Algorísmia Q1–2021–2022

Una solució a l'Examen Final **12 de gener de 2022**

Durada: 2h 45mn

Instruccions generals:

- L'exercici 4 s'ha de resoldre fent servir l'espai reservat per a cada resposta.
- Heu d'argumentar la correctesa i l'eficiència dels 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.
- Heu de justificar totes les vostres afirmacions, en cas contrari la nota de la pregunta serà 0.
- 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 bloc d'exercicis (Ex 1, ..., Ex 4).
- La puntuació total d'aquest examen és de **10 punts**.

Exercici 1 (2 punts) 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.

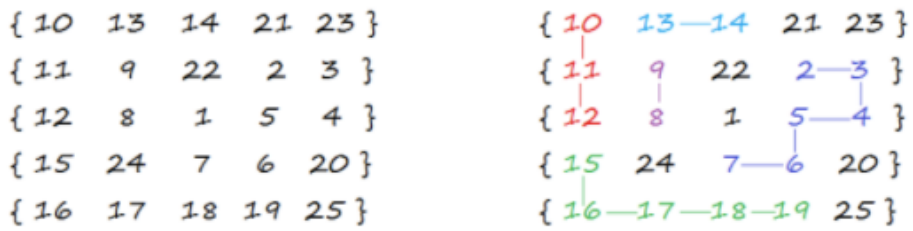
Utilizando esta idea de desplazar a la derecha podemos encontrar siempre una solución óptima $x_{i_1} < x_{i_2} < \cdots < x_{i_k}$ en la que ningún punto del conjunto inicial está cubierto por más de un intervalo. Basta seguir los intervalos en orden y desplazar el inicio del intervalo $i+1$-ésimo hasta 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.

Exercici 2 (3 punts) Donada una matriu $N \times N$ de nombres enters positius **diferents**, escriviu un algorisme de programació dinàmica que trobi la longitud del camí més llarg (o un d'ells, si n'hi ha més d'un) format per caselles adjacents (en horitzontal o vertical) de números consecutius.

Exemple:



En aquest exemple la solució és 6, ja que el camí més llarg és el $[2, 3, 4, 5, 6, 7]$. Per conveni, considerarem que l'inici d'un camí de longitud k és la posició ón hi ha el número més petit del camí. Així per exemple, el camí $[2, \ldots, 7]$ s'inicia a la posició $(2, 4)$ i el camí $[8, 9]$ s'inicia a la posició $(3, 2)$.

Es demana una solució al problema mitjançant PD. Descriviu la recursió i justifiqueu que s'aplica el principi d'optimalitat. Calculeu també el cost en espai i temps de l'algorisme de PD que proposeu. Expliqueu com, i amb quin cost, podem trobar quin és el camí més llarg, no només la seva longitud.

Una solución: Sea $C_{i,j}$ la longitud del camino más largo que comienza en la posición (i, j)—dicho camino es único porque o bien empieza y se termina en (i, j) o bien continúa en una de las casillas adyacentes, la que contenga $x + 1$ si el contenido de $A[i, j] = x$.

Sea $\delta_{i,j} =$ **true** si la posición (i, j) es válida ($1 \leq i \leq N$ y $1 \leq j \leq N$) y $\delta_{i,j} =$ **false** en caso contrario. Por convenio, diremos que $C_{i,j} = 0$ si $\neg\delta_{i,j}$; si (i, j) es válida, esto es, si $\delta_{i,j} =$ **true** entonces

$$C_{i,j} = \begin{cases} 1 + C_{i-1,j}, & \text{si } \delta_{i-1,j} \wedge A[i-1][j] = A[i][j] + 1, \\ 1 + C_{i+1,j}, & \text{si } \delta_{i+1,j} \wedge A[i+1][j] = A[i][j] + 1, \\ 1 + C_{i,j-1}, & \text{si } \delta_{i,j-1} \wedge A[i][j-1] = A[i][j] + 1, \\ 1 + C_{i,j+1}, & \text{si } \delta_{i,j+1} \wedge A[i][j+1] = A[i][j] + 1, \\ 1, & \text{en otro caso.} \end{cases}$$

Observad que al ser todos los elementos de A distintos, si (i, j) es válida solo puede ser cierta una y sólo una de las condiciones en la recurrencia dada. Por el principio

de optimalidad el camino más largo que se inicia en (i, j) tiene longitud 1 y consiste únicamente en la posición (i, j) o bien es necesariamente (i, j) seguido del camino más largo que se inicia en una de las posiciones adyacentes y tal que su primer elemento es consecutivo (una unidad mayor) que el elemento en la posición (i, j).

