Tema 1. Definiciones y conceptos básicos

1.1 Grafos. Isomorfismo de grafos

Un *grafo no dirigido* es un triple ordenado (V(G), E(G), Ψ_G), donde:

- V(G) finito y no vacío: conjunto de *vértices*
- E(G) finito: conjunto de *aristas* ($V(G)\cap E(G)\neq\varnothing$)
- Ψ_G : *función de incidencia*, asigna a cada arista $e\in E(G)$ un par no ordenado de vértices (u,v), u y v son los *extremos* de la arista e.

Un *grafo dirigido* es un triple ordenado (V(G), E(G), Ψ_G), donde:

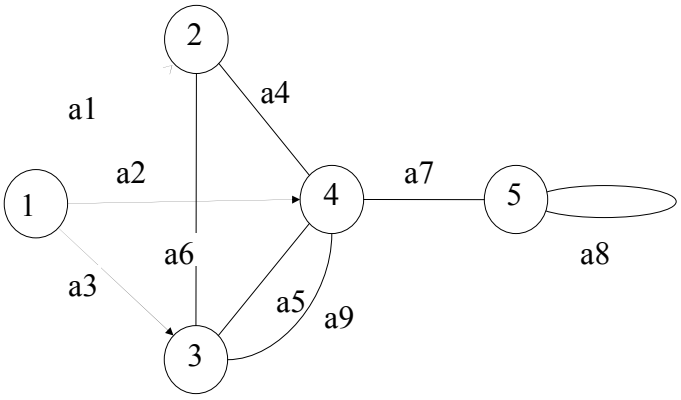
- V(G) finito y no vacío: conjunto de *vértices*
- E(G) finito: conjunto de *arcos* ($V(G)\cap E(G)\neq\varnothing$)
- Ψ_G : *función de incidencia*, asigna a cada arco $a\in E(G)$ un par ordenado de vértices (u,v); el arco a está dirigido de u a v (el arco es *saliente* de u y *entrante* en v).

Normalmente se denota un grafo simplemente como G = (V,E), siendo:

- V: conjunto de vértices
- E: conjunto de aristas o arcos

(Se entiende que cada arista (arco) es un par (par ordenado) de vértices)

Representación gráfica (grafo mixto: con arcos y aristas)



La arista a4 es *incidente* con los vértices 2 y 4

Dos vértices incidentes con una misma arista se dicen *adyacentes* (ejemplos: 2 y 4, 4 y 5)

Bucle: arista cuyos dos extremos son el mismo vértice (a8)

Aristas paralelas: tienen el mismo par de vértices extremos (a5, a9)

Arcos paralelos: tienen el mismo par ordenado de vértices.

Grafo simple: un grafo sin bucles ni aristas (arcos) paralelos. En este caso una arista (arco) se puede representar por el par (par ordenado) de vértices que une.

Dos grafos G y H son *isomorfos* si existen dos biyecciones F_V y F_E entre los conjuntos de vértices y aristas, respectivamente, de G y H, tales que: $\Psi_G(e) = (u,v) \Leftrightarrow \Psi_H(F_E(e)) = (F_V(u), F_V(v))$.

Si los grafos son simples, basta con especificar la biyección entre los conjuntos de vértices y que se cumpla: $(u,v) \in E(G) \Leftrightarrow (F_V(u), F_V(v)) \in E(H)$

Notación:

- n, n(G): número de vértices
- m, m(G): número de aristas/arcos

Algunos grafos especiales:

Grafo completo con n vértices (notación: K_n): cada par de vértices está unido por una arista.

Grafo vacío: sin aristas, *grafo trivial*: grafo vacío con un sólo vértice.

Grafo bipartido: existe una bipartición de V = X∪Y; X, Y no vacíos, tal que toda arista del grafo tiene un extremo en X y el otro en Y.

Grafo bipartido completo: es bipartido y todo vértice de X es adyacente a todo vértice de Y. Si |X|=p y |Y|=q, se denota $K_{p,q}$.

Grafo k-regular: cada vértice es incidente con k aristas exactamente.

1.2 Subgrafos

H es un *subgrafo* de G (se denota H⊆G) sii:

- V(H)⊆V(G)
- E(H)⊆E(G)
- $\Psi_H = \Psi_G|_{E(H)}$

(También se dice que G es un *supergrafo* de H)

Además, se llama:

- *Subgrafo generador*: si V(H)=V(G).
- *Subgrafo de G inducido por V'⊆V(G):* (se denota G[V']) si tiene V' como conjunto de vértices y todas las aristas de G con ambos extremos en V'.
- *Subgrafo de G inducido por E'⊆E(G)* (se denota G[E']) si tiene E' como conjunto de aristas y todos los vértices de G que son extremo de alguna arista de E'.

Notaciones:

- G-V' = G[V-V']
- G+E', (G-E'): se añaden a G (o se eliminan) las aristas de E'.

- dos grafos son *disjuntos* si no tienen vértices (y por lo tanto aristas) en común.
- dos grafos son *aristodisjuntos* si no tienen aristas en común.

1.3 Grado de un vértice

Grafo no dirigido:

grado del vértice v : d(v)= número de aristas incidentes con v (cada bucle en v cuenta dos veces)

Δ denota el *grado máximo* de un grafo y δ el *grado mínimo*.

$\Gamma(v)$ = conjunto de vértices adyacentes a v.

Grafo dirigido:

grado de entrada del vértice v: $d^-(v)$: número de arcos que entran en v.

grado de salida del vértice v: $d^+(v)$: número de arcos que salen de v.

$\Gamma(v) = \{u \in V: \text{existe el arco } (v,u)\}$

$\Gamma^{-1}(v) = \{u \in V: \text{existe el arco } (u,v)\}$

Teorema 1.1

a) En un grafo no dirigido, se cumple: $\sum_{v \in V} d(v) = 2m$.

b) En un grafo dirigido: $\sum_{v \in V} d^-(v) = \sum_{v \in V} d^+(v) = m$ *(número de arcos).*

Teorema 1.2

En cualquier grafo, el número de vértices de grado impar es par.

1.4 Caminos y conexión

Una *cadena* en G es una sucesión finita $W = v_0, e_1, v_1, e_2,..., e_k, v_k$, cuyos términos son alternativamente vértices y aristas (arcos), tal que, para $i=1,...,k$, $e_i=(v_{i-1},v_i)$:

- v_0 vértice inicial, v_k vértice final, el resto son vértices internos
- se puede denotar simplemente por la sucesión de aristas (arcos), o por la de los vértices si el grafo es simple.
- longitud de W = k (número de aristas (arcos))

Además, se llama:

- *Cadena simple*: si las aristas (arcos) $e_1,..., e_k$ son distintas.
- *Camino*: si los vértices $v_0,..., v_k$ son distintos.

Una cadena con vértice inicial u y vértice final v, se llama (u,v)-cadena. Similarmente para los caminos.

Dos vértices u y v están *conectados* en G sii existe un (u,v)-camino y un (v,u)-camino en G. Es una relación de equivalencia entre los vértices. Sean $V_1,..., V_\omega$, las clases de equivalencia.

- $G[V_i]$, i=1,..., ω, son las *componentes conexas* de G.
- ω(G) denota el número de componentes de G.
- Si ω(G)=1, se dice que G es *conexo*.

En los grafos dirigidos se distinguen dos tipos de conexión. Si G es dirigido y ω(G)=1, se dice que G es *fuertemente conexo.*

Dado un grafo dirigido G llamamos *grafo no dirigido asociado* al grafo, sea G', que resulta de sustituir cada arco por una arista con los mismos extremos que el arco sustituido. Dada una cadena en G', si sustituimos las aristas por los arcos originales, tenemos lo que se llama una *cadena débil* en G; esto es, una "cadena" salvo que la dirección de los arcos pueden no ir en la dirección de la cadena.

Análogamente se habla de: camino débil, conexión débil, componentes débilmente conexas, etc.

1.5 Ciclos y circuitos

Cadena cerrada: es una cadena con longitud positiva, en la que los vértices inicial y final coinciden.

Ciclo: es una cadena en la que los vértices internos son distintos y los vértices inicial y final coinciden.

Un *ciclo* es *par* o *impar*, según lo sea su longitud (número de aristas (arcos)).

Teorema 1.3

Un grafo no dirigido es bipartido sii no contiene ningún ciclo impar.

1.6 Representaciones matriciales

Suponemos que $V = \{v_1, ..., v_n\}$.

Matriz de adyacencia $A = [a_{ij}]_{i=1,...,n,\ j=1,...,n}$, a_{ij} = número de arcos/aristas (v_i,v_j) en G (puede ser 0)

Si el grafo es no dirigido, es simétrica.

Se cumple:

- grafo dirigido: $\sum_{j=1}^{n} a_{ij} = d^+(v_i)$, y $\sum_{j=1}^{n} a_{ji} = d^-(v_i)$
- grafo no dirigido $\sum_{j=1}^{n} a_{ij} = \sum_{j=1}^{n} a_{ji} = d(v_i)$, si no hay bucles

Si denotamos $A^2 = [a_{ij}^{(2)}]$, se cumple que $a_{ij}^{(2)}$ = número de cadenas de longitud 2 que unen v_i con v_j en G.

Supongamos también que las aristas (arcos) de G están, numerados $e_1,..., e_m$, y que no existen bucles.

Matriz de incidencia $B = [b_{ij}]_{i=1,...,n,\ j=1,...,m}$, donde:

- grafo dirigido: $b_{ij} = \begin{cases} 1, \text{si } e_j \text{ es un arco saliente de } v_i \\ -1, \text{si } e_j \text{ es un arco entrante en } v_i \\ 0, \text{en otro caso} \end{cases}$
- grafo no dirigido: $b_{ij} = \begin{cases} 1, \text{si } e_j \text{ es incidente con } v_i \\ 0, \text{en otro caso} \end{cases}$

1.7 Ejercicios

(Si no se indica lo contrario, los grafos que se mencionan en los ejercicios son no dirigidos)

1. Mostrar que si el grafo G es simple, entonces $m \leq \binom{n}{2}$ y que G es completo sii $m = \binom{n}{2}$.

   Nota: n ó n(G) denota el número de vértices del grafo, mientras que m ó m(G) denota el número de aristas o arcos.

2. Mostrar que hay 11 grafos simples no isomorfos sobre 4 vértices.
3. Demostrar que: (a) $m(K_{p,q}) = pq$. (b) si G es simple y bipartido: $m \leq n^2/4$.
4. Mostrar que: $\delta \leq 2m/n \leq \Delta$.
5. Sea S un conjunto de n puntos del plano tales que la distancia entre cualquier par de puntos es al menos 1. Demostrar que hay como máximo 3n pares de puntos a distancia exactamente 1. *Sugerencia*: Definir un grafo en el que dos puntos son adyacentes si están a distancia 1 y mostrar que el grado máximo en este grafo es 6.
6. Mostrar que en cualquier grupo de 2 ó más personas, hay siempre al menos dos con exactamente el mismo número de amigos dentro del grupo.
7. Si G tiene los vértices $v_1,v_2,...v_n$, a la secuencia $(d(v_1), d(v_2), ...,d(v_n))$ se le llama secuencia de grados de G. Mostrar que una secuencia de enteros no negativos $(d_1, d_2, ..., d_n)$ es la secuencia de grados de un grafo (simple o no) sii $\sum d_i$= par.

   Comprueba que las secuencias (7,6,5,4,3,3,2) y (6,6,5,4,3,3,1) no son secuencias de grados de un grafo simple.
8. Demostrar que si la secuencia de enteros no negativos $(d_1, d_2, ..., d_n)$, con $d_1 \geq d_2 \geq ... \geq d_n$ es la secuencia de grados de un grafo simple, entonces:

   $$\sum_{i=1}^{k} d_i \leq k(k-1) + \sum_{i=k+1}^{n} \min(k,d_i), \qquad \forall k,\ 1 \leq k \leq n.$$

   (Se puede demostrar que esta condición es también suficiente)
9. Sea G un grafo con 4 vértices y secuencia de grados (4,3,2,1). Dibujar todos los grafos sin bucles y no isomorfos, con esa secuencia de grados.
10. Probar que no existe un grafo con 7 vértices que sea 3-regular.
11. ¿Qué puedes decir de la suma de los números en cualquier fila o columna de una matriz de adyacencia?
12. Mostrar que si G es simple, los elementos de las diagonales de BB^t y de A^2 son los grados de los vértices de G (B^t es la traspuesta de B) .
13. Demostrar que si un grafo bipartido k-regular con k>0 tiene bipartición (X,Y), entonces |X|= |Y|.

14. Mostrar que si $\delta \geq 2$, entonces G contiene un ciclo.
15. Demostrar que si G es simple y $\delta \geq 2$, entonces G contiene un ciclo de longitud al menos $\delta+1$. AYUDA: Considerar un camino de longitud máxima y los vértices adyacentes al origen de dicho camino.
16. Caracterizar la matriz de adyacencia A de un grafo bipartido.
17. Probar que un grafo y su complementario no pueden ser desconexos los dos (El complementario de un grafo G=(V,E) es un grafo G^C, con el mismo conjunto de vértices que G, y se cumple que, para todo $u,v \in V$, u y v son adyacentes en G^C sii u y v no son adyacentes en G).
18. Demostrar que para toda terna de vértices u,v,w se cumple: $d(u,v) + d(v,w) \geq d(u,w)$. (Si $u,v \in V$, d(u,v) es el número de aristas del (u,v)-camino con menos aristas)
19. Demostrar que si G es simple y $\delta \geq k$, entonces G tiene un camino de longitud k.
20. Demostrar que el número de (v_i,v_j) cadenas de longitud k en G es la componente (i,j) de A^k.
21. Dada una colección de intervalos de la recta real $I = \{I_1,I_2,...,I_n\}$. El grafo intervalo para *I* es el grafo de n vértices $v_1,v_2,...,v_n$ tal que v_i es adyacente con v_j sii $I_i \cap I_j \neq \varnothing$
    a. Dibujar el grafo intervalo para los intervalos abiertos (0,2), (3,8),(1,4), (3,4), (2,5), (7,9).
    b. Demostrar que K_n es siempre un grafo intervalo para todo $n \geq 1$.
22. Dado un grupo de nueve personas, ¿es posible que cada persona le estreche la mano a exactamente tres personas distintas?
23. Demostrar que si existe una (u,v)-cadena en G, entonces también existe un (u,v)-camino en G.
24. Demostrar que G es conexo sii para toda partición de V en dos subconjuntos no vacíos V_1 y V_2, existe al menos una arista con un extremo en V_1 y el otro en V_2.
25. Demostrar que si G es simple y $\delta(G) \geq (n-1)/2$, entonces G es conexo.
26. Demostrar que si G es simple y $m > (n-1)(n-2)/2$, entonces G es conexo.
27. Demostrar que si G es simple y $\delta > \lfloor n/2 \rfloor - 1$, entonces G es conexo. Hallar un grafo simple desconexo ($\lfloor n/2 \rfloor - 1$)-regular para un n par.
28. Demostrar que si $a \in E$, entonces $\omega(G) \leq \omega(G - a) \leq \omega(G) + 1$. Ver que esto no se cumple si a es un vértice en lugar de una arista.
29. Demostrar que en un grafo conexo, dos caminos más largos siempre tienen un vértice en común.
30. Demostrar que si G es simple y conexo, pero no completo, entonces tiene tres vértices u, v y w tales que $(u,v), (v,w) \in E$, pero $(u,w) \notin E$.
31. Demostrar que si $m \geq n$, G contiene un ciclo.

Tema 2 Árboles. Complejidad algorítmica

Todo el contenido de este tema se refiere a grafos no dirigidos

2.1 Árboles y conectividad

Sea G=(V, E) un grafo no dirigido. Dado un subconjunto de vértices S⊂V, no vacío y distinto de V, la *cortadura de aristas* asociada a S es el conjunto de aristas:

$$\left[S,\overline{S}\right]=\left\{(x,y)\in E : x\in S, y\in \overline{S}; \text{o } \ y\in S, x\in \overline{S}\ \right\}$$

Obviamente, G - $\left[S,\overline{S}\right]$ es desconexo.

Se dice que e es una *arista de corte* si existe S⊂V tal que $\left[S,\overline{S}\right]=\{e\}$. Es fácil ver que e es una arista de corte sii ω(G-e) > ω(G).

