Algorísmia QT 2020–2021

Examen final **12 de Gener de 2021**

Duració: 2h 30min

Instruccions generals:

- Cal que argumenteu la correctesa i l'eficiència de tots els algorismes que proposeu. Per això podeu donar una descripció d'alt nivell de l'algorisme amb les explicacions i aclariments oportuns que permetin concloure que l'algorisme és correcte i té el cost indicat. La puntuació dependrà fortament d'aquesta argumentació.
- Podeu fer crides a algorismes que s'han vist a classe, però si la solució és una variació n'haureu de donar els detalls.
- Es valorarà especialment la claredat i concisió de la presentació.
- Entregueu **per separat** les vostres solucions de cada exercici (Ex 1, Ex 2 i Ex 3).
- Cal que resoleu l'exercici 1 fent servir un únic full.
- La puntuació total d'aquest examen és de **10 punts**.

Exercici 1 (4 punts) Cal que justifiqueu les vostres respostes. No podeu fer servir més d'un full per resoldre aquest exercici.

(a) **(0.75 punts)** Fent servir Radix Sort, podem ordenar n enters amb valor més petit o igual que $2^{\log_2(n^2)}$ en temps $O(n)$?

Una solució: Els n enters donats estan dins del rang $[0..n^2]$ i per tant es necessiten $2(\log_2 n)$ bits per a representar-los. Si fem servir base $b = \log_2 n$, el nombre de dígits necessaris és 2. Per tant, el cost de radix sort si utilitzem base $b = \log_2 n$ és $\Theta(2(n + \log_2 n)) = \Theta(n)$.

(b) **(0.75 punts)** Tenim una taula A no ordenada que conté n elements diferents. Podem obtenir en $O(n)$ passos una taula B amb $n/4$ elements d'A tals que A i B tinguin la mateixa mitjana?

Una solució: Utilitzem l'algorisme de selecció en temps lineal per a trobar els elements amb rangs $r_1 = 3n/8$ i $r_2 = 5n/8$ en A, amb cost $\Theta(n)$. Un filtrat dels elements entre el r_1-èssim i el r_2-èssim d'A (de fet, l'algorisme de selecció ja ens deixa al subvector $A[r_1 \dots r_2]$ aquests elements) ens dóna el conjunt de $n/4$ elements tals que la seva mediana és la mateixa que la del vector original A.

(c) **(0.75 punts)** Donat un graf no dirigit ponderat i connex $G = (V, E, w)$, proporcioneu un algorisme (el més eficient que pugueu) per a trobar un subgraf connex $G' = (V, E')$ tal que la suma dels pesos de les arestes a E' sigui mínima.

Una solució: S'ha d'utilitzar un algorisme eficient de càlcul de l'arbre d'expansió mínim (MST) doncs és el subgraf que es demana (connex, conté tots els vèrtexos del graf original, té pes mínim). Podem fer servir l'algorisme de Prim o el de Kruskal, amb cost $O(|E| \log |V|)$ ambdós.

(d) **(1 punt)** Un graf *unicíclic* és un graf no dirigit que conté només un cicle. Sigui donat un graf ponderat unicíclic $G = (V, E, w)$, on $w : E \to \mathbb{R}$, i un vèrtex $u \in V$. Proporcioneu un algorisme (el més eficient que pugueu) per a trobar les distàncies d'u a tots els altres vèrtexs de G en cas que sigui possible definir-les.

Una solució: Un DFS del graf amb cost $O(|V| + |E|) = O(|V|)$ ens permetrà identificar quins vèrtexos formen part del cicle (en un graf unicíclic connex $|E| = |V|$, i en general, $|E| = |V| - \#\text{CC's} + 1$) i a quina CC pertany cada vèrtex. Per tot vèrtex que no estigui a la mateixa CC que u la distància és $+\infty$.

Pels vèrtexos a la mateixa CC que u tindrem dos casos, segons el cicle formi part o no formi part de la CC d'u.

Si el cicle no forma part de la CC d'u, aquesta component connexa és un arbre i la distància d'u a qualsevol altre vèrtex v de la seva CC és la suma dels pesos de les arestes a l'únic camí d'u a v. Aquesta distància es pot calcular fent servir un BFS (o un DFS) incrementant la distància a u amb el pes de la aresta considerada al BFS.

Si el cicle forma part de la CC d'u, el primer que hem de fer és comprovar que no té pes negatiu (en temps $O(|V|)$). Si el cicle té pes negatiu, aleshores la distància no es pot definir. Altrament, la distància es pot definir. En aquest darrer cas:

a) si u forma part del cicle, siguin x i y els seus veïns al cicle i $z = u$, i
b) si u no forma part del cicle, sigui w el primer vèrtex que sí forma part del cicle visitat durant un BFS (o DFS) que comença a u; aleshores agafem x i y com els veïns de w al cicle i $z = w$.

