Algorítmica

Capítulo 3. Algoritmos Greedy

Tema 8. Greedy sobre grafos

- Algoritmos para el árbol generador minimal: Algoritmos de Kruskal y Prim
- Algoritmos para caminos mínimos: Algoritmo de Dijkstra

El Problema del Arbol Generador Minimal

- Se trata de encontrar el árbol generador minimal de un grafo, es decir, el árbol generador de mínima longitud.
- Suponemos un grafo conexo $G = (V,A)$, sobre el que hay definida una matriz de pesos $L(i,j) \geq 0$.
- Queremos encontrar un árbol $T \subseteq A$ tal que todos los nodos permanezcan conectados cuando solo se usen aristas de T, siendo la suma de los pesos de sus aristas mínima.
- Al grafo (V,T) se le llama Árbol Generador Minimal del grafo G.

El Problema del Arbol Generador Minimal

Joseph B. Kruskal investigador del Math Center de los Laboratorios Bell, en 1956 descubrió su algoritmo para la resolución del problema del Árbol Generador Minímal. Este problema es un problema típico de optimización combinatoria, que fue considerado originalmente por Otakar Boruvka (1926) mientras estudiaba la necesidad de electrificación rural en el sur de Moravia en Checoslovaquia.

- Las aplicaciones de este problema lo hacen muy importante.
  - Diseño de redes físicas.
    - teléfonos, eléctricas, hidraulicas, TV por cable, computadores, carreteras, ...
  - Análisis de Clusters.
    - Eliminación de aristas largas entre vertices irrelevantes
    - Búsqueda de cúmulos de quasars y estrellas
  - Solución aproximada de problemas NP.
    - PVC, árboles de Steiner, ...
  - Distribución de mensajes entre agentes
  - Aplicaciones indirectas.
    - Plegamiento de proteínas, Reconocimiento de celulas cancerosas, ...

El Problema del Arbol Generador Minimal

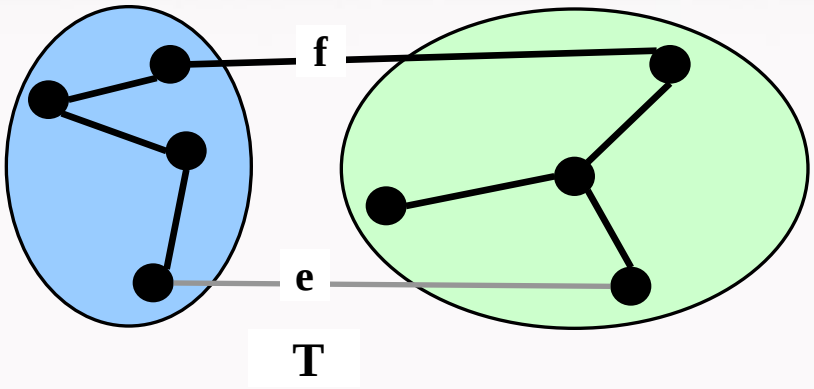
- El problema es resoluble con el enfoque greedy:
- Tenemos una lista de aristas: A partir de ella pueden darse las listas de candidatos o no a solución.
- Una solución será un conjunto de aristas que forme un AGM.
- La condición de factibilidad es que la arista que se vaya a incluir no forme un ciclo con las ya incluidas.
- El criterio de selección será escoger en cada momento la arista de mínimo peso.
- El objetivo es que la suma de los pesos de las aristas en el árbol sea mínima.

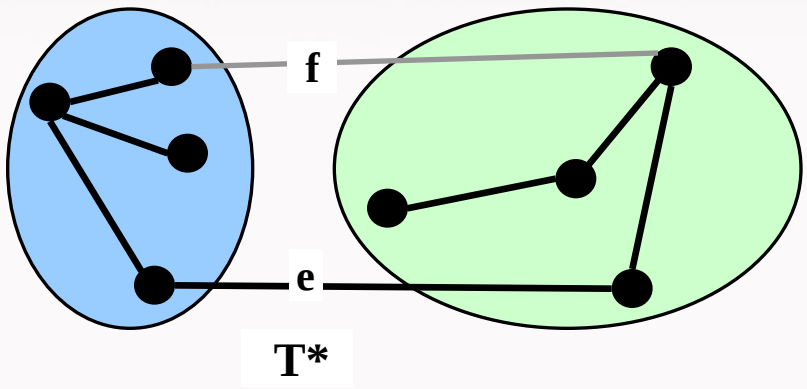
El Problema del Arbol Generador Minimal

- Se verifica la **Propiedad del AGM**:
- Sea G = (V,A) un grafo no dirigido y conexo donde cada arista tiene una longitud conocida. Sea $U \subseteq V$ un subconjunto propio (lo que significa que U no puede coincidir con V) de los nodos de G. Si (u,v) es una arista tal que $u \in U$ y $v \in V - U$ y, además, es la arista del grafo que verifica esa condición con el menor peso, entonces existe un AGM T que incluye a (u,v).
- Esta propiedad garantizará que el algoritmo greedy que diseñemos proporcione la solución óptima del problema, y por tanto que funcione correctamente.

Demostración de la propiedad del AGM

- **Demostración por contradicción**: Supongamos que T es un AGM que no contiene a e = (u,v)





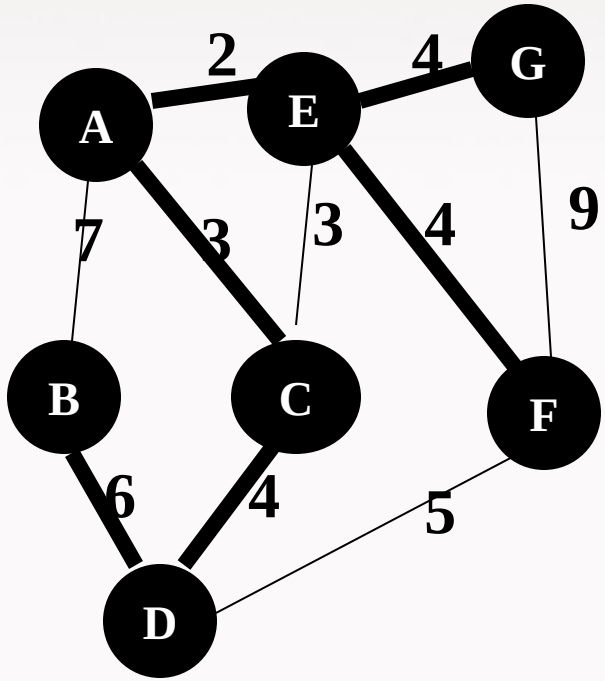
- Si añadimos e a T se crea un ciclo C. Si rompemos ese ciclo (eliminando una arista f conectando U y V), obtenemos un nuevo AGT* que (por incluir a e) tendrá menor longitud que T. Eso es una contradicción

