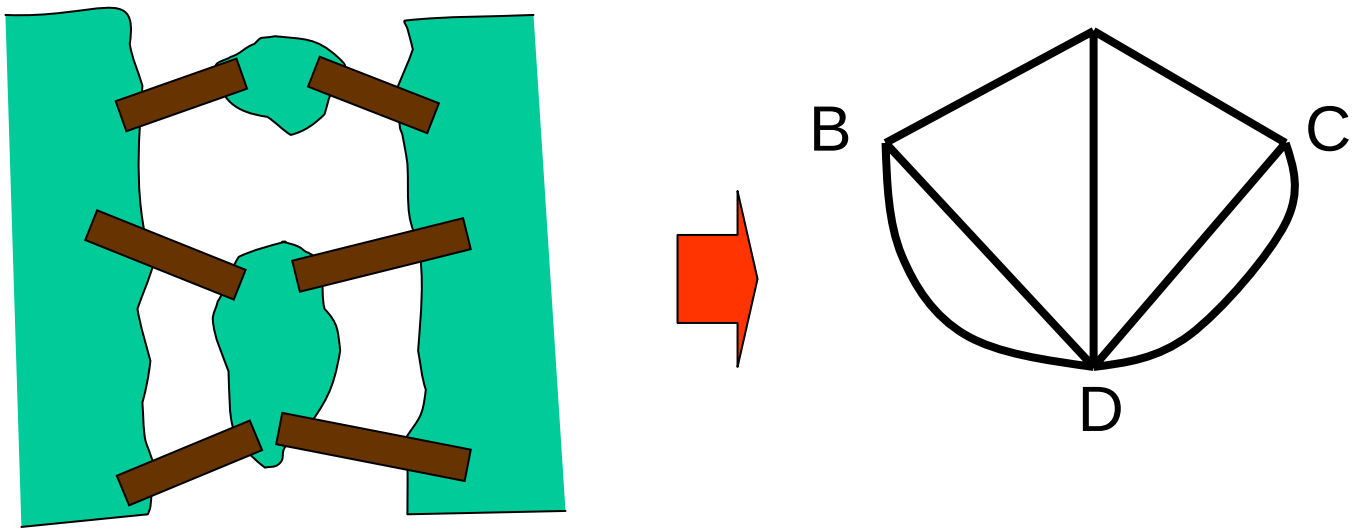
# Teoría de Grafos

# Introducción

- Königsberg, s.XVIII




- Euler resuelve este problema mediante la teoría de grafos: sólo puede haber un ciclo euleriano cuando todos los nodos tienen un número par de aristas incidentes

# Interés de la teoría de grafos

- Sirve para resolver problemas que se puedan representar mediante una red o un grafo
  - camino mínimo
  - inventarios
  - etc
- Proporciona algoritmos más eficientes que los que hemos visto hasta ahora

# Algoritmos eficientes

- La eficiencia se puede medir en función de:
  - la capacidad de almacenamiento requerida
  - el tiempo de ejecución: complejidad
- La complejidad depende del número de operaciones elementales f(n) y de la dimensión del problema n

$$\theta(g(n)) \text{ si } \lim_{n\to\infty} \frac{f(n)}{g(n)} = cte$$

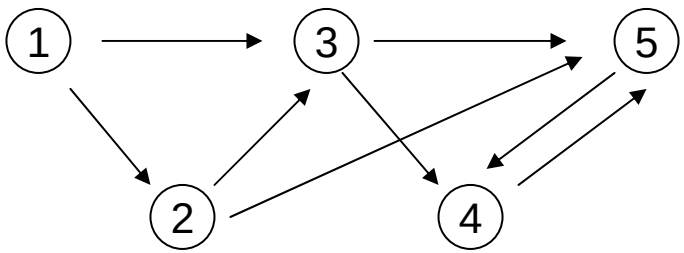
- Criterio del tiempo máximo o peor de los casos
- Tipos de algoritmo
  - polinomial: $\theta(n^p)$
  - no polinomial: $\theta(m^n)$

# Definiciones básicas (I)

**Grafo**: G(V, A) conjunto de nodos unidos por:

arcos: grafo dirigido

aristas: grafo no dirigido




**Arista**: arco sin orientación

**Bucle**: arco que empieza y acaba en el mismo punto

**Multigrafo**: grafo en el que un par de vértices están unidos por más de una arista

# Definiciones básicas (II)

**Cadena**: secuencia de aristas que une un par de nodos

**Camino**: cadena con la misma orientación en todos los arcos

**Ciclo**: cadena que une un nodo con él mismo

**Circuito**: ciclo dirigido (ciclo y camino)

**Grafo simple**: no multigrafo, sin bucles

**Grafo conexo**: si cualquier par de vértices está unido al menos por una cadena

**Árbol**: grafo conexo sin ciclos

**Árbol generador o de extensión** (spanning tree): árbol que incluye a todos los vértices del grafo

# Representación de grafos

- Matriz de incidencia de un grafo $B(b_{ik})$
  - $b_{ik}$ = 1 si el nodo i pertenece a la arista k
  - $b_{ik}$ = 0 en otro caso
- Matriz de incidencia de un grafo dirigido $B(b_{ik})$
  - $b_{ik}$ = +1 si el arco k acaba en el nodo i
  - $b_{ik}$ = -1 si el arco k empieza en el nodo i
  - $b_{ik}$ = 0 en otro caso
- Matriz de adyacencia $A(a_{ij})$ (simétrica si no dirigido)
  - $a_{ij}$ = 1 si existe arco o arista de i a j
  - $a_{ij}$ = 0 en otro caso

# Árbol de extensión mínima

Dado un grafo G=(V, E) con pesos en las aristas, se pretende obtener un árbol que conecte a todos los vértices y la suma de los pesos sea mínima

Algoritmos ávidos, avaros (greedy):

- en cada etapa escogen la elección óptima para esa etapa
- en general son heurísticos
- en este caso la solución es óptima

# Algoritmo de Prim

$C_k$: conjunto de nodos conectados en la iteración k

$\overline{C}_k$: nodos no conectados aún

n: nodos

T: árbol

PASO 1:

$C_0=\emptyset$, $\overline{C}_0=V$. Se toma un vértice cualquiera $i \in C_0$

$C_1=\{i\}$, $\overline{C}_1=V-\{i\}$, k=2, $T=\emptyset$

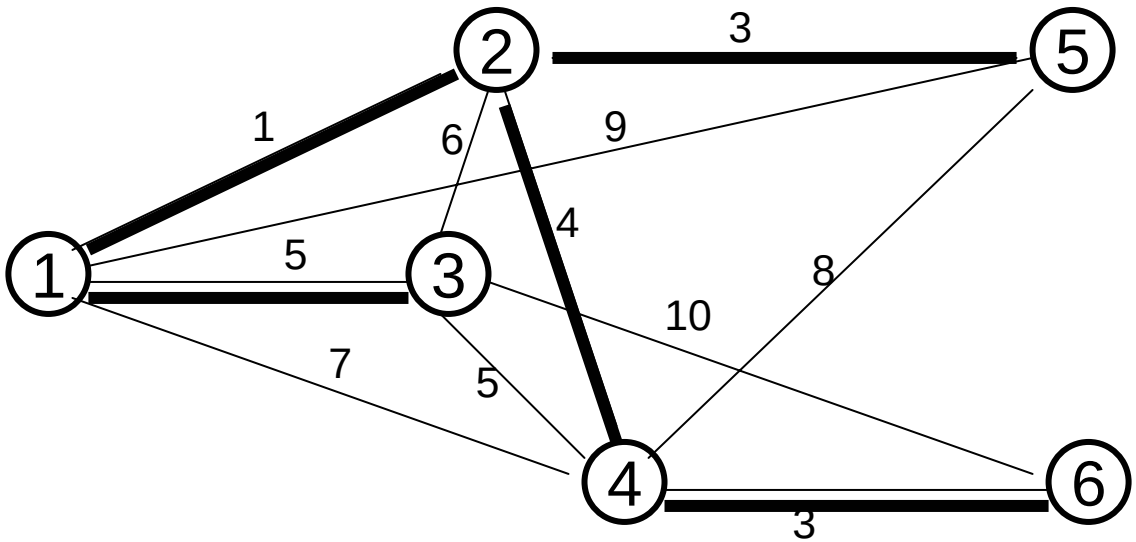
PASO 2:

Seleccionar $j \in \overline{C}_{k-1}$ que sea el que se une a algún vértice de $C_{k-1}$ con la arista de menor peso $e_{k-1}$

$C_k=C_{k-1} \cup \{j\}$, $\overline{C}_k=\overline{C}_{k-1}-\{j\}$, $T=T \cup \{e_{k-1}\}$

Si k=n, parar. Si no, k=k+1 y repetir paso 2.

# Ejemplo nº1



# Problemas de camino mínimo

Encontrar un camino de longitud mínima de un origen a un destino, sobre un grafo dirigido

Son problemas muy importantes:

- en sí mismos
- subrutinas

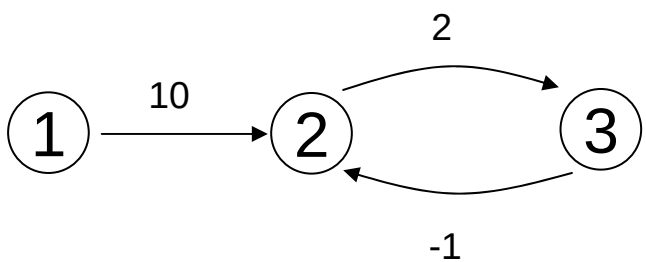
Circuitos con longitud negativa: no tienen solución, suponemos que no existen

# Ecuaciones de Bellman

$$u_1 = 0$$

$$u_j = \min_{k \neq j} \{u_k + d_{kj}\},\ j = 2..n$$

$u_j$: etiquetas que, una vez hechas permanentes, son la longitud del camino mínimo de 1 a j




Hacen falta algoritmos más eficientes para resolver las ecuaciones de Bellman

# Algoritmo de Dijkstra

Válido cuando las longitudes de los arcos son positivas

P: nodos etiquetados de forma permanente
T: nodos etiquetados de forma transitoria
$u_j$: longitud camino mínimo
pred(j): predecesor del vértice j

PASO 0

$u_1=0$, $u_j=d_{1j}$ ($\infty$ si no existe arco), P={1}, T={2,...,n}, pred(j)= 1

PASO 1 (Designar etiqueta permanente)

Buscar $k \in T$ / $u_k=\min \{ u_j\}$, $T=T-\{k\}$, $P=P \cup \{k\}$

Si T=Ø, PARAR. Si no, ir al PASO 2

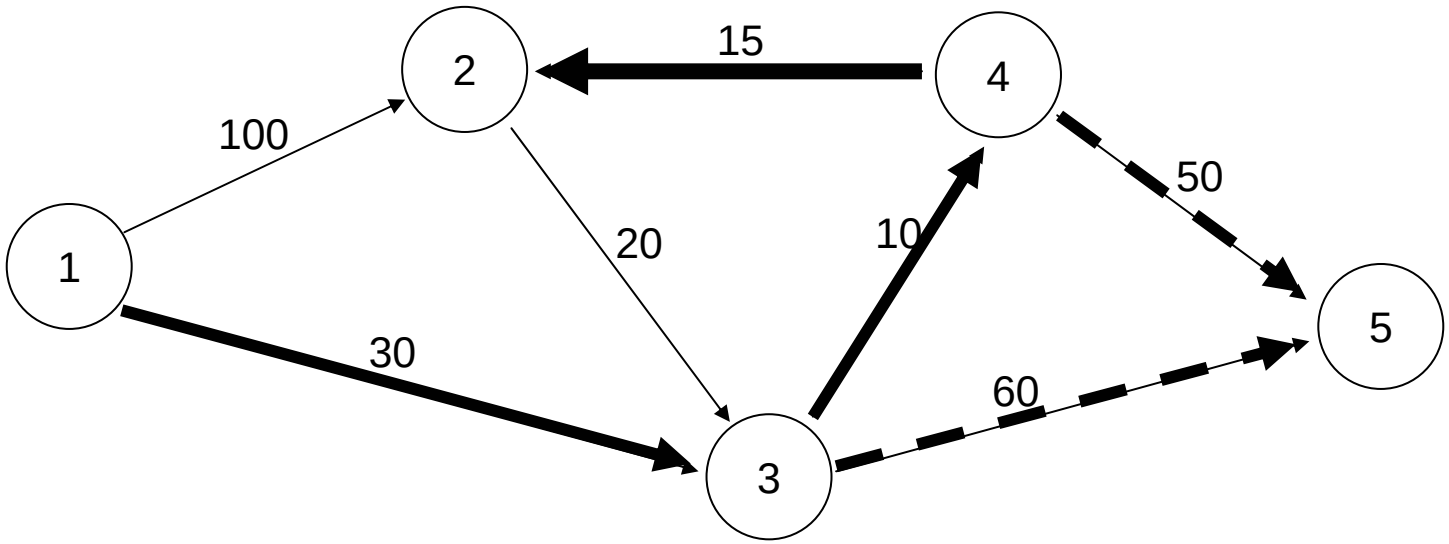
PASO 2 (Revisar etiquetas transitorias)