Teorema 2.1

e es una arista de corte de G sii e no está contenida en ningún ciclo de G.

2.2 Árboles

Un grafo acíclico es aquel que no contiene ciclos. Un *árbol* es un grafo conexo y acíclico.

Teorema 2.2

En un árbol, cualquier par de vértices está unido por un único camino.

Teorema 2.3

Si G es un árbol, entonces m = n-1.

Teorema 2.4

Un grafo conexo es un árbol sii toda arista del grafo es de corte.

Dado un grafo G, un *árbol generador* de G es un subgrafo generador de G que es árbol.

Teorema 2.5

Si G es conexo, entonces tiene un árbol generador.

Teorema 2.6

Si G es conexo, entonces m ≥ n-1.

Teorema 2.7

Sea T un árbol generador de un grafo conexo y sea e una arista de G que no está en T, entonces T+e contiene un único ciclo.

Teorema 2.8

Un árbol generador de G (conexo) es un subgrafo generador acíclico y maximal de G, y viceversa.

Teorema 2.9

Las siguientes proposiciones son equivalentes:

a) *G es un árbol.*
b) *G es conexo y m = n-1.*
c) *G es acíclico y m = n-1.*

Un vértice de un grafo G se dice que es *vértice de corte* si $\omega(G-v) > \omega(G)$, o v tiene un bucle.

Teorema 2.10

Un vértice v, de un árbol G, es de corte sii d(v)>1.

Teorema 2.11

a) *Todo árbol no trivial contiene al menos dos vértices de grado 1.*
b) *Todo grafo conexo y no trivial contiene al menos dos vértices que no son de corte.*

Teorema 2.12

Sea T un árbol generador de G:

a) *Sea e una arista de G, $e \notin T$, y sea f una arista del único ciclo de T+e, entonces T+e-f es un árbol generador de G.*
b) *Sea e una arista de T, y sea $f \neq e$ una arista de G con un extremo en cada componente de T-e, entonces T-e+f es una árbol generador de G.*

2.3 Algunos conceptos de complejidad algorítmica

Los conceptos y definiciones de este apartado no son completamente rigurosos. Su objetivo es el de dar a conocer los conceptos y resultados más importantes de complejidad algorítmica y de la teoría de la NP-completitud a unos niveles intuitivos y aplicados al campo de la Investigación Operativa.

Sea $\mathcal{P}$ un *problema de optimización* que consiste en encontrar una solución óptima, según cierto criterio, entre un conjunto de soluciones posibles. Normalmente $\mathcal{P}$ está definido en función de ciertos parámetros. Un *ejemplo* de $\mathcal{P}$ está especificado por unos valores concretos de esos parámetros.

Ejemplo.

P: *calcular el máximo común divisor de n números enteros*. Ejemplo de $\mathcal{P}$: {3456, 987, 57}.

Se dice que un algoritmo *resuelve* el problema $\mathcal{P}$ cuando su aplicación a cualquier ejemplo de $\mathcal{P}$ proporciona la solución en un número finito de pasos.

Se define el *tamaño de un ejemplo* de un problema como el número de dígitos que se usan para especificar los valores de los parámetros que definen el ejemplo. Suponemos que se utiliza un sistema de representación "razonable". En el ejemplo anterior el tamaño sería 13, incluyendo los signos que separan los números y dos delimitadores que indican el principio y final de los datos (representando los números en sistema decimal).

La eficiencia de un algoritmo para resolver un determinado problema se mide por el número de operaciones que necesita para resolver un ejemplo en función de su tamaño.

Notación $O(\cdot)$: se dice que una función f(t), definida sobre los números naturales, es $O(g(t))$ si existe un número real c y un número natural t_0, tal que $f(t) \leq c \cdot g(t)$ para todo $t \geq t_0$.

Se dice que un *algoritmo* para el problema $\mathcal{P}$ es *polinómico* si el número máximo de operaciones que realiza para resolver cualquier ejemplo de tamaño t del problema, está acotado superiormente por un polinomio en t. Si el polinomio es de grado p, esto es equivalente a decir que el número de operaciones es $O(t^p)$. En el caso de que el número de operaciones no pueda ser acotado por un polinomio, se dice que el algoritmo es *exponencial*. Dado que las funciones exponenciales crecen muy rápidamente con la variable independiente, un algoritmo exponencial suele ser eficiente sólo para ejemplos de tamaño pequeño. Sin embargo, un polinomio tiene un crecimiento mucho más lento en general, sobre todo si el grado del polinomio es bajo, y por lo tanto puede ser eficiente para tamaños mucho más grandes.

Para representar un problema de grafos hace falta un número de dígitos que depende del número de vértices y el de aristas o arcos, así como de los posibles costes, capacidades, etc. asociados a los vértices y/o aristas/arcos. Notar que para representar un número u en el sistema decimal hacen falta $\lfloor \log u \rfloor + 1$ dígitos (esto es, del orden de log u). Así pues, el tamaño de un ejemplo de un problema de grafos depende normalmente de n, m y log U (siendo U el mayor entero en valor absoluto que aparece en los datos). Por lo tanto, se puede demostrar que *un algoritmo para un problema de grafos es polinómico* si necesita un número de operaciones que está acotado superiormente por un polinomio en n, m y/o log U.

Un problema se dice que está en la *clase* P si existe un algoritmo polinómico para resolverlo. Los problemas de la clase P son problemas que pueden ser resueltos muy eficientemente.

La clase NP es más difícil de definir, nos limitaremos a definirla para los problemas de optimización. Se puede decir que un problema de optimización está en la clase NP, si el coste de una solución posible de un ejemplo del problema se puede calcular con un algoritmo polinómico. Naturalmente es mucho más fácil calcular el coste de una solución que encontrar la solución óptima; como consecuencia, casi todos

los problemas de optimización están en la clase *NP*. Muchos de los problemas de la clase *NP* son considerados muy difíciles y no se conocen algoritmos polinómicos para resolverlos.

Conjetura: ¿*P*= *NP*?

Un problema $\mathcal{P}$ se dice que es *NP-difícil* si tiene la siguiente propiedad:

> "si existiera un algoritmo polinómico para $\mathcal{P}$, existirían algoritmos polinómicos para todos los problemas de la clase *NP* "

(En algunos problemas se utiliza también el término *NP-completo*)

Por lo tanto, si se encontrara un algoritmo polinómico para un problema *NP*-difícil, la consecuencia sería que *P* = *NP*. Existen muchos problemas de todo tipo que son *NP* -difíciles; y hay muy pocas esperanzas de que se consiga encontrar un algoritmo polinómico que resuelva cualquiera de ellos.

2.4 Árbol generador de coste mínimo (SST)

Sea G=(V, E) un grafo conexo no dirigido con n vértices y m aristas. Sea $c_{ij} = c(i,j)$ el *coste de la arista* $(i,j)\in E$.

El coste de un árbol generador de G se define como la suma de las aristas que contiene. El problema de encontrar el *árbol generador* de G con *coste mínimo* (abreviadamente SST, de *Shortest Spanning Tree*) tiene aplicaciones en gran cantidad de situaciones en las que se busca conectar una serie de elementos con coste mínimo.

Por simplicidad, si T es un árbol generador de G, su conjunto de aristas se denotará también por T.

Algoritmo de Kruskal para el SST

Inicio. Sea T el subgrafo generador vacío de G y sea E':=E.
Mientras |T| < n-1
    Sea e la arista de menor coste de E'; hacer E':=E'-e.
    Si T+e no contiene un ciclo, hacer T:=T+e.
FinMientras

La demostración de que el algoritmo es válido se hace por inducción sobre |T|. Si |T| = k < n-1, T es unión de subárboles. Dados dos subárboles T_s y T_r de T, definimos:

$$d(T_s, T_r) = \min \{c_{ij} : i\in T_s, j\in T_r, (i,j)\in E\}$$

Teorema 2.13

Sea T, con |T| < n-1, un subgrafo generador de G contenido en un SST y sea T_s uno de sus subárboles. Sea (v_s, v_{j*}) la arista que cumple:

$$c(v_s, v_{j*}) = min \{ d(T_s, T_r): T_r \text{ subárbol de T distinto de } T_s\}$$

entonces T + (v_s, v_{j*}) está contenido en un SST.

Inicialmente, T es vacío, por lo que se cumple la hipótesis del **Teorema 2.13**. En cada iteración del algoritmo de Kruskal, se añade la arista de menor coste que no forma ciclo y, por lo tanto, esta arista es la de menor coste entre dos subárboles cualesquiera.

Complejidad del algoritmo de Kruskal.

Para averiguar de forma eficiente si la adición de una arista cierra un ciclo, se utiliza una etiqueta para cada vértice:

l(i) = s si el vértice i pertenece al subárbol s

Es evidente, entonces, que la arista (i,j) cerraría un ciclo al ser añadida a T sii l(i) = l(j).

La asignación de etiquetas se hace de la siguiente forma. Inicialmente (cuando T = ∅), se hace l(i) := i, para todo i∈V. Si se añade una arista (i,j) a T, supongamos que l(i) < l(j), entonces basta con:

Hacer l(v) := l(i), para todo v tal que l(v)=l(j).

Para aplicar el algoritmo, podemos ordenar de menor a mayor coste todas las aristas, con un número de operaciones del orden de $m \cdot \log_2 m < m \cdot \log_2 n^2 = 2\, m \cdot \log_2 n < 2mn$ (las desigualdades se deben cumplir para números "grandes"). Para saber si cierta arista cierra un ciclo, basta comparar la etiqueta de los vértices (una operación), y se prueba, como máximo con las m aristas. Además, para actualizar las etiquetas cada vez que se añade efectivamente la arista, hay que cambiar, como máximo, la etiqueta de n vértices; como se añaden n-1 aristas, en total, por este concepto tenemos del orden de $n \cdot (n-1)$ operaciones. En resumen, el algoritmo es $O(mn)$.

Teorema 2.14

El SST es el árbol generador para el que el coste de la arista de mayor coste que contiene es mínimo.

Otros problemas relacionados

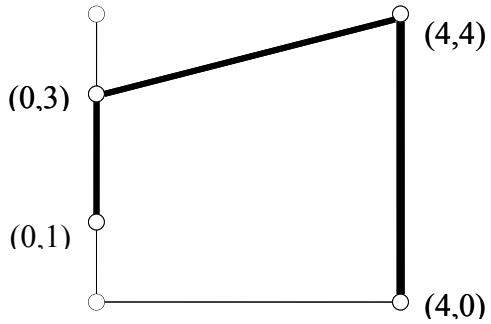
La demostración del **Teorema 2.13** se basa en el hecho de que el coste de un árbol disminuye si se sustituye una arista por otra de coste menor. Por lo tanto, el algoritmo de Kruskal es válido siempre que el coste del árbol sea una función simétrica en todos sus componentes (depende del conjunto de aristas, no del orden en que aparecen) y creciente.

Por ejemplo, suponer que los vértices del grafo representas nudos de comunicación y la existencia de una arista (i,j) indica que se puede transmitir un mensaje de i a j con una probabilidad p_{ij} de que sea interceptado por una persona no autorizada. Se trata de encontrar un grafo conexo en el que se pueda transmitir un mensaje a todos los nudos de la red con una probabilidad mínima de que sea interceptado, esto es, un árbol generador T que minimice $1-\prod_{(i,j)\in T}(1-p_{ij})$. Dado que esta función de coste cumple los requisitos mencionados, podemos utilizar el algoritmo de Kruskal para construirlo.

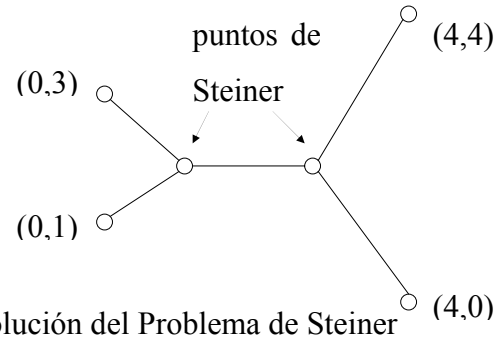
El Problema de Steiner

Si se trata de conectar n ciudades por una red de carreteras de longitud total mínima, la solución es el SST siempre que no se permitan enlaces entre los segmentos de carreteras que estén fuera de las ciudades. En la práctica, estos enlaces son corrientes y pueden permitir construir una red de menor longitud que el SST. Éste es en realidad el llamado *Problema de Steiner*: dado un conjunto P de n puntos en el plano, conectarlos con líneas rectas de longitud total mínima, permitiendo que dos o más rectas se encuentren en puntos distintos de los de P.

El problema de Steiner en grafos consiste en encontrar un árbol de coste mínimo, que conecte un subconjunto de los vértices de un grafo $V'\subseteq V$, pudiéndose utilizar o no, el resto de vértices. Ambos problemas, el de Steiner y el de Steiner en grafos son NP-difíciles. Los únicos algoritmos eficientes para este tipo de problemas son los *algoritmos heurísticos*: algoritmos que no garantizan la obtención de la solución óptima.



Longitud del SST = 10.123



Solución del Problema de Steiner
longitud = 9.196

2.5 Ejercicios

1. Un hidrocarburo saturado es una molécula C_pH_q en la cual cada átomo de carbono tiene 4 enlaces, cada átomo de hidrógeno tiene un enlace y ninguna sucesión de enlaces forma un ciclo. Mostrar que para todo p entero positivo, C_pH_q puede existir solamente si $q = 2p + 2$.
2. Dibuja todos los hidrocarburos saturados con fórmula química C_6H_{14}.
3. Demostrar que todo árbol es un grafo bipartido.
4. Demostrar que si todo par de vértices de un grafo sin bucles G, están conectados por un único camino, entonces G es un árbol.
5. Un grafo acíclico es llamado también un bosque. Demostrar que: a) Cada componente conexa de un bosque es un árbol, b) G es un bosque sii $m = n - \omega$ (ω es el número de componentes conexas).
6. Mostrar que si G es un árbol con $\Delta \geq k$, entonces G tiene al menos k vértices de grado uno.
7. Un centro de G es un vértice u tal que $\max_{v \in V} d(u,v)$ es lo más pequeño posible. Demostrar que un árbol tiene o bien un centro o bien dos centros adyacentes.
8. Demostrar que si G es un bosque con exactamente 2k vértices de grado impar, entonces existen k caminos aristodisjuntos $P_1, P_2, ..., P_k$ en G tal que $E(G)= E(P_1) \cup E(P_2) \cup ... \cup E(P_k)$ (E(H) denota el conjunto de aristas de H).
9. Mostrar que un árbol con sólo dos vértices de grado 1 es un camino.
10. Mostrar que si G no tiene bucles y tiene exactamente un árbol generador, entonces es un árbol.
11. Sea G conexo y $a \in A$. Demostrar que;
    a) a está en cada árbol generador de G sii a es una arista de corte de G.
    b) a no está en ningún árbol generador de G sii a es un bucle.
12. Mostrar que si G es simple, contiene al menos $m-n+\omega$ ciclos distintos (ω es el numero de componentes conexas).
13. Sea F un bosque maximal de G. Mostrar que:
    a) Para cada componente conexa H de G, $F \cap H$ es un árbol generador de H
    b) $m(F) = n(G) - \omega(G)$
14. Probar que el complementario de un árbol no trivial es, o bien conexo, o bien consta de un vértice aislado y un subgrafo completo.
15. Adaptar el algoritmo de Kruskal para resolver el problema de conexión con pre-asignaciones: Conectar un número de pueblos por medio de una red de carreteras de coste mínimo, con el requerimiento adicional de que ciertos pares de pueblos estén unidos directamente.
16. ¿Puede ser adaptado el algoritmo de Kruskal para hallar:
    a) un árbol generador de coste máximo? b) un bosque maximal de coste mínimo?.

17. Dado $v \in V$, sea a_v una arista incidente con v de coste mínimo.

    a).- Demostrar que existe un árbol generador de coste mínimo que contiene a a_v.

    b).- ¿Puede $A'=\{a_v, v \in V\}$ contener un ciclo?.

    c).- En el caso en que G[A'] sea acíclico: ¿Existe siempre un árbol generador de coste mínimo que contiene a A'?.

18. Considera el algoritmo que intenta hallar un árbol generador de coste mínimo T* eliminando las aristas de coste máximo, una a una, de un grafo conexo G siempre y cuando no desconecten el grafo. Probar que T* es en efecto un árbol generador de coste mínimo.

