Algorísmia QT 2022–2023

Examen Final **12 de gener de 2023**

Durada: 3 hores

Exercici 1 (2 punts)

Considereu un camí entre dos vèrtexs s i t en un graf no dirigit G amb pesos a les arestes. L'amplada del camí és el mínim entre els pesos de les arestes del camí. El *coll d'ampolla* (bottleneck) entre s i t és l'amplada del camí més ample entre s i t.

Suposem que G és connex.

1. Demostreu que qualsevol arbre d'expansió **màxim** d'G conté camins de màxima amplada entre qualsevol parell de vèrtexs s i t.
2. Descriviu un algorisme que, donats dos vèrtexs s i t i un pes W, determini en temps $O(|V|+|E|)$ si el coll d'ampolla entre s i t és $\leq W$ o no.

Una solució:

1) Podem demostrar-ho per contradicció. Suposem que T és un arbre d'expansió del graf G que és màxim però que el camí $p_T : s \rightsquigarrow t$ que uneix dos vèrtexs s i t a T no és de màxima amplada.

   Sigui $e \in E$ l'aresta de menys pes del cami p_T. Si esborrem e de T aleshores els nodes s i t deixen d'estar connectats a T (perquè T no té cicles i, doncs, el camí entre ells era únic). Anomenem T_{-e} a l'arbre T sense aquesta aresta.

   Si p_T no era el camí de màxima amplada entre s i t al graf G vol dir que n'hi ha altres camins que uneixen aquests dos vèrtexs a G amb amplada superior a la de p_T. Sigui p'_G el camí de més amplada que uneix s i t a G. Afegim a T_{-e} una aresta de p'_G que no tanqui cicle. Tenim la garantia de poder afegir una aresta sense provocar cicle perquè els dos nodes estan desconnectats a T_{-e}. Anomenem T' a aquest nou arbre.

   El pes de l'aresta esborrada era el mínim del camí p_T, però p_T no era el camí d'amplada màxima entre s i t; per tant, sabem que tota aresta del nou camí que s'ha format entre s i t té un pes més gran que l'aresta e esborrada. Això significa que el nou arbre d'expansió T' ha de tenir, per força, un pes total més gran que el T original. La qualcosa contradiu la definició de l'arbre d'expansió màxim T original.

2) El problema de cercar el camí amb més amplada entre dos vèrtexs donats d'un graf ponderat es coneix amb el nom de *problema del camí més ample* (en anglès, *Widest path problem*) o *problema de la ruta de capacitat màxima* (en anglès, *Maximum capacity path problem*). És possible adaptar la majoria dels algorismes coneguts per trobar els camins més curts (Dijkstra, Bellman-Ford, etc.) per calcular els camins més amples, modificant-los per tal que utilitzin la mesura de l'amplada en lloc de la suma de pesos (o la longitud) del camí.[1] Tanmateix, aquestes adaptacions requereixen més temps del demanat. També requereix més temps qualsevol solució que passi per construir

[1] També requeriran la conversió del graf no dirigit, a dirigit.

l'arbre d'expansió màxim. I encara requereixen més temps (exponencial!) les solucions que miren tots els camins possibles entre els dos vèrtexos.

Donats un graf $G = (V, E, w)$ no dirigit, connex i ponderat, dos vèrtexs $s, t \in V$ i un pes W, se'ns demana un algorisme per decidir si el coll d'ampolla entre s i t és $\leq W$ o no. Un algorisme senzill per fer-ho és el següent:

funció BOTTLENECK_AFITAT $(G = (V, E, w), s, t \in V, W \in \mathbb{Z})$
  $E' \leftarrow \{e \in E \mid w(e) \leq W\}$
  Sigui $G' = (V, E \setminus E')$ ▷ No considerar les arestes de G amb pes $\leq W$
  si $\exists$ camí $p : s \rightsquigarrow t$ a G' **aleshores**
    retornar NO
  altrament
    retornar SÍ

Considerant un implementació de G en llista d'adjacències, esborrar-ne les arestes amb pes inferior o igual a W té cost $O(|E|)$. Trobar si existeix un camí entre s i t es pot fer mitjançant un algorisme de recorregut sobre G, bé sigui DFS o BFS. Començant per un dels dos vèrtexs, només s'haurà de comprovar que s'arriba a l'altre. Això té cost $O(|V| + |E|)$ i, per tant, el cost total és el demanat. Observeu, però, que en realitat no cal esborrar de G les arestes d'E'; n'hi ha prou a no tenir-les en compte quan es fa el recorregut sobre G. Per tant, l'algorisme es redueix, en realitat, a un recorregut (DFS o BFS) modificat que obvia les arestes amb pes $\leq W$.