Calculem les distàncies $d_1[v]$ d'u a la resta de vèrtexos v en $G_1 = G \setminus \{(z, x)\}$ i les distàncies $d_2[v]$ d'u a la resta de vèrtexos v en el graf $G_2 = G \setminus \{(z, y)\}$. Tant G_1 com G_2 són arbres i les distàncies es poden calcular en temps $O(|V|)$. Finalment les distàncies buscades són $d[v] = \min(d_1[v], d_2[v])$.

El cost total de l'algorisme és $O(|V|)$.

(e) **(0.75 punts)** Si la capacitat de les arestes d'una xarxa $\mathcal{N}$ és $\leq c$, per una certa constant c, quin algorisme és més eficient en cas pitjor per trobar un flux de valor màxim a $\mathcal{N}$: Ford-Fulkerson o Edmonds-Karp?

Una solució: El cost en cas pitjor d'Edmonds-Karp (EK) és $O(nm^2)$, independentment del valor de c. El cost en cas pitjor de Ford-Fulkerson (FF) és $O(Cm)$, on C és la capacitat del min-cut. Donat que $C \leq cm$ i c és constant, el cost de Ford-Fulkerson ens queda $O(m^2)$, i per tant és més eficient FF que EK.

Exercici 2 (3 punts) (Rèpliques)

Tenim un sistema S format per la concatenació de n subsistemes S_1, S_2, ..., S_n. Per a cada subsistema S_i es coneix la seva probabilitat de fallida ϕ_i. La probabilitat p_{corr} que el sistema funcioni correctament és la probabilitat que **tots** els subsistemes funcionin correctament, és a dir:

$$p_{\text{corr}} = \Pi_{1 \leq i \leq n}(1 - \phi_i)$$

Ara bé, per tal d'augmentar la probabilitat que el sistema funcioni correctament podem replicar els subsistemes; així, si posem $x_i > 0$ rèpliques del subsistema S_i la probabilitat que falli passa a ser

$$\phi_i' = \phi_i^{x_i},$$

donat que només fallarà si les x_i rèpliques fallen. Desgraciadament tenim un pressupost limitat de $B \in N$ euros en total, el cost del subsistema S_i és $v_i \in N$ i només hi ha y_i unitats del subsistema S_i.

Es demana que plantegeu la resolució d'aquest problema mitjançant un algorisme de programació dinàmica (PD) que calculi el nombre de rèpliques x_1, ..., x_n, amb $x_i \geq 1$, $1 \leq i \leq n$, tal que la probabilitat que S funcioni correctament sigui màxima. Les dades del problema són:

- les probabilitats de fallida ϕ_i, $1 \leq i \leq n$,
- el pressupost $B \geq v_1 + \ldots + v_n$ (es podrà comprar una unitat de cada subsistema, com a mínim),
- l'*stock* $y_i \geq 1$ de cada subsistema (s'ha de complir $x_i \leq y_i$), i
- el valor v_i de cada unitat del subsistema S_i (s'ha de complir $\sum_i v_i x_i \leq B$).

En particular, es demana que:

(a) Proporcioneu la recurrència que ens permeti calcular la màxima probabilitat de funcionament correcte, donades les restriccions d'stock i pressupost.

(b) Desenvolupeu un algorisme de PD a partir de la recurrència.

(c) Calculeu el cost en temps i espai del vostre algorisme i demostreu que és polinòmic en n i B.

(d) Descriviu com ampliar/modificar l'algorisme per tal d'obtenir els valors x_1, ..., x_n que donen la solució òptima.

Una solución: Nuestra solución de PD para este problema se asemeja bastante a la solución PD para el problema de la mochila (*knapsack*). Pero la decisión sobre cada "objeto" no es binaria: podrá ponerse entre 1 y y'_i copias, donde y'_i es el máximo de copias del objeto en cuestión, limitado por el número de copias y_i en stock y por el dinero disponible para comprarlas.

- Sea $P(i;C)$ la máxima probabilidad de funcionamiento correcto de los subsistemas i a n, $1 \leq i \leq n$, con un presupuesto de C euros.

  Consideremos en primer lugar el caso base $P(n;C)$. Esta probabilidad será la que se obtiene al usar el máximo número posible de replicas, solo limitado por el stock y_n y por el dinero disponible (podremos comprar como máximo $\lfloor C/v_n \rfloor$ réplicas). Es decir, para toda n y toda C, tomaremos $x^*_n := \min(y_n, \lfloor C/v_n \rfloor)$ y

  $$P(n;C) = 1 - \phi_n^{x^*_n},$$

  si $x^*_n \geq 1$, y $P(n;C) = 0$ en caso contrario.

