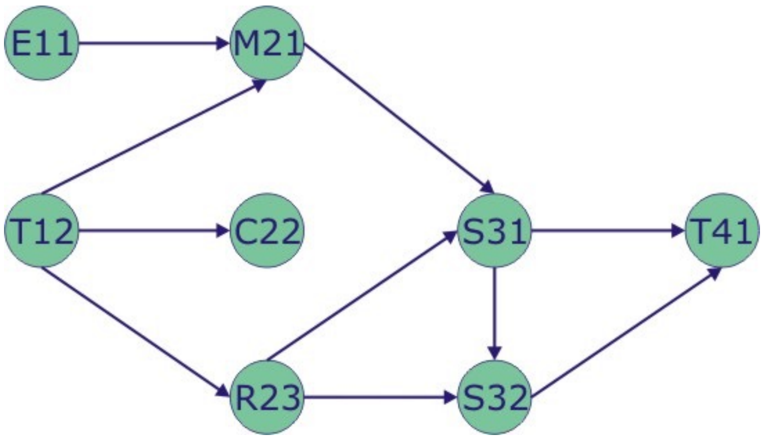
Tema 6
Grafos II: Algoritmos fundamentales

1. Ordenación topológica
2. Matriz de caminos: algoritmo de Warshall
3. Problema de los caminos más cortos con un solo origen: algoritmo de Dijkstra
4. Problema de los caminos más cortos entre todos los pares de vértices: algoritmo de Floyd-Warshall
5. Problema del flujo de fluidos
6. Problema del árbol de expansión de coste mínimo: algoritmos de Prim y Kruskal

1. Ordenación topológica

- **Ordenación topológica T de un grafo acíclico (sin ciclos)**
  - ordenación lineal de los vértices, tal que si hay un camino de v_i a v_j, entonces v_j aparece DESPUÉS de v_i en la ordenación T.
- Ejemplo: GDA que representa la estructura de prerrequisitos de 8 cursos.

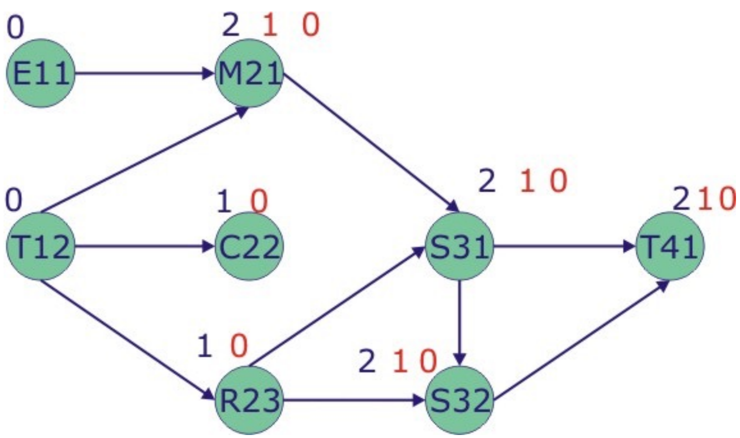


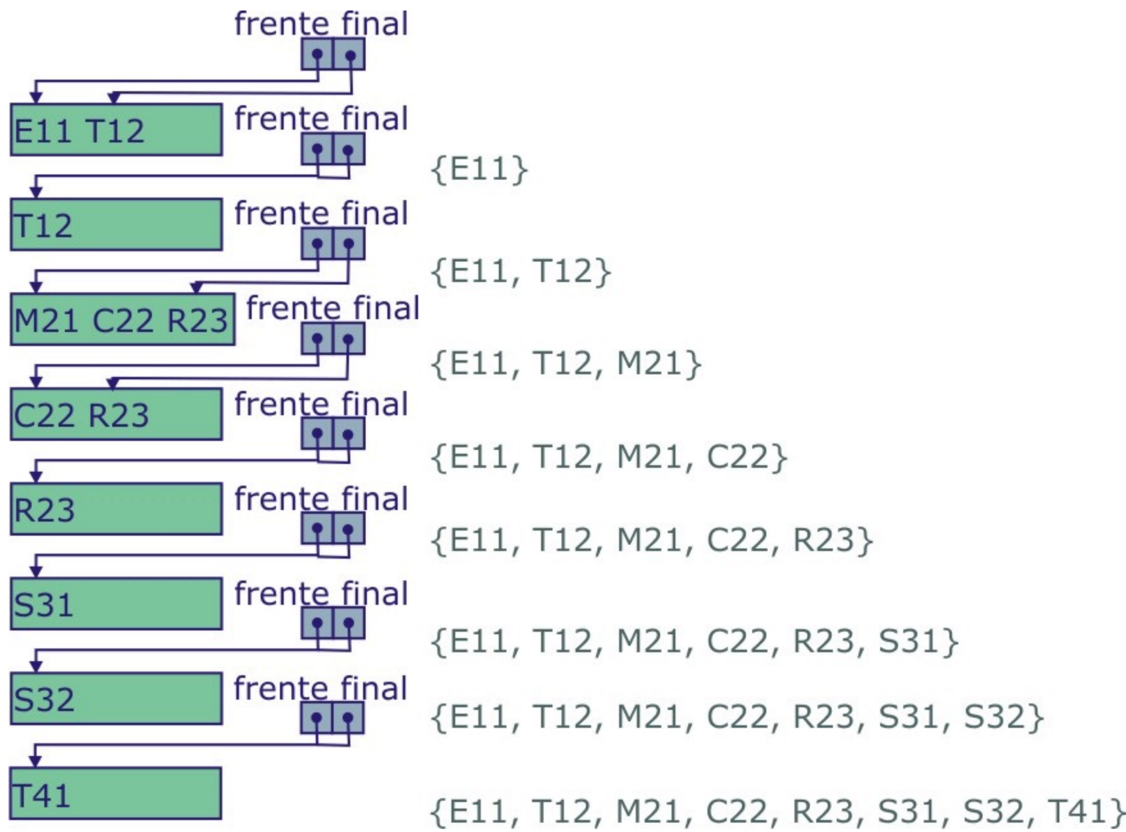
  - Arco (r,s): el curso ***r*** debe terminarse antes de empezar el curso ***s***.
  - Ordenación topológica: cualquier secuencia de cursos que cumpla los prerrequisitos.
    - Para un GDA no tiene porqué existir una única ordenación topológica
      - E11 – T12 – M21 – C22 – R23 – S31 – S32 – T41
      - T12 – E11 – R23 – C22 – M21 – S31 – S32 – T41

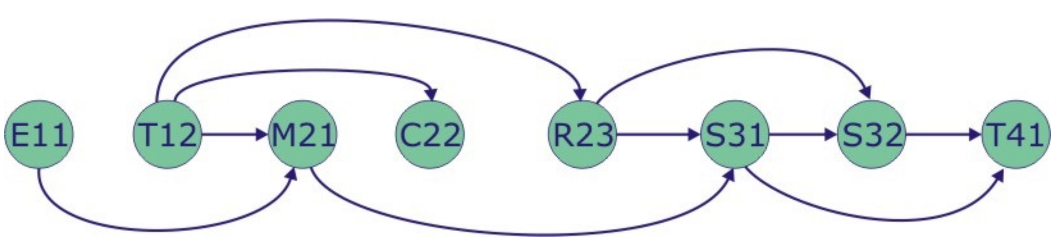
1. Ordenación topológica

- Algoritmo para la obtención de una ordenación topológica T
  - Buscamos vértices sin predecesores (`grado de entrada=0`), y los metemos en una cola.
  - Se saca el vértice en el frente de la cola y pasa a formar parte de la ordenación topológica T
  - Eliminamos los arcos que salen del vértice anterior (⇒disminuir `gradent` de sus adyacentes)
    - Los nuevos vértices con `gradent=0` se meten en la cola
  - Seguimos hasta que la cola esté vacía