19. El algoritmo de Kruskal puede proporcionar diferentes árboles generadores de coste mínimo del mismo grafo G dependiendo de cómo se deshagan los empates cuando las aristas son ordenadas. Demostrar que para cada árbol generador de coste mínimo T, existe una manera de ordenar las aristas de G en el algoritmo de Kruskal para que el algoritmo proporcione T.

20. Describir un algoritmo eficiente que, dado un grafo G, halle un árbol generador de G tal que el mayor coste de sus aristas sea mínimo entre todos los árboles generadores.

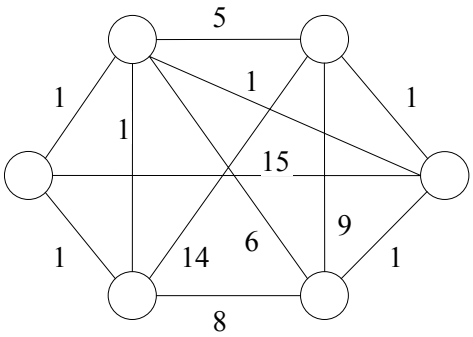
21. Sea T un árbol generador de coste mínimo de un grafo G. ¿Cómo se podría modificar T si se añadiese a G un nuevo vértice y aristas incidentes con él?

22. Sea a una arista de coste mínimo en G. Demostrar que existe un árbol generador de coste mínimo que la contiene.

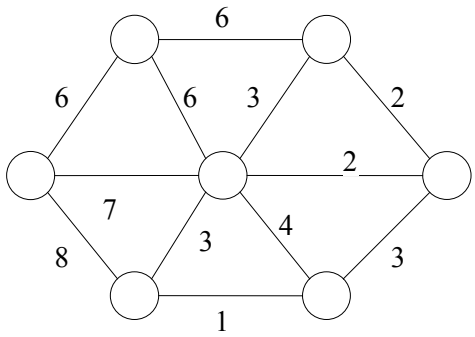
23. Sea C un ciclo de G y a una de sus aristas de coste máximo. Demostrar que existe un árbol generador de coste mínimo en G-a que es también un árbol generador de coste mínimo de G.

24. Mostrar que si en cada cortadura de G existe una única arista de coste mínimo, entonces el grafo tiene un único árbol generador de coste mínimo. Mostrar que el contrario no es cierto.

25. Encontrar el árbol generador de mínimo peso en los grafos



(a)



(b)

Tema 3. Caminos más cortos

3.1 Introducción

Sea G = (V, E) un grafo dirigido en el que cada arco e=(i,j) tiene asociado un coste (o longitud) c_{ij}. El coste de una cadena entre dos vértices s y t es la suma de los costes de los arcos que contiene. Llamaremos *cadena más corta* de s a t a una cadena s-t con coste mínimo (o longitud mínima).

Si en G existe un ciclo con coste total negativo (abreviadamente: *ciclo negativo*) y existe algún camino de s a t que pasa por alguno de sus vértices, entonces, recorriendo el ciclo repetidamente, podemos construir cadenas de s a t con coste menor que cualquier número (problema no acotado). Por otro lado, si no existen tales ciclos, siempre habrá una cadena más corta de s a t que sea en realidad un camino, por lo tanto, en este caso, es equivalente hablar de *camino más corto* o cadena más corta. En este tema veremos algoritmos polinómicos para calcular el camino más corto en un grafo sin ciclos negativos y estudiaremos también otros problemas relacionados. Sin embargo, el problema de encontrar el camino más corto entre dos vértices dados en un grafo con ciclos negativos es NP-difícil.

Si el grafo es no dirigido, podemos transformarlo en uno dirigido desdoblando cada arista en dos arcos opuestos con el mismo coste. Sin embargo, si existe una arista con coste negativo, al desdoblarla aparece un ciclo negativo, por lo que no se podrá, en este caso, aplicar los algoritmos que vamos a estudiar. (Sin embargo, si el grafo no dirigido original no tiene ciclos negativos, es posible calcular caminos más cortos en tiempo polinómico utilizando técnicas basadas en acoplamientos).

Todos los algoritmo que veremos en este tema son de Programación Dinámica y permiten calcular los caminos más cortos desde un vértice dado s al resto de vértices del grafo con un coste computacional similar al de calcular el camino más corto desde s a un único vértice t.

3.2 Ecuaciones de Bellman

Supongamos que no existen ciclos negativos.

Definimos $c_{ij} = \infty$ para todo $(i,j) \notin E$. Por simplicidad, supondremos que queremos calcular los caminos más cortos desde el vértice 1 al resto de vértices. Sea:

u_j = longitud del camino más corto del vértice 1 al vértice j.

Es obvio que $u_1 = 0$.

Notar que si P es un camino más corto entre dos vértices s y t, entonces cualquier sección del camino es también un camino más corto entre los vértices extremos de la sección (Principio de optimalidad de la Programación Dinámica).

Por lo tanto, si P es un camino más corto de 1 a j y k es el vértice anterior a j en ese camino, se cumplirá que $u_j = u_k + c_{kj}$. Entonces, es evidente que se cumplen las *ecuaciones de Bellman*:

(3.1)
$$\begin{cases} u_1 = 0 \\ u_j = \min\{u_k + c_{kj} : k \neq j\} \quad j = 2,...,n \end{cases}$$

Estas ecuaciones no permiten, en general, calcular el valor de cada u_j, sin embargo veremos casos en los que es posible resolverlas. Supongamos que hemos resuelto estas ecuaciones y conocemos los u_j para todo vértice j. Entonces, el vértice anterior a j en el camino más corto de 1 a j, es el vértice k que cumple que $u_j = u_k + c_{kj}$, el vértice anterior al k será un vértice r que cumpla $u_k = u_r + c_{rk}$, y así sucesivamente. De esta forma, podemos determinar los vértices que forman el camino más corto de 1 a j.
Si todos los ciclos de G tienen longitud estrictamente positiva, se puede demostrar que las ecuaciones de Bellman tienen una única solución que representa las longitudes de los caminos más cortos desde el vértice 1 al resto de vértices. Si existen ciclos de longitud cero, estas ecuaciones pueden tener varias soluciones (ver ejercicio 1) aunque los algoritmos que vamos a estudiar proporcionan las longitudes correctas.

3.3 Grafos acíclicos. Método del camino crítico

Si G es un grafo sin ciclos, es posible ordenar los vértices de forma que las ecuaciones (3.1) se pueden resolver recursívamente. Notar que para calcular u_j, sólo es necesario considerar los vértices k para los que $(k,j) \in E$ (para el resto, $c_{kj}=\infty$).

Teorema 3.1

Un grafo dirigido es acíclico sii es posible numerar sus vértices de forma que se cumple que si $(i,j) \in E$ entonces $i<j$.

Para ello se numera como primer vértice un vértice con grado de entrada cero y se eliminan los arcos que salen de él, a continuación se numera como segundo un vértice con grado de entrada cero en el grafo resultante; y así sucesivamente hasta numerar todos los vértices. Este método tiene complejidad $O(n^2)$ ya que, para numerar un vértice hay que examinar los grados de entrada de los vértices aún sin numerar (máximo n) y luego modificar los grados de entrada del resto (máximo n).
Si los vértices están numerados de esta forma, las ecuaciones (3.1) pueden ser reemplazadas por:

(3.2)
$$\begin{cases} u_1 = 0 \\ u_j = \min\{u_k + c_{kj} : (k,j) \in E\} \quad j = 2,...,n \end{cases}$$

Estas ecuaciones permiten calcular ordenadamente u_1, u_2, ...,, u_n, ya que para calcular u_j, solo es necesario conocer u_1, u_2, ...,, u_{j-1} (si $(k, j)\in E$, entonces $k<j$). Se puede comprobar fácilmente que el cálculo de todos los u_j requiere un número de operaciones que es $O(m)$.

Para calcular los caminos más cortos desde un vértice distinto del 1, basta con eliminar todos los arcos entrantes en él y los vértices con una numeración menor, numerarlo como 1 y aplicar las ecuaciones (3.2).

Secuenciación de proyectos. El método del camino crítico.

Considerar la ejecución de un proyecto que incluye la realización de un gran número de tareas o actividades que están relacionadas entre sí por relaciones de precedencia: para poder realizar una actividad determinada, es necesario que otras actividades hayan sido ya realizadas. La ejecución de este tipo de proyectos hace necesaria una planificación racional del orden en que deben ser realizadas las actividades y cuándo podrán ser efectuadas. Los métodos que se utilizan para este problema se designan con el nombre de PERT (Project Evaluation Research Task) y uno de ellos es el del *camino crítico*.

Cada una de las actividades que componen el proyecto se representa por un vértice. Si para realizar una actividad i, es necesario haber realizado la actividad j, se incluye un arco (j,i), con un coste c_{ji} que representa el tiempo que debe transcurrir desde el comienzo de la actividad j al de la i; y esto para todas las relaciones de precedencia. El resultado es un grafo acíclico ya que la existencia de un ciclo implicaría que el proyecto es irrealizable. Se incluyen en el grafo dos nuevos vértices s y t que representan el comienzo y la finalización del proyecto, respectivamente. Es evidente que el coste del camino más largo (de mayor coste) de s a t representa el mínimo tiempo necesario para completar el proyecto. El camino correspondiente se llama camino crítico ya que las actividades que están incluidas en el camino determinan la duración mínima del proyecto, y cualquier retraso en la ejecución de cualquiera de ellas implica un retraso en la finalización del proyecto (notar que puede haber varios caminos críticos con el mismo coste).

El camino más largo puede calcularse reemplazando c_{ij} por $-c_{ij}$, para todos los arcos $(i,j)\in E$ y aplicando (3.2) (ya que el grafo es acíclico, no existen ciclos negativos) o, simplemente sustituyendo el mínimo por el máximo en (3.2). Sin embargo, el problema de calcular el camino más largo en un grafo no acíclico y con costes no negativos es NP-difícil en general.

3.4 Grafos con longitudes no negativas. Método de Dijkstra

Sea G=(V,E) un grafo dirigido en el que las longitudes c_{ij} son no negativas. En este caso se puede aplicar un algoritmo más eficiente que en el caso general, que veremos en la siguiente sección.

En este algoritmo se asocia una etiqueta u_j a cada vértice j. Ciertas etiquetas se designan como variables y el resto fijas. Si una etiqueta u_j es fija, su valor es igual a la longitud del camino más corto de 1 a j, si es

variable, es simplemente una cota superior a esa longitud. El conjunto de vértices con etiquetas fijas en cualquier momento se denota por P y el de los que tienen etiquetas variables, por T. Se utiliza también la etiqueta predecesor p_j que indicará el vértice anterior al j en el camino más corto de 1 a j.

Algoritmo de Dijkstra

Inicio. Hacer $u_1 := 0$, $u_j := c_{1j}$ y $p_j := 1$, para todo j tal que $(1,j) \in E$; $u_j := \infty$, para el resto,

$P := \{1\}$, $T := \{2, 3, \ldots, n\}$.

Mientras 1 = 1

1) Encontrar el k para el que $u_k = \min \{u_j : j \in T\}$. Hacer $P := P \cup \{k\}$, $T := T - \{k\}$. Si $T = \varnothing$, PARAR,
2) Para todo arco (k,j) tal que $j \in T$, hacer: $u_j := \min \{u_j, u_k + c_{kj}\}$. Si u_j ha cambiado, hacer $p_j := k$.

FinMientras

Notar que, inicialmente sólo hay una etiqueta fija, la del vértice 1, con valor $u_1=0$. En cada iteración, la etiqueta con menor valor entre las etiquetas variables, digamos u_k, se convierte en fija, entonces u_k es la longitud del camino más corto de 1 a k y se actualizan los valores de las etiquetas variables, u_j, comparándolas con $u_k + c_{kj}$, que representa la longitud del camino más corto hasta k, más el arco (k,j), esto es, un camino de 1 a j. El algoritmo acaba cuando todas las etiquetas son fijas.

La demostración de que el algoritmo proporciona los caminos más cortos del vértice 1 al resto de vértices, se puede hacer demostrando que, al final de cada iteración, la etiqueta de cada vértice en P es la longitud del camino más corto, mientras que, para los vértices en T, u_j es la longitud del camino más corto con la restricción de que los vértices internos del camino estén en P.

Los caminos se pueden construir utilizando recursivamente la etiqueta p_j que es el vértice anterior al j en el camino más corto hasta j.

Complejidad del algoritmo: en cada iteración, tanto en (1) como en (2) se hacen como máximo n comparaciones y asignaciones. Dado que el número de iteraciones es n, la complejidad es $O(n^2)$. Sin embargo, existen implementaciones del algoritmo que logran una complejidad menor: $O(m+n\log n)$.

3.5 Método de Bellman-Ford de aproximaciones sucesivas.

El algoritmo de Dijkstra sólo es válido en el caso de que todas las longitudes sean no negativas, lo que permite fijar la etiqueta de ciertos vértices antes de que acabe de ejecutarse el algoritmo. En caso de que existan longitudes negativas, las siguientes ecuaciones proporcionan un método recursivo que permite calcular las longitudes de los caminos más cortos por aproximaciones sucesivas, siempre que no existan ciclos negativos.

$$\textbf{(3.3)} \qquad \begin{cases} u_1^{(1)} = 0 \\ u_j^{(1)} = c_{1j} \quad \forall j \text{ tal que } (1,j) \in E,\ u_j^{(1)} = \infty, \text{ para el resto} \\ u_j^{(t+1)} = \min\{u_j^{(t)}, \min\{u_k^{(t)} + c_{kj} : (k,j) \in E\}\} \quad t = 1,2,..., j = 1,...,n \end{cases} \qquad \text{(b)}$$

Notar que $u_j^{(1)} \geq u_j^{(2} \geq u_j^{(3)} \geq ...$ y que en cada iteración se revisan todas las etiquetas considerando todos los arcos del grafo. Vamos a ver que estas etiquetas son aproximaciones sucesivas de los u_j, la longitud de los caminos más cortos, y que convergen a estos valores en un número finito de pasos.

Teorema 3.2

Si no hay ciclos negativos, y se aplican recursivamente las ecuaciones (3.3), *entonces, para todo t y todo vértice j,* $u_j^{(t)}$ *es la longitud del camino más corto del vértice 1 al j, sujeto a la condición de que el camino tenga como máximo t arcos.*

Dado que un camino puede tener como máximo n-1 arcos, es obvio que $u_j^{(n-1)} = u_j$. Por otro lado, es fácil ver que si para un t < n-1 se cumple que $u_j^{(t)} = u_j^{(t-1)}$, para todo j; entonces será $u_j^{(p)} = u_j^{(t)}$ para todo p > t y para todo j; y por lo tanto $u_j^{(t)} = u_j$ para todo j y habremos calculado los caminos más cortos en menos de n-1 iteraciones.

Observar que las ecuaciones (3.3) no evitan que la longitud mínima calculada corresponda a una cadena que contenga un ciclo ya que puede ocurrir que $u_j^{(t+1)} = u_k^{(t)} + c_{kj}$, y que la cadena que corresponde a la etiqueta $u_k^{(t)}$ pase por el vértice j, con lo que la cadena cuya longitud es $u_j^{(t+1)}$ contiene un ciclo de longitud negativa o cero. De hecho, las etiquetas $u_j^{(t)}$ corresponden en realidad a cadenas más cortas por lo que la única garantía de que el resultado final corresponda a caminos más cortos es que no existan ciclos negativos. Por otro lado, si existe un ciclo negativo alcanzable desde el vértice 1, a partir de cierta iteración las etiquetas de los vértices del ciclo corresponderán en realidad a cadenas que contienen al ciclo negativo. De hecho, si $u_j^{(n)} < u_j^{(n-1)}$ para algún vértice j, podemos asegurar que la cadena cuya longitud es $u_j^{(n)}$ contiene un ciclo negativo. Por lo tanto, las ecuaciones (3.3) pueden ser utilizadas para detectar la existencia de ciclos negativos.

Por último, se puede demostrar fácilmente por inducción que, en cada iteración de la aplicación de las ecuaciones (3.3), es suficiente considerar para la minimización en (b) los vértices k para los que $u_k^{(t)}$ representa la longitud del camino más corto de 1 a k con exactamente t arcos, y que estos vértices son precisamente aquellos para los que $u_k^{(t)} < u_k^{(t-1)}$.

