \leq Q$, tal que al finalizar el algoritmo $T[i, q]$ contenga la puntuación máxima que se puede obtener considerando las categorías $C_i, \ldots, C_k$ cuando solo disponemos de $q$ monedas.

De acuerdo con la estructura de suboptimalidad, $T$ se pude definir usando la siguiente recurrencia:

$$T[i,q] = \begin{cases} 0 & q = 0 \\ \max_{\{0 \leq j \leq n_k, v_{k,j} \leq q\}} p_{k,j} & i = k \\ \max_{\{0 \leq j \leq n_i, v_{i,j} \leq q\}} \{p_{i,j} + T[i+1, q - v_{i,j}]\} & \text{otherwise} \end{cases}$$

La tabla se puede rellenar siguiendo un recorrido por filas, empezando por la fila $k$. La puntuación máxima la obtendremos en la posición $T[1, Q]$.

Para obtener la solución óptima, de la forma habitual guardaremos en otra tabla $X$ el argumento de cada uno de los máximos en la recurrencia. Luego extraeremos una solución empezando en la posición $X[1, Q]$, que nos indicará el elemento $e_{1,s_1}$, a partir de aquí consultarmos la posición $X[2, Q - v_{1,s_1}]$, para obtener el segundo elemento y continuaremos hasta construir una solución óptima.

Para analizar el coste del algoritmo, considero el valor $N = \sum_{i=1}^{k} n_i$, la entrada contiene $O(N)$ números. Como $Q \leq 1000$, las tablas tienen $O(k)$ posiciones. Para rellenar la posición $[i, q]$ de $T$ iy $X$, tenemos que acceder a como mucho $n_i$ valores, por lo que rellenar $T$ y $X$ tiene coste total $O(QN) = O(N)$. Obtener la solución, después de calcular $X$ tiene coste $O(k)$. El coste total del agoritmo es $O(N)$ que es lineal en el tamaño de la entrada.

**Una solució:** Sigui $G_{i,q}$ la puntuació màxima que podem aconseguir equipant items de les categories 1 a $i$ amb un presupost $q \geq 0$. Un cas base trivial és $G_{i,0} = p_{1,0} + \ldots + p_{i,0}$, per a tota $i$, $1 \leq i \leq k$, és a dir, la suma de les puntuacions dels items gratuits $e_{1,0}$, $e_{2,0}$, $\ldots$, $e_{i,0}$. D'altra banda, si $q < 0$ llavors $G_{i,q} = 0$. Finalment un altre cas base seria $G_{i,q} = 0$, per a qualsevol $i \leq 0$ i qualsevol presupost $q$, doncs no equipem cap item. En general, si $1 \leq i \leq k$ i $q \geq 0$ tindrem

$$G_{i+1,q} = \max\{G_{i,q-v_{i+1,j}} + p_{i+1,j} \,|\, 0 \leq j \leq n_{i+1}\},$$

això és, agafarem l'item $j$, $0 \leq j \leq n_{i+1}$ de la categoria $i+1$ i la seva puntuació $p_{i+1,j}$ es suma a la màxima puntuació que podem obtenir amb items de les categories 1 a $i$, amb un pressupost de $q - v_{i+1,j}$; maximitzarem sobre totes les opcions possibles de $j$. Tenim doncs $\Theta(k \cdot Q)$ subproblemes i el cost de calcular-ne el subproblema $(i+1, q)$ serà $\Theta(n_i)$. Si definim una matriu $G$ de talla $(k+1) \times (Q+1)$ en primer lloc omplirem la fila 0 i la columna 0 utilitzant els casos base; i després omplirem la resta de la matriu de dalt ($i = 1$) a baix ($i = k$), i d'esquerra ($q = 1$) a dreta ($q = Q$). Per a omplir la posició $G[i][q]$ farem un bucle amb $n_i$ iteracions. La puntuació òptima buscada és $G_{k,Q}$.

```
vector<vector<int>> G(k+1, vector<int>(Q+1,0));
vector<vector<int>> p; // p[i][j] = puntuacio de l'item j de la categoria i, 1≤i≤k
vector<vector<int>> v; // v[i][j] = valor de l'item j de la categoria i, 1≤i≤k
                       // p[i].size() == v[i].size() = n_i+1
for (int q = 0; q <= Q; ++q) G[0][q] = 0;
for (int i = 1; i <= k; ++i) G[i][0] = G[i-1][0] + p[i][0];

for (int i = 1; i <= k; ++i) {
  for (int q = 1; q <= Q; ++q) {
    G[i][q] = p[i][0] + G[i-1][q]; // usant l'item gratis de la categoria i
    for (int j = 1; j < p[i].size(); ++j)
      if (q - v[i][j] >= 0) {
        if (G[i][q] < G[i-1][q-v[i][j]] + p[i][j])
          G[i][q] = G[i-1][q-v[i][j]] + p[i][j];
      }
  }
}

return G[k][Q];
```