- Ejemplo:


0
E11
2 1 0
M21
0
T12
1 0
C22
2 1 0
S31
2 1 0
T41
1 0
R23
2 1 0
S32

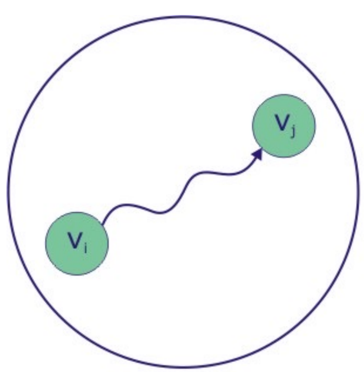

frente final
E11 T12
{E11}
T12
{E11, T12}
M21 C22 R23
{E11, T12, M21}
C22 R23
{E11, T12, M21, C22}
R23
{E11, T12, M21, C22, R23}
S31
{E11, T12, M21, C22, R23, S31}
S32
{E11, T12, M21, C22, R23, S31, S32}
T41
{E11, T12, M21, C22, R23, S31, S32, T41}


E11
T12
M21
C22
R23
S31
S32
T41

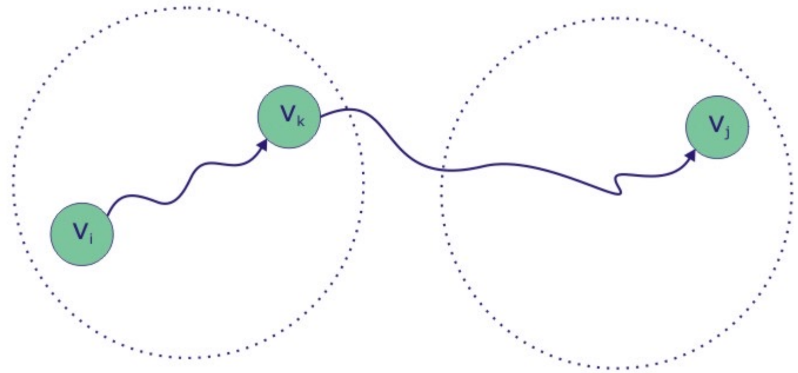
2. Matriz de caminos: Algoritmo de Warshall

- Warshall propuso un algoritmo eficiente para calcular la **matriz de caminos**
- $\forall\ k=0,1,2,\ldots,n$ se define la matriz P_k como:
  - $P_k(i,j) = \begin{cases} TRUE & \text{si hay camino de i a j que use a lo sumo como vértices intermedios el } 1, 2, \ldots, k \\ FALSE & \text{en otro caso} \end{cases}$
- **Warshall** encuentra una relación entre las matrices P_{k-1} y P_k que nos permite, partiendo de P_0 (matriz de adyacencia), encontrar $P_n=P$ (matriz de caminos) por sucesivas iteraciones.
- La relación para encontrar los elementos de P_k es:

$$P_k(i,j)=P_{k-1}(i,j) \text{ OR } [P_{k-1}(i,k) \text{ AND } P_{k-1}(k,j)]$$

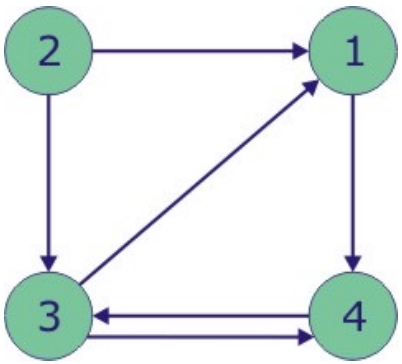


Camino de v_i a v_j



Camino de v_i - v_k - v_j

- **Ejemplo:**



$$P_0 = A = \begin{pmatrix} F & F & F & T \\ T & F & T & F \\ T & F & F & T \\ F & F & T & F \end{pmatrix} \qquad P_1 = \begin{pmatrix} F & F & F & T \\ T & F & T & T \\ T & F & F & T \\ F & F & T & F \end{pmatrix}$$

$P_1(2,4)=P_0(2,4)$ OR $(P_0(2,1)$ AND $P_0(1,4))$

$$P_2 = \begin{pmatrix} F & F & F & T \\ T & F & T & T \\ T & F & F & T \\ F & F & T & F \end{pmatrix} \qquad P_3 = \begin{pmatrix} F & F & F & T \\ T & F & T & T \\ T & F & F & T \\ T & F & T & T \end{pmatrix} \qquad P_4 = \begin{pmatrix} T & F & T & T \\ T & F & T & T \\ T & F & T & T \\ T & F & T & T \end{pmatrix}$$

$P_3(4,1)=P_2(4,1)$ OR $(P_2(4,3)$ AND $P_2(3,1))$

$P_3(4,4)=P_2(4,4)$ OR $(P_2(4,3)$ AND $P_2(3,4))$

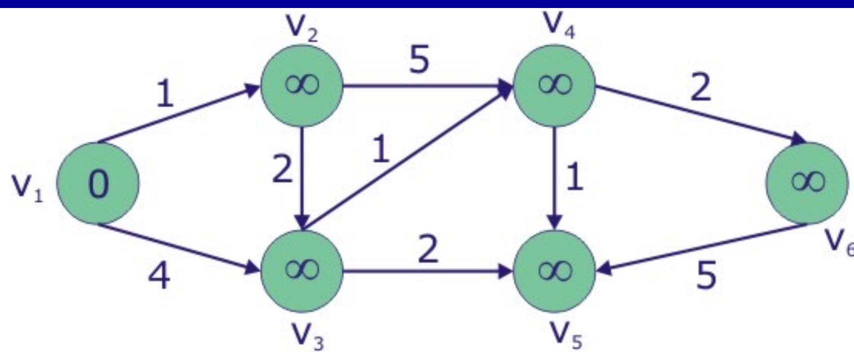
3. Algoritmo de Dijkstra

- **El algoritmo de Dijkstra calcula el camino más corto desde un vértice a todos los demás**
- **Aplicación**: Calcular la ruta que en menor tiempo nos lleva desde un punto (nuestra casa, por ejemplo) a un conjunto de lugares de la ciudad. Los vértices intermedios son paradas de Bus o Metro.
- **Es un algoritmo voraz clásico** (resuelve el problema en sucesivos pasos, seleccionando en cada paso la solución óptima) donde los candidatos son los vértices del grafo.
  - **C**=conjunto de vértices candidatos; **S**=conjunto de los ya escogidos.
  - Camino especial=camino que parte del vértice origen y que tiene todos los vértices dentro de **S** excepto posiblemente el último.
  - **D**=vector de distancias que mantiene la longitud del camino especial más corto desde el origen a cualquier vértice de **G**.
  - **A** es la matriz de costes (pesos).

- **Algoritmo:**
  - Inicialmente, **C**=todos los vértices excepto el origen (el **1**) y **S**=vértice origen (el **1**). Así, **D(j)=A(1,j)** para **j=2,…,n**
  - Sea **i** el vértice de **C** con **D(i)** mínimo.
  - Eliminamos **i** de **C** y lo ponemos en **S**.
  - Para cada vértice **j** de **C** actualizamos **D(j)** como **D(j)=mínimo[D(j),D(i)+A(i,j)]**
  - Si el esquema anterior lo repetimos **n-1** veces y **n** es el número de vértices, tendremos en **D** la longitud de los caminos mínimos que, partiendo del vértice **1**, llegan a cada uno de los restantes vértices.
  - Para recuperar el camino de longitud mínima en el algoritmo de Dijkstra, basta con añadir al algoritmo un array auxiliar **P**, de tal manera que **P(j)** contiene siempre el predecesor inmediato del camino especial mínimo que va del vértice **1** al vértice **j**.