El Problema del Arbol Generador Minimal

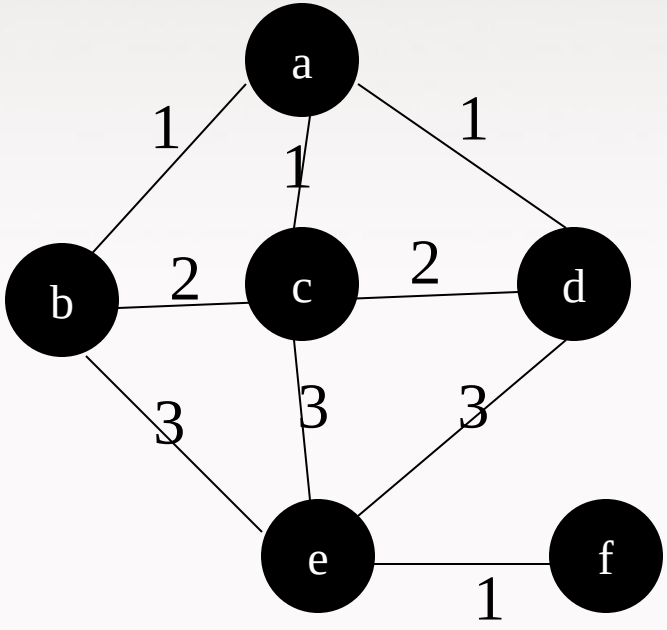
Algoritmo de Kruskal

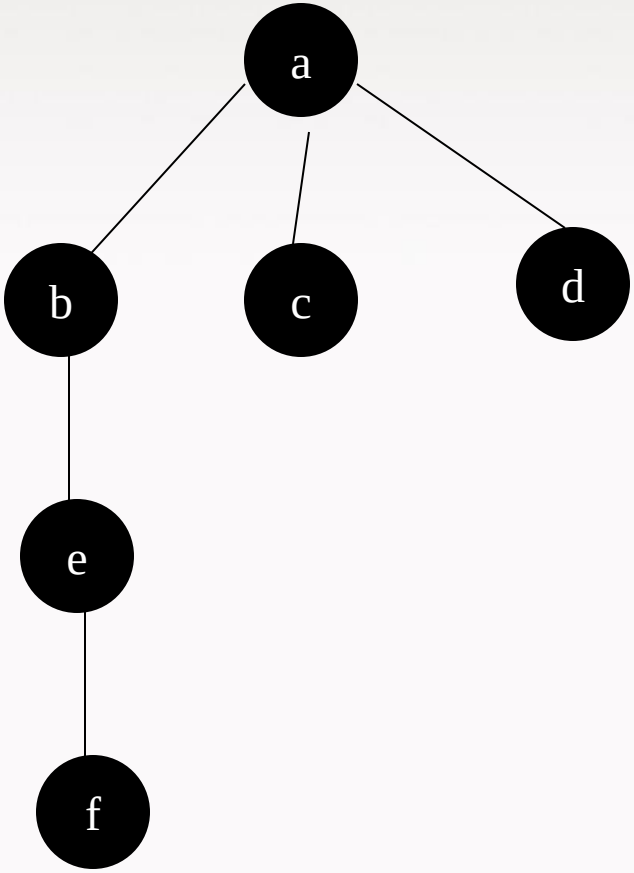
- Ordenar las aristas de forma creciente de costo
- Repetir
  - Coger la arista mas corta.
  - Borrar la arista de E
  - Aceptar la arista si no forma un ciclo en el arbol parcial,

    Rechazarla en caso contrario,
- Hasta que tengamos |V|-1 aristas correctas.
  - Si nuestro grafo tiene n vértices y a aristas, el tiempo de este algoritmo es O(a log a)



Ejemplo





(a,b), (a,c), (a,d), (e,f), (b,c), (c,d), (b,e), (c,e), (d,e)

Implementación del algoritmo

FUNCION KRUSKAL (G: GRAFO): Conjunto de aristas.

Ordenar las aristas por longitudes crecientes

N = |V|

T = ∅

Inicializar N conjuntos (1 elemento cada)

Repetir

{U,V} = arista mas corta aun no considerada

COMP U = BUSCA (U)

COMP V = BUSCA (V)

SI COMPU ≠ COMPV ENTONCES

UNIR (COMPU, COMPV)

T = T ∪ ({U,V})

Hasta que |T| = N – 1

DEVOLVER (T)

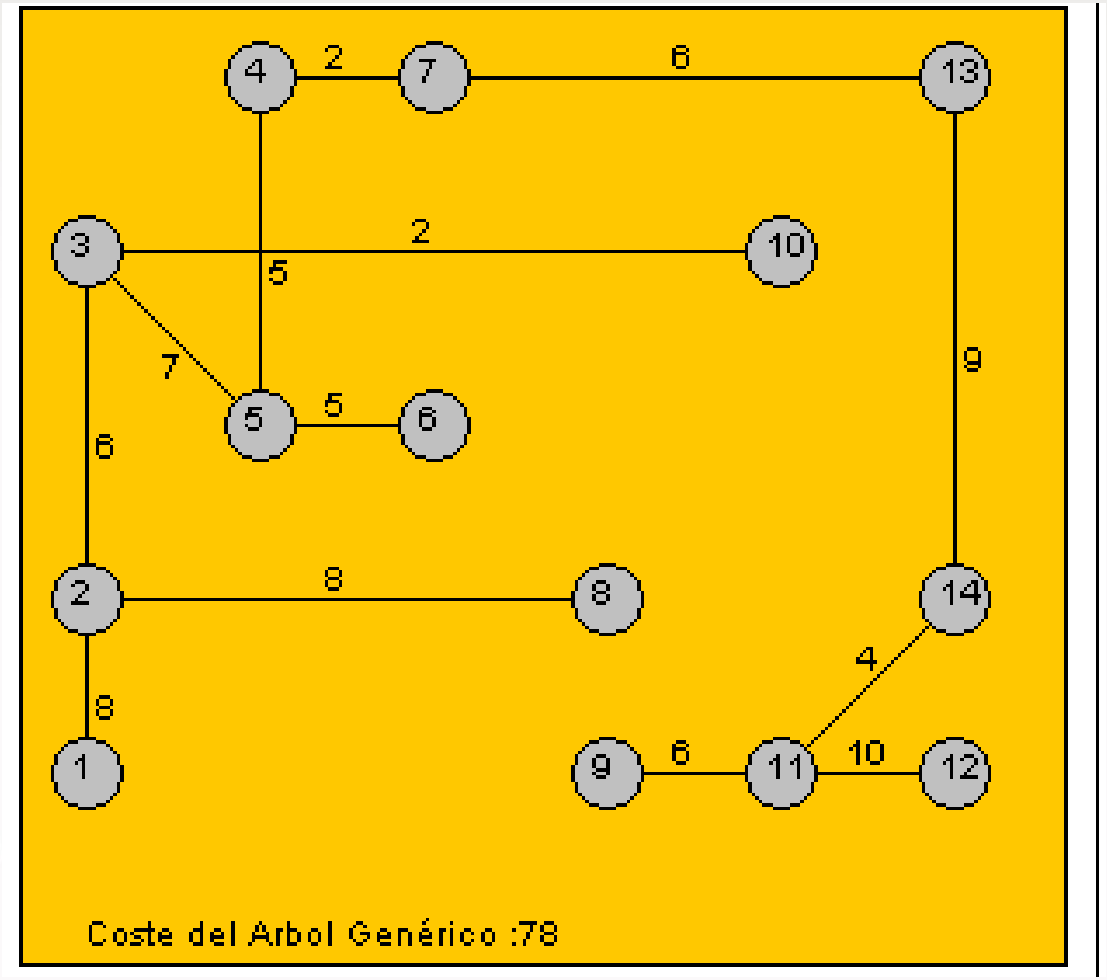
Análisis del algoritmo

- Donde:
  - Ordenar las aristas es O(a log a) para “a” aristas.
  - N es el número de vértices en el grafo.
  - T es el conjunto donde construiremos AGM.
  - BUSCA ( ) lleva a cabo la búsqueda de la arista que se considere y es (si se hace con búsqueda binaria) O (log n).
  - UNION es O (log n) y lleva a cabo la unión de dos conjuntos.
- El orden del algoritmo de Kruskal es O (a log a), pero como tenemos n vértices, y en un grafo conexo siempre se verifica:

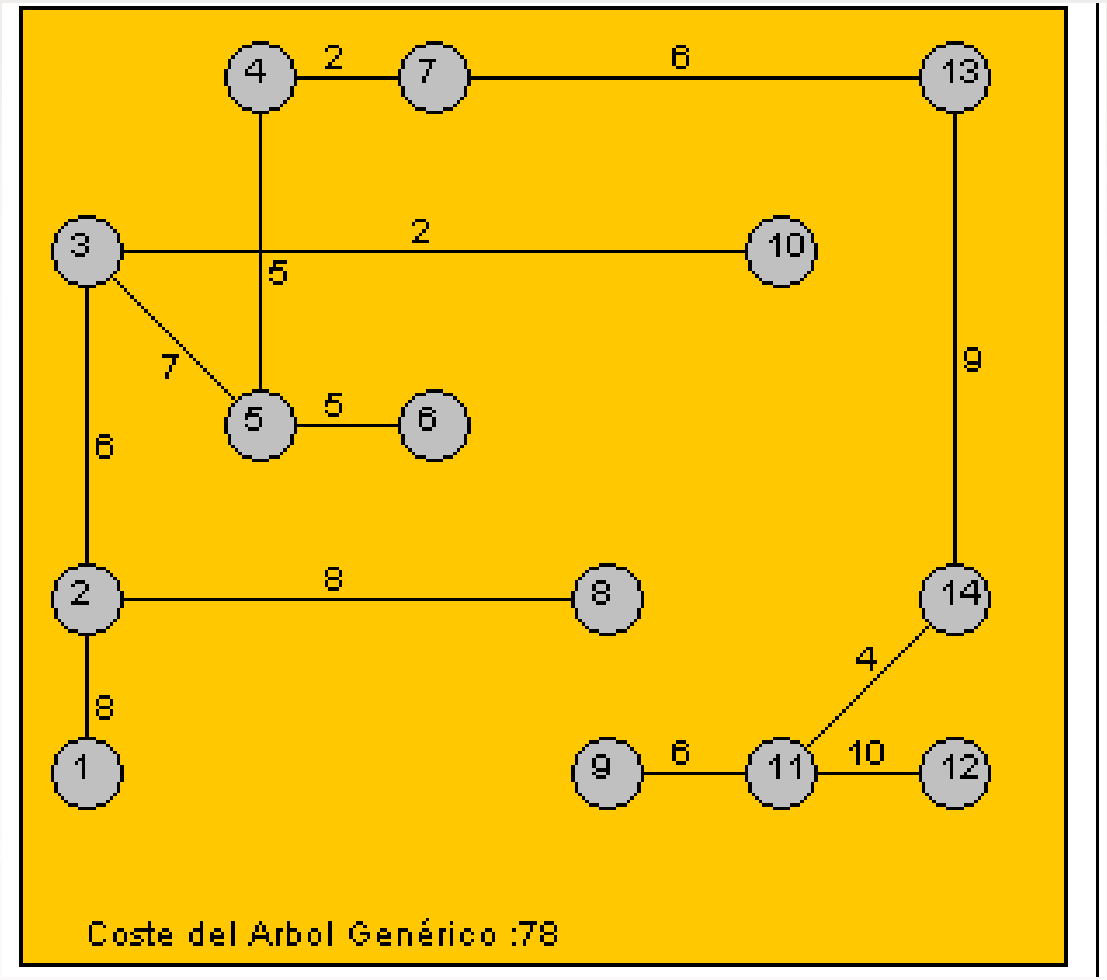
$$n-1 \leq a < \frac{n(n-1)}{2}$$

se tiene que el algoritmo es O(a log n)

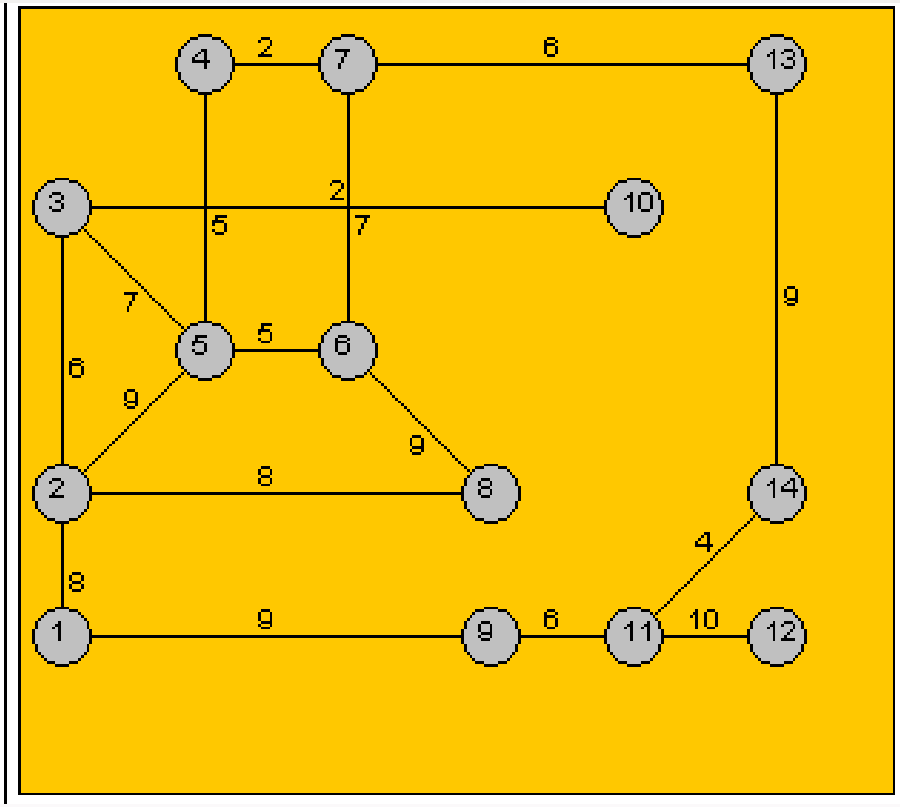
Ejemplo-1

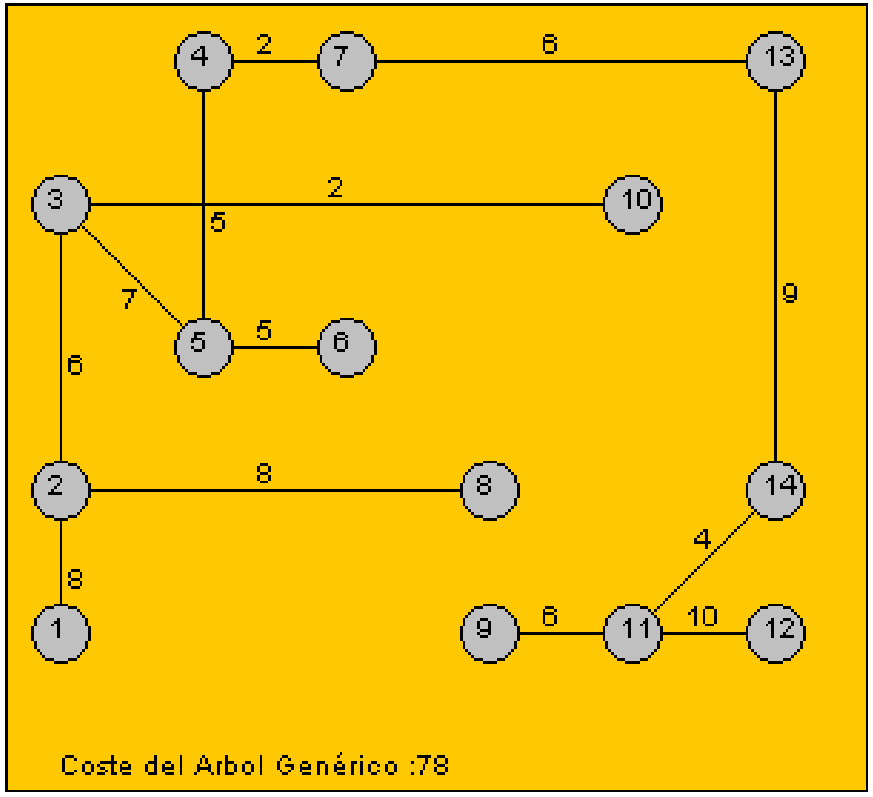

4
2
7
6
13
3
2
10
5
7
9
5
5
6
6
2
8
8
14
4
8
1
9
6
11
10
12
Coste del Arbol Genérico :78