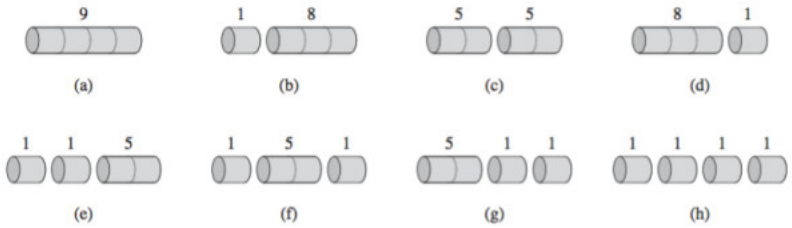
Al graf G' poden passar dues coses: que els vèrtexs s i t estiguin connectats o no. Si no hi ha camí entre ells a G', vol dir que tots els camins que els unien a G feien servir alguna de les arestes eliminades/no considerades i, per tant, el coll d'ampolla entre ells era menor o igual que W. En canvi, si s i t encara estan connectats a G' vol dir que entre ells hi ha camins d'amplada més gran de W i, per tant, el coll d'ampolla també ho serà.

Exercici 2 (3 punts)

Els recursos naturals són cada vegada més escassos i hem de reduir-ne i optimitzar-ne l'ús. A la indústria metal·lúrgica això és especialment important perquè treballen principalment amb minerals com a matèria primera. Per aquesta raó hi ha molts processos d'optimització que es desenvolupen i s'apliquen en aquest àmbit. A nivell econòmic, sovint això implica haver de treure el màxim profit econòmic de la quantitat de recursos disponible en un determinat moment.

Tenim una gran barra d'acer de longitud n, i volem tallar-la en trossos per a destinar-los a diferents usos. Cada tros de barra d'acer, en funció de la seva llargària, té un cost al mercat. Un tros de llargària i, amb $i \in \mathbb{Z}^+$ i $1 \leq i \leq n$, val p_i euros. Observeu que les llargàries dels trossos són unitats enteres.

La figura següent en mostra un exemple amb les 8 possibles formes de tallar una barra d'acer de llargària 4. Sobre cadascun dels trossos s'indica el preu que se n'obté ($\{p_1 = 1, p_2 = 5, p_3 = 8, p_4 = 9\}$). La solució òptima és l'opció (c) –tallar la barra en dos trossos de longitud 2– que dóna un guany de 10€.



Es demana:

a) De quantes formes diferents es pot tallar una barra de llargària n?
Raoneu la resposta.

b) Dissenyeu un algorisme, el més eficient que pugueu, per a decidir com tallar una barra d'acer de longitud n en trossos, de manera que es maximitzi el guany total que se n'obté. El vostre algorisme ha de dir quants talls s'han de fer en total, i a on.

Assumirem que fer un tall a la barra no indueix cap cost afegeix. Observeu que no es demana cap requisit sobre el nombre de talls a fer; podeu fer qualsevol nombre de talls entre 0 i $n-1$.

Una solució:

a) 2^{n-1}, perquè hi ha $n-1$ llocs on podem triar fer talls, i a cada lloc, o fem un tall o no el fem.

b) Un algorisme naïf per a resoldre aquest problema exploraria les $O(2^n)$ formes de tallar la barra (apartat (a)) i retornaria la que maximizés el guany obtingut. Naturalment, aquesta solució no és opció per a nosaltres. Podem resoldre'l de manera més eficient si apliquem programació dinàmica. Podem fer-ho perquè el problema presenta les dues condicions necessàries: subestructura òptima,[2] i problemes superposats .[3]

Representem una solució òptima a un subproblema mitjançant la següent recurrència: sigui $S(i)$ el preu màxim que podem obtenir en tallar un tros de la barra de llargada i que comprèn l'inici de la mateixa (és a dir, el tros que va des de l'inici fins als i metres). Segons aquesta definició, la solució al problema queda representada pel terme $S(n)$ de la recurrència. Tenint en compte la subestructura òptima, la resta de solucions òptimes parcials es calculen de la següent forma:

[2]Fixat un punt de tall a la barra, la solució òptima que s'hi pot aconseguir requereix resoldre de manera òptima els dos nous trossos que aquest punt de tall produeix a la barra. Altrament sempre podríem crear una solució millor i, per tant, la considerada no seria òptima.