3. Algoritmo de Dijkstra

- Ejemplo



P=(0,_,_,_,_,_)

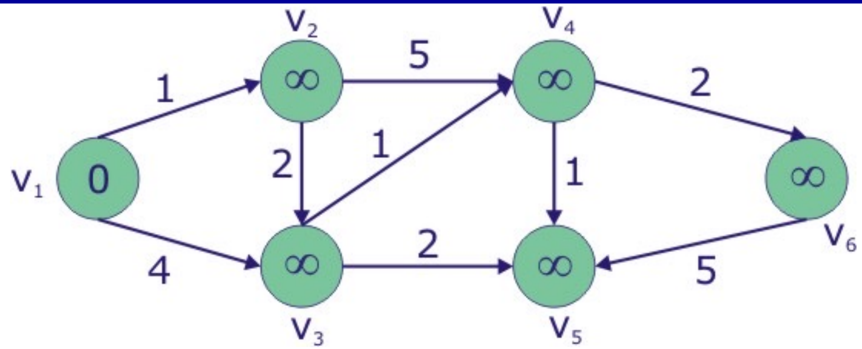
D=(0,∞,∞,∞,∞,∞)

Seleccionamos el nodo con valor mínimo en D (en este caso V1), y lo añadimos a S

S=V1

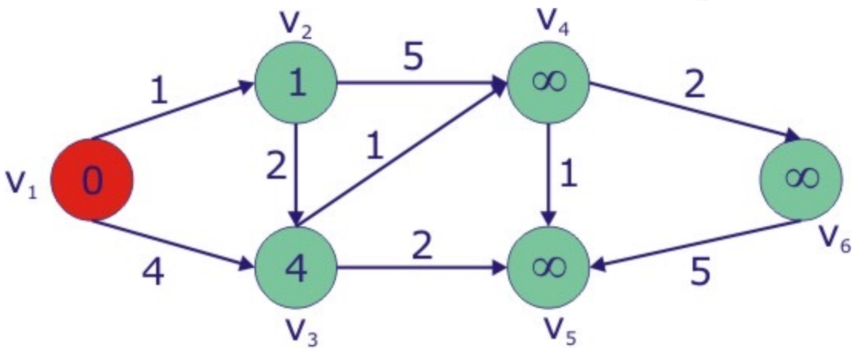
C=V2,V3,V4,V5,V6

- Ejemplo



P=(0,_,_,_,_,_)

D=(0,∞,∞,∞,∞,∞)



Recalculamos P y D

P=(0,1,1,_,_,_)

D=(0,1,4,∞,∞,∞)

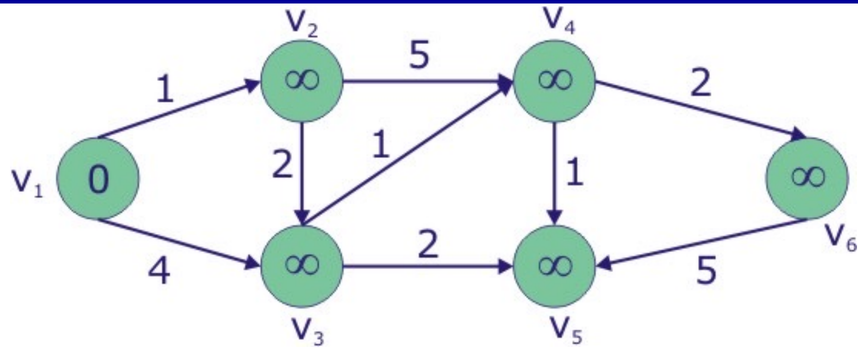
Seleccionamos el nodo con valor mínimo en D que pertenezca al conjunto C (ahora V2), y lo añadimos a S

S=V1,V2

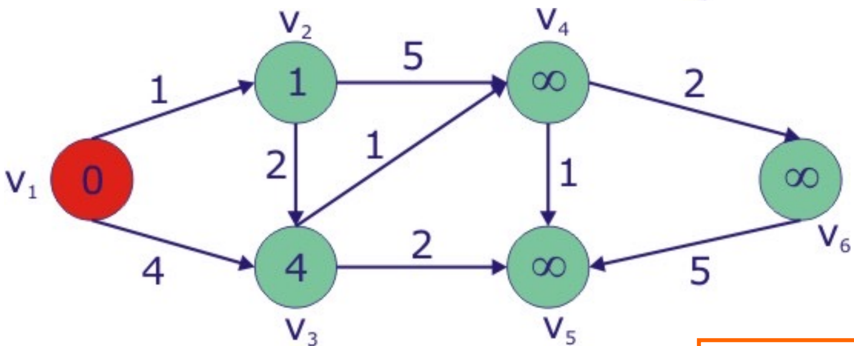
C=V3,V4,V5,V6

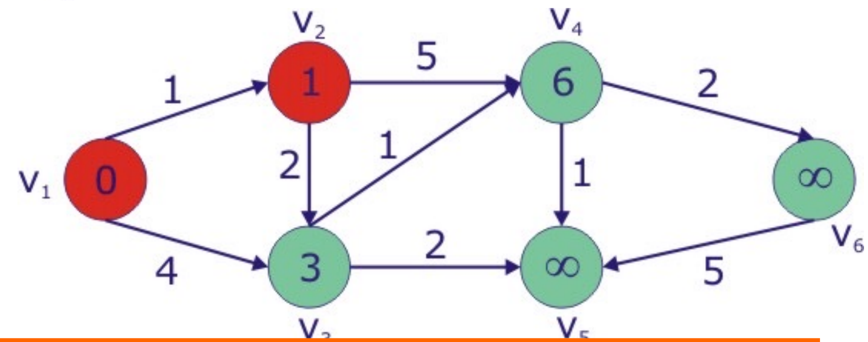
3. Algoritmo de Dijkstra

- Ejemplo



P=(0,_,_,_,_,_)





P=(0,1,1,_,_,_)

D=(0,1,4,∞,∞,∞)

Recalculamos P y D P=(0,1,2,2,_,_)

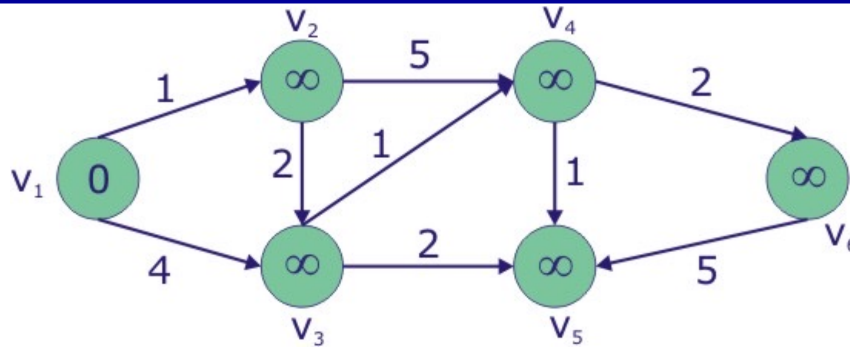
D=(0,1,3,6,∞,∞)