Ejemplo-1

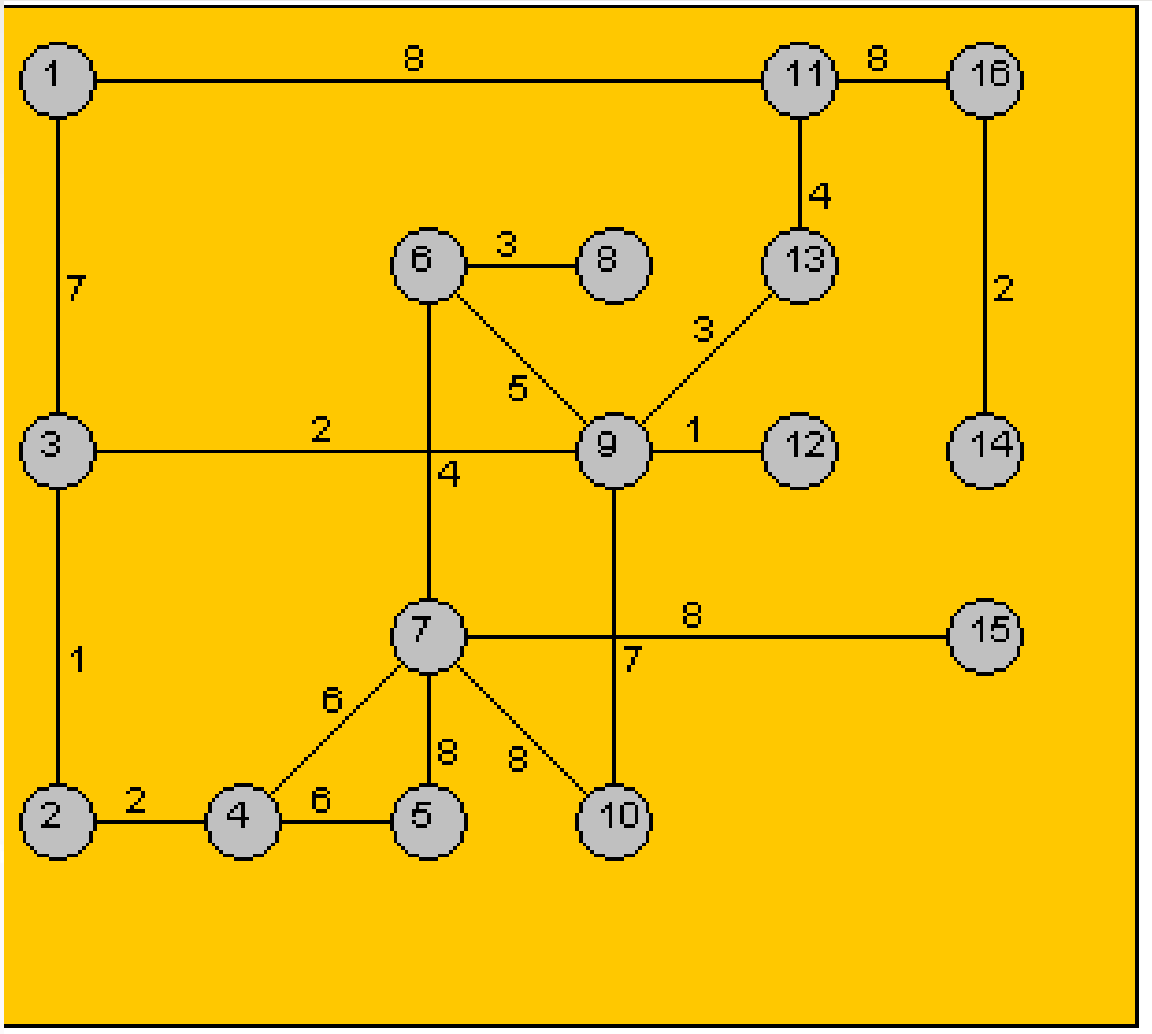

4
2
7
6
13
3
2
10
5
7
9
5
5
6
6
2
8
8
14
4
8
1
9
6
11
10
12
Coste del Arbol Genérico :78

Ejemplo-1

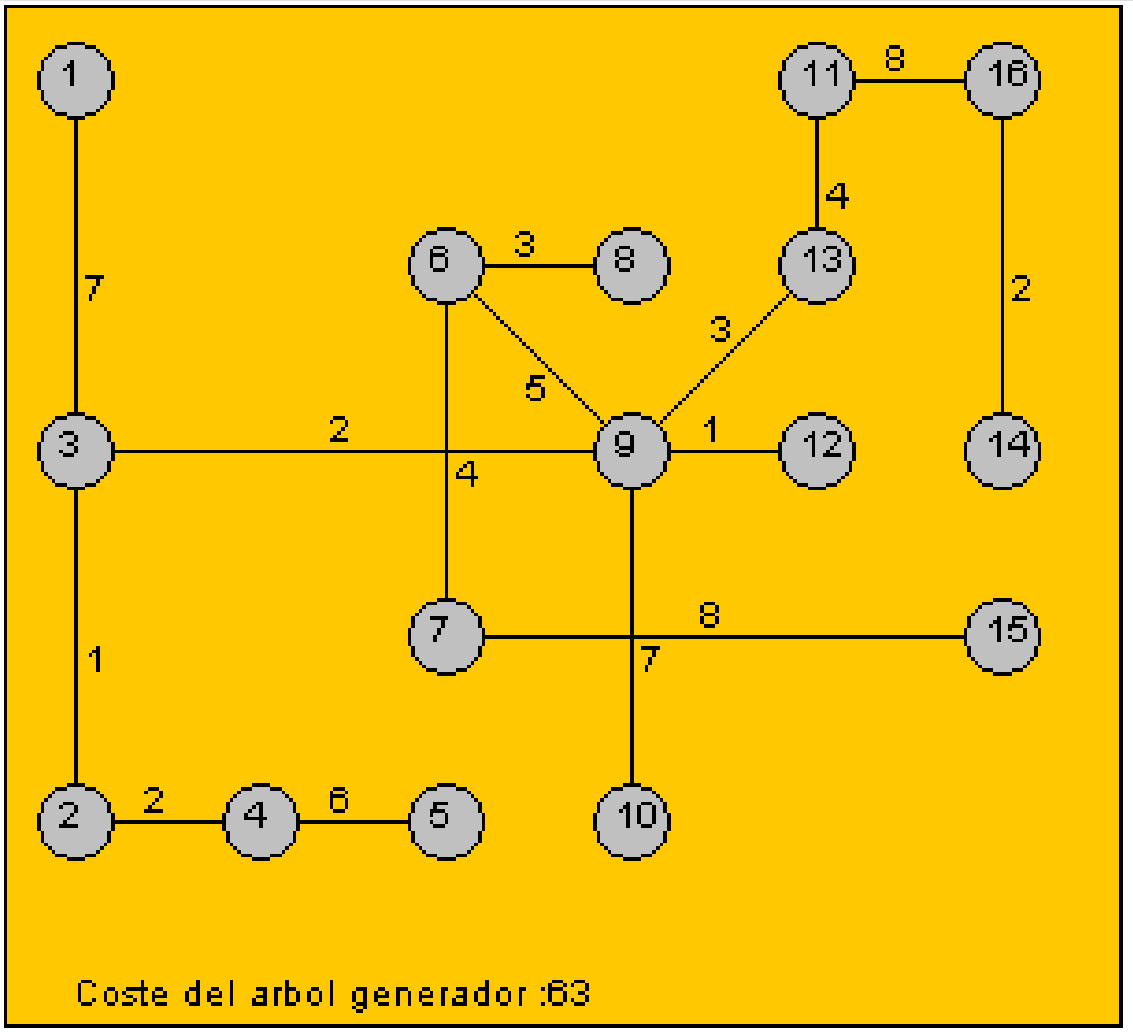

4
2
7
6
13
3
2
10
5
7
7
9
5
5
6
6
9
9
2
8
8
14
4
8
1
9
9
6
11
10
12


4
2
7
6
13
3
2
10
5
7
9
5
5
6
6
2
8
8
14
4
8
1
9
6
11
10
12
Coste del Arbol Genérico :78