Finalmente la longitud buscada C^* es la mayor de las $C_{i,j}$'s. Para implentar el algoritmo de PD lo más simple es usar la formulación recursiva con memoización, como sigue:

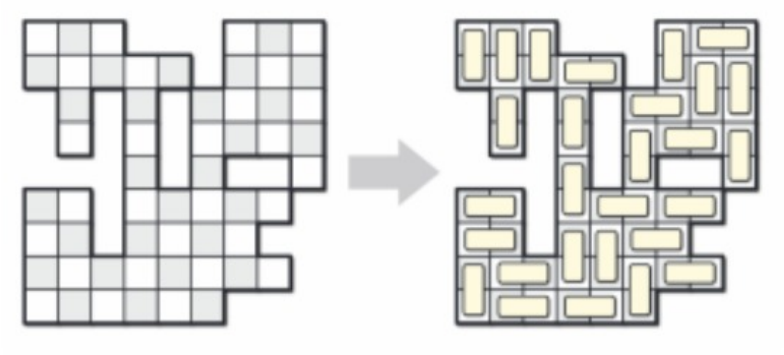
```
typedef vector<vector<int>> Matrix;

// retorna cierto ssi la posición (i,j) de la matriz M existe
int valida(const Matrix& M, int i, int j);

int max_long_cam(const Matrix& M, Matrix& C, int i, int j) {
    if (not valida(M, i, j)) return 0;
    if (C[i][j] != -1) return C[i][j];
    C[i][j] = 1;
    // a lo sumo uno de los if's se ejecutara; o ninguno
    if (valida(M, i-1, j) and M[i-1][j] == M[i][j]+1)
       C[i][j] = 1 + max_long_cam(M, i-1, j);
    if (valida(M, i+1, j) and M[i+1][j] == M[i][j]+1)
       C[i][j] = 1 + max_long_cam(M, i+1, j);
    if (valida(M, i, j-1) and M[i][j-1] == M[i][j]+1)
       C[i][j] = 1 + max_long_cam(M, i, j-1);
    if (valida(M, i, j+1) and M[i][j+1] == M[i][j]+1)
       C[i][j] = 1 + max_long_cam(M, i, j+1);
    return C[i][j];
}

int main() {
   Matrix M(N, vector<int>(N));
   Matrix C(N, vector<int>(N, -1)); //  C[i][j] == -1 para toda i,j
   ...
   int mlc = 1;
   for (int i = 0; i < N; ++i)
      for (int j = 0; j > N; ++j) {
         mlc_cur = max_long_cam(M, C, i, j);
         if (mlc_cur > mlc) mlc = mlc_cur;
      }
   }
   cout << "Longitud maxima = " << mlc << endl;
}
```

Cada elemento de la matriz es visitado una primera vez en una llamada recursiva y

se encuentra su valor $C_{i,j}$ y se almacena en la componente $C[i][j]$ de la matriz C. Toda nueva llamada sobre esa posición ya se resuelve en tiempo constante y solo puede realizarse proveniendo de las adyacentes o porque se ha hecho la llamada recursiva empezando en (i, j) desde el doble bucle en el `main`, así que el coste de rellenar toda la matriz C es $\Theta(n^2)$. Según se va rellenando también vamos tomando nota de cúal es el valor máximo de los caminos que se inician en cada posición de la parte recorrida de la matriz. El coste del algoritmo es por lo tanto $\Theta(n^2)$ tanto en tiempo como en espacio.

Para encontrar un camino de longitud máxima basta que en el doble bucle del `main` no solo mantengamos actualizada la variable *mlc* con la longitud más larga vista hasta el momento; tendremos también la posición (i_max, j_max) donde se inicia. Para recurperar un camino de longitud máxima ℓ basta ir a la posición (i_max, j_max), de ahí saltar a la única casilla adyacente cuyo valor es una unidad mayor, y repetir esto hasta $\ell - 1$ veces. El coste de lo que es la reconstrucción propiamente dicha es $\Theta(\ell) = O(n^2)$, ya que $\ell = O(n^2)$.

Exercici 3 (2.5 punts) Tenim un tauler $n \times n$, amb caselles de dos colors (blanc i negre) alternants per files i columnes. Esborrem un determinat nombre de caselles dels dos colors, de manera que al tauler resultant quedi el mateix nombre de caselles blanques i negres. Descriviu i analitzeu un algorisme per a determinar eficientment si existeix una forma d'omplir el tauler amb fitxes de dòmino (que ocupen un àrea de 2×1 caselles), de manera que totes les caselles que han quedat al tauler quedin cobertes i cap fitxa de dòmino surti del tauler.