Seleccionamos el nodo con valor mínimo en D (ahora V3), lo añadimos a S

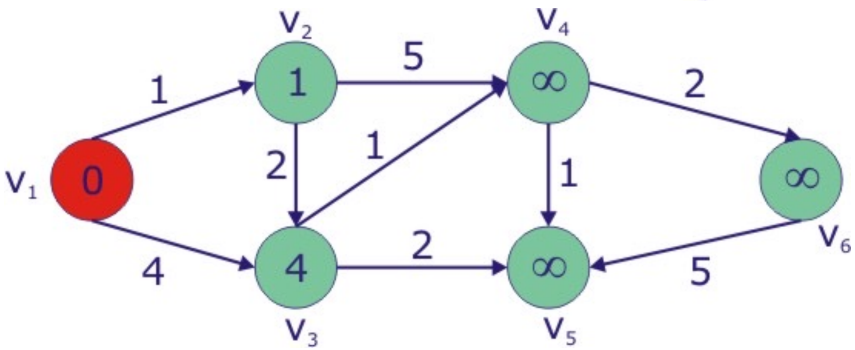
S=V1,V2,V3

C=V4,V5,V6

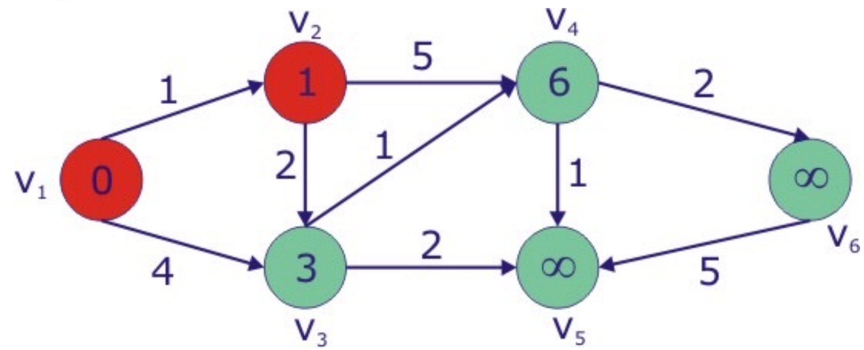
- Ejemplo



P=(0,_,_,_,_,_)

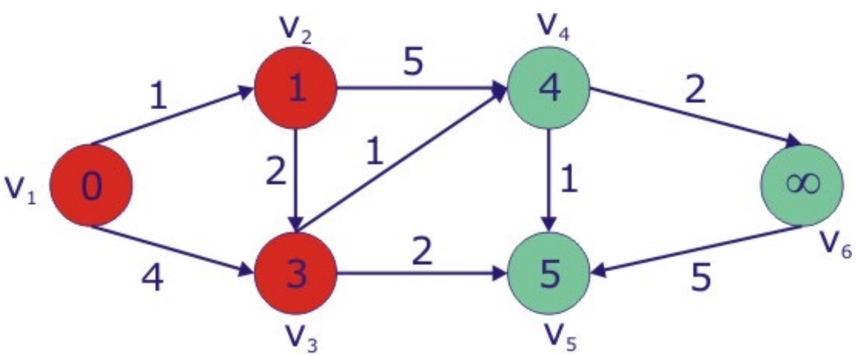


P=(0,1,1,_,_,_)



P=(0,1,2,2,_,_)

D=(0,1,3,6,∞,∞)



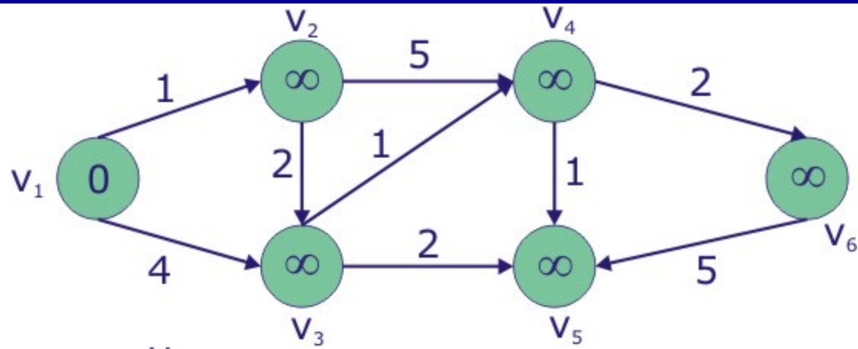
Recalculamos P y D

P=(0,1,2,3,3,_)

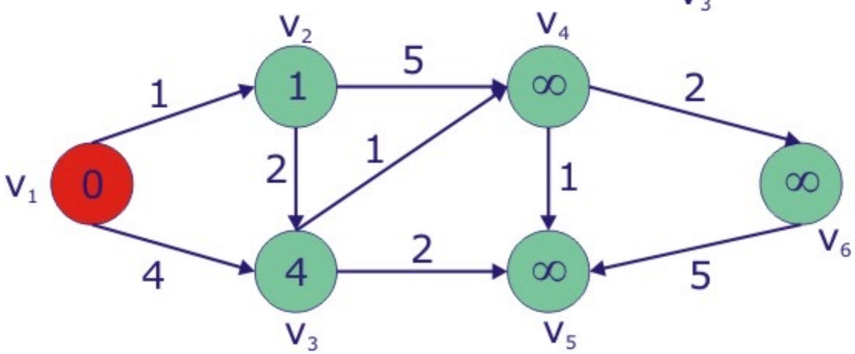
D=(0,1,3,4,5,∞)

Seleccionamos V4 =>

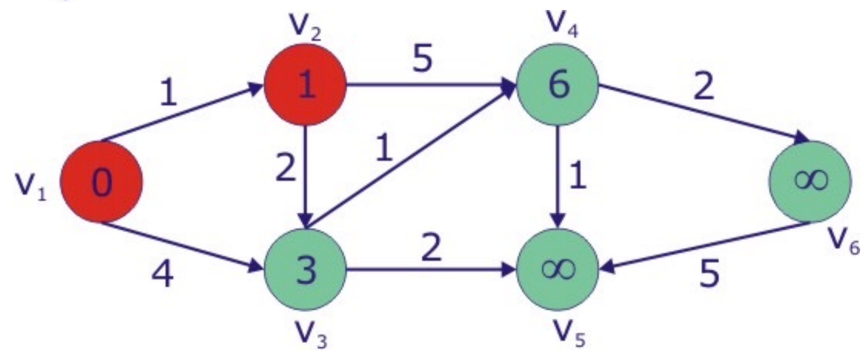
- Ejemplo



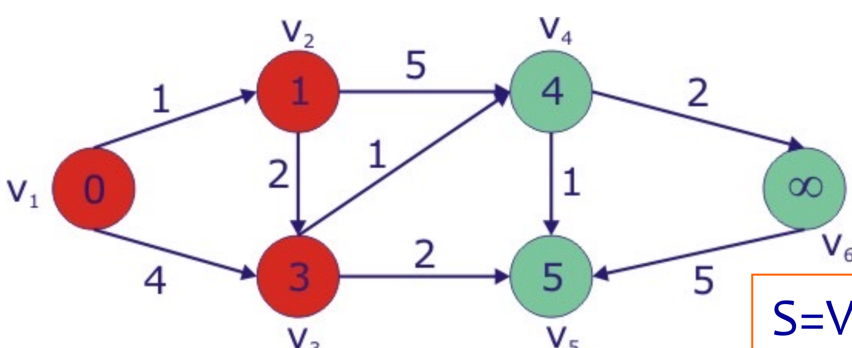
P=(0,_,_,_,_,_)



P=(0,1,1,_,_,_)



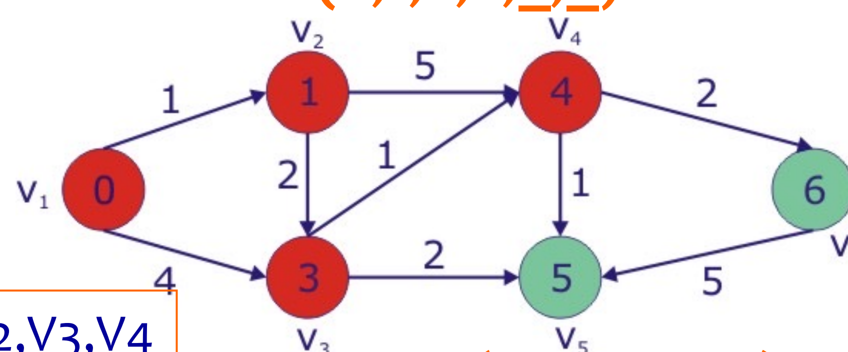
P=(0,1,2,2,_,_)