Estas observaciones permiten establecer el algoritmo de una forma más eficiente, aunque no afectan su complejidad. Como ya hemos dicho, las operaciones de minimización en (3.3) (b), se tienen que hacer, como máximo para t = 1, 2, ..., n-1, y, para cada valor de t, el cálculo de los mínimos para cada vértice j implica realizar $d^-(j)$ sumas y comparaciones, por lo que, considerando todos los vértices, el número total de operaciones es del orden de m (recordar que la suma de los grados $d^-(j)$ para todo los vértices es m). Así, la complejidad del algoritmo es O(mn).

Denotaremos por Γ(i) al conjunto de vértices j para los que existe el arco (i,j); y por Γ(S) a la unión de los conjuntos Γ(i) para todos los vértices i∈S.

Algoritmo de Bellman- Ford

Inicio. Hacer t:= 1, $u_1^{(1)} := 0$, $u_j^{(1)} := c_{1j}$, $p_j := 1$, para todo j∈Γ(1) y $u_j^{(1)} := \infty$ para el resto de vértices.

Hacer S:= Γ(1).

Paso 1. Para todo vértice j∈Γ(S), hacer $u_j^{(t+1)} := \min\{u_j^{(t)}, \min\{u_k^{(t)} + c_{kj} : k \in S, (k,j) \in E\}$; si

$u_j^{(t+1)} < u_j^{(t)}$, y $u_j^{(t+1)} = u_k^{(t)} + c_{kj}$ hacer $p_j := k$.

Para el resto de vértices, hacer $u_j^{(t+1)} := u_j^{(t)}$.

Paso 2

- Si t ≤ n-1 y $u_j^{(t+1)} = u_j^{(t)}$ para todo j, estos valores representan los valores de los caminos más cortos. PARAR.
- Si t < n-1 y $u_j^{(t+1)} < u_j^{(t)}$ para algún j, ir al *Paso 3*.
- Si t = n-1 y $u_j^{(t+1)} < u_j^{(t)}$ para algún j, existe un ciclo negativo en el grafo. PARAR.

Paso 3. Actualizar el conjunto S := {j∈V: $u_j^{(t+1)} < u_j^{(t)}$}, hacer t:= t+1 y volver al *Paso 1*.

3.6 Grafos con tiempos de recorrido

Supongamos que además de la longitud c_{ij} existe un entero positivo $t_{ij} > 0$ asociado a cada arco (i,j) del grafo que representa, por ejemplo, el tiempo necesario para recorrer ese arco (puede representar también una carga que haya que recoger en el arco). Se plantea el problema de encontrar los caminos más cortos desde un vértice al resto con la condición de que los caminos no requieran más de cierta cantidad de tiempo T. Definimos:

$u_j(t)$ = longitud del camino más corto desde el vértice 1 al j, sujeto a la condición de que el camino requiera no más de t unidades de tiempo.

Se puede demostrar fácilmente que se cumplen las siguientes ecuaciones:

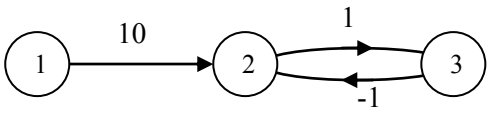
(3.4)
$$\begin{cases} u_j(t) = \infty \text{ para todo } j \in V \text{ y todo } t < 0 \\ u_1(0) = 0 \\ u_j(t) = \min\{u_j(t-1), \min\{u_k(t-t_{kj}) + c_{kj} : (k,j) \in E\} \quad t = 1,2,..., j = 1,...,n \end{cases}$$

Estas ecuaciones permiten calcular recursivamente $u_j(t)$ para t = 1, 2, ..., T (suponemos que los tiempos t_{ij} son enteros) con un número de operaciones que es $O(n^2T)$. Observar que las ecuaciones (3.4) son una generalización de las ecuaciones de Bellman-Ford, en las que la restricción impuesta a los caminos se refería al número máximo de arcos utilizados.

Considerar el problema de encontrar una ruta para un vehículo desde un origen, x, a un destino, z, dados, dentro de un grafo en el que sólo se puede repostar en ciertos vértices (x y z están entre estos vértices) y al recorrer un arco (i,j) tiene un consumo de carburante t_{ij}. Si la capacidad del depósito de carburante es T, el consumo realizado desde que sale de un vértice en el que puede repostar hasta que llega a otro de esos vértices, no puede ser mayor que T. El problema es encontrar la ruta con longitud mínima sujeta a la condición de que el vehículo no se quede en ningún momento sin carburante. Dado que puede ser que el camino más corto de x a z tenga un consumo mayor que T, no es posible resolver este problema directamente. En cambio, se puede resolver en dos fases. Primero aplicamos las ecuaciones (3.4) n veces, tomando como vértice inicial cada uno de los vértices en los que se puede repostar; así podemos calcular $u_{ij}(T)$: longitud del camino más corto entre i y j con un consumo no mayor que T. A continuación resolvemos un problema de camino más corto (sin otra restricción) entre el vértice x y el z, en un grafo que contiene sólo los vértices en los que se puede repostar y cuyos arcos tienen como longitudes los valores $u_{ij}(T)$ previamente calculados.

3.7 Ejercicios

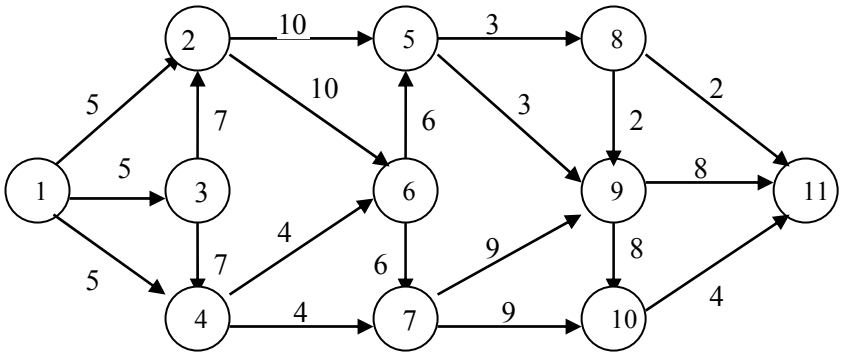
1. Considerar el grafo de la figura. Mostrar que para este grafo las ecuaciones de Bellman no tienen una única solución finita. Caracterizar el conjunto de todas las soluciones.



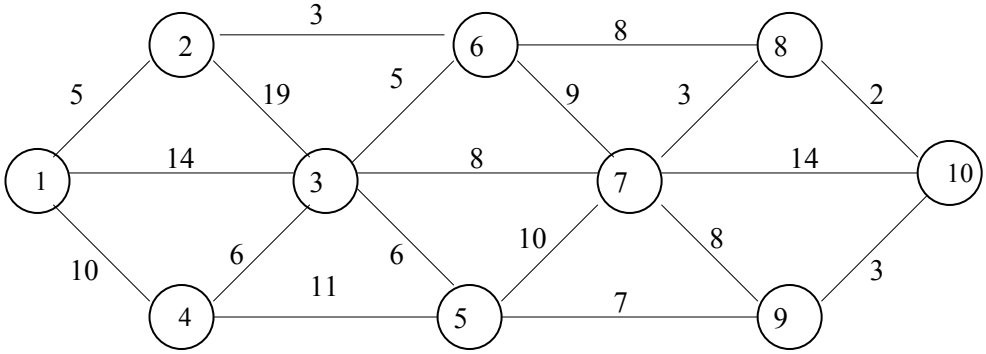
2. Rescribir las ecuaciones de Bellman para cada uno de los siguientes casos:
   a) c_{ij} = probabilidad de que el arco (i,j) exista,
      u_i = probabilidad asociada al camino "más seguro" desde el vértice 1 al j.
   b) c_{ij} = capacidad del arco (i,j),
      u_i = capacidad del camino de máxima capacidad desde el vértice 1 al j.

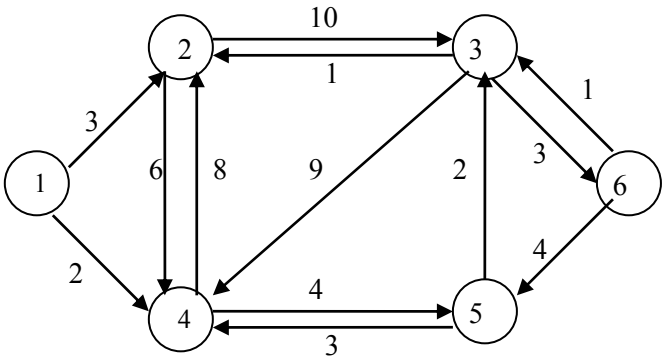
      (la capacidad de un camino es la mínima de las capacidades de sus arcos)

3. Considerar el diagrama PERT de la figura. Encontrar los caminos críticos y calcular el máximo retraso admisible para la actividad 5 que no produzca por sí solo un retraso en el proyecto global (los vértices 1 y 11 representan el inicio y final del proyecto, respectivamente).



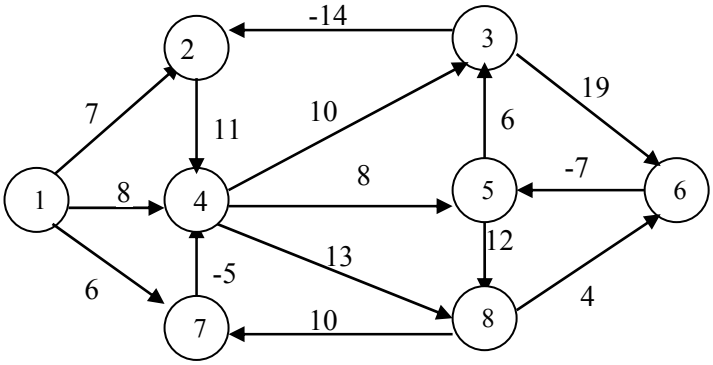
4. Utilizando el algoritmo de Dijkstra, calcular los caminos más cortos desde el vértice 1 al resto de vértices en los grafos de las figuras siguientes. Determinar en ambos casos la arborescencia de caminos más cortos.





5. Desarrollar un algoritmo, similar al de Dijkstra, para determinar el camino más corto entre dos subconjuntos disjuntos de vértices A y B (que se define como el camino más corto entre todos los caminos que tienen el vértice inicial en A y el vértice final en B). Analizar el número de sumas y comparaciones requeridas por el algoritmo.

6. Aplicar el algoritmo de Bellman-Ford para calcular los caminos más cortos, desde el vértice 1 al resto de vértices, en el grafo de la figura, o detectar un circuito negativo si existe.



7. Repetir el problema anterior reemplazando el coste del arco (4,5) por -8.

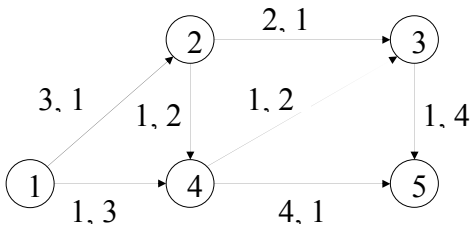
8. Supongamos que $u_j^{(t)}$ denota la longitud de la cadena más corta desde el vértice 1 al j, sujeto a la condición de que la cadena contenga exactamente t arcos (contando los arcos repetidos si los hay) ¿cómo deberían modificarse las ecuaciones de Bellman-Ford para adaptarse a este nuevo significado?

9. Mostrar que el problema de encontrar el camino más corto entre dos vértices que tenga un número dado de arcos, en un grafo sin circuitos negativos es NP-difícil. *Sugerencia*: demostrar que este problema incluye al Problema del Viajante Abierto: "encontrar el camino de coste mínimo de s a t (dos vértices dados) que pase por todos los vértices del grafo", que es NP-difícil.

10. Mostrar que, en el algoritmo de Bellman-Ford, en cada iteración al menos una nueva etiqueta $u_j^{(t)}$, para algún vértice j≠1, pasa a ser la longitud del camino más corto de 1 a j; esto es, $u_j^{(t)} = u_j$ (suponiendo que no hay ciclos negativos).

11. Demostrar que, si en la aplicación de las ecuaciones de Bellman-Ford a un grafo G, en una iteración t, $1 \leq t \leq n-1$, existen al menos n-t vértices j para los cuales se cumple: $u_j^{(t+1)} < u_j^{(t)}$, entonces G contiene un ciclo negativo.

12. Determinar los caminos más cortos del vértice 1 al resto de vértices en el grafo de la figura, que emplean como máximo 4 unidades de tiempo (los números representan la longitud y el tiempo de cada arco, en este orden).



Tema 4. Acoplamientos

4.1 Acoplamientos

Sea G = (V, E) un grafo no dirigido. Un subconjunto M de aristas se dice que es un *acoplamiento* de G si no contiene bucles y no hay dos aristas incidentes con el mismo vértice.

Un vértice v∈V se dice M-*saturado* si alguna arista de M es incidente con v, en caso contrario se dice M-insaturado. Si todo vértice de G es M-saturado, se dice que M es un *acoplamiento perfecto*.

M es un acoplamiento máximo de G si no hay otro acoplamiento M' de G tal que |M'| > |M|.

Ejemplo:

(a) Acoplamiento máximo (b) Acoplamiento perfecto

Un camino M-*alternado* es un camino cuyas aristas son alternativamente de E-M y de M. Por ejemplo, en el grafo de la figura (a): 5, 8, 1, 7, 6. Un camino M-*aumentado* es un camino M-alternado con los vértices inicial y final M-insaturados.

Teorema 4.1

Un acoplamiento M de G es máximo sii G no contiene ningún camino M-aumentado.

4.2 Acoplamientos y recubrimientos en grafos bipartidos

Sea S ⊆ V en G = (V, E). Llamamos *entorno* de S en G al conjunto de vértices de G que son adyacentes a algún vértice de S; lo denotaremos por N(S).

Teorema 4.2 (Hall)

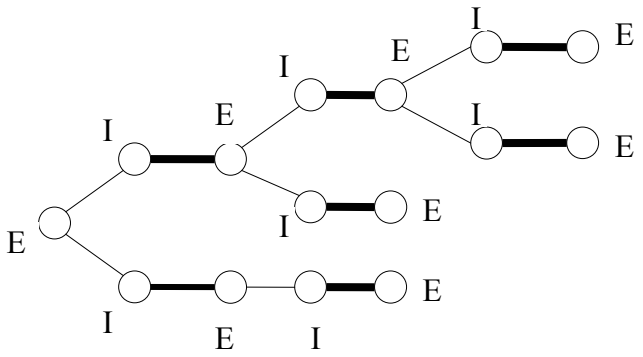
Sea G un grafo bipartido con bipartición (X, Y), entonces, G tiene un acoplamiento que satura todo vértice de X sii:

(4.1) $|N(S)| \geq |S|$ *para todo* $S \subseteq X$

Un *árbol alternado*, relativo a un acoplamiento M, es un árbol T que cumple:

a) Un vértice de T es M-insaturado y se le llama raíz de T.
b) Todos los caminos de T que empiezan en la raíz son M-alternados.
c) Todos los caminos maximales desde la raíz de T son de cardinalidad par (por lo tanto la última arista de estos caminos siempre es de M).

Ejemplo:



Etiquetamos los vértices de un árbol alternado con I (interior) o E (exterior) según que el camino desde la raíz a ese vértice sea de cardinalidad impar o par, respectivamente. Llamamos predecesor de un vértice v, distinto de la raíz, al vértice anterior a v en el único camino desde la raíz a v. Si w es el predecesor de v, entonces diremos que v es un sucesor de w (puede no ser el único). Las siguientes propiedades de los árboles alternados son inmediatas.

- Todos los vértices extremos de caminos más largos desde la raíz, tendrán etiqueta E.
- Todo vértice etiquetado I tiene un único sucesor.
- El número de vértices etiquetados E es una unidad mayor que el número de vértices etiquetados I.

Un *árbol húngaro* es un árbol alternado de G, en el que todas las aristas de G incidentes con vértices etiquetados E, tienen el otro extremo en vértices del árbol etiquetados I.

Teorema 4.3

Si G es bipartido y k-regular, entonces G tiene un acoplamiento perfecto.

Se llama recubrimiento de un grafo G a un subconjunto de vértices, $K \subseteq V$, tal que toda arista de G tiene al menos un vértice extremo en K. Un recubrimiento K es mínimo en G si no existe otro recubrimiento K' tal que $|K'| < |K|$.