[3]En el plantejament recursiu del problema ens trobem que s'ha de resoldre diverses vegades el mateix subproblema.

$$S(i) = \begin{cases} 0, & \text{si } i = 0 \\ \max\limits_{0 \le j < i} \{S(j) + p_{i-j}\}, & \text{altrament} \end{cases}$$

Observeu que, al cas recursiu, el que es fa es cercar el millor *punt de tall* j on fer l'últim tall a la barra (tros des de $j+1$ a i, que té llargària $i-j$).

Per implementar aquesta solució i poder guardar les solucions dels subproblemes necessitem una taula de mida n, on la posició i enmagatzemarà el valor de $S(i)$. En total s'han de resoldre n subproblemes i cadascun d'ells requereix temps $O(n)$. El cost en espai és, doncs, $O(n)$ i el cost temporal $O(n^2)$.

Se'ns demana també reconstruir la solució, és a dir, explicitar quants talls s'han de fer i a quines posicions per obtenir el benefici màxim que calcula la recurrència. Això ho podem fàcilment amb una estructura auxiliar d'espai $O(n)$ que guardi memòria, per al càlcul de cadascun dels $S(i)$, de quina ha estat la j (punt de tall) que ha produït el màxim. Possible implementació:

```
funció BottomUpTallaBarraComplet (p, n)
    S[0..n] ← {0, 0, .., 0}          ▷ Vector que guarda els valors de la recurrència
    T[0..n]                           ▷ Vector on guardarem els talls
    per a i = 1 to n fer
        q = −∞
        per a j = 1 to i fer
            si q < S[j] + p[i − j] aleshores
                q = S[j] + p[i − j])          ▷ Arrosseguem el màxim
                T[i] = j          ▷ En acabar el bucle, (T[i] = j)⟹(j = arg_max{S(j) + p_{i−j}})
        S[i] = q
    retornar S, T

funció PrintSolution (p, n)
    (S, T) ←BottomUpTallaBarraComplet(p, n)
    mentre n > 0 fer
        Print(T[n])
        n = n − T[n]
```

NOTA: De manera anàloga, es podria plantejar la solució simètrica, és a dir, que $S(i)$ fos el preu màxim que es pot obtenir en tallar un tros de la barra que va des de la posició i fins al final (tros de llargada $n-i$). En aquest cas, el punt de tall es buscaria entre i i n.

Una altra solució:

Podem també representar una solució òptima a un subproblema mitjançant la següent recurrència: sigui $S(i, j)$ el preu màxim que podem obtenir en tallar un tros de la barra comprès entre les posicions i i j, considerant que es talla *després* de la posició i. Aquesta representació és més genèrica i ens permet considerar com a subproblema qualsevol tros de la barra, des de qualsevol posició.[4] Com veurem, això no ens aporta cap benefici.

Segons aquesta definició, la solució al problema queda representada pel terme $S(0, n)$ de la recurrència. Tenint en compte la subestructura òptima, la resta de solucions òptimes parcials es calculen de la següent forma:

$$S(i,j) = \begin{cases} 0, & \text{si } i = j = 0 \\ p_j, & \text{si } j > i = 0 \\ \max\limits_{i \le k < j} \{S(i,k) + p_{j-k}\}, & \text{altrament} \end{cases}$$

Amb aquesta representació hem de resoldre un nombre de subproblemes d'ordre $O(n^2)$ i cadascun d'ells requereix temps $O(n)$. El cost total necessari és, doncs, $O(n^3)$. A més, en implementar-ho, necessitarem fer ús d'un espai també d'ordre $O(n^2)$. Aquesta solució és, doncs, pitjor que la primera proposada, tant en temps com en espai.

[4] En contraposició a la primera solució proposada, que sempre considerava un tros de la barra original però començant des del principi de la barra (els "prefixos" de la barra).