P=(0,1,2,3,3,_)

S=V1,V2,V3,V4

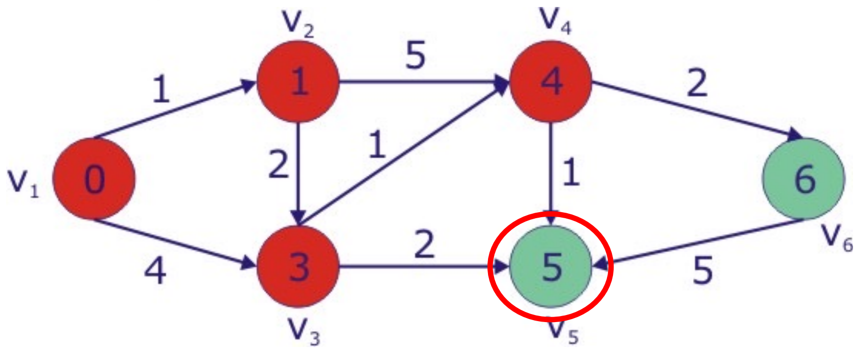
C=V5,V6



P=(0,1,2,3,4,4)

D=(0,1,3,4,5,6)

- Ejemplo



S=V1,V2,V3,V5

C=V6

P=(0,1,2,3,4,4)

D=(0,1,3,4,5,6)

Al seleccionar v5 en el siguiente paso no hay arco a v6, por lo tanto no hay cambios en P ni en D

El camino mínimo de v1 a v6 tiene longitud 6

La secuencia de vértices que hacen el camino mínimo es:

v1-v2-v3-v4-v6

- **El algoritmo de Floyd-Warshall calcula el camino más corto entre todos los pares de vértices**
  - *Robert W. Floyd es profesor de la Stanford University y en 1978 fue galardonado con el prestigioso premio A.M. Turing que otorga la ACM para reconocer las contribuciones de naturaleza técnica realizadas a la comunidad informática. El premio le fue concedido por tener una influencia clara en las metodologías para la creación de software eficiente y fiable, y por ayudar a fundar las siguientes áreas de la informática: teoría de análisis sintáctico, semántica de los lenguajes de programación, verificación automática de programas, síntesis automática de programas y análisis de algoritmos.*
  - *El algoritmo de Floyd-Warshall fue ideado por Floyd en 1962 basándose en un teorema de Warshall también de 1962.*
- Sea **G** un grafo dirigido valorado, **G=(V,A)**.
  - Suponemos que los vértices están numerados de **1** a **n**
  - La matriz **A** es una matriz de pesos, de modo que todo arco **(i,j)** tiene asociado un peso c_{ij}
- Queremos encontrar la matriz **D** de n×n elementos tal que cada elemento D_{ij} sea el coste mínimo de los caminos que van del vértice **i** al vértice **j**.

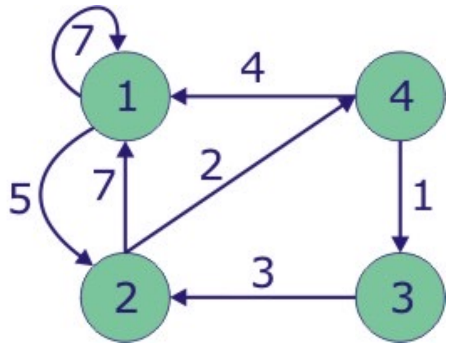
- Algoritmo de Floyd-Warshall
  - Sigue los mismos pasos que el algoritmo de *Warshall* para encontrar la matriz de caminos
  - Se genera iterativamente la secuencia de matrices D_0, D_1, …, D_k, …, D_n cuyos elementos tienen el significado:
    - $D_0(i,j)=c(i,j)$ coste (peso) del arco de i a j (∞ si no hay arco; 0 cuando i=j)
    - $\forall$k=1,2,…,n, $D_k(i,j)$=longitud del camino mínimo de i a j usando los vértices 1, 2, …, k
      - Si este camino usa vértice k: $D_k(i,j)=D_{k-1}(i,k)+D_{k-1}(k,j)$
      - Si este camino NO usa vértice k: $D_k(i,j)=D_{k-1}(i,j)$

    $$D_k(i,j)=\text{minimo}(D_{k-1}(i,j),\ D_{k-1}(i,k)+D_{k-1}(k,j))$$

    - D_n será la matriz de caminos mínimos del grafo
  - Igual que en el algoritmo de Dijkstra, por cada vértice queremos guardar el índice del último vértice que ha conseguido que el camino sea mínimo del i al j; en caso de que el camino sea directo tiene un 0. Para ello usamos una **matriz de vértices predecesores P**.

4. Algoritmo de Floyd-Warshall

- Ejemplo



$$P(i,j) = \begin{cases} i & si\ existe\ arco\ (i,j), con\ i \neq j \\ 0 & en\ otro\ caso \end{cases}$$

$$D_0 = \begin{pmatrix} 0 & 5 & \infty & \infty \\ 7 & 0 & \infty & 2 \\ \infty & 3 & 0 & \infty \\ 4 & \infty & 1 & 0 \end{pmatrix} \qquad P = \begin{pmatrix} - & 1 & - & - \\ 2 & - & - & 2 \\ - & 3 & - & - \\ 4 & - & 4 & - \end{pmatrix}$$

$$D_1 = \begin{pmatrix} 0 & 5 & \infty & \infty \\ 7 & 0 & \infty & 2 \\ \infty & 3 & 0 & \infty \\ 4 & 9 & 1 & 0 \end{pmatrix} \quad P = \begin{pmatrix} - & 1 & - & - \\ 2 & - & - & 2 \\ - & 3 & - & - \\ 4 & 1 & 4 & - \end{pmatrix} \quad D_2 = \begin{pmatrix} 0 & 5 & \infty & 7 \\ 7 & 0 & \infty & 2 \\ 10 & 3 & 0 & 5 \\ 4 & 9 & 1 & 0 \end{pmatrix} \quad P = \begin{pmatrix} - & 1 & - & 2 \\ 2 & - & - & 2 \\ 2 & 3 & - & 2 \\ 4 & 1 & 4 & - \end{pmatrix}$$

$$D_3 = \begin{pmatrix} 0 & 5 & \infty & 7 \\ 7 & 0 & \infty & 2 \\ 10 & 3 & 0 & 5 \\ 4 & 4 & 1 & 0 \end{pmatrix} \quad P = \begin{pmatrix} - & 1 & - & 2 \\ 2 & - & - & 2 \\ 2 & 3 & - & 2 \\ 4 & 3 & 4 & - \end{pmatrix} \quad D_4 = \begin{pmatrix} 0 & 5 & 8 & 7 \\ 6 & 0 & 3 & 2 \\ 9 & 3 & 0 & 5 \\ 4 & 4 & 1 & 0 \end{pmatrix} \quad P = \begin{pmatrix} - & 1 & 4 & 2 \\ 4 & - & 4 & 2 \\ 4 & 3 & - & 2 \\ 4 & 3 & 4 & - \end{pmatrix}$$

Camino mínimo de v1 a v4: longitud=7 (v1-v2-v4)

P(1,4)=2 : v1→…→v2→v4   P(1,2)=1 : v1→v2→v4

Camino mínimo de v1 a v3: longitud=8 (v1-v2-v4-v3)

P(1,3)=4: v1→…→v4→v3 P(1,4)=2: v1→…→v2→v4→v3 P(1,2)=1 : v1→v2→v4→v3

Concepto del flujo (i)

- Flujo=forma de enviar objetos de un lugar a otro.
- Ejemplos:
  - Flujo de mercancías entre centros de producción y distribución
  - Flujo de personas entre lugares de residencia y centros de trabajo
  - Sistema de tuberías para transporte de agua
    - Cada arco es un tubo y el "peso" representa la capacidad de la tubería en litros/minuto
    - Los nodos son los puntos de unión de los diversos tubos
- Objetivo: transportar la máxima cantidad de flujo desde un punto de partida (llamado fuente s) hacia un punto final (llamado sumidero t).
- Un grafo con factor de peso es una estructura ideal para modelar estas situaciones.