  Otro caso base trivial es $P(i;C)$ con $C \leq 0$. Por convenio podemos decir que $P(i;C) = 0$ para cualquier i, $1 \leq i \leq n$, si $C \leq 0$, puesto que no podemos adquirir ni una sola unidad de los subsistemas requeridos. De hecho, este razonamiento también sirve si $C < v_i + \cdots + v_n$.

  En general, si definimos $y'_i := \min(y_i, \lfloor C/v_i \rfloor$ tendremos

  $$P(i;C) = \max_{1 \leq x \leq y'_i} \left\{ (1 - \phi_i^x) \cdot P(i+1; C - x \cdot v_i) \right\}, \qquad (*)$$

  En la solución óptima pondremos un cierto número de réplicas x que estará necesariamente entre 1 y y'_i; entonces la probabilidad de funcionamiento correcto será $(1 - \phi_i^x)$ —la probabilidad de que el subsistema S_i, con x réplicas, funcione correctamente— por la probabilidad óptima con la que funcionan los subsistemas $i+1$ a n y descontando del presupuesto C el coste xv_i de las réplicas del subsistema i. Como queremos maximizar $P(i;C)$ se tomará el x que nos de el valor máximo entre las opciones posibles.

- Consideraremos ahora de manera conjunta los tres "apartados" siguientes del problema.

  El algoritmo de PD comienza definiendo dos matrices bidimensionales P y X con n filas ($n+1$ filas, pero la fila 0 se "descarta") y $B+1$ columnas cada una. En $P[i][C]$ guardaremos $P(i;C)$ y en $X[i][C]$ el valor x^*_i que maximiza la probabilidad $P(i;C)$. Esto es, si $i < n$ entonces $x_i = X[i][C]$ es el valor con el cual alcanzamos el máximo en la recurrencia $(*)$ para $P(i;C)$. Para $i = n$, $x^*_n = X[n][C] = \min(y_n, C/v_n)$.

  El valor deprobabiliad buscado es el que obtendremos en $P[1][B]$. No se necesitarán otras estructuras de datos adicionales y por lo tanto el coste en espacio será $\Theta(nB)$.

  Todas las entradas de P y X se inicializan a 0, y después se rellenan las filas n-ésimas de P y X:

```
vector<vector<double>> P(n+1,vector<double>(B+1, 0.0));
vector<vector<int>> X(n+1,vector<int>(B+1, 0));

for (int C = v[n]; C <= B; ++C) {
    P[n][C] = 1-power(phi[n], min(y[n], C/v[n]));
    X[n][C] = min(y[n],C/v[n]);
}
// N.B. power(x,y) calcula x elevado a y
```

y a partir de esta "base de la recursión", las restantes filas, de abajo ($i = n - 1$) hacia arriba ($i = 1$):

```
for (int i = n-1; i >= 1; --i)
    for (int C = 0; C <= B; ++C)
      for (int x = 1; x <= min(y[i], C/v[i]); ++x)
          if (P[i][C] < (1-power(phi[i],x)) * P[i+1][C-v[i]*x]) {
            P[i][C] = (1-power(phi[i],x)) * P[i+1][C-v[i]*x];
            X[i][C] = x;
          }
```

El coste de calcular cada $P[i][C]$ es $O(\min(y_i, C/v_i)) = O(B)$. Por tanto el coste del algoritmo (en tiempo) puede acotarse superiormente por $O(nB^2)$. No obstante esta cota es bastante pesimista. Por ejemplo si el número máximo de réplicas de cualquier subsistema es Y entonces el coste del algoritmo será $O(nBY)$ y es habitual que $Y \ll B$. Puede refinarse aún más si se tiene en cuenta el coste por réplica; si el coste por réplica más barata es V ($v_i \geq V$) entonces el coste del algoritmo de PD es $O(nB \min(Y, B/V))$.

La solución óptima puede reconstruirse muy fácilmente con un simple recorrido secuencial y coste $\Theta(n)$:

```
int remain_budget = B;
for (int i = n; i >= 1; --i) {
  xopt[i] = X[i][remain_budget];
  remain_budget = remain_budget - xopt[i] * v[i];
}
```

Exercici 3 (3 punts) (Vacunació).

La Generalitat ha de planificar els trasllats de les vacunes Covid des de l'aeroport del Prat als diferents centres de distribució de la vacuna a Catalunya. Per dur-ho a terme, ha creat un graf de distribució format per les localitats catalanes a on es poden emmagatzemar vacunes amb les condicions de conservació adients (incloent-hi el propi aeroport). Les connexions entre localitats garanteixen que el transport es pot portar a terme des d'una localitat fins a qualsevol altra, seguint una o més connexions, en temps suficientment curt per tal de no trencar la cadena del fred.