Poner para todo $j \in T$, $u_k = \min \{u_j, u_k + d_{kj}\}$ . Ir al PASO 1

Si se modifica la etiqueta, pred(j) = k

Complejidad: $\theta(n^2)$

# Ejemplo nº2



# Algoritmo de Bellman-Ford

- Es un algoritmo más general que el anterior
- No válido si existen circuitos de longitud negativa

$u_j$: longitud camino mínimo de 1 a j
$u_j^m$: long. camino mínimo de 1 a j usando m o menos arcos

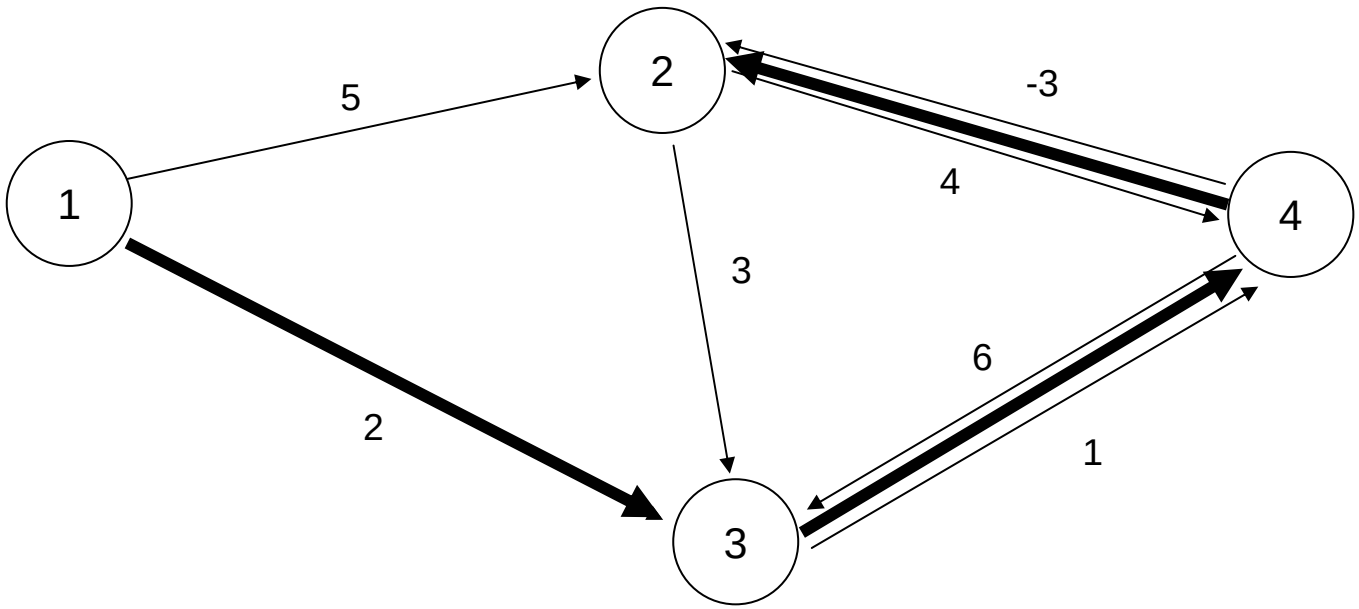
INICIO: $u_1^1=0$, $u_j^1 = d_{1j}$

ITERACIÓN m: $u_j^{m+1} = \min\{ u_j^m, \min\{u_k^m + d_{kj}\}\}$

PARAR: cuando $u_j^m = u_j^{m+1}$, para todo j

Complejidad: $\theta(n^3)$

# Ejemplo nº3



# Problemas de flujo máximo en red

- Un grafo G(V,U) dirigido, sin bucles y conexo es una red de transporte si:
  - existe una única fuente
  - existe un único sumidero
  - G está valorado
  - Cada vértice es alcanzable desde la fuente, y el sumidero es alcanzable desde todos los vértices
- El objetivo es transportar la mayor cantidad posible de la fuente al sumidero

# Conceptos de flujo máximo (I)

- Flujo compatible:
  - No supera la capacidad de ningún arco
  - Verifica la ley de conservación de flujo

$$\forall i \neq s,t \quad \sum_{j/(j,i)\in U} \varphi_{ji} - \sum_{j/(i,j)\in U} \varphi_{ij} = 0$$

  - El flujo que sale de s es el que llega a t
- Corte
  - Conjunto de aristas cuya eliminación desconecta a G en dos componentes tal que s y t no están en la misma
  - Capacidad de un corte: suma de las capacidades de los arcos que lo forman

# Conceptos de flujo máximo (II)

Si $\varphi$ es un flujo compatible,entonces

$$V(\varphi) \leq C(P, P')$$ para todo flujo compatible y corte

Además,

$$\max V(\varphi) = \min C(P, P')$$

El máximo flujo que se puede enviar coincide con la mínima capacidad de un corte

# Algoritmo de Ford-Fulkerson

Es un algoritmo de etiquetado

PASO 1: Flujo compatible = 0, Etiquetar s $(-, \infty)$

PASO 2: Si no se pueden etiquetar más vértices: PASO 6

Si se pueden etiquetar: PASO 3

PASO 3: Sea i el último etiquetado y j no etiquetado unido a i

Si $(i,j) \in U$ y $\varphi_{ij} < C_{ij}$, $\delta_j = \min\{\delta_i, C_{ij} - \varphi_{ij}\}$, etiqueta j: $(i+, \delta_j)$