Si K es un recubrimiento y M un acoplamiento de G, por cada arista de M tiene que haber un vértice distinto en K y, por lo tanto, se cumplirá: $|M| \leq |K|$. Concretamente, si M* es un acoplamiento máximo y K* es un recubrimiento mínimo, se cumple: $|M^*| \leq |K^*|$. Vamos a ver que si G es un grafo bipartido, se cumple la igualdad, aunque esto no es cierto para cualquier grafo.

Teorema 4.4 (König)

En un grafo bipartido, el número de aristas de un acoplamiento máximo es igual al número de vértices de un recubrimiento mínimo.

4.3 El Problema de Asignación Óptima.

Una empresa dispone de p trabajadores $\{x_1, x_2, ..., x_p\}$ para realizar p trabajos $\{y_1, y_2, ..., y_p\}$, y el coste de asignar cada trabajador, x_i, a cada trabajo y_j es w_{ij}. El *Problema de Asignación Óptima* consiste en asignar cada trabajador a un trabajo distinto, de forma que el coste total de las p asignaciones sea mínimo.

Definimos un grafo G, bipartido y completo, con bipartición (X, Y), donde $X=\{x_1, x_2, ..., x_p\}$, e $Y=\{y_1, y_2, ..., y_p\}$, y la arista (x_i, y_j) tiene coste w_{ij}. El problema de Asignación Óptima es equivalente a encontrar en G el acoplamiento perfecto de coste mínimo. Vamos a ver un algoritmo polinómico para este problema. Notar que la suposición de que el grafo es completo no es restrictiva, ya que, si alguna arista no existe, podemos añadirla con coste infinito.

Definimos *etiquetado admisible* como una función de valores reales, l(.), sobre el conjunto de vértices $X \cup Y$, que cumple: $l(x_i) + l(y_j) \leq w_{ij}$, para todo $x_i \in X$, $y_j \in Y$. Llamaremos a l(v), *etiqueta* del vértice v.

Se puede obtener un etiquetado admisible inicial como sigue:

- Para i = 1, ..., p, hacer $l(x_i) := \min \{w_{ij} : j = 1, ..., p\}$.
- Para j = 1, ..., p, hacer $l(y_j) := \min \{w_{ij} - l(x_i) : i = 1, ..., p\}$.

Si l es un etiquetado admisible, denotamos por $E_l = \{(x_i, y_j) \in E: l(x_i) + l(y_j) = w_{ij}\}$. Llamaremos *subgrafo igualdad* al subgrafo generador cuyo conjunto de aristas es E_l, y lo denotaremos por G_l.

Teorema 4.5

Sea l un etiquetado admisible. Si G_l contiene un acoplamiento perfecto M, entonces M* es un acoplamiento óptimo de G.*

El algoritmo de Kuhn-Munkres está basado en este teorema. Se parte de un etiquetado admisible inicial, se forma el subgrafo igualdad y se construye en él un acoplamiento inicial (que puede ser vacío). Este acoplamiento es mejorado iterativamente por medio de caminos aumentados, que son encontrados desarrollando un árbol alternado cuyo conjunto de vértices denotaremos por A (Paso 2). Cuando el acoplamiento no puede ser mejorado (es máximo en el subgrafo igualdad), entonces se cambia el etiquetado admisible (Paso 3) de forma que el árbol alternado construido puede ser prolongado en el nuevo subgrafo igualdad. Este proceso se repite hasta que se encuentre un camino aumentado. En ese momento, se modifica el acoplamiento (su tamaño aumenta en una unidad) y se empieza a construir un nuevo árbol alternado, y así sucesivamente hasta obtener un acoplamiento perfecto, que será óptimo.

Algoritmo de Asignación Óptima (Kuhn- Munkres)

Inicio Encontrar un etiquetado admisible l, construir G_l y tomar como acoplamiento M = ∅.

Paso 1 Si X es M-saturado. PARAR, M es un acoplamiento óptimo.

En otro caso, sea u∈X un vértice M-insaturado. Tomar u como raíz de un árbol alternado, marcarlo E y hacer A={u}.

Paso 2 *Mientras* exista un vértice marcado E, sin explorar:

Sea x_0 un vértice marcado E y sin explorar.

Para toda arista (x_0, y_j) de G_l, con $y_j \notin A$, hacer:

a) Si y_j es M-saturado, sea $(x_i, y_j) \in M$. Marcar y_j como I, x_i como E y hacer $A := A \cup \{y_j, x_i\}$ y añadir las aristas (x_0, y_j) y (x_i, y_j) al árbol alternado.

b) Si y_j es M-insaturado, el camino de de u a y_j en el árbol alternado es un camino M-aumentado, sea E(P) el conjunto de aristas del camino. Hacer $M := M \Delta E(P)$, rechazar el árbol alternado, borrar las marcas E, I de los vértices, e ir al *Paso 1*.

FinPara-toda-arista

FinMientras

No quedan vértices E por explorar (árbol húngaro); ir al *Paso 3*.

Paso 3 Calcular $\alpha_l = \min \{w_{ij} - l(x_i) - l(y_j)$: para todo $x_i \in A$, marcado E, y para todo $y_j \notin A\}$.

Para todo vértice *v*, hacer: $l(v) := \begin{cases} l(v) + \alpha_l & \text{si v tiene marca E} \\ l(v) - \alpha_l & \text{si v tiene marca I} \\ l(v) & \text{en otro caso} \end{cases}$

Construir el nuevo subgrafo igualdad G_l, e ir al *Paso 2*. El árbol alternado se conserva, pero todos sus vértices se consideran no explorados.

Si el árbol alternado generado en el Paso 2 no puede ser prolongado, llamando S al conjunto de vértices etiquetados E, se cumple $|N_{G_l}(S)| < |N(S)|$, lo que demuestra que el subgrafo igualdad no contiene un acoplamiento perfecto y es necesario cambiar el etiquetado admisible y el subgrafo igualdad. Todas las aristas del árbol y del acoplamiento siguen estando en el nuevo subgrafo igualdad, por lo que no tenemos que empezar un nuevo árbol alternado, sino que se puede seguir desarrollando el que ya tenemos. De hecho, el cambio de etiquetado hace que aparezcan nuevas aristas en el subgrafo igualdad, que permitirán desarrollar el mismo árbol.

Complejidad

El número total de árboles alternados que será necesario desarrollar es como máximo p. Para construir un árbol alternado, hay que explorar los vértices marcados E, para lo que se analizan las aristas incidentes

con cada uno, lo que implica del orden de p operaciones para cada vértice, en total, p^2 operaciones para construir el árbol, sin contar las operaciones del paso 3. En el paso 3 también se hacen del orden de p^2 operaciones y es posible que tenga que ser ejecutado p veces durante la construcción del árbol (cada vez que se ejecuta podremos añadir al menos una arista y un vértice al árbol). Por lo tanto, cada árbol implica un número de operaciones del orden de p^3, y dado que el número de árboles construidos es p, el algoritmo es $O(n^4)$. Sin embargo, existen implementaciones de este algoritmo que son $O(n^3)$.

Ejemplo

Utilizamos la siguiente matriz de costes de asignación

	1	2	3	4	5	6	7	8
1	13	21	20	12	8	26	22	11
2	12	36	25	41	40	11	4	8
3	35	32	13	36	26	21	13	37
4	34	54	7	8	12	22	11	40
5	21	6	45	18	24	34	12	48
6	42	19	39	15	14	16	28	46
7	16	34	38	3	34	40	22	24
8	26	20	5	17	45	31	37	43

Utilizando el etiquetado inicial que se ha descrito, los costes reducidos son:

$l(y_j)$ →	5	0	0	0	0	2	0	3

$l(x_i)$ ↓		1	2	3	4	5	6	7	8
8	1	0	13	12	4	0	16	14	0
4	2	3	32	21	37	36	5	0	1
13	3	17	19	0	23	13	6	0	21
7	4	22	47	0	1	5	13	4	30
6	5	10	0	39	12	18	26	6	39
14	6	23	5	25	1	0	0	14	29
3	7	8	31	35	0	31	35	19	18
5	8	16	15	0	12	40	24	32	35

Primer cambio de etiquetado:

$l(y_j)$ →	5	0	-1	0	0	2	-1	3

$l(x_i)$ ↓		1	2	3	4	5	6	7	8
8	1	0	13	13	4	0	16	15	0
5	2	2	31	21	36	35	4	0	0
14	3	16	18	0	22	12	5	0	20
8	4	21	46	0	0	4	12	4	29
6	5	10	0	40	12	18	26	7	39
14	6	23	5	26	1	0	0	15	29
3	7	8	31	36	0	31	35	20	18
5	8	16	15	1	12	40	24	33	35

Segundo cambio de etiquetado:

$l(y_j)$ →	5	0	-1	0	0	2	-1	3
$l(x_i)$ ↓	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**
8	**1**	0	13	13	4	0	16	15	0
5	**2**	2	31	21	36	35	4	0	0
14	**3**	16	18	0	22	12	5	0	20
8	**4**	21	46	0	0	4	12	4	29
6	**5**	10	0	40	12	18	26	7	39
14	**6**	23	5	26	1	0	0	15	29
3	**7**	8	31	36	0	31	35	20	18
6	**8**	15	14	0	11	39	23	32	34

Tercer cambio de etiquetado:

$l(y_j)$ →	5	0	-5	-4	0	2	-1	3
$l(x_i)$ ↓	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**
8	**1**	0	13	17	8	0	16	15	0
5	**2**	2	31	25	40	35	4	0	0
14	**3**	16	18	4	26	12	5	0	20
12	**4**	17	42	0	0	0	8	0	25
6	**5**	10	0	44	16	18	26	7	39
14	**6**	23	5	30	5	0	0	15	29
7	**7**	4	27	36	0	27	31	16	14
10	**8**	11	10	0	11	35	19	28	30

4.4 Acoplamiento Perfecto de Coste Mínimo

Dado un grafo cualquiera G, con un número par de vértices y cuyas aristas tienen asociado un coste, el Problema del Acoplamiento Perfecto de Coste Mínimo consiste en encontrar un acoplamiento perfecto en G (si existe) que tenga coste total mínimo. En el apartado anterior hemos visto un algoritmo que es válido para el caso en que el grafo es bipartido. Edmonds (1965) generalizó este algoritmo para aplicarlo a un grafo cualquiera. El algoritmo de Edmonds es bastante más complejo que el de Kuhn-Munkres y no lo estudiaremos; sin embargo su complejidad es $O(n^3)$, aunque normalmente exige mayor coste computacional resolver el problema en un grafo general que en un grafo bipartido del mismo tamaño.

4.5 Grafos eulerianos. El Problema del Cartero Chino

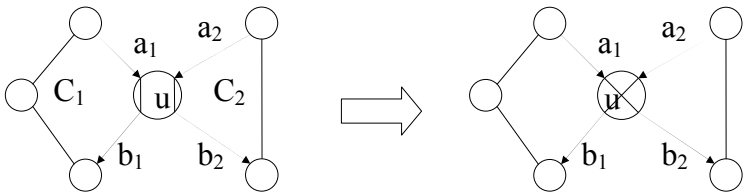
Dado un grafo no dirigido G, un *tour* de G es una cadena cerrada que atraviesa cada arista de G al menos una vez. Si atraviesa todas las aristas exactamente una vez, se dice que es un *tour euleriano*. G es un

grafo euleriano si contiene un tour euleriano. El problema de caracterizar los grafos eulerianos fue propuesto por primera vez por Euler (1736) (de ahí el nombre) en lo que se considera el nacimiento de la Teoría de Grafos.

Teorema 4.6

Un grafo conexo no dirigido es euleriano sii no tiene vértices de grado impar.

Si un grafo es euleriano, es muy fácil (complejidad $O(m)$) construir un tour euleriano: empezando en cualquier vértice, construir una cadena añadiendo cada vez una arista incidente con el último vértice de la cadena, sin repetir aristas, hasta volver al vértice inicial, con lo que tendremos una cadena cerrada. Si contiene a todas las aristas, hemos acabado, en caso contrario, se eliminan de G las aristas de la cadena cerrada y se vuelve a empezar una cadena empezando en un vértice que tenga alguna arista incidente. Y así sucesivamente hasta que no queden aristas en G. A continuación, se van uniendo las cadenas cerradas obtenidas, que tengan vértices comunes, hasta obtener una única cadena cerrada que es tour euleriano. La forma de unir dos cadenas C_1 y C_2 que contengan un mismo vértice es la siguiente: sea u el vértice común, supongamos que la arista anterior a u en C_1 es a_1 y la posterior b_1 (transición a_1-u-b_1), asimismo, sean a_2 y b_2 las aristas anterior y posterior a u, respectivamente, en C_2 (transición a_2-u-b_2), entonces, para unir C_1 y C_2, cruzamos las dos transiciones: al llegar a u por C_1, haremos a_1-u-b_2, y al volver a u por C_2, haremos transición a_2-u-b_1. Ver figura.



Sea G un grafo conexo no dirigido en el que cada arista tiene asociado un coste. El *Problema del Cartero Chino* consiste en encontrar un tour para G que tenga coste total mínimo; el coste del tour es la suma de los costes de las aristas que contiene, teniendo en cuenta las repeticiones. Este problema tiene muchas aplicaciones en situaciones en las que hay que encontrar una ruta para un vehículo que pase por todas las aristas de un grafo, por ejemplo: un cartero tiene que repartir las cartas en cierto distrito, por lo tanto tiene que recorrer un tour que pase por todas las calles al menos una vez, preferiblemente con una distancia (o tiempo) total mínima. Similarmente, buscar rutas óptimas para la recogida de basuras, inspección de redes eléctricas o de ferrocarril, etc. son problemas que están relacionados con éste.

La solución del problema es posible combinando el cálculo de caminos más cortos y de acoplamientos:

1) Calcular los caminos más cortos entre todos los pares de vértices impares de G.

2) Construir un grafo completo G_1 cuyos vértices son los vértices impares de G y los costes de las aristas son los costes de los caminos más cortos entre los vértices correspondientes. Sea M el acoplamiento perfecto de coste mínimo en G_1.
3) Para cada arista (v_i, v_j) de M, duplicar en G las aristas del camino más corto entre v_i y v_j. El resultado es un grafo con todos los vértices pares y, por lo tanto, euleriano. Un tour euleriano en este grafo es un tour de coste mínimo para G (el tour pasa dos veces por las aristas que corresponden a los caminos más cortos añadidos).

Teorema 4.7

El tour construido con el método anterior es un tour con coste total mínimo.

4.6 Ejercicios

1. Encontrar el número de acoplamientos perfectos en K_{2n} y $K_{n,n}$.

2. Mostrar que un árbol tiene, como máximo, un acoplamiento perfecto.

3. Dos personas juegan sobre un grafo eligiendo, alternativamente, vértices distintos: v_0, v_1, v_2 ... , tales que v_{i+1} debe ser adyacente a v_i, para $i \geq 0$. El último jugador que es capaz de elegir un vértice, gana. Mostrar que el primer jugador tiene una estrategia de éxito seguro si, y sólo si, G no tiene ningún acoplamiento perfecto.

4. Mostrar que es imposible, utilizando rectángulos de dimensiones 1x2, cubrir completamente un cuadrado 8x8 al que se le ha quitado, en dos esquinas opuestas, un cuadrado 1x1.

5. Mostrar que un grafo bipartido G tiene un acoplamiento perfecto si, y sólo si, $|N(S)| \geq |S|$, para todo $S \subseteq V$.

6. Una línea de una matriz es una fila o una columna de la matriz. Demostrar que el mínimo número de líneas que contienen todos los unos de una matriz de ceros y unos, es igual al máximo número de unos que se pueden escoger de manera que no haya dos en la misma línea.

7. Sean A_1, A_2, ..., A_m subconjuntos de un conjunto S. Un sistema de representantes distintos para la familia $\{ A_1, A_2, ..., A_m \}$ es un subconjunto $\{a_1,a_2,...,a_m\}$ de S tal que $a_i \in A_i$, $1 \leq i \leq m$, y $a_i \neq a_j$ para $i \neq j$. Demostrar que $(A_1, A_2, ..., A_m)$ tiene un sistema de representantes distintos sii $\left|\bigcup_{i\in J} A_i\right| \geq |J|$ para todos los subconjuntos J de $\{1,2,...,m\}$.