Per a cada localitat, fora del Prat, es coneixen la quantitat màxima de caixes de vacunes que es poden emmagatzemar de forma segura.

Per raons de seguretat, el nombre total de caixes de vacunes que poden circular per una localitat no pot superar la capacitat d'emmagatzematge de la localitat.

(a) Dissenyeu un algoritme que, donats el graf de distribució, la quantitat de caixes que es pot emmagatzemar a cada localitat, les localitats que són centres de distribució i, per a cada centre de distribució, el nombre de caixes de vacunes que es volen traslladar des del magatzem de l'aeroport a ell, determini si és possible fer el trasllat respectant les restriccions descrites abans.

Podeu suposar que al magatzem del Prat hi ha més caixes de vacunes de les que es volen distribuir.

(b) En cas que el trasllat no es pugui portar a terme, esteneu l'algorisme precedent per tal d'identificar un conjunt de localitats on un increment de seva capacitat d'emmagatzematge garantiria poder portar a terme el trasllat.

Una solución:

(a) Podemos resolver el problema como un problema de circulación con demandas en una red adecuada. La limitación de transporte está en la cantidad de cajas de vacunas que pueden atravesar una localidad.

Sea $G = (V, E)$ el grafo que nos proporcionan y $a(u)$, $u \in V$, la cantidad de cajas que se pueden almacenar la localidad u. Suponemos que $s \in V$ es el nodo que representa el almacén del Prat. Si $D \subset V - \{s\}$ es el conjunto de centros de distribución, se nos dará $b(w) > 0$, $w \in D$, el número de cajas de vacunas que queremos trasladar desde el Prat al centro de distribución w.

Asumimos que el grafo que nos dan es dirigido, si no lo fuese, tal y como hemos visto en clase se convertiría en dirigido poniendo los arcos (u, v) y (v, u) para cada arista $\{u, v\}$ del grafo no dirigido.

La red $\mathcal{N}$ tiene

- Nodos: V y V', siendo V' una copia de los nodos en V. Si $u \in V$, entonces u' representa su copia en V'.
- Aristas y capacidades:

$$\begin{array}{rl} \{(u, u') \mid u \in V\} & \text{capacidad } a(u) \\ \{(u', v) \mid (u, v) \in E\} & \text{capacidad } \infty \end{array}$$

- Demandas:

$$u \in D, d(u) = b(u)$$
$$d(s) = -\sum_{u \in D} b(u)$$
$$u \notin D, u \neq s, d(u) = 0$$

La duplicación de los nodos y la capacidad de la arista que conecta un nodo con su copia garantizan que, en un flujo entero, nunca pasan por un nodo más cajas de las que es posible almacenar en él.

Teniendo en cuenta que el aeropuerto tiene suficientes cajas para satisfacer la demanda, el problema de circulación tiene solución si y solo si podemos realizar el transporte requerido ya que las restricciones solicitadas están garantizadas por la capacidad de las aristas.

Para analizar el coste, asumo que G tiene n vértices y m aristas. Así $\mathcal{N}$ tiene $2n$ vértices y $n + m$ aristas. Como no tenemos cotas en el valor de las demandas y $-d(s)$ puede ser grande, el coste del algoritmo viene dado por el análisis de EK con coste $O(NM^2) = O(n(n + m)^2)$.

(b) Si la circulación no existe, lo que podemos hacer es considerar la red de flujo utilizada para resolver el problema de circulación y el flujo con valor máximo que hemos calculado en dicha red. Utilizando el grafo residual, obtenemos un corte con capacidad mínima. Como ninguna de las aristas del corte puede tener capacidad ∞, ya que no permite el transportar todas las cajas requeridas, cada arista de este corte identifica un vértice.

Miraríamos en cuanto se incrementa el valor del flujo asignando capacidad ∞ a las aristas del corte. Si con esto no es suficiente para que el problema de circulación tenga solución. Repetiremos el proceso, con la nuevo red y el nuevo flujo máximo, tantas veces como sea necesario. El algoritmo acabará en el momento en que la circulación buscada exista.

El conjunto de aristas (u, u') con capacidad ∞, nos proporciona el conjunto de nodos pedidos. Observemos que en cada paso el corte con capacidad mínima no puede contener ninguna de las aristas a las que hemos asignado capacidad ∞, por lo que como mucho repetiremos este proceso $O(n)$ veces, cada una de ellas con el coste de EK $O(n(n + m)^2)$. Así el coste total del algoritmo es $O(n^2(n + m)^2)$.