Exercici 3 (3 punts)

Tenim un conjunt $\mathcal{M} = \{m_1, \ldots, m_N\}$ de màquines i un conjunt d'operaris $\mathcal{W} = \{w_1, \ldots, w_T\}$. Cada màquina m_j pot funcionar un màxim d'λ_j unitats de temps al llarg d'un torn (consecutivament o espaides) i cada operari w_k pot treballar un màxim de ϵ_k unitats de temps al llarg d'un torn (també consecutivament o espaides, en una mateixa màquina o en màquines diferents). A més se'ns dona una matriu de booleans on $C_{jk} =$ **true** si i només si l'operari k sap operar la màquina j.

El nostre objectiu és assignar unitats de temps de les màquines als operaris de manera que es maximitzi el nombre total d'unitats de temps en les quals les màquines funcionen. Una solució és un conjunt de tripletes (m_j, w_k, t_{jk}) amb $m_j \in \mathcal{M}$, $w_k \in \mathcal{W}$ i $t_{jk} > 0$ de manera que: **1)** C_{jk} és cert; **2)** fixada una màquina m_j, la suma dels t_{jk} per a totes les tripletes amb primera component igual a m_j és $\leq \lambda_j$; **3)** fixat un operari w_k, la suma dels t_{jk} per a totes les tripletes amb segona component igual a w_k és $\leq \epsilon_k$.

a) Dissenyeu un algorisme eficient per trobar una solució que maximitzi el nombre total d'unitats de temps en les quals estan actives les màquines durant el torn. Justifica la correctesa i analitza el seu cost en funció d'N (nombre de màquines), de T (nombre d'operaris) i del nombre màxim d'unitats amb màquines funcionant $\Lambda = \lambda_1 + \cdots + \lambda_N$.

Un CEO de l'empresa s'ha adonat que no és rendible posar en marxa una màquina si la feina a fer fa que aquesta màquina s'utilitzi en un nombre petit d'unitats de temps. Ha calculat, per a cada màquina, un enter r_i indicant el nombre d'unitats de temps que garanteix el rendiment econòmic de l'ús de la màquina i.

b) Dissenyeu un algorisme eficient per determinar si hi ha una solució en el que el nombre d'unitats de temps en les quals estan actives les màquines durant el torn és l'obtingut a l'apartat a) garantint, a més, que cada màquina s'utilitza r_i o més unitats de temps. Justifiqueu-ne la correctesa i analitzeu-ne el cost en funció d'N (nombre de màquines), de T (nombre d'operaris) i del nombre màxim d'unitats amb màquines funcionant $\Lambda = \lambda_1 + \cdots + \lambda_N$.

Una solució:

a) Solucionaremos el problema obteniendo el flujo máximo en una red s-t $\mathcal{N} = \langle V, E, c\rangle$ construída con los datos del problema.

- Los vértices de la red son las máquinas, los operarios y adicionalemnte una fuente s y un sumidero t. Esto es, $V = \{s, t\} \cup \mathcal{M} \cup \mathcal{W}$, y $|V| = 2 + N + T$.
- Habrá una arista (s, m_j) por cada máquina m_j con capacidad λ_j, una arista (w_k, t) por cada operario w_k con capacidad ϵ_k y una arista (m_j, w_k) por cada máquina m_j y operario w_k tales que $C_{jk} =$ **true**; su capacidad diremos que es $+\infty$ (puede también ponerse $c(m_j, w_k) = \min\{\lambda_j, \epsilon_k\}$ por ejemplo). En total $|E| \leq N + T + N \times T$.

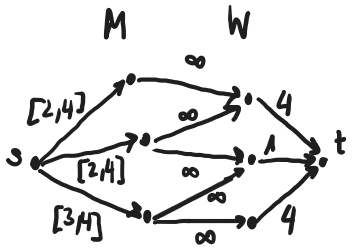
Un flujo válido f sobre $\mathcal{N}$ nos da una solución al problema de asignación máquinas-operarios $\{(m_j, w_k, t_{jk})\}$ con $t_{jk} = f(m_j, w_k)$ si (m_j, w_k) es una arista en $\mathcal{N}$; las condiciones de capacidad y conservación del flujo garantizan que ninguna máquina m_j está activa más de λ_j unidades en el turno y que ningún operario trabaja más de ϵ_k unidades en el turno. Puesto que queremos maximizar el número total de unidades de tiempo con máquinas activas lo que estamos es buscando el flujo máximo en $\mathcal{N}$.