8. Demostrar la siguiente generalización del Teorema de Hall.
   Si G es un grafo bipartido con bipartición (X,Y), el número de aristas en un acoplamiento máximo es $|X| - \max_{S\subseteq X}\{|S| - |N(S)|\}$.

9. Deducir el Teorema de Hall a partir del teorema de König.

10. Una diagonal de una matriz nxn es un conjunto de n elementos, no dos de los cuales pertenecen a la misma fila o columna. El peso de una diagonal es la suma de los pesos de sus elementos. Encontrar la diagonal de mínimo peso de la matriz:

$$\begin{pmatrix} 4 & 5 & 8 & 10 & 11 \\ 7 & 6 & 5 & 7 & 4 \\ 8 & 5 & 12 & 9 & 6 \\ 6 & 6 & 13 & 10 & 7 \\ 4 & 5 & 7 & 9 & 8 \end{pmatrix}$$

11. Construir métodos que permitan resolver los siguientes problemas de acoplamientos en grafos bipartidos:
    a) Calcular el acoplamiento perfecto de coste mínimo cuando |X| = |Y| pero el grafo no es completo.
    b) Calcular el acoplamiento de coste mínimo que satura todo X, cuando |X| < |Y|.
    c) Calcular el acoplamiento perfecto de coste máximo cuando |X| = |Y| y el grafo es completo.
    d) Calcular el acoplamiento de coste mínimo cuando |X| = |Y| y el grafo es completo. (notar que el acoplamiento no tiene porqué ser perfecto y entonces las aristas de coste positivo no a parecerán nunca en la solución)
    e) Calcular el acoplamiento máximo de coste mínimo cuando |X| = |Y| y el grafo no es completo.

12. Adaptar el algoritmo de Kuhn-Munkres para calcular el acoplamiento de coste mínimo que sature todos los vértices de X en un grafo bipartido G(X,Y) con |X| < |Y|, en el que todo vértice de X es adyacente a todo vértice de Y (notar que el Teorema 4.5 no es aplicable en este caso).

13. Un taller posee 6 máquinas taladradoras. Un cierto día, llegan 5 piezas que necesitan ser taladradas. Cada pieza puede ser trabajada en cualquiera de las 6 máquinas. Cada pieza ha de procesarse en una única máquina y cada máquina sólo puede procesar una pieza. El número de horas máquina que se requeriría para realizar cada trabajo en cada máquina aparece en la tabla siguiente. Hallar la forma de asignar las piezas a las máquinas con mínimo número total de horas máquina.

		Pieza				
		A	**B**	**C**	**D**	**E**
Máquina	**1**	5	7	6	4	9
	2	8	10	3	4	7
	3	6	11	5	4	7
	4	5	8	7	3	9
	5	3	6	4	2	7
	6	3	7	5	3	7

14. Una agencia inmobiliaria tiene a la venta una variedad de casas y un número de potenciales compradores. Cada comprador potencial está interesado posiblemente en más de una casa. La inmobiliaria puede estimar muy ajustadamente cuánto pagaría cada comprador por cada casa en la que está interesado. Si la inmobiliaria recibe un 7% de cada venta. ¿Cuál sería la asociación comprador/casa que le proporcionaría un beneficio máximo?
    ESTIMACIONES EN MILLONES DE PESETAS

    Comprador 1/casa1: 12 Comprador 1/casa3: 13 Comprador 1/casa4: 10’5

    Comprador 2/casa1: 13 Comprador 2/casa2: 12’5 Comprador 3/casa1: 12’5

    Comprador 3/casa4: 16 Comprador 3/casa5: 18 Comprador 4/casa3: 13

    Comprador 4/casa1: 14 Comprador 4/casa2: 13 Comprador 5/casa3: 10’5

    Comprador 5/casa1: 12 Comprador 6/casa3: 17’5 Comprador 6/casa1: 10

Comprador 7/casa4: 15 Comprador 7/casa3: 13 Comprador 7/casa5: 17'5

15. Como parte de su proceso de fabricación, un fabricante de altavoces estéreos debe emparejar altavoces individuales antes de poder venderlos. Para medir la calidad del emparejamiento, la compañía genera coeficientes de emparejamiento para cada posible par. Un coeficiente bajo indica una buena pareja y un coeficiente alto indica un mal emparejamiento. Actualmente debe emparejar 6 altavoces entre si, cuyos coeficientes de acoplamiento son:

Altavoz	1	2	3	4	5	6
1	-	22	13	6	7	8
2	22	-	3	12	14	5
3	13	3	-	8	9	1
4	6	12	8	-	9	2
5	7	14	9	9	-	12
6	8	5	1	2	12	-

Hallar el máximo número de pares de altavoces que se pueden formar de manera que ningún coeficiente de emparejamiento sea superior a 9. Emparejar el máximo número de altavoces de manera que la suma de los coeficientes de acoplamiento de las parejas formadas sea mínima. (*Nota*. El grafo no es bipartido, utilizar el programa *Netsolve*)

16. Un conjunto independiente en una matriz nxn es un conjunto de n componentes de la matriz tal que no hay dos de ellos en la misma fila ni en la misma columna. El peso de un conjunto independiente es la suma de las componentes que hay en él. Hallar un conjunto independiente de mínimo peso de la matriz:

4	5	8	10	11
7	6	5	7	4
8	5	12	9	6
6	6	13	10	7
4	5	7	9	8

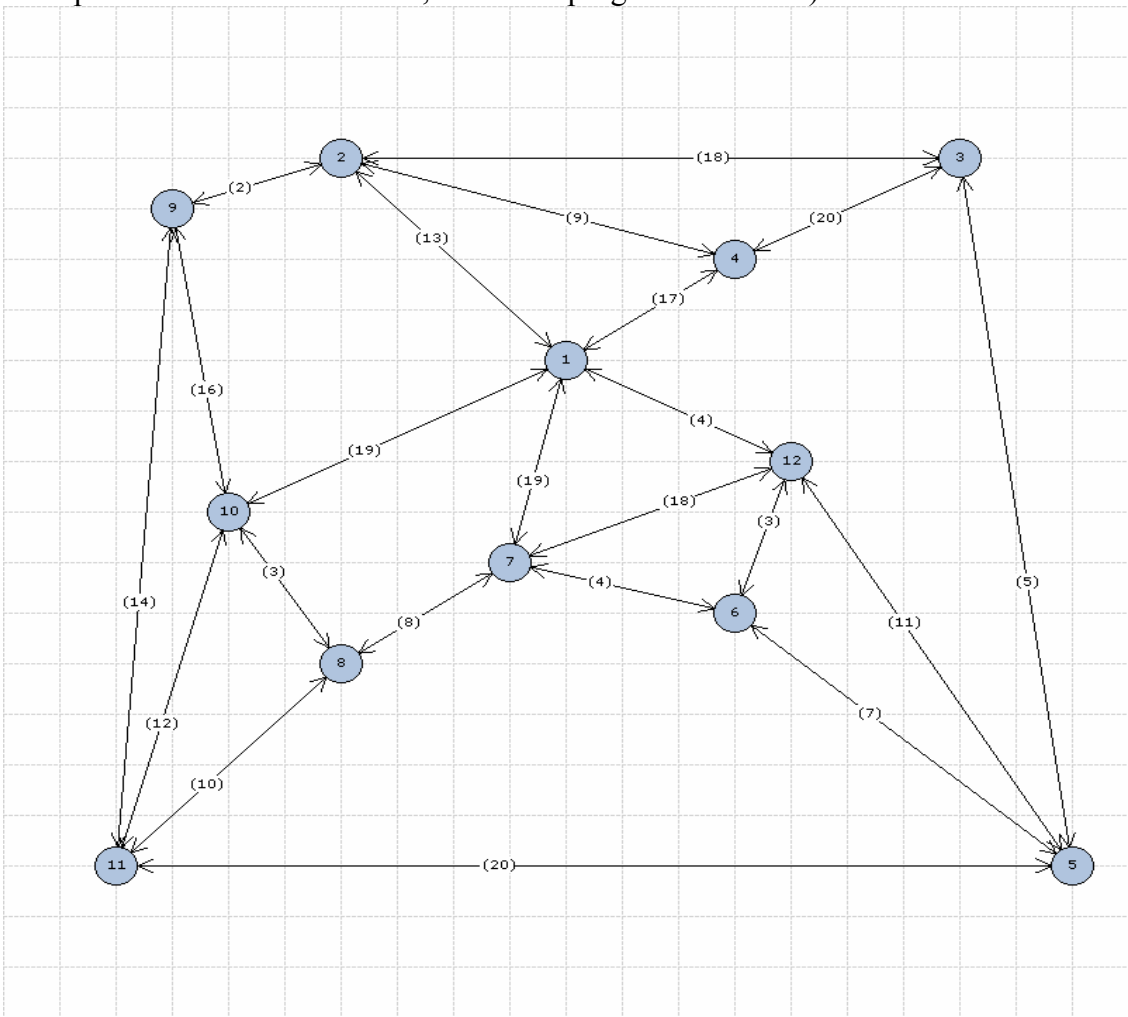
17. Un primer ministro quiere reorganizar su consejo de ministros el cual consta de 5 ministros que ocupan 5 carteras. El conoce las preferencias de cada ministro por cada cartera, que pueden ser expresadas por una puntuación del 1 al 5, donde 1 indica la máxima preferencia y 5, la mínima. La tabla de preferencias es la siguiente:

	c 1	*c* 2	*c* 3	*c* 4	*c* 5
m 1	3	2	1	4	5
m 2	2	5	4	1	3
m 3	1	3	2	5	4
m 4	5	4	1	3	2
m 5	3	5	2	1	4

En la situación actual el ministro i ocupa la cartera i, i=1,...,5. ¿Es posible reorganizar el consejo de ministros de manera que cada ministro ocupe una cartera que prefiera más que la que ocupa

actualmente? ¿Cuál sería la reorganización del consejo de ministros que produciría una mayor satisfacción global?

18. Resolver el Problema del Cartero Chino definido en por el grafo de la figura. (*Nota*. Para resolver el problema de acoplamiento de coste mínimo, utilizar el programa *Netsolve*)



Tema 5 Flujo máximo. Conectividad

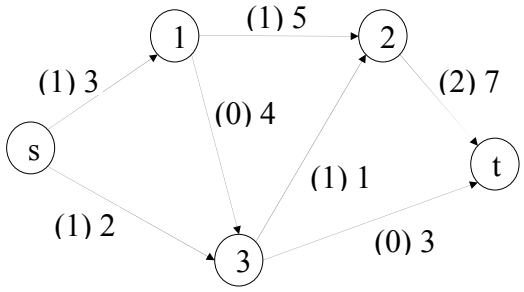
5.1 Problemas de flujos. El Problema del flujo máximo

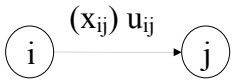
(En este tema trabajaremos con grafos dirigidos, excepto en el apartado 5.8)

Sea un grafo dirigido G = (V, A) con:

- *capacidades* $u_{ij} \geq 0$, definidas para todo arco $(i,j) \in A$,
- dos vértices especiales: s, *fuente*, y t, *sumidero*.

El *Problema del flujo máximo* consiste en enviar la máxima cantidad de flujo posible desde la fuente s al sumidero t, teniendo en cuenta que el flujo que atraviesa cada arco no puede ser mayor que su capacidad y que el flujo se conserva en todos los vértices que atraviesa, esto es, el flujo que entra en un vértice es igual al flujo que sale, excepto en los vértices s y t.





x_{ij} : flujo por el arco (i,j)

Aplicaciones directas: redes de conducciones, suministro, telecomunicaciones, etc.

Notación: $\delta^+(i) = \{(i,j): (i,j) \in A\}$, $\delta^-(i) = \{(j,i): (j,i) \in A\}$.

Más formalmente, un flujo es un vector $x = [x_{ij}]_{(i,j)\in A}$ y el Problema del flujo máximo consiste en encontrar un flujo que sea una solución óptima del problema:

Max v

s. a.

$$\textbf{(5.1)} \qquad \sum_{(i,j)\in\delta^+(i)} x_{ij} - \sum_{(j,i)\in\delta^-(i)} x_{ji} = \begin{cases} v & \text{si} \quad i = s \\ 0 & \text{si} \quad i \neq s,t \\ -v & \text{si} \quad i = t \end{cases} \qquad \forall i \in V$$

$$\textbf{(5.2)} \qquad 0 \leq x_{ij} \leq u_{ij} \qquad \forall (i,j) \in A$$

Se dice que $x = [x_{ij}]_{(i,j)\in A}$ es un *flujo posible* si cumple (5.1) y (5.2) y v es el *valor del flujo*. El primer miembro de la restricción (5.1) se llama *flujo neto* en i, esto es,

flujo neto en i = flujo saliente de i - flujo entrante en i

Notar que el flujo nulo, x = 0, es siempre un flujo posible, cualquiera que sea la red G. un problema más general es aquel en el que existe una cota inferior para la cantidad de flujo que pasa por cada arco. En este caso, las restricciones (5.2) se escribirían:

$$l_{ij} \le x_{ij} \le u_{ij} \quad \forall (i,j) \in A$$

Esta generalización se puede resolver con métodos similares a los que vamos a estudiar, aunque, en este caso, el flujo nulo no es siempre posible, por lo que encontrar un flujo posible no es trivial.

5.2 Un algoritmo para el problema del flujo máximo

Recordar que un camino débil en un grafo dirigido es un camino en el que no se tiene en cuenta la dirección de los arcos.

Dado un camino débil de w a v: $v_0 = w$, a_1, v_1, a_2, v_2, ... , a_k, $v_k = v$, un arco a_i es:

- *hacia adelante*, si $a_i = (v_{i-1}, v_i)$,
- *hacia atrás*, si $a_i = (v_i, v_{i-1})$.

Dado un camino débil P, de w a v, P denotará también el conjunto de arcos del camino además:

- P^+ denota el conjunto de arcos hacia adelante de P,
- P^- denota el conjunto de arcos hacia atrás de P,

En el algoritmo que vamos a estudiar, se parte de un flujo inicial que va aumentando sucesivamente de valor gracias a la utilización de caminos aumentados.

Dado un flujo x, P es un *camino aumentado* de w a v, para x, si:

a) $x_{ij} < u_{ij}$, para $(i,j) \in P^+$,

b) $x_{ij} > 0$, para $(i,j) \in P^-$.

La capacidad residual del arco (i,j) se define como:

- $r_{ij} = u_{ij} - x_{ij} > 0$, si $(i,j) \in P^+$,
- $r_{ij} = x_{ij} > 0$, si $(i,j) \in P^-$.

la capacidad residual del camino P se define como $\delta = \min \{r_{ij}: (i,j) \in P\} > 0$.

Teorema 5.1

Dado un flujo x de s a t con valor v y un camino aumentado P, de s a t para x, con capacidad residual δ, es posible construir un flujo x' de s a t con valor v + δ.

Es evidente pues que si existe un camino aumentado para x de s a t, entonces x no es un flujo máximo. Vamos a ver que también se cumple al contrario: si un flujo no es máximo, entonces existe un camino aumentado.

Dado un conjunto de vértices S, tal que s∈S y t∉S, llamamos *cortadura s-t* asociada a S al conjunto de arcos con un extremo en S y el otro en su complementario. Este conjunto se denota: $[S,\overline{S}]$

$$[S,\overline{S}]=\{(i,j)\in A: i\in S, j\in \overline{S}, \text{o, } j\in S, i\in \overline{S}\}$$

Los arcos (i,j)∈ $[S,\overline{S}]$, tales que $i\in S, j\in \overline{S}$, se llaman *arcos hacia adelante* de la cortadura, mientras que los arcos (i,j)∈ $[S,\overline{S}]$, tales que $j\in S, i\in \overline{S}$, se llaman *arcos hacia atrás*.

Denotaremos $(S,\overline{S})=\{(i,j)\in A: i\in S, j\in \overline{S}\}$, luego $[S,\overline{S}]=(S,\overline{S})\cup(\overline{S},S)$.

La *capacidad de una cortadura*, $u[S,\overline{S}]$, se define como $u[S,\overline{S}]=\sum_{(i,j)\in(S,\overline{S})}u_{ij}$. Notar que en el cálculo de la capacidad de la cortadura sólo intervienen los arcos hacia adelante.