Ejemplo-2

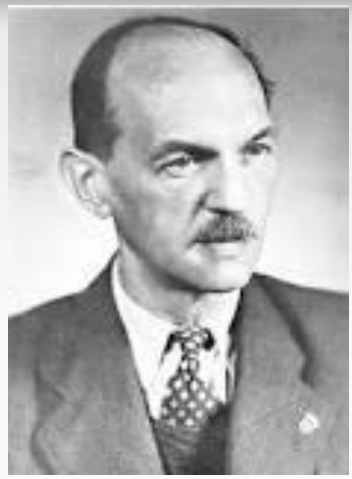


Ejemplo-2


1
11
8
16
4
6
3
8
13
7
2
3
5
3
2
9
1
12
14
4
7
8
15
1
7
2
2
4
6
5
10
Coste del arbol generador :63

Algoritmo de Prim

- Es otro método de resolver el problema del AGM pero también se basa en la construcción del algoritmo en función de la propiedad del AGM.
- En el algoritmo de Kruskal partíamos de la selección de la arista más corta que hubiera en la lista de aristas, lo que implica un crecimiento desordenado del AGM.
- Para evitarlo, el algoritmo de Prim propone que el crecimiento del AGM sea ordenado (a partir de una raíz).
- El algoritmo fue diseñado en 1930 por el matemático checo Vojtech Jarnik y luego de forma independiente por Robert C. Prim en 1957



Vojtech Jarnik

Robert C. Prim

Algoritmo de Prim

- La idea del algoritmo de Prim es la siguiente:
  - Se toma un conjunto U de nodos, que inicialmente contiene al nodo raíz.
  - Formamos el conjunto T de soluciones (aristas).
  - En cada etapa el algoritmo búsca la arista más corta que conecta U con V – U, siendo V el conjunto de candidatos.
  - Añade el vértice obtenido al conjunto U y la arista obtenida a T.
  - En cada instante, las aristas que están en T constituyen un AGM para los nodos que están en U.
  - Esto lo hacemos hasta que U = n.

Implementación del Algoritmo de Prim

FUNCION PRIM (G = (V, A)) conjunto de aristas.

(Inicialización)

T = ∅ (Contendrá las aristas del AGM que buscamos).

U = un miembro arbitrario de V

MIENTRAS $|U| = N$ HACER

BUSCAR e = (u,v) de longitud mínima tal que

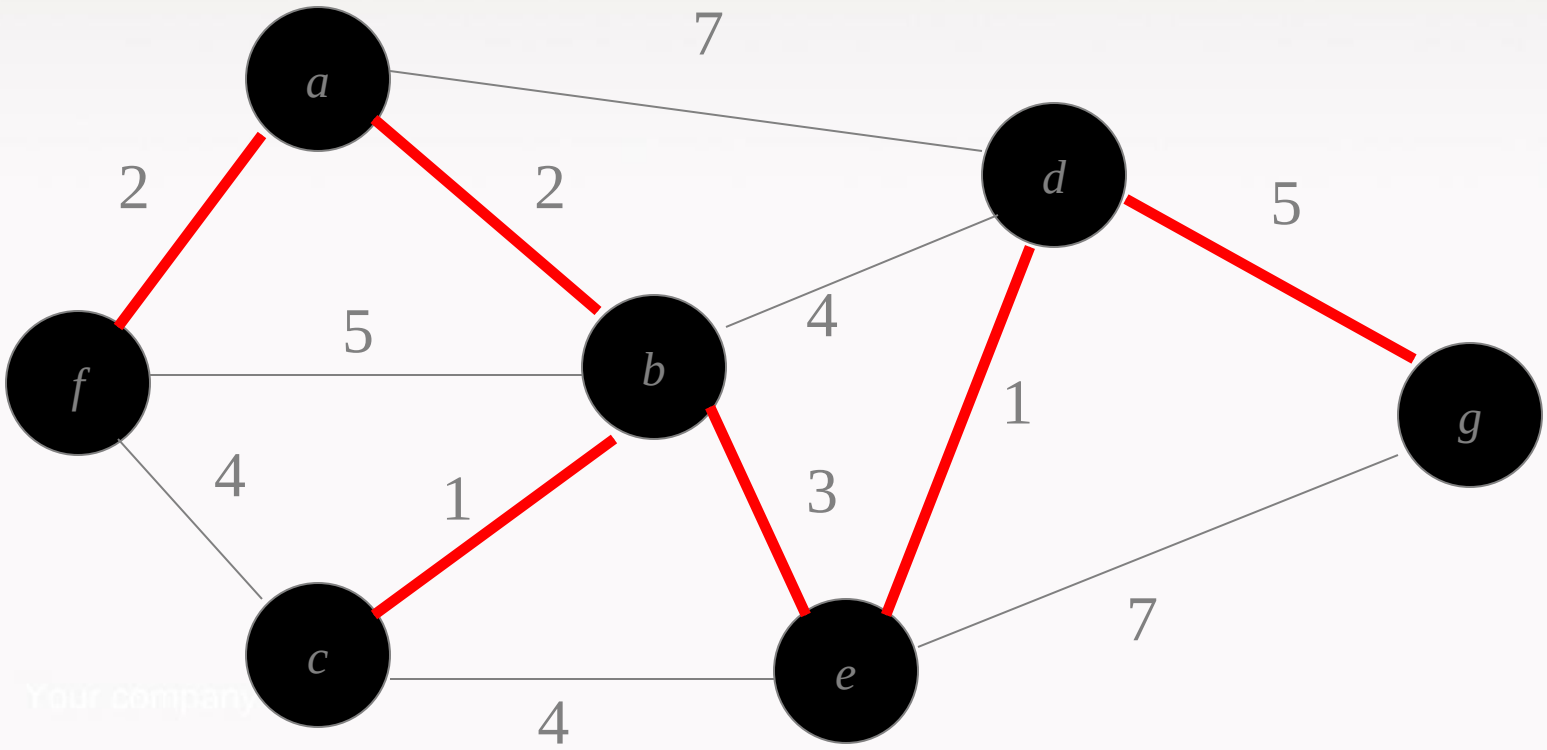
$u \in U$ y $v \in V - U$

T = T + e

U = U + v

DEVOLVER (T)

Ejemplo del Algoritmo de Prim



Implementación del Algoritmo de Prim

- Para estudiar la eficiencia de Prim, es necesario elaborar un poco más su implementación, por lo que suponemos:
- L [I , J] ≥ 0 una matriz de distancias.
- MasProximo [x] es un vector que nos da el nodo U que está más cercano al vértice x.
- DistMin [x] es un vector que nos da la distancia entre x y MasProximo [x].