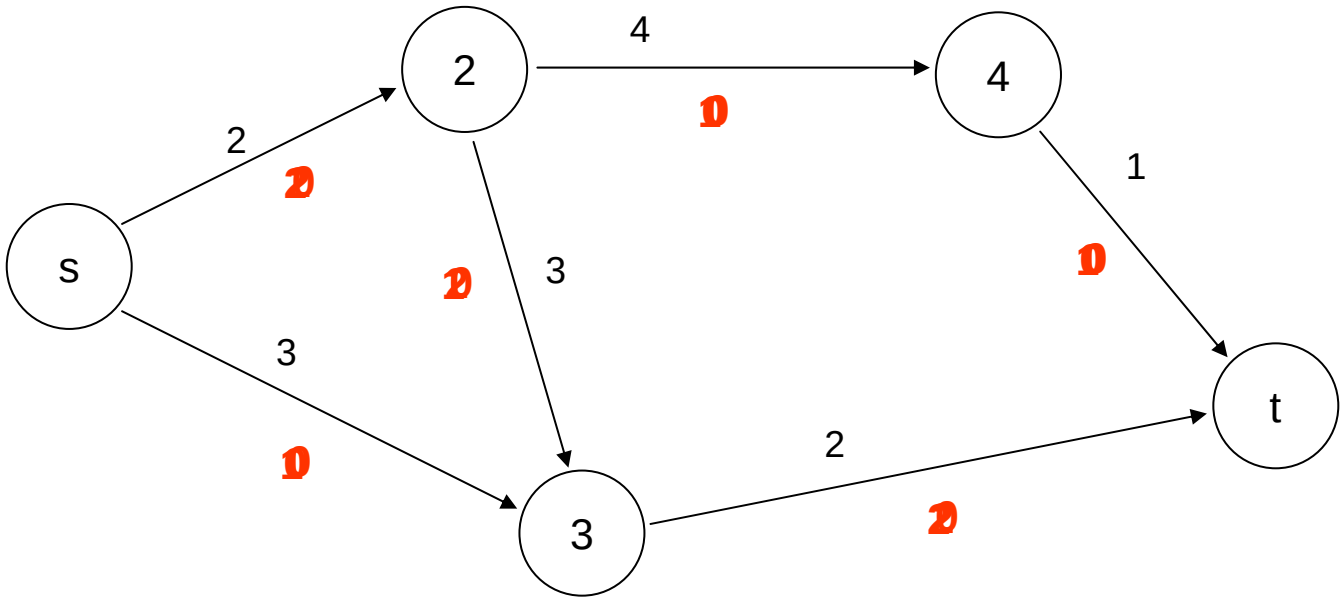
Si $(j,i) \in U$ y $\varphi_{ij} > 0$, $\delta_j = \min\{\delta_i, \varphi_{ij}\}$, etiqueta j: $(i-, \delta_j)$

PASO 4: Si el sumidero está etiquetado ir a PASO 5. Si no, PASO 2

PASO 5: Aumentar el flujo en los vértices etiquetados en $\delta_t$ (disminuyendo en las etiquetas negativas). Borrar etiquetas.

PASO 6: El flujo es máximo. El corte de mínima capacidad son los vértices etiquetados.

# Ejemplo nº4




Flujo máximo: 3

Corte de mínima capacidad:

P= (s, 2, 3, 4)

P'= (t)

# Formulación en programación lineal

$$Max\, V$$

$$s.a.$$

$$0 \le \varphi_{ij} \le C_{ij}$$

$$\sum_{j/(j,i)\in U} \varphi_{ji} - \sum_{j/(i,j)\in U} \varphi_{ij} = \begin{cases} -v \text{ si } i = s \\ 0 \text{ si } i \neq s,t \\ v \text{ si } i = t \end{cases}$$

# Problemas de flujo con coste mínimo

- El objetivo es enviar una cantidad fija de flujo de una fuente s a un sumidero t con coste total mínimo
- Se suele resolver por programación lineal

$$Min \sum_{(i,j)\in U} d_{ij}\varphi_{ij}$$

$$\sum_{i/(s,i)\in U} \varphi_{si} - \sum_{i/(i,s)\in U} \varphi_{is} = \theta$$

$$\sum_{i/(t,i)\in U} \varphi_{ti} - \sum_{i/(i,t)\in U} \varphi_{it} = -\theta$$

$$\sum_{i/(j,i)\in U} \varphi_{ji} - \sum_{i/(i,j)\in U} \varphi_{ij} = 0, \forall j \neq s,t$$

$$0 \leq \varphi_{ij} \leq C_{ij}, \forall (i,j) \in U$$

# Ciclos y caminos eulerianos

- Ciclo/camino euleriano:
  - Si existe un ciclo/camino que pasa por todos los vértices y atraviesa exactamente una vez cada arista
- Teorema (Euler, 1766)
  - Sea G(V,E) un grafo o multigrafo. Existe un ciclo euleriano si y sólo si G es conexo y todos su vértices son de grado par
  - G posee un camino euleriano si y sólo si G es conexo y tiene a lo sumo dos vértices de grado impar (se resuelve uniendo esos vértices con una arista ficticia y buscando un ciclo euleriano)

# Algoritmo ciclo euleriano

Se parte de la matriz de adyacencia $A=(a_{ij})$

PASO 1: Si existe i t.q. $\sum a_{ij}$ no es par: NO existe ciclo

En otro caso, ir a PASO 2. p=0

PASO 2: Elegir fila k no nula de la matriz A. p=p+1

a) Definir n=k y $C=\{v_k\}$

b) Buscar m tal que $a_{nm}>0$

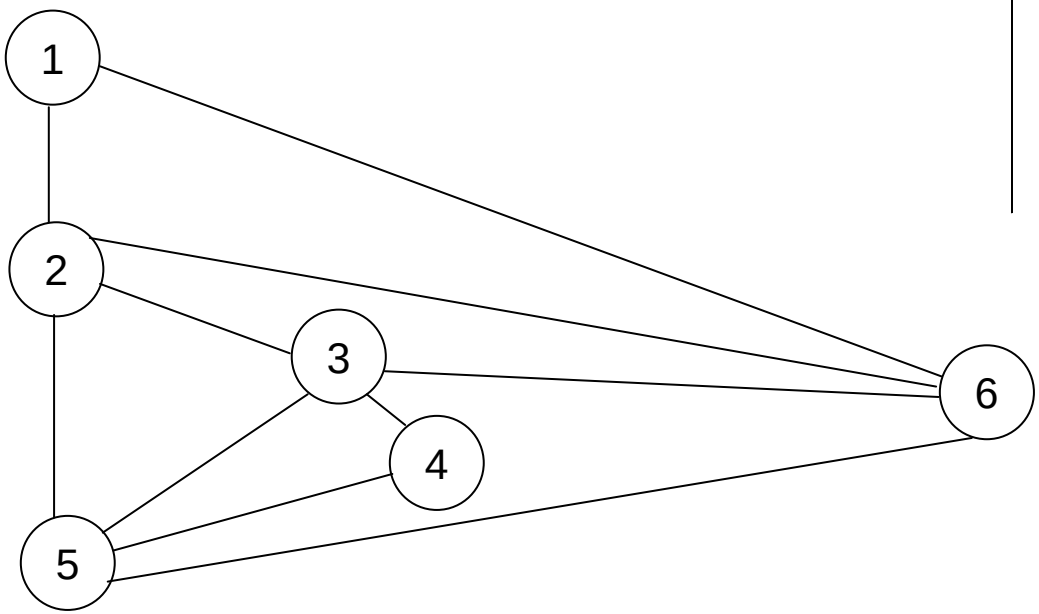
c) Incluir $v_m$ en C. Hacer $a_{nm}=a_{mn}=0$

d) Hacer n=m. Si $n \neq k$ ir a b). Si no, almacenar $C_p$ e ir a PASO 3

PASO 3: Si existe $a_{ij} \neq 0$ ir a PASO 2. Si no, PASO 4

PASO 4: Buscar 2 ciclos $C_p$ con un vértice común. Sustituir el vértice por el ciclo. Repetir hasta que sólo quede un ciclo