Teorema 5.2

Sea x un flujo de s a t con valor v, entonces: $v=\sum_{(i,j)\in(S,\overline{S})}x_{ij}-\sum_{(i,j)\in(\overline{S},S)}x_{ij}$

Teorema 5.3

El valor de cualquier flujo de s a t es menor o igual que la capacidad de cualquier cortadura s-t.

Una *cortadura s-t* se dice que es *mínima* si tiene mínima capacidad entre todas las cortaduras s-t.

Teorema 5.4

Sea x un flujo de s a t con valor v y sea $[S,\overline{S}]$ *una cortadura s-t tal que* $u[S,\overline{S}]=v$*, entonces x es un flujo máximo de s a t y* $[S,\overline{S}]$ *es una cortadura s-t mínima.*

Teorema 5.5

Un flujo x de s a t es un flujo máximo de s a t sii no existe ningún camino aumentado para x de s a t.

Teorema 5.6 (Teorema de flujo máximo - cortadura mínima)

Si en un grafo dirigido con capacidades, existe el flujo máximo de s a t, su valor es igual a la capacidad de la cortadura s-t mínima.

Algoritmo de Flujo máximo (Ford&Fulkerson (1956))

Inicio. Construir un flujo inicial x, de s a t (por ejemplo x := 0).

Paso 1. Buscar un camino aumentado para x de s a t. Sea P este camino, si no existe, PARAR, x es un flujo máximo.

Paso 2. Sea $\delta := \min \{r_{ij}: (i,j)\in P\} > 0$.

Hacer para cada arco $(i,j)\in P$

- si $(i,j)\in P^+$, $x_{ij} := x_{ij} + \delta$,
- si $(i,j)\in P^-$, $x_{ij} := x_{ij} - \delta$,

FinHacer

Volver al *Paso 1*.

Complejidad del algoritmo de flujo máximo

Si las capacidades son racionales, podemos convertirlas en enteras multiplicándolas por un número apropiado y la solución no cambia (aunque el valor del flujo máximo quedaría multiplicado por el mismo número). Supongamos que las capacidades son enteras; entonces, en cada iteración (Pasos 1 y 2) el valor del flujo aumenta al menos en una unidad. El valor del flujo máximo está acotado superiormente por la capacidad de cualquier cortadura s-t, por ejemplo $u[\{s\}, V-\{s\}] \le (n-1)U$, siendo $U = \max\{u_{ij} : (i,j)\in A\}$. Por lo tanto el número de iteraciones será, como máximo, (n-1)U. Veremos más adelante que existe un algoritmo para buscar un camino aumentado en el Paso 1 cuya complejidad es O(m), por lo tanto, la complejidad del algoritmo de Ford&Fulkerson es O(mnU).

Notar que el algoritmo no es polinómico dado que para representar U hacen falta log U dígitos y U es una función exponencial de log U ($U = 10^{\log U}$). Además, si las capacidades son números irracionales, no está garantizada la finitud del algoritmo. Sin embargo, si en el Paso 1 se busca un camino aumentado de s a t con un número mínimo de arcos, entonces, se puede demostrar que el número de iteraciones es polinómico, independientemente de cómo sean las capacidades.

Un *flujo* x se dice que es entero si x_{ij} es *entero* para todo arco $(i,j)\in A$.

Teorema 5.7 (Propiedad de integridad)

La formulación del problema de Flujo máximo como un problema de Programación Lineal (PL) tiene la propiedad de integridad, i.e. si las capacidades son enteras, existe un flujo máximo entero.

5.3 Algoritmo BFS para la búsqueda de un camino aumentado

Este algoritmo explora la totalidad del grafo empezando en el vértice s y utilizando la estrategia FIFO (First In First Out), esto es, los vértices son explorados en el mismo orden en que son encontrados. Se utiliza una Lista de vértices que han sido alcanzados y no explorados (inicialmente contiene sólo a s); en cada iteración se selecciona el primer vértice de la lista, se elimina de la lista y se exploran todos los arcos "usables" incidentes con ese vértice. Para cada uno de estos arcos, si el otro extremo es un vértice no alcanzado, se añade al final de la Lista. El algoritmo acaba cuando el vértice t es alcanzado por primera vez, o bien, la Lista se queda vacía antes de que esto ocurra, en cuyo caso la conclusión es que no existe ningún camino aumentado de s a t.

Algoritmo BFS para el camino aumentado

Inicio Hacer

- Lista := (s), l(s) := 0, l(v) := ∞ $\forall v \in V-\{s\}$
- pred(v) := 0, cap(v) := ∞, $\forall v \in V-\{s\}$,
- i := 1.

Mientras Lista $\neq \varnothing$ y l(t) = ∞

- Seleccionar el primer vértice de Lista, sea v,
- *Para Todos* los arcos usables a incidentes con v
    - Sea w el otro extremo del arco a. *Si* l(w) = ∞ hacer:
        - pred(w) := v, l(w) := i, cap(w) := min{cap(v), r_a}
        - Lista := Lista + w (añadirlo al final)
        - i := i+1

    FinSi

    FinParaTodos
- Eliminar v de Lista

FinMientras

PARAR.

- Si l(t) < ∞, existe camino aumentado de s a t.
- En otro caso, $\left[T, \overline{T}\right]$ es una cortadura mínima, siendo T = {i: l(i) < ∞}.

Supongamos que se están explorando los arcos incidentes con el vértice v. Sea a uno de estos arcos; si el otro extremo del arco a ya había sido alcanzado, se pasa al siguiente arco, en otro caso, sea w el otro extremo, entonces se dice que el arco a es *usable*:

a) si a = (v,w) (arco hacia adelante) y $x_{vw} < u_{vw}$,

b) si a = (w,v) (arco hacia atrás) y $x_{wv} > 0$.

En el caso (a), se define $r_a = u_{vw} - x_{vw}$, y en el caso (b) $r_a = x_{vw}$. Si el arco a es usable, el otro extremo, w, es etiquetado *alcanzado* y es introducido al final de la Lista. En resumen, un vértice w es etiquetado alcanzado sii existe un camino aumentado desde s a w.

La etiqueta pred(i) permite obtener el camino aumentado de s a t en caso de que t sea alcanzado. En caso contrario, S = {i∈V: etiqueta(i) ≠ ∅} determina una cortadura $\left[S, \bar{S}\right]$ mínima.

Es fácil demostrar que este algoritmo:

- tiene complejidad *O*(m),
- el camino aumentado que proporciona, en caso de que exista, es mínimo (tiene un número mínimo de arcos).

Teorema 5.8

Si en cada iteración del algoritmo de flujo máximo se utiliza un camino aumentado mínimo de s a t para modificar el flujo, entonces el número de iteraciones es, como máximo, mn/2.

Teorema 5.9

*El algoritmo de flujo máximo que utiliza el algoritmo BFS para buscar el camino aumentado en cada iteración (*Edmonds&Karp (1972)*), tiene complejidad $O(m^2n)$; por lo tanto es una algoritmo polinómico.*

Otros algoritmos para el problema del flujo máximo:

Karzanov (1974), complejidad $O(n^2m)$; Cheriyan & Maheswari (1989), complejidad $O\left(n^2\sqrt{m}\right)$.

5.4 Descomposición de un flujo en caminos

Dado un flujo x de s a t, se trata de encontrar un conjunto de caminos s-t y un flujo para cada uno de ellos, de forma que la suma de los flujos de estos caminos sea igual al valor del flujo x, y el flujo que atraviesa cada arco (i,j), en el conjunto de caminos, sea también igual a x_{ij}.

Algoritmo de descomposición en caminos

Inicio. Flujo de s a t $x = [x_{ij}]_{(i,j)\in E}$ con valor v. Hacer k := 1.

Mientras $v > 0$,

1) Buscar un camino P_k de s a t que utilice arcos (i,j) con $x_{ij} > 0$. Sea $v(P_k) := \min \{ x_{ij}: (i,j)\in P_k \}$.
2) Para cada arco $(i,j)\in P_k$, hacer $x_{ij} := x_{ij} - v(P_k)$.
3) Escribir P_k y $v(P_k)$, hacer $v := v - v(P_k)$ y k := k+1.

FinMientras

El algoritmo que busca un camino de s a t en (1) es una variante del algoritmo BFS en el que los arcos "usables" son los arcos hacia adelante con flujo positivo. Notar que la transformación del flujo en (2) resta la misma cantidad de flujo entrante y saliente para cada vértice, por lo que la conservación de flujo en cada vértice sigue cumpliéndose. Además, por lo menos un arco pasa de tener un flujo positivo a tener flujo cero, por lo que el número de iteraciones será, como máximo, m. Por lo tanto, este algoritmo es $O(m^2)$.

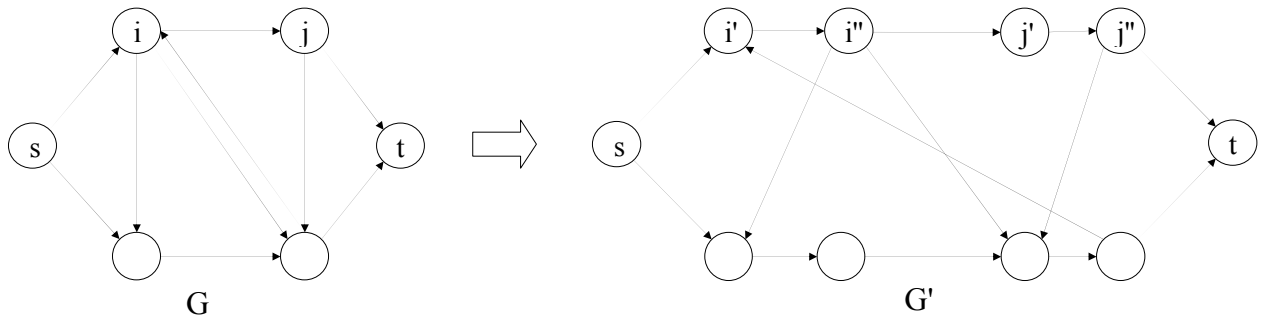
5.5 Flujos con capacidades en los vértices

En algunos casos, además de la limitación en cuanto al flujo que puede atravesar un arco, se limita también la cantidad de flujo que puede atravesar un vértice. Para todo $v\in V$, sea c_v la máxima cantidad de flujo que puede atravesar el vértice v. La siguiente transformación permite reducir el problema al problema de flujo máximo con capacidades sólo en los arcos.

Dado el grafo G, construimos el grafo G' de la siguiente forma:

- Partir cada vértice $i\in V-\{s,t\}$ en dos vértices i' y i" y añadir el arco (i',i").
- Reemplazar cada arco $(s,j)\in E$, por el arco (s,j') y reemplazar cada arco (j,t) por el arco (j",t).
- Reemplazar el resto de arcos de G, $(i,j)\in E$ con $i,j\in V-\{s,t\}$, por los arcos (i",j').

Ejemplo:



Notar que, en G', un camino que entra en el vértice i', tiene que usar necesariamente el arco (i',i") y seguir por algún arco, sea (i",j'), que corresponde a un arco (i,j) de G. Por lo tanto, es fácil ver que existe una

correspondencia biunívoca entre los caminos de s a t en G y los caminos de s a t en G'. Además, si un camino pasa por el vértice i en G, el correspondiente camino en G' pasa por el arco (i',i''). Por lo tanto basta poner una capacidad c_i a cada arco (i',i'') de G', y calcular el flujo máximo de s a t en G'. El flujo correspondiente en G, cumplirá la condición de que, por cada vértice $i \in V$, no pase una cantidad de flujo mayor que c_i.

5.6 Flujos en grafos no dirigidos

Para calcular un flujo máximo en un grafo dirigido, basta sustituir cada arista (i,j) por un par de arcos opuestos: (i,j) y (j,i), ambos con la misma capacidad. Existe una correspondencia biunívoca entre los flujos en un grafo no dirigido y los flujos no solapados (que se definen a continuación) en el grafo dirigido que resulta de sustituir cada arista por dos arcos opuestos.

Un *flujo solapado* es aquel para el que, para algún par de vértices i y j, existe un flujo positivo de i a j y también de j a i, esto es: $x_{ij}>0$ y $x_{ji}>0$. Dado un flujo solapado x, siempre es posible construir un flujo no solapado equivalente (con el mismo valor) haciendo, para cada par de vértices i, j para los que existen dos arcos opuestos, $x_{ij} := x_{ij} - \delta_{ij}$, y , $x_{ji} := x_{ji} - \delta_{ij}$, siendo $\delta_{ij} = \min\{ x_{ij}, x_{ji} \}$.

5.7 El Problema del Flujo de coste mínimo

Sea un grafo dirigido G = (V, A), en el que cada arco (i,j) tiene asociado un *coste* c_{ij} y una *capacidad* u_{ij}; y cada vértice $i \in V$ tiene asociada una demanda b_i, de forma que $\sum_{i \in V} b_i = 0$. El problema del flujo de coste mínimo consiste en encontrar un flujo en el que el flujo neto en cada vértice i sea igual a b_i, el flujo por cada arco (i,j) cumpla $x_{ij} \le u_{ij}$, y tenga un coste total mínimo. Se puede formular como un problema de Programación Lineal:

$$\text{Min} \sum_{(i,j) \in E} c_{ij} x_{ij}$$

s. a.

$$\sum_{(i,j) \in \delta^+(i)} x_{ij} - \sum_{(j,i) \in \delta^-(i)} x_{ji} = b_i \qquad \forall i \in V$$

$$0 \le x_{ij} \le u_{ij} \qquad \forall (i,j) \in A$$

Este problema incluye como caso particular al problema del flujo máximo y tiene muchas aplicaciones. Se puede resolver con algoritmos polinómicos muy eficientes.

5.8 Determinación de la conectividad de un grafo

Utilizando el algoritmo BFS es posible determinar si existe un camino desde un vértice s al resto de vértices; basta considerar como arco "usable" los arcos hacia adelante y hacer que el algoritmo acabe cuando Lista = ∅. Una vez finalizado el algoritmo, para cada vértice i, se cumple que etiqueta(i) ≠ ∅, sii

existe un camino de s a i en el grafo. Si aplicamos el algoritmo n veces tomando cada vez como vértice s un vértice distinto, podemos determinar los pares de vértices que están conectados y, por lo tanto, las componentes fuertemente conexas del grafo. El caso de un grafo no dirigido, se reduce al caso dirigido sustituyendo cada arista por dos arcos opuestos (existen forma más eficientes de calcular las componentes de un grafo, tanto en el caso dirigido como en el caso no dirigido).

Si un grafo es conexo, puede estudiarse el "grado de conectividad" de distintas formas. Estudiaremos estos conceptos sólo para los grafos no dirigidos (en el caso dirigido existen definiciones similares).

Sea G = (V, E) un grafo no dirigido; por simplicidad, supondremos que no tiene bucles.

Conectividad por vértices

Una *cortadura de vértices* es un subconjunto $V' \subseteq V$, $V' \neq V$, tal que G-V' es desconexo. Si $|V'|=k$, se llama k-cortadura de vértices. Si $v \in V$ y $\omega(G-v) > \omega(G)$, se dice que v es un *vértice de corte*. Se define la *conectividad* de un grafo G, no completo, como el mínimo k para el que G tiene una k-cortadura de vértices; se denota por K(G). La conectividad de un grafo completo (o que tenga como subgrafo generador un grafo completo), se define como n-1. Notar que si G es trivial o desconexo, K(G) = 0.

Un grafo G se dice k-*conexo* si $K(G) \geq k$, lo que equivale a decir que quitando menos de k vértices, el grafo no se desconecta.

Conectividad por aristas

Se define la *aristoconectividad* de un grafo G, K'(G), como el mínimo k para el que G tiene una k-cortadura de aristas (esto es, una cortadura de aristas con k aristas). Si G es trivial, se define K'(G) = 0. Se dice que un grafo G es k-aristoconexo si $K'(G) \geq k$.

Teorema 5.10

Para cualquier grafo G, se cumple que $K(G) \leq K'(G) \leq \delta(G)$.

Diremos que dos caminos son *aristodisjuntos* si no tienen ninguna arista en común y diremos que son *internamente disjuntos* si no tienen ningún vértice interno en común (se exceptúan los vértices extremos del camino).