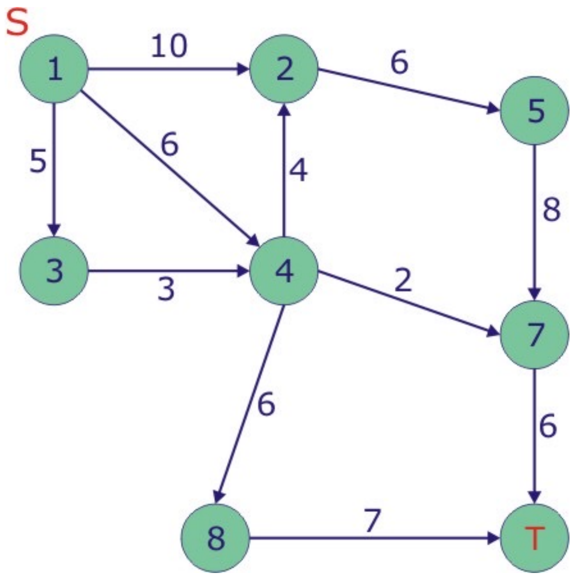
5. Problema del flujo de fluidos

Ejemplo 1: Planificación de rutas alternativas para circulación en horas punta.

- Suponemos punto origen con fuerte tráfico, por ejemplo una ciudad dormitorio
- El punto destino representa el centro, zona comercial, de la gran ciudad.
- Entre ambos puntos se pueden establecer diversas rutas de entrada, cada una de ellas con una capacidad de tráfico máxima, dato que es conocido y expresado en miles de vehículos por hora.

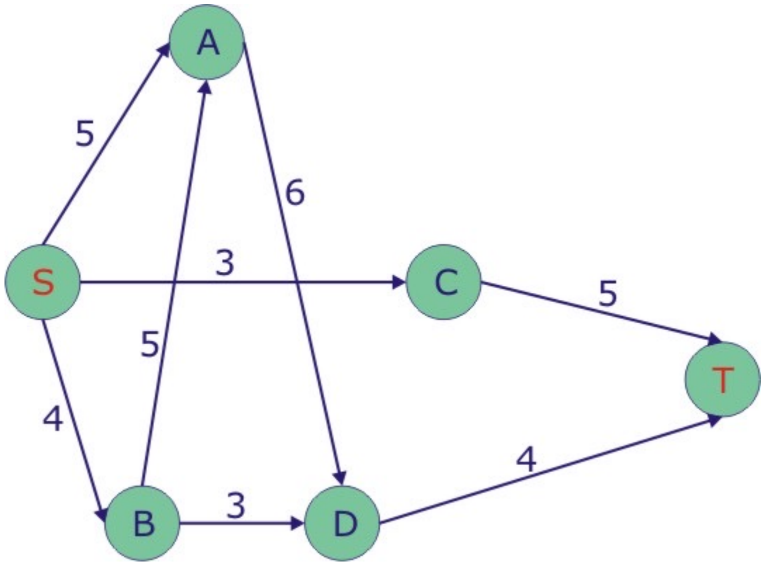
Problema: encontrar el flujo máximo de vehículos que pueden desplazarse del origen al destino en hora punta, y conocer cuál será el índice de utilización de cada una de las rutas de entrada.

Problema de optimización



Ejemplo 2: Maximizar la cantidad de agua transportada por un sistema de tuberías.

- Cada arco representa un tubo y el factor de peso es la capacidad en litros/minuto
- Los vértices representan puntos en los cuales las tuberías están unidas
- El agua es transportada de un tubo a otro.
- Dos nodos, s y t, representan la fuente de emisión de agua y el punto de consumo, respectivamente.
- **Problema: maximizar la cantidad de agua que fluye desde el vértice fuente hasta su punto de consumo.**



Concepto del flujo (ii)

- Formulación matemática de flujo
  - Se llama flujo a una función Fij definida en A (arcos) que verifica las propiedades:

1. ***COTA INFERIOR: Fij≥0 $\forall(i,j) \in A$***

POSITIVO

2. ***CONSERVACIÓN DE CANTIDAD TOTAL***

$$\boxed{\sum_i Fij - \sum_j Fji = 0 \quad \forall i \in V,\ i \neq s,\ i \neq t}$$

Lo que entra = lo que sale

- V es el conjunto de nodos
- s es el nodo inicial del flujo
- t es el nodo destino del flujo

3. ***COTA SUPERIOR: Fij≤Uij $\forall(i,j) \in A$***

- *Uij*=capacidad del arco *(i,j)*

Flujo <= Capacidad

Algoritmo del aumento del flujo: algoritmo de Ford-Fulkerson (i)

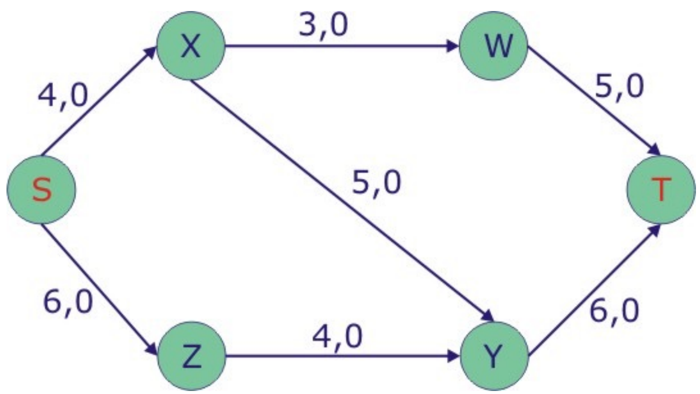
- Este es uno de los algoritmos más sencillos y a la vez eficientes para determinar el flujo máximo en una red, partiendo de un nodo fuente S y teniendo como destino el nodo sumidero T.
- La idea básica del algoritmo es partir de una función de flujo, *flujo cero*, e iterativamente ir mejorando el flujo.
- La mejora se da en la medida que el flujo de S a T aumenta, teniendo en cuenta las condiciones que ha de cumplir la función de flujo:
  - Flujo que entra=flujo que sale
  - El flujo no puede superar la capacidad del arco
- Los arcos de la red pueden clasificarse en 3 categorías:
  - No modificable: arcos cuyo flujo no puede aumentarse ni disminuirse, por tener capacidad 0 o tener un coste prohibitivo
  - Incrementable: arcos cuyo flujo puede aumentarse pues transportan un flujo inferior a su capacidad
  - Reducible: arcos cuyo flujo puede ser reducido