Podemos usar el algoritmo de Ford-Fulkerson con coste $O((|V| + |E|) \cdot C)$ donde C es una cota superior del valor del flujo máximo. El valor Λ es una tal cota superior (es la capacidad del corte $\langle s, V \setminus \{s\}\rangle$). Por lo tanto, en términos de N, T y Λ el coste del algoritmo será $O(NT\Lambda)$.

b) Para esta nueva variante simplemente agregaremos cotas inferiores de flujo a las aristas (s, m_j). Lo demás (vértices, aristas, capacidades máximas) no cambia. Tendremos $\ell(s, m_j) = r_j$. En las restantes aristas e de la red pondremos $\ell(e) = 0$. Pero tendremos que ser un poco más ingeniosos para resolver el problema, no podemos aplicar ni Ford-Fulkerson ni Edmonds-Karp directamente sobre el nuevo problema.

Podemos añadir una arista (t, s) con capacidad máxima $+\infty$ y el valor del flujo máximo obtenido en el apartado (a) como cota mínima, y poner demanda $d(v) = 0$ para todo vértice, para convertir el problema en uno de circulación con demandas y cotas inferiores. El problema de circulación con demandas y cotas inferiores puede resolverse en tiempo $O((\Lambda + R)NT)$, siendo $R = \sum_j r_j$, mediante la transformación del problema en uno de máximo flujo utilizando las reducciones apropiadas (circulación con demandas y cotas inferiores $\rightarrow$ circulación con demandas $\rightarrow$ flujo máximo).

Es importante observar que el flujo máximo obtenido en el apartado (a) anterior no necesariamente satisface la condición de que para cada máquina m_j se le asignan r_j unidades de tiempo al menos, aunque exista un flujo así; es decir, el flujo $f^*(s, m_j)$ retornado por FF o EK no es necesariamente $\geq r_j$ para todo j aunque exista un flujo máximo que sí lo cumple. En general pueden existir flujos de valor $\leq |f^*|$ que sí cumplan con las restricciones originales y la nuevas restricciones de uso mínimo de las máquinas. O no existir.



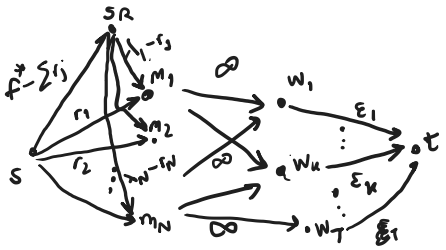
Por ejemplo en la red de la figura arriba un flujo máximo sin restricciones de flujo mínimo sería $(m_1, w_1, 4)$, $(m_2, w_2, 1)$ y $(m_3, w_3, 4)$ y su valor total sería 9. Pero no cumple la condición de que m_2 sea usada al menos $r_2 = 2$ unidades de tiempo. El flujo $(m_1, w_1, 2)$, $(m_2, w_1, 2)$ y $(m_3, w_3, 4)$ sí cumple todas las restricciones y su valor es 8, no es máximo, pero sí factible. El flujo $(m_1, w_1, 2)$, $(m_1, w_2, 1)$, $(m_2, w_1, 2)$, $(m_3, w_3, 4)$ es máximo, ya que tiene valor 9 y además cumple todas las restricciones, incluyendo las de mínimo uso de las máquinas. Pero no necesariamente es la solución que nos devuelve el algoritmo de maxflow del apartado (a).

Tampoco basta poner capacidad r_j en las aristas (s, m_j), aplicar maxflow y comprobar que el flujo máximo satura todas esas aristas. Eso sería suficiente para garantizar la existencia de un flujo que respeta todas las cotas mínimas de uso de las máquinas, pero no garantiza que existe un flujo que cumple las restricciones y cuyo valor es igual al del caso sin restricciones.

Otra solución alternativa, pasa por resolver el apartado previo y obtener el valor del flujo máximo f^*. Entonces construimos una nueva red s-t $\mathcal{N}'$, cuyos vértices son $M \cup W \cup \{s_R\} \cup \{s, t\}$. Pondremos una arista (s, s_R) con flujo $f^* - \sum_{1 \leq j \leq N} r_j$. Aristas (s, m_j) con capacidad r_j cada una, para toda máquina m_j, y aristas (s_R, m_j) con capacidad $\lambda_j - r_j$. Para toda máquina m_j y operario w_k, aristas (m_j, w_k) si $C_{jk} = \textbf{true}$, con capacidad $+\infty$. Para cada operario aristas (w_k, t) con capacidad ϵ_k.