El cost de l'algorisme de PD és $\Theta(k \cdot Q)$ en espai i $\Theta(n \cdot Q)$ en temps, si definim $n = \sum_{1 \le i \le k} n_i$. Alternativament podem donar una cota superior del cost una mica menys "fina": $O(kQ \max_i\{n_i\})$. Per reconstruir la solució òptima es pot definir una matriu $J[i][q]$ que ens dona l'índex $j$ del item de la categoria $i$ utilitzat en la solució òptima del subproblema $(i, q)$. El cost de en temps de la fase de reconstrucció és $\Theta(k)$.

```
...
vector<vector<int>> J(k+1,vector<int>(Q+1));

for (int q = 0; q <= Q; ++q) G[0][q] = 0;
for (int i = 1; i <= k; ++i) { G[i][0] = G[i-1][0] + p[i][0]; J[i][0] = 0; }

for (int i = 1; i <= k; ++i) {
  for (int q = 1; q <= Q; ++q) {
    G[i][q] = p[i][0] + G[i-1][q]; // usant l'item gratis de la categoria i
    J[i][q] = 0; // <= anotem quina es l'eleccio optima; de moment, l'item gratuit
    for (int j = 1; j < p[i].size(); ++j)
      if (q - v[i][j] >= 0) {
        if (G[i][q] < G[i-1][q-v[i][j]] + p[i][j]) {
          G[i][q] = G[i-1][q-v[i][j]] + p[i][j];
          J[i][q] = j;  // <= anotem quina es l'eleccio optima
      }
  }
}

// reconstruccio
vector<int> item(k+1);
int i = k; int q = Q;
item[i] = J[i][q];
while (i >= 1) {
  q -= v[i][item[i]]; --i;
  item[i] = J[i][q];
}
```

**Exercici 3 (3 punts) (BcnTrans)**

BcnTrans és una empresa de transport que s'encarrega de la distribució de mercaderies des del port de Barcelona cap a diferents destinacions. A partir de la seva experiència durant dècades, BcnTrans ha extret un mapa amb les rutes viables entre diferents localitats per als seus camions. Aquesta informació s'ha modelitzat com un graf dirigit $G = (V, E)$. L'empresa té previstes diferents condicions de transport, i els camions necessaris per fer-ho, però volen saber el nombre màxim de rutes des del port de Barcelona cap a la destinació (o destinacions) finals, d'acord amb d'altres requisits.

- **Transport pesat:** Donat un enter $k$ i una localitat de destinació, volen rutes des del port de Barcelona cap a la destinació final de manera que una aresta de $E$ no formi part de més de $k$ rutes.
- **Transport perillós:** A més de les condicions de l'apartat anterior, ara cada localitat té assignat un nivell de perillositat que indica el nombre màxim de camions que la poden travessar.
- **Transport amb destinació:** Les mateixes condicions de l'apartat anterior, però ara se sap el nombre de camions i és té un conjunt de destinacions. Per a cada destinació se sap, a més, el nombre de camions que han d'arribar-hi.

Dissenyeu algorismes per resoldre aquests problemes. Analitzeu-ne el seu cost i justifiqueu-ne la seva correctesa.