# Ejemplo nº5




| | | | | | |
|---|---|---|---|---|---|
| 0 | 1 | 0 | 0 | 0 | 1 |
| 1 | 0 | 1 | 0 | 1 | 1 |
| 0 | 1 | 0 | 1 | 1 | 1 |
| 0 | 0 | 1 | 0 | 1 | 0 |
| 0 | 1 | 1 | 1 | 0 | 1 |
| 1 | 1 | 1 | 0 | 1 | 0 |

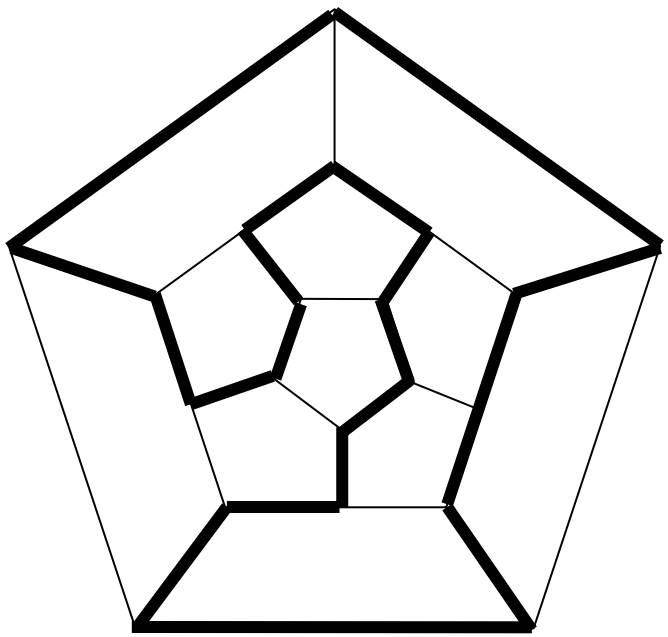
3-2-1-6-2-5-3-4-5-6-3

# Problema del cartero chino

- Es una variante del programa anterior
- Se permite pasar más de una vez por cada arista
- Se busca el recorrido de longitud mínima
- Si existe un ciclo euleriano, esa es la solución
- Si no, hay que pasar por alguna arista más de una vez
- Variantes
  - Cartero motorizado (grafo dirigido)
  - Parte del recorrido prefijado
  - Vuelta a la base cada cierto tiempo

# Recorridos hamiltonianos (I)

Hamilton, 1859: Recorrer los vértices de un dodecaedro pasando por ellos una y sólo una vez



# Recorridos hamiltonianos (II)

- Se dice que G(V,E) tiene un ciclo/camino hamiltoniano si existe un ciclo/camino que contenga todos los vértices de G
- Dado un ciclo hamiltoniano, la eliminación de cualquier arista da un camino hamiltoniano. al revés, no siempre es factible
- No existen condiciones necesarias y suficientes

# Algoritmo circ. hamiltonianos

- Sirve para grafos y digrafos
- Fácilmente adaptable para buscar el circuito hamiltoniano más corto
- Bastante ineficiente
- Idea básica: Si A es la matriz de adyacencia, el elemento (i,j) de $A^k$ es el número de caminos de longitud k que van de i a j
  - k=n-1 : caminos hamiltonianos
  - k=n : circuitos hamiltonianos

# Algoritmo circ. hamiltonianos

PASO 1: Construir una matriz $M_1$ obtenida de la matriz de adyacencia reemplazando cada elemento distinto de 0 pr la hilera ij, salvo en la diagonal

PASO 2: Construir M eliminando la primera letra de cada hilera

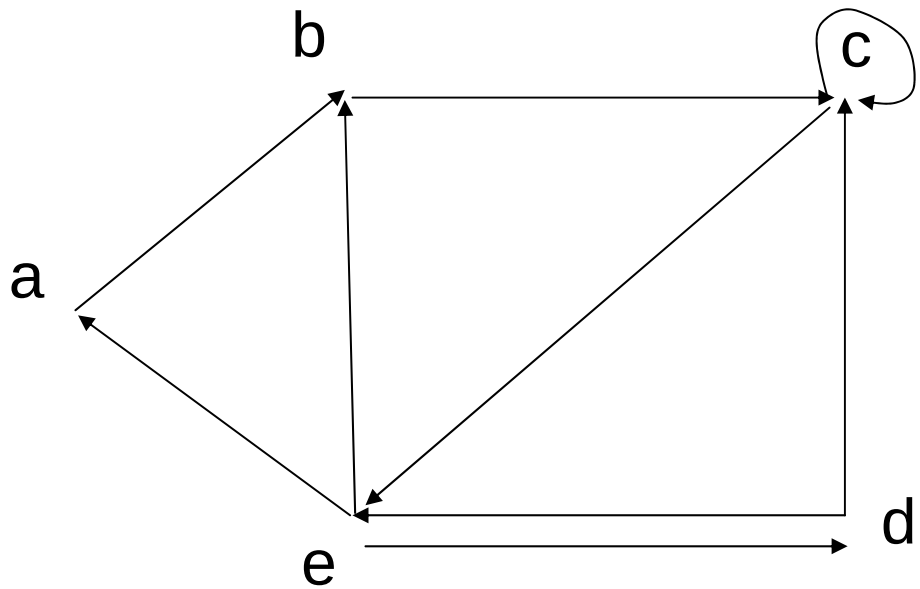
PASO 3: Calcular $M_j$, para todo j entre 1 y n

$M_j(r,s) = M_{j-1}(r,t)M(t,s)$, donde ningún elemento es cero ni tiene vértices comunes (en ese caso, se hace cero)

(es el conjunto de caminos de r a s usando j aristas)