5. Problema del flujo de fluidos

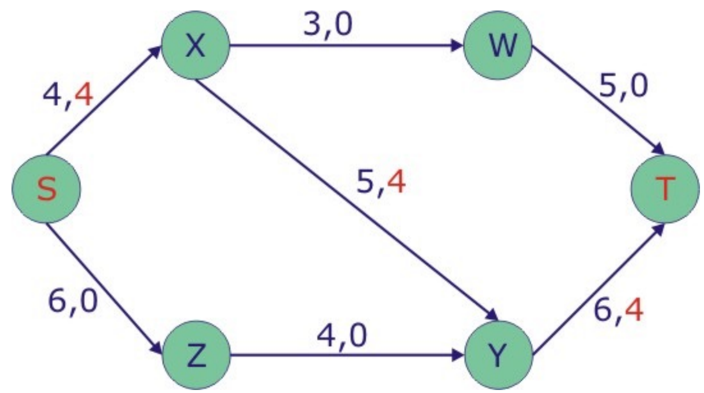
Algoritmo del aumento del flujo: algoritmo de Ford-Fulkerson (ii)

1. Parte de un flujo inicial `Fij=0`. Determina arcos incrementables y reducibles. Marca vértice fuente S.
2. Marcar una cadena de aumento de flujo, que vaya de la fuente al sumidero.
   - Aumentar el flujo al máximo aumento de flujo permitido por la cadena.
   - Actualizar los flujos de cada arco con las unidades de aumento.
     - Cadenas de arcos incrementables: `Mínimo { (Uij-Fij), ∀(i,j)∈P }`
     - Cadenas de arcos reducibles: `Mínimo { Fij, ∀(i,j)∈P’ }`
     - Cadenas de arcos incrementables-reducibles: mínimo de las dos cantidades anteriores
     - Este mínimo se llama **aumento de flujo máximo de la cadena**
   - Borrar todas las marcas, salvo la del vértice S, y repetir desde paso 2.
3. Repetir hasta que no existan cadenas de aumento de flujo.

Ford-Fulkerson: Ejemplo

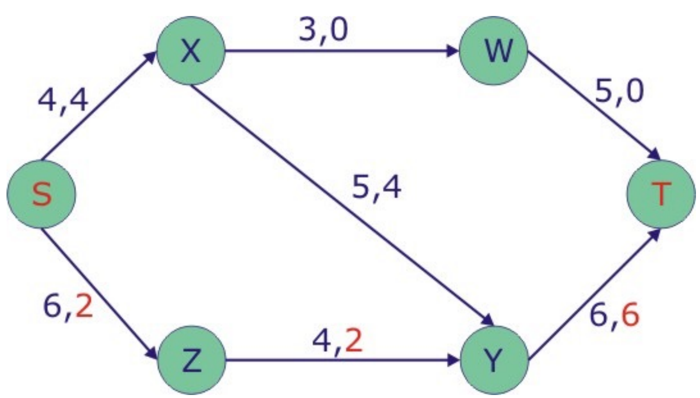


Fij=0



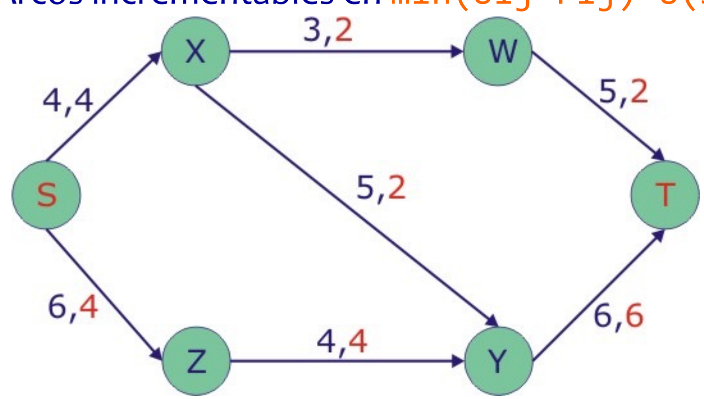
Cadena S-X-Y-T

Arcos incrementables en `min(Uij-Fij)=U(S,X)=4`



Cadena S-Z-Y-T

Arcos incrementables en `min(Uij-Fij) = = U(Y,T)-F(Y,T)=6-4=2`



Cadena S-Z-Y-X-W-T

Arcos incrementables: (S,Z), (Z,Y), (X,W), (W,T)

`min(Uij-Fij)=U(Z,Y)-F(Z,Y)=4-2=2`

Arcos reducibles: (X,Y)

`min Fij=4`

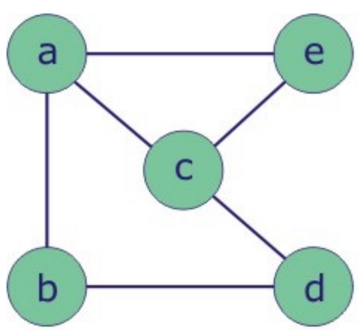
`MINIMO(2,4)=2`

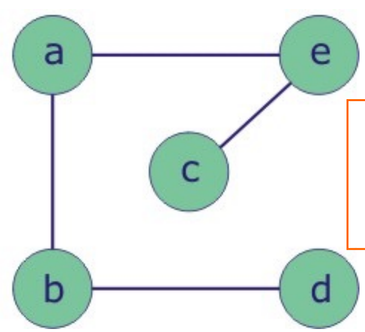
Flujo máximo=8

6. Árbol de expansión de coste mínimo

- El **cálculo del árbol de expansión de coste mínimo** se aplica sobre grafos **no dirigidos**, que se emplean para modelar relaciones simétricas entre entes, representados por los vértices del grafo.
- **Definiciones**:
  - **Red conectada**: red en la que cualquier par de vértices pueden unirse mediante un camino.
  - **Árbol de expansión**: árbol que contiene a todos los vértices de una red.
    - Buscar un árbol de expansión de una red es una forma de averiguar si la red es conectada.

Grafo no dirigido





Árbol de expansión:

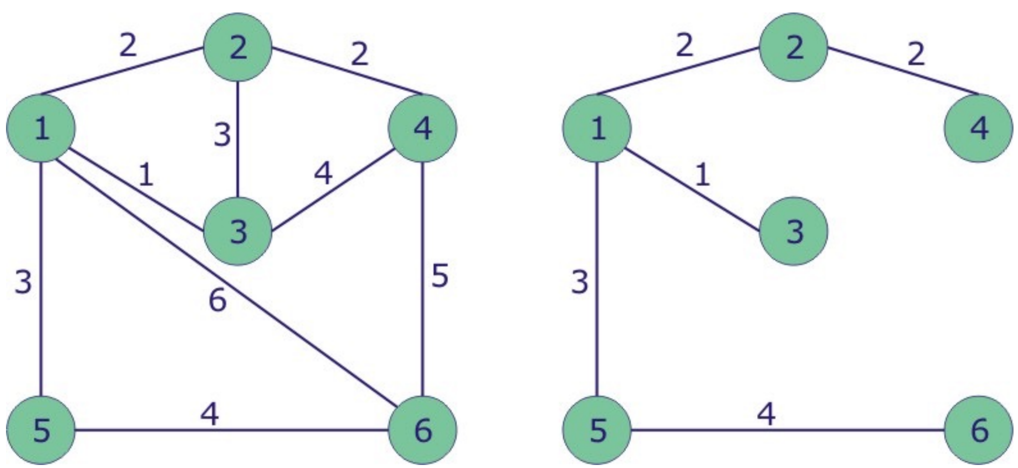
El grafo es una red conectada

6. Árbol de expansión de coste mínimo

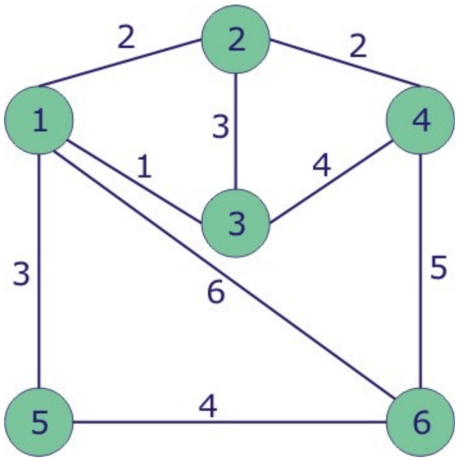
- **Árboles de expansión de coste mínimo:**
  - Aplicaciones: diseño de redes de comunicación en todas sus vertientes.
  - Ejemplo: Red telefónica
  - Ejemplo: Pueblos del ayuntamiento de Santiago:
    - Cada par de pueblos está conectado por un camino vecinal
    - El peso de un arco o camino directo entre dos pueblos viene dado por la distancia en km.
    - Se quiere construir una red de carriles-bici de tal forma que cada par de pueblos esté comunicado a un coste mínimo.
    - *El planteamiento, dado un grafo no dirigido ponderado y conexo, es encontrar el subconjunto del grafo compuesto por todos los vértices, con conexión entre cada par de vértices y sin ciclos, cuya suma de los costes de los arcos sea mínima.*