Implementación del Algoritmo de Prim

```
Funcion Prim (L[1...n, 1...n]: conjunto de aristas
{al comienzo solo el nodo 1 se encuentra en U}
    T = ∅ (contendrá las aristas del AGM)
    Para i = 2 hasta n hacer
        MasProximo [i] = 1; DistMin [i] = L[i, 1]
    Repetir n – 1 veces
        min = ∞
        Para j = 2 hasta n hacer
            Si 0 ≤ DistMin [j] < min entonces min = DistMin [j]
        T = T + (MasProximo [k], k)
        DistMin [k] = -1 (estamos añadiendo k a U)
        para j = 2 hasta n hacer
            si L [j, k] < DistMin [j] entonces
                DistMin [k] = L [j, k];
                MasProximo [j] = k
    Devolver T.
```

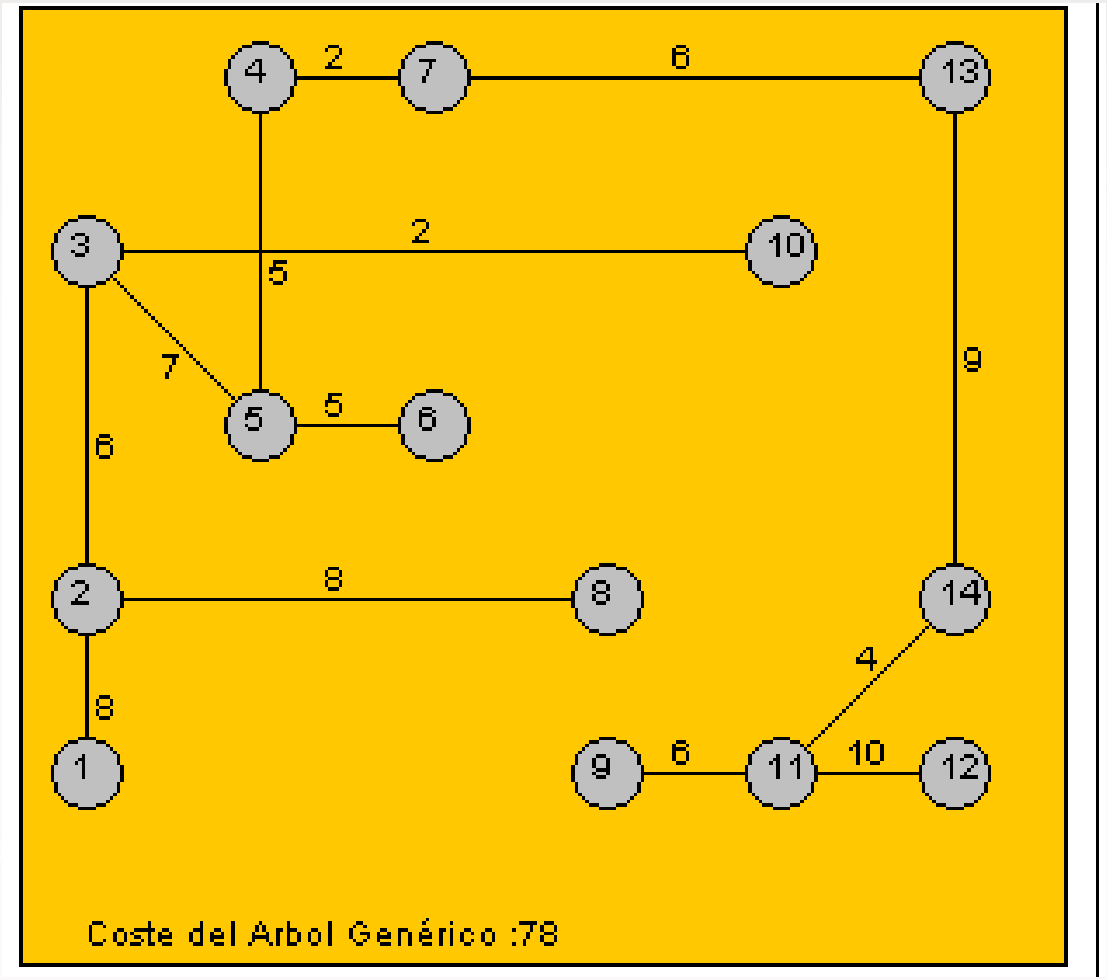
Análisis del algoritmo de Prim

- El bucle principal del algoritmo se ejecuta n – 1 veces.
- En cada iteración, el bucle “para” anidado requiere un tiempo O(n). Por tanto, el algoritmo de Prim requiere un tiempo $O(n^2)$.
- Como el Algoritmo de Kruskal era O(a log n), siendo a el numero de aristas del grafo,
- ¿Que algoritmo usar Prim o Kruskal?
- Sabemos que

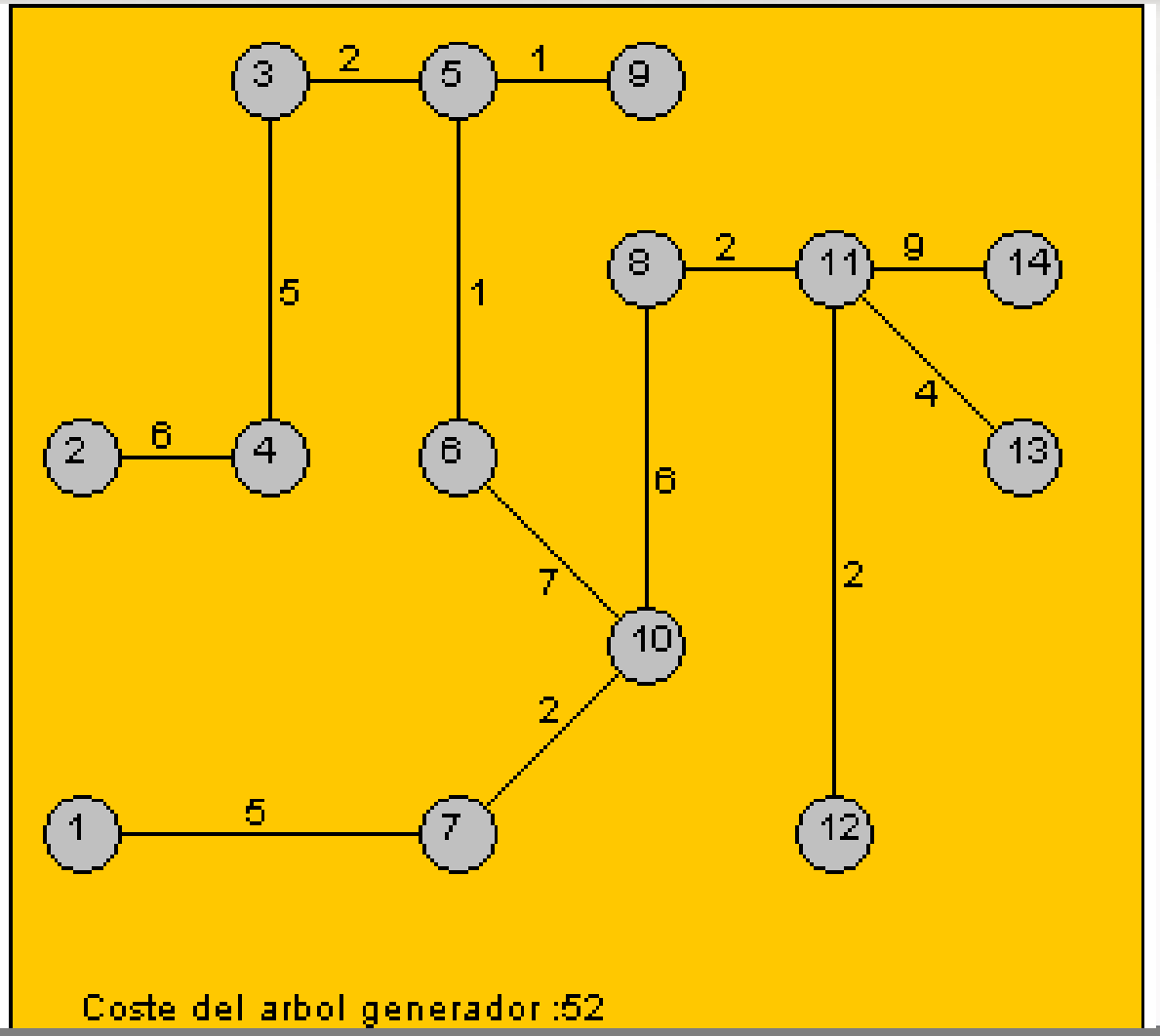
$$n-1 \leq a < \frac{n(n-1)}{2}$$

- Luego en grafos poco densos, lo mejor seria emplear Kruskal, y si el grafo es muy denso, entonces el de Prim