Dado un grafo no dirigido y conexo G, asignamos *a cada arista una capacidad igual a* 1. Es evidente que el valor del flujo máximo entre dos vértices i y j en este grafo es igual al número de caminos aristodisjuntos entre i y j en G y, al mismo tiempo, por el teorema de flujo máximo - cortadura mínima, este número es igual al mínima cortadura de aristas cuya eliminación destruye todos los caminos entre i y j.

Si calculamos el flujo máximo entre un vértice fijo i y el resto de vértices del grafo, aplicando n-1 veces el algoritmo de flujo máximo, la cortadura con menos aristas encontrada es la mínima cortadura de aristas de G y el número de aristas que contiene es igual a K'(G).

La conectividad por vértices puede determinarse también utilizando el algoritmo de flujo máximo. Para ello basta asignar a cada vértice una capacidad igual a 1 (ver sección 5.5): el valor del flujo máximo entre dos vértices i y j será igual al número de caminos internamente disjuntos entre i y j, y también será igual al mínimo número de vértices cuya eliminación destruye todos los camino de i a j. Similarmente, aplicando (n-1)n/2 veces (para todos los pares de vértices) el algoritmo de flujo máximo a este grafo, podemos calcular la conectividad de G, K(G).

Utilizando los mismos razonamientos se puede demostrar fácilmente el siguiente teorema.

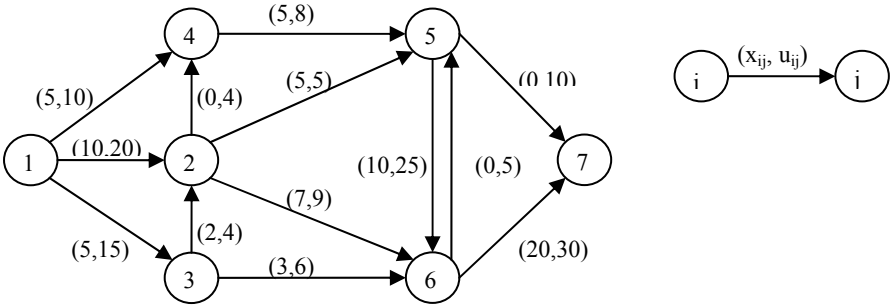
Teorema 5.11 (Menger)

a) Un grafo es k-aristoconexo sii entre todo par de vértices existen al menos k caminos aristodisjuntos.

b) Un grafo con $n \geq k+1$ vértices es k-conexo sii entre todo par de vértices no adyacentes existen al menos k caminos internamente disjuntos.

5.9 Ejercicios

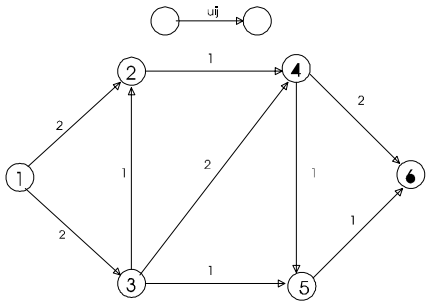
2. Varias familias van a participar en un recorrido turístico por las Montañas Nevadas en vehículos compartidos. Para minimizar las posibilidades de disputas, se quiere asignar los individuos a coches de forma que no haya dos miembros de una familia en el mismo coche. Formular este problema como un problema de flujos en redes.
3. Considerar la red de la figura con el flujo posible dado.



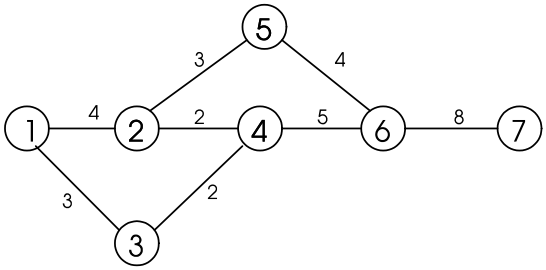
(a) Especificar cuatro cortaduras 1-7 en la red, cada una conteniendo cuatro arcos hacia adelante. Listar la capacidad y el flujo a través de cada una de estas cortaduras.

(b) Calcular el flujo máximo de 1 a 7. Calcular la cortadura 1-7 mínima. Descomponer el flujo máximo en caminos 1-7.

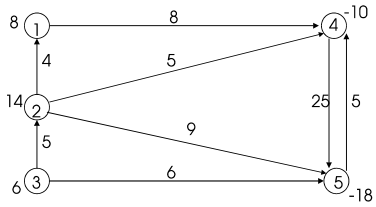
4. Resolver el problema de flujo máximo de 1 a 6 en el grafo de la figura. Descomponer el flujo máximo en flujos a través de caminos 1-6. Especificar una cortadura 1-6 mínima.



5. Calcular el flujo máximo de 1 a 7 en el grafo no dirigido de la figura. Especificar la cortadura 1-7 mínima en este grafo.



6. Mostrar cómo se puede transformar un problema de flujo máximo con varias fuentes y varios sumideros en el que hay que maximizar el flujo total saliente de todas las fuentes, en uno con una sola fuente y un sólo sumidero.
7. Demostrar que si añadimos cualquier número de arcos entrantes, con cualquier capacidad, al nudo fuente, el valor del flujo máximo no cambia. Similarmente, si añadimos cualquier número de arcos salientes, con cualquier capacidad, al nudo sumidero, el valor del flujo máximo no cambia.
8. Suponer que se quiere resolver un problema de flujo máximo que contiene arcos en paralelo, pero el programa del que disponemos para resolverlo no admite esta posibilidad. ¿Cómo podríamos usar este programa para resolver nuestro problema?
9. Flujo posible: Los nudos del grafo de la figura representan puertos que tienen una determinada oferta/demanda de cierta mercancía, representada al lado de cada nudo. Los arcos representan posibles envíos de mercancía con una determinada capacidad que se ha representado al lado de cada arco. Determinar si es posible satisfacer las ofertas y demandas dadas resolviendo un problema de flujo máximo.



10. Programación distribuida: Un programa tiene 4 módulos que pueden ejecutarse en dos procesadores distintos PA y PB. Los costes de ejecución de estos módulos en cada procesador vienen dados por la tabla (a). Por otro lado, si dos módulos i,j se ejecutan en procesadores distintos, se incurre en un coste de comunicación; estos costes se representan, para cada par de módulos en la tabla (b). Determinar en qué procesador se debe ejecutar cada módulo para minimizar el coste total formulando un problema de cortadura mínima.

	1	**2**	**3**	**4**
PA	6	5	19	4
PB	4	10	3	8

(a)

	1	**2**	**3**	**4**
1	0	5	0	0
2	5	0	6	2
3	0	6	0	1
4	0	2	1	0

(b)

11. La tabla siguiente contiene los datos de un problema de secuenciación en máquinas paralelas con cuatro tareas: tiempo de procesamiento (en días), día en que están disponibles, y día límite, para cada

tarea. Suponiendo que se dispone de dos máquinas cada día, resolver el ejemplo con el algoritmo de Ford&Fulkerson.

tarea (j)	**1**	**2**	**3**	**4**
Procesamiento (p_j)	2.5	3.1	5.0	1.8
Disponible (r_j)	1	5	0	2
Límite (d_j)	5	9	8	7

12. Suponer que una red contiene un nudo, distinto de la fuente, sin ningún arco entrante, ¿se puede eliminar ese nudo sin afectar al valor del flujo máximo? Similarmente, ¿se puede eliminar un nudo, distinto del sumidero, que no tenga ningún arco saliente?
13. Un comandante está situado en un nudo p de una red de comunicaciones G y sus subordinados están situados en un conjunto S de nudos de la red. Sea u_{ij} el esfuerzo requerido para destruir el arco (i,j) de la red. El problema es determinar el mínimo esfuerzo requerido para impedir que el comandante se pueda comunicar con cualquiera de sus subordinados. ¿Cómo se puede resolver este problema?
14. En el ejemplo de la Secuenciación de Máquinas en Paralelo, ¿cómo se puede modelizar la situación en que el número de máquinas disponibles es distinto cada día? Aplica el método que propongas al siguiente ejemplo. Hay que realizar 4 tareas; la tabla siguiente proporciona el tiempo de procesamiento (en días), día en que están disponibles, y día límite, para cada tarea. Tres máquinas están disponibles los días 1, 2, 4 y 5, dos máquinas los días 3 y 6, y cuatro máquinas el resto de días.

tarea (j)	**1**	**2**	**3**	**4**
Procesamiento (p_j)	1.5	1.25	2.1	3.6
Disponible (r_j)	3	1	3	5
Límite (d_j)	5	4	7	9

15. Suponer que se conoce un flujo máximo de s a t en una red. Mostrar cómo se puede construir una cortadura s-t mínima con un tiempo adicional O(m).
16. Demostrar que si $x_{ij} = u_{ij}$ para cierto arco (i,j), en todo flujo máximo, entonces este arco debe ser un arco hacia adelante en alguna cortadura mínima.
17. Un departamento universitario con p profesores, F_1, F_2,..., F_p, va a ofrecer p asignaturas, C_1, C_2,..., C_p, en el semestre siguiente y cada profesor va a impartir exactamente un asignatura. Cada profesor solicita las dos asignaturas por las que tiene mayor preferencia. Decimos que una asignación de

cursos es posible si a cada profesor se le asigna una de las dos asignaturas que ha solicitado. ¿Cómo se puede determinar si existe una asignación posible?

18. Un flujo es par si para cada arco $(i,j)\in A$, x_{ij} es par; es impar si para cada arco $(i,j)\in A$, x_{ij} es impar. Para cada una de las afirmaciones siguientes, demostrarla o encontrar un contraejemplo:
    a.- Si las capacidades de todos los arcos son pares, la red tiene un flujo máximo par.
    b.- Si las capacidades de todos los arcos son impares, la red tiene un flujo máximo impar.

19. Para cada una de las afirmaciones siguientes, demostrarla o encontrar un contraejemplo:
    a.- Si x es un flujo máximo, si $(i,j), (j,i)\in A$, entonces se cumple que $x_{ij} = 0$ o $x_{ji} = 0$.
    b.- Para cualquier red existe un flujo máximo x para el cual, si $(i,j), (j,i)\in A$, entonces se cumple que $x_{ij} = 0$, o $x_{ji} = 0$.
    c.- Si multiplicamos la capacidad de cada arco por un número positivo λ, la cortadura mínima no cambia.
    d.- Si sumamos un número positivo λ a la capacidad de cada arco, la cortadura mínima no cambia.

20. Si la capacidad de un arco disminuye en k unidades, ¿en cuánto, como máximo, puede cambiar el valor del flujo máximo? Poner ejemplos en los que se alcance el máximo y el mínimo cambio posible respectivamente.

21. Si la capacidad de un arco aumenta k unidades, ¿en cuánto, como máximo, puede cambiar el valor del flujo máximo? Poner un ejemplo en el que se alcance el máximo cambio posible.

22. Sean $[S,\overline{S}]$ y $[T,\overline{T}]$ dos cortaduras s-t cualesquiera de la red dirigida G. Demostrar que se cumple

$$u[S,\overline{S}] + u[T,\overline{T}] \geq u[S\cup T,\overline{S\cup T}] + u[S\cap T,\overline{S\cap T}].$$

23. Demostrar que si $[S,\overline{S}]$ y $[T,\overline{T}]$ son ambas cortaduras s-t mínimas entonces las cortaduras $[S\cup T,\overline{S\cup T}]$ y $[S\cap T,\overline{S\cap T}]$ son también cortaduras s-t mínimas.

24. Suponer que se conoce un flujo máximo en una red que tiene capacidades enteras. Sugerir un algoritmo para convertir este flujo en un flujo máximo entero. ¿Cuál es la complejidad del algoritmo? (*Sugerencia*: modificar el flujo a lo largo de ciclos débiles).

<u>En los ejercicios siguientes, se supone que los grafos son no dirigidos</u>

25. Sea G un grafo conexo con matriz de adyacencia Y. ¿Qué se puede decir de Y si: a) v_i es un vértice de corte b) (v_i, v_j) es una arista de corte?

26. Demostrar que un grafo conexo simple que tiene exactamente dos vértices que no son vértices de corte es un camino.
27. ¿Cuál es el número máximo de vértices de corte en un grafo con n vértices?.
28. Demostrar que si G es conexo y cada vértice tiene grado par, entonces: $\omega(G - v) \leq d(v)/2$, $\forall v \in V$.
29. Demostrar que si:
    a) cada grado en G es par, entonces G no tiene aristas de corte.
    b) G es un grafo k regular bipartido con $k \geq 2$, entonces G no tiene aristas de corte.
30. Sean G conexo y $S \subset V$ con $S \neq \emptyset$. Demostrar que la cortadura de aristas [S,V-S] es una ligadura de G sii G[S] y G[V-S] son conexos. (Nota: una ligadura es una cortadura de aristas minimal, i. e. no contenida estrictamente en otra cortadura).
31. Demostrar que toda cortadura de aristas es unión disjunta de ligaduras.
32. Sea G un grafo conexo con $n \geq 3$. Demostrar que:
    a) Si G tiene una arista de corte, entonces G tiene un vértice de corte.
    b) El recíproco de a) no es necesariamente cierto.
33. a) Demostrar que si G es k-aristo conexo, k>0, y si A' es un conjunto de k aristas de G, entonces $\omega(G-A') \leq 2$.
    b) Para k>0, hallar un grafo k-conexo G y un conjunto V' de k vértices de G tal que $\omega(G-V') > 2$.
34. Demostrar que si G es k'-aristo conexo (k-conexo), entonces $m \geq k'n/2$ ($m \geq kn/2$).
35. a.- Mostrar que si G es simple y $\delta(G) \geq n - 2$, entonces $K(G) = \delta(G)$.
    b.- Hallar un grafo simple G con $\delta(G) = n-3$ y $K(G) < \delta(G)$.
36. Sea G un grafo conexo sin bucles con $n \geq 3$. Se dice que G es un bloque si no tiene vértices de corte. Demostrar que si G es un bloque con $n \geq 3$, si se subdivide una de sus aristas el grafo resultante es también un bloque. Nota. *Subdividir* una arista (i, j) consiste en sustituirla por las aristas (i,u), (u,j), siendo u un nuevo vértice,
37. Demostrar que las siguientes afirmaciones son equivalentes:
    a) G es un bloque.
    b) Cada par de vértices de G están en un ciclo común.
    c) Cada par de aristas de G están en un ciclo común.
    d) Cada par de vértice y arista están en un ciclo común.
    e) Dados dos vértices y una arista, existe un camino que une los dos vértices y que contiene a la arista.
    f) Dados tres vértices distintos, existe un camino uniendo cualesquiera dos de ellos que contiene al tercero.

g) Dados tres vértices distintos, existe un camino uniendo cualesquiera dos de ellos que no contiene al tercero.

38. Demostrar que si v es un vértice de corte de un grafo sin bucles G, entonces v no es un vértice de corte de G^C.

39. Demostrar que un vértice v de un grafo sin bucles G es un vértice de corte sii existen dos vértices u, w adyacentes a v tales que v está en cada uw-camino.

40. Comprobar si es cierta o falsa la siguiente afirmación:

Un grafo conexo sin bucles con $n \geq 3$ es un bloque sii dados dos vértices y una arista existe un camino que une los dos vértices y que no contiene a la arista.

41. Demostrar que: Un grafo conexo sin bucles con al menos 2 aristas es un bloque sii cualquier dos aristas adyacentes están sobre un ciclo.

42. Sea G conexo con $n \geq 3$. Las siguientes afirmaciones son equivalentes:

a) G no tiene aristas de corte.
b) Cada par de vértices están sobre un ciclo.
c) Cada par de vértice y arista están sobre un ciclo.
d) Cada par de aristas están sobre un ciclo.
e) Dados un par de vértices y una arista, existe un camino uniendo los vértices que contiene a la arista.
f) Dados un par de vértices y una arista, existe un camino uniendo los vértices que no contiene a la arista.
g) Dados tres vértices existe una cadena que une cualquier par de ellos y que contiene al tercero.

43. Dar un ejemplo para mostrar que si P es un u-v camino en un grafo 2-conexo G, entonces G no contiene necesariamente un u-v camino Q internamente disjunto con P.

44. Demostrar que si G es r regular y $K(G) = 1$, entonces $K'(G) \leq \lfloor r/2 \rfloor$.

45. Demostrar que si G es conexo, entonces $K(G) = 1 + \min\{K(G-v);\ v \in V\}$.