**Una solució:** Para resolver estos problemas utilizaré modelos de red de flujo obtenida a partir de $G$. El nodo $s$ será el puerto de BCN. En la red de flujo correspondiente, una unidad de flujo de $s$ a $t$ representa la ruta que sigue un camión desde el puerto hasta su destino final.

- **Transport pesat:** En este caso $\mathcal{N} = (G, s, t, c)$, siendo $s$ el puerto y $t$ la localidad de destino. Para asegurar que ninguna arista forma parte de más de $k$ rutas nos basta con asignar a $c(e) = k$, para $e \in E$.

  El valor del flujo máximo en $\mathcal{N}$ nos indicará el máximo número de rutas cumpliendo las condiciones.

  Para analizar el coste, podemos tener en cuenta que todas las aristas tienen la misma capacidad, y que por ello el algoritmo de FF nos garantiza que en cada iteración incrementamos el flujo en $k$ unidades. Como el valor del flujo máximo es $O(k|V|)$, ya que $s$ no puede tener más vecinos. El coste del algoritmo es $O(|V|(|V| + |E|)$. Si utilizamos el análisis de EK tendremos $O(|V||E|(|V| + |E|)$ que es asintóticamente peor.
- **Transport perillós:** En este caso $\mathcal{N} = (G', s, t, c)$. Para obtener $G'$ duplicaremos el conjunto de vértices incluyendo una copia $V'$, tendremos aristas $(v, v')$ con capacidad el número máximo de camiones que pueden pasar por $v$, para $v \in V$ y

aristas $(v', u)$ con capacidad $k$ cuando $(v, u) \in E$. Tomaremos $s$ el puerto en $V'$ y $t$ la localidad de destino en $V$.

Con esta construcción en un flujo válido ninguna arista pertenece a más de $k$ rutas y no pasarán por ninguna ciudad más camiones de los permitidos. Así el valor del flujo máximo nos indica el número máximo de rutas posibles respetando las restricciones.

En este caso no podemos asegurar que en cada iteración de FF se incremente el flujo en $k$ unidades, lo que nos da un coste $O(k|V|(|V| + |E|)$. El análisis usando EK sigue dándonos $O(|V||E|(|V| + |E|)$. Al no tener cotas al valor de $k$ el mejor coste es $O(|V||E|(|V| + |E|)$.

- **Transport amb destinació:** Notemos que si el número de rutas obtenido en el apartado anterior es menor que el número total de camiones $d$, sabemos que el transporte no se puede llevar a cabo. Si no es así, podemos resolver este apartado como un problema de circulación con demandas. La red será la misma del apartado anterior añadiendo aristas conectando los nodos de destino a $s$ con capacidad $\infty$. El puerto de barcelona ($s$) tendrá demanda $-d$, el número de camiones, y cada ciudad de destino $u$ demanda $d_u$, el número de camiones que tienen que llegar a $u$.

  La existencia de una circulación en esta red es equivalente a poder encontrar ruta que permitan circular a los $d$ camiones de origen a destino respetando las restricciones.

  El coste del algoritmo, como desconocemos cotas para $k$ y $d$, es el mismo que el del apartado anterior: $O(|V||E|(|V| + |E|)$.

**Comentario** El último apartado también se puede resolver utilizando una red de flujo con un nodo adicional $t$ y conectando las ciudades destino con $t$ con una arista con capacidad el número de camiones que lleguen a destino. El algoritmo obtine un flujo $f^*$ de valor máximo. El transporte es posible siempre y cuando $d \leq |f^*|$, si no no.

**Exercici 4 (3 punts)**

**4.1** (0.75 punts) Donats un conjunt $S$ amb $n$ enters positius i un altre enter $x \leq 100n$, es pot decidir en temps $O(n)$ si hi ha 2 elements a $S$ que sumin $x$?

**Una solució:** Primer notem que com que els enters son positius, per poder sumar $x$ amb un altre element necessitem que els dos elements siguin menors que $x$. Això vol dir que només cal tractar valors que siguin menors que $x$.