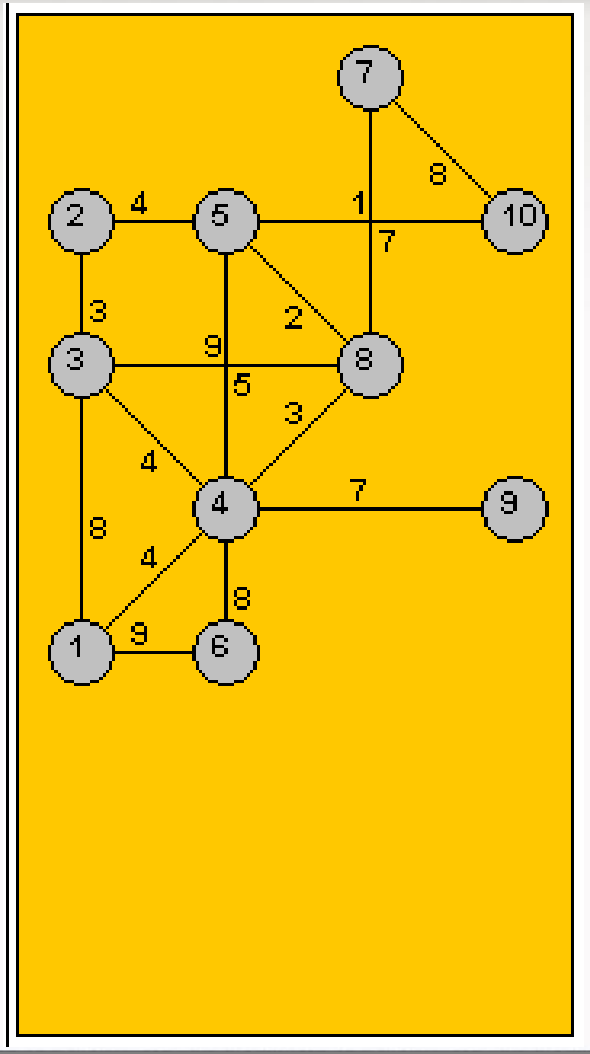
Ejemplo-1


4
2
7
6
13
3
2
10
5
7
9
5
5
6
6
2
8
8
14
4
8
1
9
6
11
10
12
Coste del Arbol Genérico :78

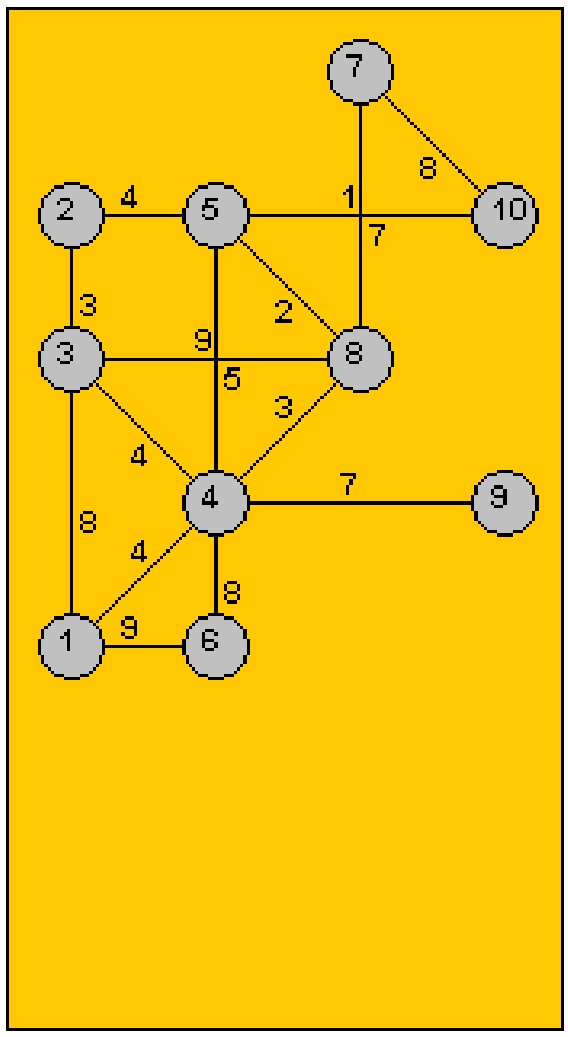
Ejemplo-1


3
2
5
1
9
5
1
8
2
11
9
14
4
2
6
4
6
13
6
7
2
10
2
1
5
7
12
Coste del arbol generador :52

Ejemplo-2: Prim usando matriz de adyacencia



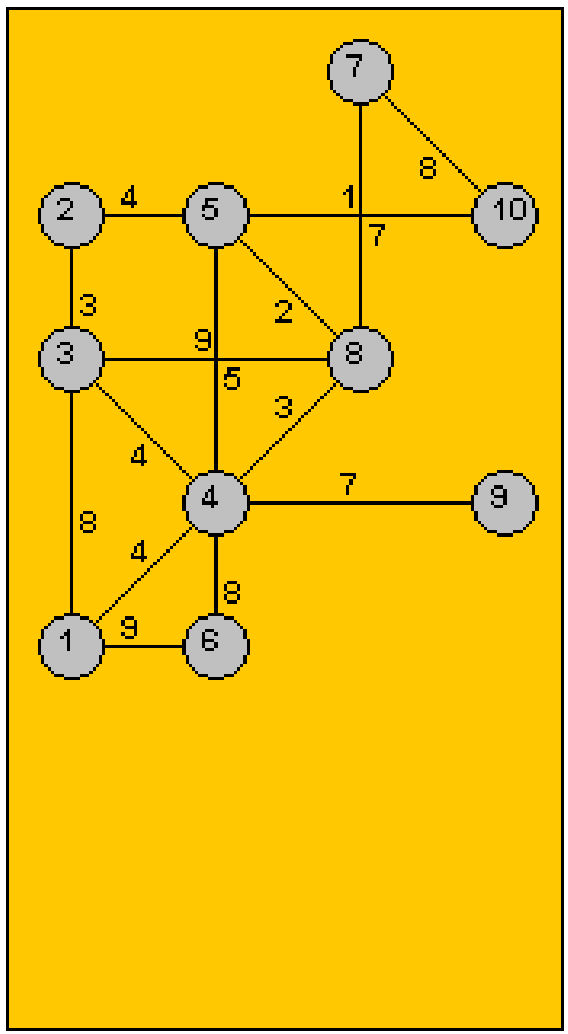
Ejemplo-2: Prim usando matriz de adyacencia



Matriz de adyacencia

	1	2	3	4	5	6	7	8	9	10	
1			8	4		9					✓
2			3		4						✓
3	8	3		4				9			✓
4	4		4		5	8		3	7		✓
5		4		5				2		1	✓
6	9			8							✓
7								7		8	✓
8			9	3	2		7				✓
9				7							✓
10					1		8				✓

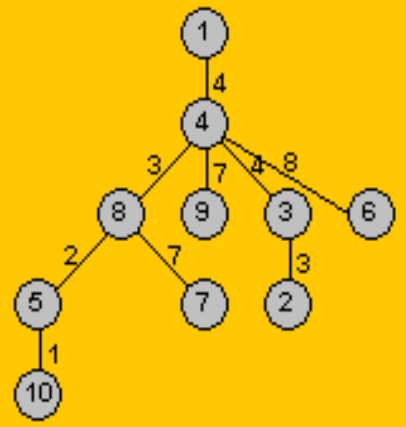
Ejemplo-2: Prim usando matriz de adyacencia



Matriz de adyacencia

	1	2	3	4	5	6	7	8	9	10	
1			8	4		9					✓
2			3		4						✓
3	8	3		4				9			✓
4	4		4		5	8		3	7		✓
5		4		5				2		1	✓
6	9			8							✓
7								7		8	✓
8			9	3	2		7				✓
9				7							✓
10					1		8				✓