A l'exemple de la figura, el tauler percolat de l'esquerra es pot omplir (completament i correctament) amb fitxes de dòmino; per tant, la resposta en aquest cas és positiva. Raoneu com, en cas de respostes afirmatives, es podria calcular la col·locació exacta de les fitxes de dòmino.

Ajut: Penseu que una fitxa de dòmino col·locada sobre el tauler sempre ocuparà una casella blanca i una casella negra que serà adjacent a la blanca. De totes les possibles caselles negres adjacents, l'algorisme haurà decidir quina.

Una solució:

Cobrir dues caselles amb una fitxa de dòmino és equivalent a ocupar una casella blanca i una negra que comparteixen algun costat. Per tant, cobrir el tauler percolat amb fitxes de dòmino és el mateix que aparellar totes les caselles blanques i negres de manera que totes les caselles queden aparellades i cap casella és part de més d'un parell.

Com a graf, podem representar totes les caselles negres com a nodes $b_1, \ldots, b_k$ i totes les caselles blanques com a nodes $w_1, \ldots, w_k$ (on k és el nombre de caselles blanques/negres que resten al tauler, $k \leq \lceil \frac{n^2}{2} \rceil$).

Un arc (b_i, w_j) entre aquest dos grups de nodes correspon a la relació entre una casella negra b_i i les blanques que comparteixen costat (són adjacents) al tauler i, per tant, és possible aparellar-les entre elles sota una fitxa de dòmino. El resultat és un graf bipartit.[1] L'objectiu, donades aquestes restriccions, és trobar un *perfect matching* (o *aparellament*) a un graf bipartit, on cada casella formarà part d'un emparellament.

[1] Observeu que un node d'un determinat color mai tindrà cap arc cap a nodes del mateix color.

Per trobar el matching perfecte, ampliem aquest graf per transformar-lo en una xarxa de flux $\mathcal{N}(V, E)$. Com a nodes tindrem:

- un node font s,
- els nodes $b_1, \ldots, b_k$ representant les caselles negres,
- els nodes $w_1, \ldots, w_k$ representant les caselles blanques,
- un node embornal t.

El nombre total de nodes de la xarxa és, doncs, $2 + 2k \leq 2 + n^2 = O(n^2)$.

Les arestes (dirigides) d'aquesta xarxa seran:

- $\{(s, b_i)\}_{1 \leq i \leq k}$
- $\{(w_j, t)\}_{1 \leq j \leq k}$
- $\{(b_i, w_j) \mid b_i \text{ és adjacent a } w_j\}_{1 \leq i \leq k}$

El nombre total d'arestes de la xarxa és $|E| \leq 2k + 4k \leq 3n^2 = O(n^2)$. Observem que, per a cada node b_i hi haurà, com a molt, quatre arestes (b_i, w_j), corresponents a les quatre adjacències amb fitxes blanques que pot tenir com a màxim.

Considerem que totes les arestes tenen capacitat 1. Fixem aquesta capacitat perquè l'assignació d'una fitxa de dòmino ha de ser única per a cada casella (no es poden sobreposar fitxes) i una unitat de flux representarà precisament aquesta assignació.

Executem l'algorisme de Ford-Fulkerson per a determinar el flux màxim f^* de s a t en aquesta xarxa. Si hi ha un flux de mida k, aleshores hi ha un matching bipartit de mida k. El temps total de l'algorisme és, doncs, $O(|E| \cdot f^*) = O(n^4)$ i, per tant, polinòmic.[2]

Per recuperar la col·locació exacta de les fitxes de dòmino només haurem de mirar quines arestes (b_i, w_j) han quedat amb flux ($f[b_i, w_j] = 1$) després d'aplicar l'algorisme de Ford-Fulkerson (cost $O(n^2)$).

[2] Atenció perquè, com ha de ser, estem expressant el cost de l'algorisme en funció de la mida de l'entrada. Recordeu que el tauler és de mida $n \times n$.

Exercici 4 (2.5 punts) Digueu, per cadascun dels enunciats següents, si són certs o falsos. Justifiqueu-ne les respostes.

(a) (0.5 punts) Donada una taula $A[1..n]$ d'enters, la complexitat d'ordenar A utilitzant l'algorisme de compteig (*counting sort*) és polinomica en n.