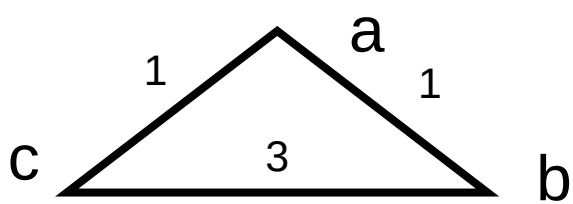
PASO 4: Para encontrar los circuitos hamiltonianos basta ver de los caminos cuáles pueden conectarse los vértices inicial y final

# Ejemplo nº6



# Problema del viajante (TSP)

- Consiste en encontrar el circuito de longitud mínima en un grafo valorado que:
  - visite cada vértice exactamente una vez
  - visite cada vértice al menos una vez
- Grafos dirigidos: TSP asimétrico
- No tiene por qué ser un circuito simple




- Puede reducirse a buscar un circuito hamiltoniano en un grafo modificado

# Algoritmo bioptimal

PASO 0: Seleccionar origen s. Elegir t t.q. su distancia a s sea mínima. Hacer l=t

PASO 1: Seleccionar t entre los no visitados t.q. d(l,t) sea mínima. Añadir t al recorrido y hacer l=t

Si hay nudos que no estén en el recorrido: PASO 1

Si no, añadir s al recorrido: PASO 2

PASO 2: El recorrido será $x_1$, $x_2$, ..., $x_n$, $x_1$. Longitud L, i=1

PASO 3: j=i+2

PASO 4: Considerar recorrido $x_1$, .., $x_i$, $x_j$, $x_{j-1}$, .., $x_{i+1}$, $x_{j+1}$, .., $x_n$

Si L' < L : Se considera nuevo recorrido: PASO 2

Si L' > L: PASO 5

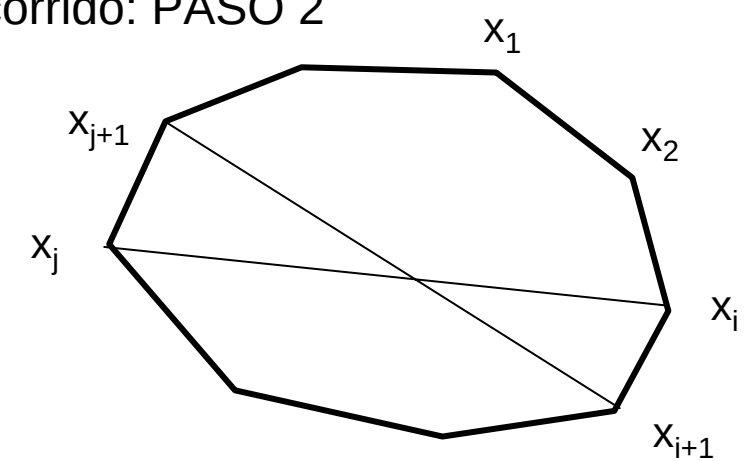
PASO 5: j=j+1

Si j≤n : PASO 4

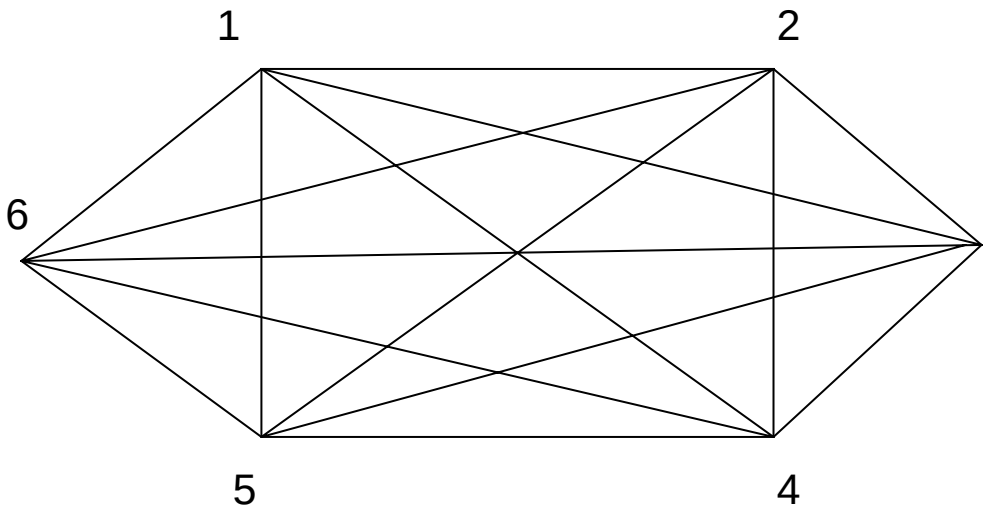
Si j>n, i=i+1

Si i ≤n-2 : PASO 3

Si i >n-2 : PARAR



# Ejemplo nº7




| | | | | | |
|---|---|---|---|---|---|
| ∞ | 13 | 12 | 18 | 7 | 14 |
| 13 | ∞ | 21 | 26 | 15 | 25 |
| 12 | 21 | ∞ | 11 | 6 | 4 |
| 18 | 26 | 11 | ∞ | 12 | 14 |
| 7 | 15 | 6 | 12 | ∞ | 9 |
| 14 | 25 | 4 | 14 | 9 | ∞ |

# Control y prog. de proyectos

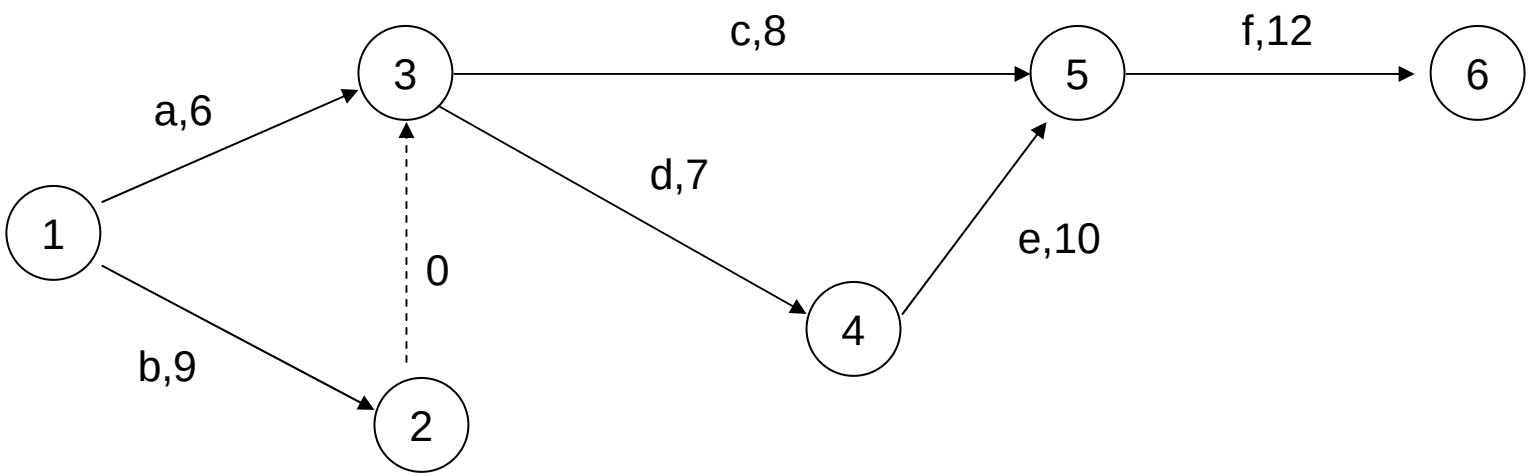
- Dos técnicas fundamentales:
  - CPM (Critical Path Method)
    - Determinista
  - PERT (Program Evaluation and Review Technique)
    - Probabilista
- Desarrolladas a finales de los 50
- Se basan en formular los proyectos como redes
- Sirven para programar y controlar proyectos complejos, con muchas actividades interrelacionadas