Faig servir una variació inspirada en l'algorisme d'ordenació per conteig. A una taula $T$ de $x$ elements, fent una pasada per els elements a $S$, contabilitzo quantes ocurrències de cada valor menor que $x$ apareixen a $S$.

Després, per $i = 1$ fins a $x/2$, si $T[i]$ i $T[x - i]$ son més grans que 1, la resposta es sí. L'únic cas que em de mirar amb cura és quan $x$ és parell, en aquest cas si arribem a $i = x/2$ hem de comprobar que $T[x/2] \geq 2$.

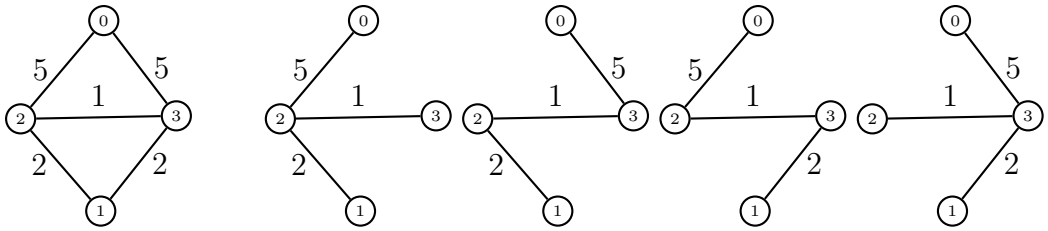
**4.2** (0.75 punts) Ens donen un graf no dirigit i ponderat $G = (V, E, w)$ mitjançant les llistes d'adjacència dels seus vèrtexs. El pes mitjà d'un vèrtex $u$ es defineix com la suma dels pesos de les arestes $(u, v) \in E$ dividit pel grau d'$u$. Es pot trobar en temps lineal un vèrtex amb pes mitjà més gran o igual que el de tres quartes parts dels vèrtexs, i més petit o igual que el de l'altre quart?

**Una solució:** El pes mitjà d'un vertex el podem obtenir recorrent le seva llista d'adjecència. Per tant podem obtenir els pesos mitjans de tots els vèrtexs en temps $O(n + m)$.

Després fem servir l'algorisme de selecció lineal per trobar l'element que ocupa la posició $3n/4$ en la llista de vèrtexs respecte els seus pesos mitjans. Això ho podem fer amb cost $O(n)$. L'algorisme té cost $O(n + m)$ i per tant és lineal en la mida de l'entrada.

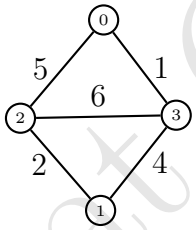
**4.3** (0.5 punts) Tenim un graf no dirigit amb pesos a les arestes on, a més, cada pes apareix com a molt dues vegades. Tenim com a molt dos MSTs diferents en aquest graf?

**Una solució:** No, tal i com es pot ver al graf de sota. Els pesos compleixen la propietat demanada però el graf en té 4 MSTs diferents.




**4.4** (0.5 punts) Si totes les arestes d'un graf no dirigit i connex tenen pesos positius i diferents, és únic el camí més curt entre qualsevol parell de vèrtexs?

**Una solució:** NO, tal i com es pot ver al graf de sota. Els pesos compleixen la propietat demanada però en el graf ehi han tres camins amb distancia 6 del node 2 al 3. Tots tres son de distància mínima.




**4.5** (0.5 punts) Tenim una xarxa de flux $\mathcal{N}$ i un flux $f$ amb valor màxim a ella. Per cada aresta $e$ ens donen el cost $c_e$ de transportar una unitat de flux a través seu. Doneu un algorisme per determinar si $f$ té cost mínim entre tots els fluxos de valor màxim.

**Una solució:** Tal i com he vist a classe si $f$ no té cost mínim al graf residual de $G_f$ quan assignem pes $c_e$ a les arestes cap en davant i pes $-c_e$ a les arestes cap enrrera no hi han cicles amm pes negatiu.

Hem vist a clase com comprobar aquesta propietat en temps $O(nm)$ afegint al graf un node adicional connectat amb tots els nodes amb pes 0 i executant l'algorisme de Bellman Ford.