Sea f' un flujo máximo en la nueva red. El flujo que sale de m_j será la suma del que llega por (s, m_j) más el que llega por (s_R, m_j). El primero es $\leq r_j$, el segundo es $\leq \lambda_j - r_j$. En definitiva la máquima m_j será usada a lo sumo λ_j unidades y siempre se cumplirá que $|f'| \leq |f^*|$. Supongamos que existe un flujo en $\mathcal{N}'$ de valor igual al de f^*. Entonces al aplicar maxflow sobre $\mathcal{N}'$ lo encontraremos y necesariamente habremos de saturar las aristas (s, m_j) de capacidad r_j,

enviado el resto del flujo a través de la arista (s, s_R) de capacidad $|f^*| - \sum_j r_j$ y saturándola también. Es decir, el valor del flujo saliente de s es $|f^*|$. Si no existe un flujo que satisface las cotas mínimas o cuyo valor es $= |f^*|$ alguna de las aristas que sale de s **no** se podrá saturar.



Exercici 4 (0.5 punts x 4 = 2 punts)

1. Es poden ordenar en temps $O(n \lg \lg n)$, n enters diferents amb valors entre 1 i $n \lg n$?

   Una solució: Cierto. Si tomamos base $b = n^c$ con $0 < c < 1$, p.e., $b = \sqrt{n}$, el coste de LSD radix sort es $\Theta((n+b)\log_b f(n))$ siendo $f(n) = n \lg n$. Con $b = n^c$ tenemos

   $$\log_b f(n) = \frac{\lg(n \lg n)}{c \lg n} = \frac{1}{c} + O\left(\frac{\log \log n}{\log n}\right),$$

   y el coste es $\Theta((n+b)\log_b f(n)) = \Theta(n) = o(n \lg \lg n)$. Puede tomarse una base $b = o(n^c)$ para toda $c > 0$ y conseguir que el coste de LSD sea $O(n \log \log n)$, pero tiene que ser $b = \omega((\log n)^k)$ para toda $k > 0$; tomar $b = n^c$ con $c \in (0,1)$ es la opción más sencilla para demostrar que la afirmación es correcta.

2. Considereu un algorisme que té com a entrada $3n$ enters diferents donats en tres vectors no ordenats de n elements i que retorna dos enters x i y tals que n dels valors donats són $\leq x$, n d'ells tenen valor entre x i y, i els altres n són $\geq y$. És cert que aquest algorisme ha de tenir complexitat $\Omega(n \lg n)$?

   Una solució: Falso. Si ponemos los $3n$ enteros en un vector A (coste $\Theta(n)$) y utilizamos el algoritmo de selección de coste lineal $\Theta(n)$ para hallar los elementos $(n+1)$-ésimo y el $2n$-ésimo de A tendremos los elementos x e y pedidos y se obtienen con coste $o(n \log n)$.

3. Es pot trobar en temps lineal el camí més llarg d'un graf dirigit acíclic $G = (V, E)$?

   Una solució: Cierto. Un recorrido en orden topológico del DAG nos permite dividir el DAG en "capas". Las raíces (vértices sin predecesores) del DAG constituyen la capa 0. Cuando un vértice v de la capa i es visitado en el orden topológico sus sucesores se asignan a la capa $i+1$ (un vértice w podía estar asignado a la capa j por ser sucesor de un vértice de la capa $j-1$ pero si también es sucesor de un vértice de la capa $i > j-1$ se actualizará a w como vértice de la capa $i+1 > j$). Un camino de longitud j máxima en G nos lleva de la capa 0 a una cierta "hoja" de la capa j máxima. El coste del algoritmo es lineal respecto al tamaño del DAG.

4. Considereu un digraf amb pesos a les arestes $G = (V, E)$; $w : E \to Z$. Siguin $s, t \in V$, i sigui C el camí $s \rightsquigarrow t$ més curt a (G, w). Sense modificar la topologia de G, definim nous pesos sobre les arestes w', de manera que $\forall e \in E$: $w'(e) = 2w(e)$. Continuarà sent C el camí més curt a (G, w')?

   Una solució: Todo camino π en (G, w) es un camino en (G, w') y viceversa. Para cualquier camino π entre s y t en G se cumple $w'(\pi) = 2w(\pi)$. En consecuencia, si π de s a t es mínimo con los pesos w entonces también es mínimo para los pesos w' y viceversa.