# Red de actividades

- Es la representación de las actividades del proyecto
- Reglas de construcción de la red:
  - Cada actividad se representa por un (y sólo un) arco, cuya dirección es la de progreso en el proyecto. Asociado a cada arco debe figurar el nombre de la actividad y su duración
  - Los nodos representan sucesos (inicio o término) de actividades. Debe existir uno inicial y otro final
  - Dos nodos sólo pueden estar conectados por un arco (se utilizan actividades ficticias)
  - Para asegurar la representación adecuada de las preferencias, al añadir una actividad hay que contestar a:
    - ¿Qué actividades la preceden?
    - ¿Qué actividades la siguen?
    - ¿Cuáles deben ocurrir a la vez?

# Ejemplo nº8

| ACTIVIDADES | Antecesores | Duración |
|---|---|---|
| A-Capacitar trabajadores | - | 6 |
| B- Compra materia prima | - | 9 |
| C- Fabricar producto 1 | A,B | 8 |
| D- Fabricar producto 2 | A,B | 7 |
| E- Probar producto 2 | D | 10 |
| F- Ensamblar productos 1y 2 | C,E | 12 |



# Camino crítico (CPM)

- Supone conocidas las duraciones de las actividades
- El objetivo es hallar
  - la duración total del proyecto
  - las actividades críticas
  - las holguras para las actividades
- Actividad crítica: la que no se puede retrasar o adelantar sin afectar a la duración total del proyecto
- Holgura: Tiempo en que se puede retrasar alguna actividad sin que afecte a la duración total del proyecto (las actividades críticas tienen holgura nula)
- Camino crítico: el formado por las actividades críticas

# Algoritmo CPM (I)

Para cada nodo j se define

$\square_j$: instante más temprano en que puede ocurrir

$\Delta_j$: instante más tardío en que puede ocurrir

$D_{ij}$: duración de la actividad (i,j)

FASE HACIA DELANTE: Cálculo de instantes más tempranos

1) $\square_1 = 0$

2) $\square_j = \max\{\square_p+D_{pj}, \square_q+D_{qj}, ...\}$, donde p,q son nodos unidos a j y cuyos valores $\square$ ya han sido calculados

3) Si j=n, parar. Si no, j=j+1 y repetir el paso 2)

4) $\square_n$ : duración total del proyecto

FASE HACIA ATRÁS: Cálculo de instantes más tardíos

1) $\Delta_n=\square_n$

2) $\Delta_j = \min\{\Delta_p-D_{jp}, \Delta_q-D_{jq}, ...\}$, donde p,q son nodos unidos a j y cuyos valores $\Delta$ ya han sido calculados

3) Si j=1, parar. Si no, j=j-1 y repetir el paso 2)

# Algoritmo CPM (II)

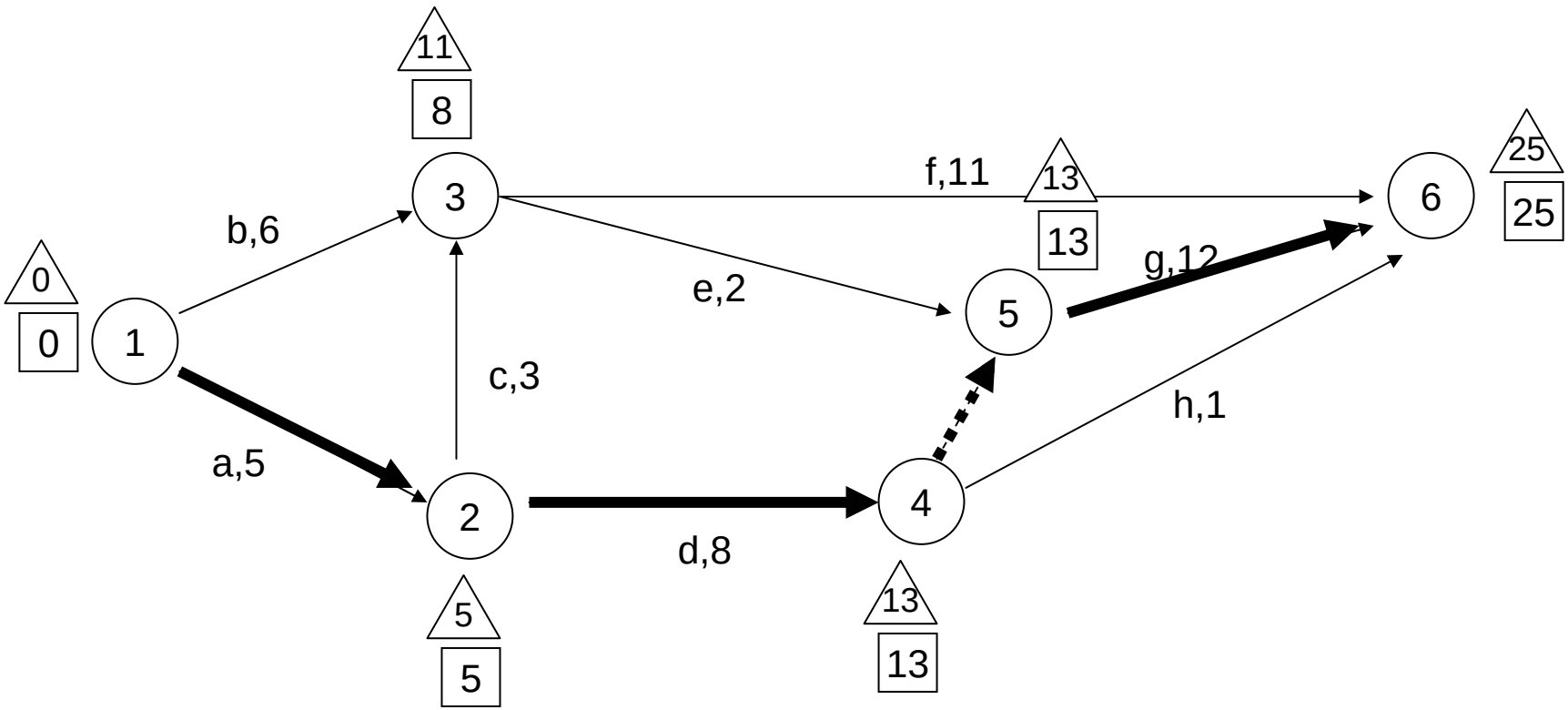
Una actividad (i, j) será crítica si:

1) $\Delta_i = \square_i$

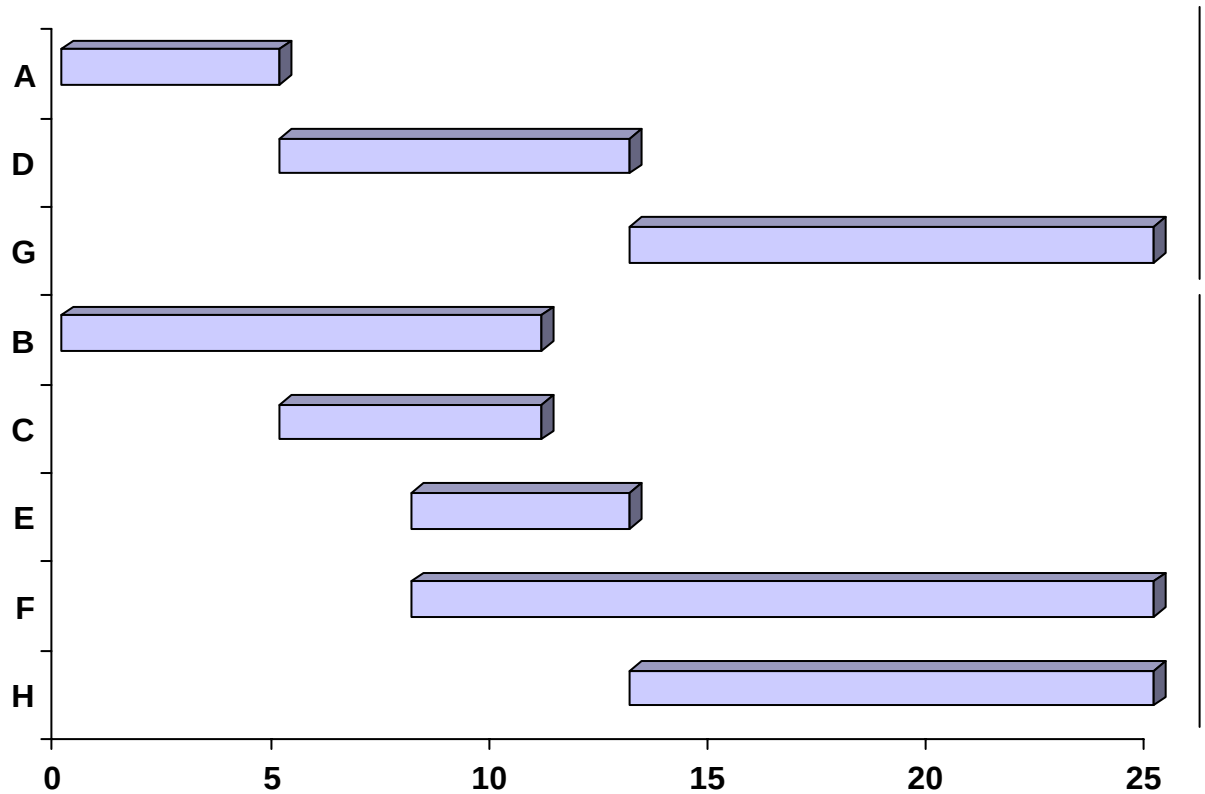
2) $\Delta_j = \square_j$

3) $\Delta_j - \Delta_i = \square_j - \square_i = D_{ij}$

# Ejemplo nº9



# Ejemplo nº9


A
D
G
B
C
E
F
H
0
5
10
15
20
25
ACTIVIDADES CRÍTICAS
ACTIVIDADES NO CRÍTICAS

# Holguras

## Holgura total ($TF_{ij}$)

Cantidad en que se puede retrasar la actividad o aumentar su duración sin retrasar el proyecto (suponiendo que no se retrasan otras)

$$TF_{ij} = \Delta_j - \square_i - D_{ij}$$

## Holgura libre ($FF_{ij}$)

Cantidad en que se puede retrasar la actividad o aumentar su duración sin afectar a otras actividades

$$FF_{ij} = \square_j - \square_i - D_{ij}$$

Por definición, $FF_{ij} \leq TF_{ij}$

# Ejemplo nº9

| Actividades no críticas | $D_{ij}$ | TF | FF |
|---|---|---|---|
| B | 6 | 11-0-6=5 | 8-0-6=2 |
| C | 3 | 11-5-3=3 | 8-5-3=0 |
| E | 2 | 13-8-2=3 | 13-8-2=3 |
| F | 11 | 25-8-11=6 | 25-8-11=6 |
| H | 1 | 25-13-1=11 | 25-13-1=11 |

# PERT (I)

- Se considera aleatoria la duración, con una esperanza y una varianza
- Supuestos
  - Todas las variables son independientes
  - La variable $D_{ij}$ tiene una distribución beta
  - El camino crítico no cambia, siempre es el más largo
  - La variable CP (duración total) tiene una distribución normal

# PERT (II)

Resultados

$$E(D_{ij}) = \frac{a_{ij} + 4m_{ij} + b_{ij}}{6}$$

$$V(D_{ij}) = \frac{(b_{ij} - a_{ij})^2}{36}$$

$$E(CP) = \sum_{i,j} E(D_{ij})$$

$$V(CP) = \sum_{i,j} V(D_{ij})$$

$$P(CP \leq X) = P\left( \frac{CP - E(CP)}{\sqrt{V(CP)}} \leq \frac{X - E(CP)}{\sqrt{V(CP)}} \right)$$