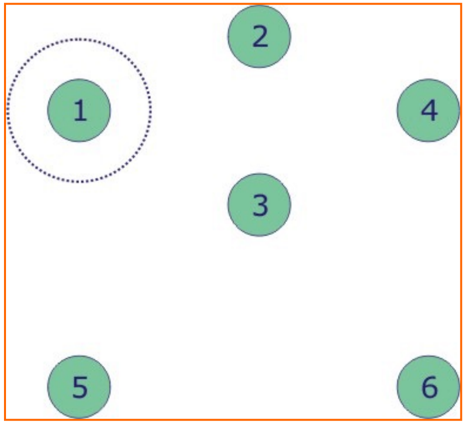
HAY QUE ENCONTRAR EL ÁRBOL DE EXPANSIÓN DE COSTE MÍNIMO

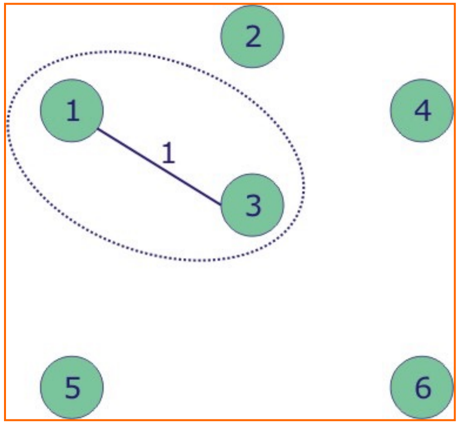
- **Algoritmo de Prim para encontrar el árbol de expansión de coste mínimo**
  - Algoritmo voraz (en cada paso se añade **el arco más corto** al árbol)
    - Se parte de un grafo G=(V,A) que es una red. Suponemos V={1, 2, … , n}.
    - Se asigna un vértice inicial a W: W={1}.
    - Se hace crecer el árbol de expansión, añadiendo en cada iteración otro vértice v∈V-W, tal que si u∈W, el arco (u,v) es el más corto.
    - El proceso termina cuando V=W.

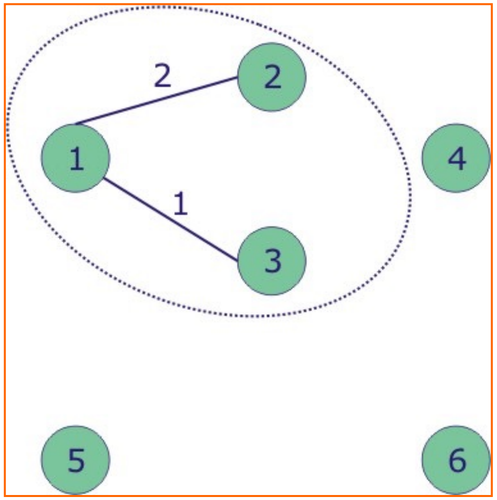


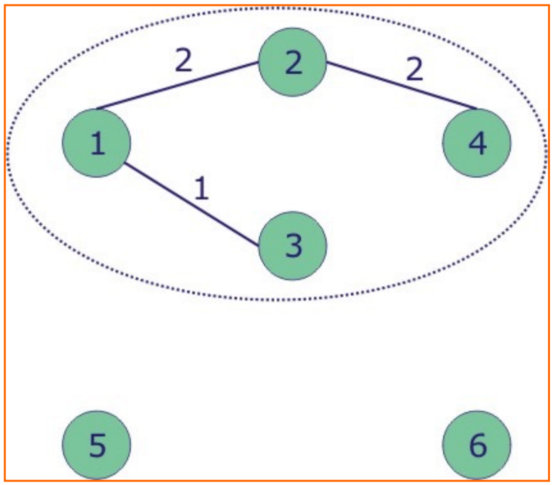
- Ejemplo paso a paso (Prim):

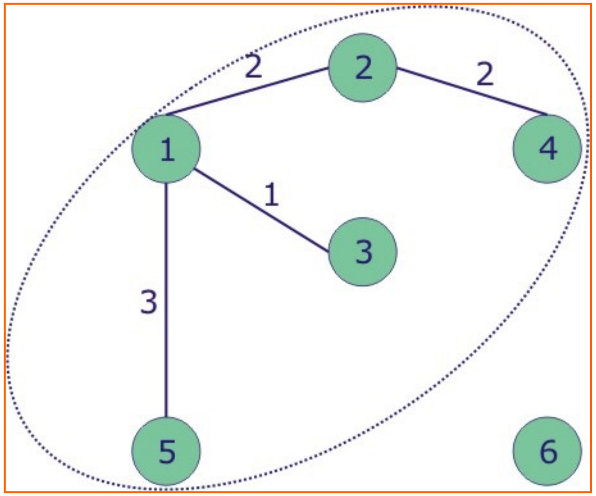


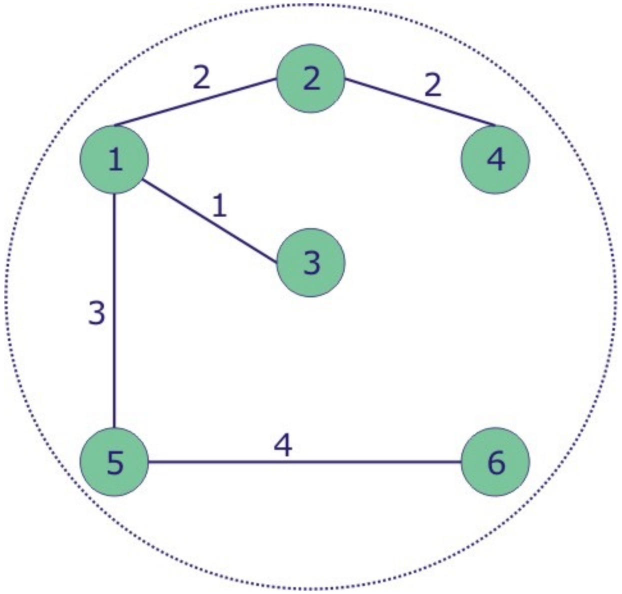












- ## Algoritmo de Kruskal para encontrar el árbol de expansión de coste mínimo
  - Se parte de un grafo G=(V,A) que es una red. Suponemos V={1, 2, … , n}.
    - El algoritmo comienza con un grafo T con los mismos vértices que V *pero sin arcos*.
    - Para construir componentes conexas cada vez mayores, se examinan los arcos de A en orden creciente de coste
      - Si el arco conecta 2 vértices pertenecientes a componentes conexas distintos, se añade el arco a T.
      - En otro caso, se descarta el arco, pues puede provocar un ciclo.
    - Cuando todos los vértices estén en un solo componente, T es un árbol de expansión de coste mínimo del grafo G.

- Ejemplo paso a paso (Kruskal):