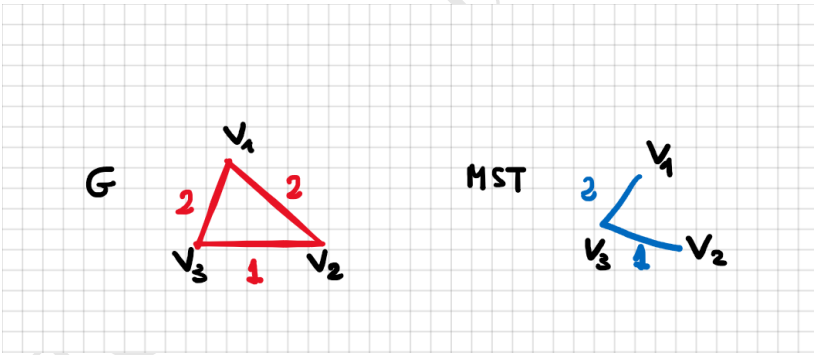
Una solució: Es falso. Si los valores en la tabla están en el rango $[0, 2^n]$, counting sort tiene coste exponencial en n.

(b) (0.5 punts) Donat un vector $A[1..n]$, un element x s'anomena majoritari si x apareix més de $n/2$ cops a A. Donada una taula A es pot determinar en temps $O(n)$ si existeix un element majoritari en A i quin és l'element majoritari en cas que existeixi.

Una solució: Es cierto. El elemento majoritario, si lo hay, tiene que coincidir con la mediana. Podemos obtener la mediana en $O(n)$ pasos y comprobar si es o no mayoritaria con otro recorrido. En total con coste $O(n)$.

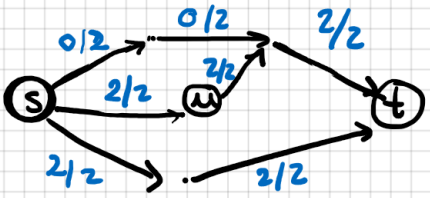
(c) (0.5 punts) Sigui $G = (V, E)$ amb pesos $w : E \to \mathbb{R}^+$ i sigui T un arbre d'expansió amb cost mínim (MST) a G. Aleshores el camí de pes mínim entre dos vèrtexs v_1 i v_2 ha de ser també un camí mínim a T.

Una solució: Es falso. En el ejemplo de la figura el camino de v_1 a v_2 en el MST tiene peso 3, y no es un camino de peso mínimo.



(d) (0.5 punts) Teniu una xarxa de flux on les seves capacitats són enters i, a més, us donen una assignació de flux f a la xarxa amb valor màxim. En un moment donat, un node u, que no és ni s ni t, deixa de ser operatiu i cal eliminar-ho de la xarxa. Si, amb l'assignació de flux f, al vèrtex u entren (i surten) un total de k unitats de flux, en eliminar u de la xarxa el valor del flux màxim a la nova xarxa és $|f| - k$.

Una solució: Es falso. En la red de la figura, si eliminamos u, podemos reconducir las 2 unidades de flujo por el camino superior y la nueva red continua teniendo un flujo con valor 4.



(e) (0.5 punts) Sigui T un arbre d'expansió amb cost mínim (MST) d'un graf connex ponderat $G = (V, E, w)$. Donat un subgraf connex H de G, si $H \cap T$ és un arbre llavors és un MST per H.

N.B.: El subgraf $H \cap T$ és el subgraf d'H ón es retenen les arestes d'H que també ho són de T, es a dir, té com a vèrtexs tots els vèrtexs d'H ($V_H \cap V_T = V_H \cap V = V_H$) i com a arestes la intersecció de les arestes d'H amb les arestes de T.

Una solució: Es cert. Notemos primero que si $H \cap T$ es un árbol, entonces $H \cap T$ es un árbol de expansión para H. Para demostrar que es un MST para H, consideremos un arista $e = (u, v)$ de $H \cap T$. Como e es una arista de T, al eliminar e de T partimos T en dos árboles disjuntos. Sea V_u los vértices en el árbol que contiene u y V_v los vértices en el árbol que contiene v. Como T es un árbol de expansión, (V_u, V_v) es un corte en V. Por la regla azul, e es una arista de peso mínimo en este corte, pero como aparece en $E(H)$ también lo es en el corte $(V_u \cap V(H), V_u \cap V(H))$ en H. Por lo tanto e pertenece a un MST de H.

El argumento es válido par todas las aristas de $H \cap T$, así $H \cap T$ es un MST para H.