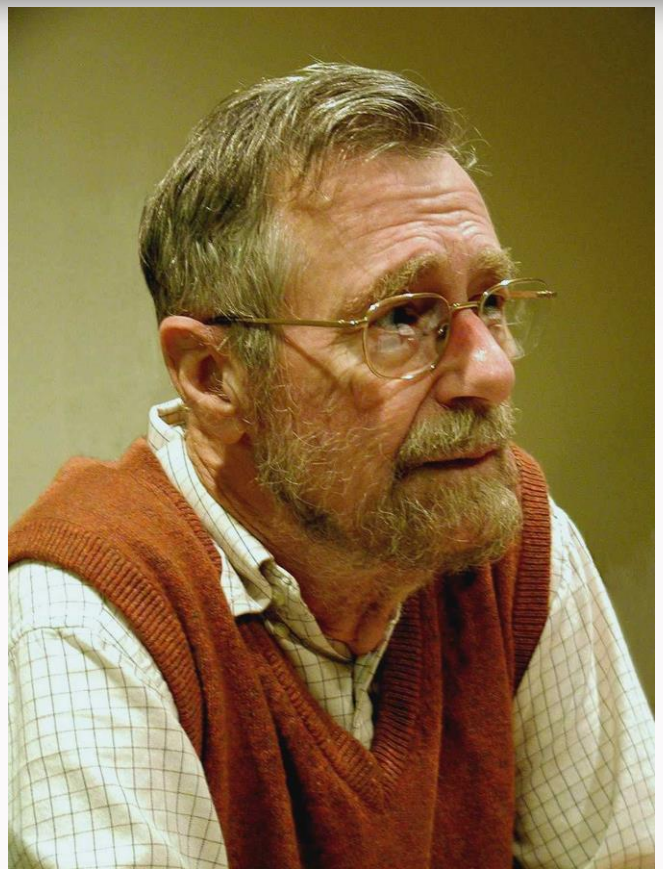
Árbol Generador



Coste del árbol genérico : 39

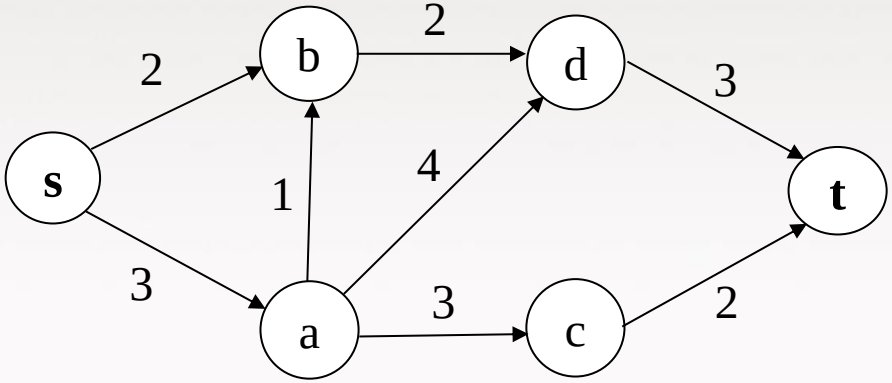
Algoritmos de Camino Mínimo

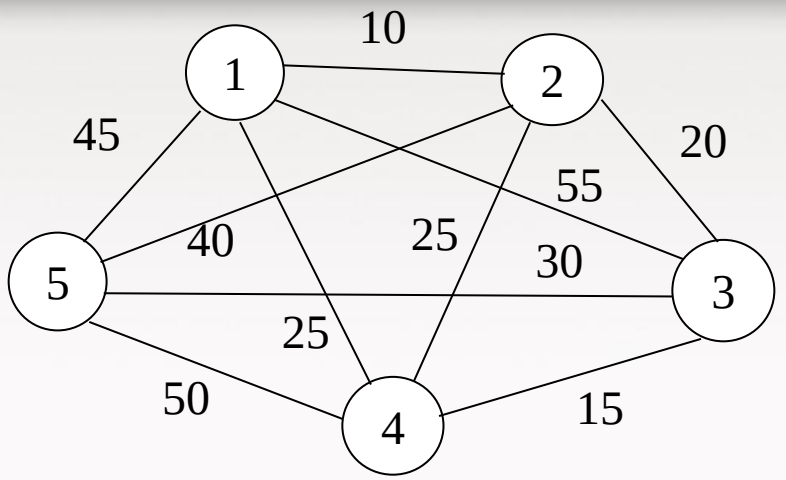
- El problema de determinar el camino mínimo (de longitud mínima) entre dos vértices de un grafo es de una importancia extraordinaria en todas las ramas de las Ingenierías, y en particular en Inteligencia Artificial
- El algoritmo que resuelve este problema de manera mas eficiente es el conocido **Algoritmo de Dijkstra**, que diseño el mismo Edsger W. Dijkstra en 1959, cuando tenia 29 años
- Murió el 6 de agosto de 2002.

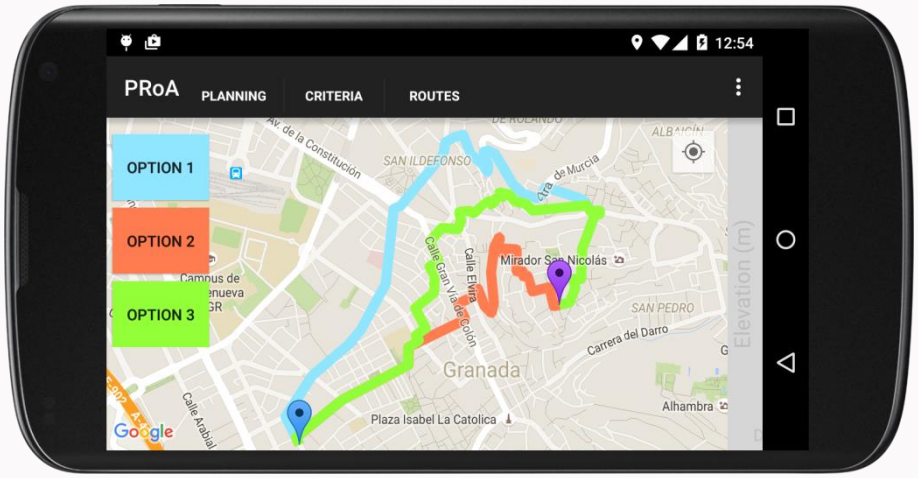


“El esfuerzo de utilizar las máquinas para emular el pensamiento humano siempre me ha parecido bastante estúpido. Preferiría usarlas para emular algo mejor.” (Edsger W. Dijkstra)

Algoritmos de Camino Mínimo







El problema del Camino Mínimo

- Suponemos:
- Un grafo dirigido G = (V,E) con E un conjunto de arcos y V un conjunto de vértices.
    - Una distancia definida entre los nodos que viene dada por una matriz L[1...n, 1...n] ≥ 0 .
- Se trata de hallar la distancia de los caminos mínimos desde un nodo raíz (el nodo 1) a todos los demás.
- Es un problema típicamente Greedy, en el que se identifican fácilmente las seis características.
- La aplicación del enfoque greedy conduce al **Algoritmo de Dijkstra,** del que las principales características son las siguientes

Algoritmo de Dijkstra

- Llamamos S al conjunto de los nodos elegidos. S contendrá los nodos cuya distancia desde el origen es mínima
- Inicialmente, S solo contiene el origen y cuando finalice el algoritmo, S contendrá todos los nodos del grafo.
- En cada etapa, elegiremos aquel nodo cuya distancia al origen sea menor, para ponerlo en S.

Algoritmo de Dijkstra

- Diremos que un camino del origen a otro nodo es **especial** si todos los nodos intermedios están en S.
- En cada etapa del algoritmo se emplea un vector D que contiene la longitud del camino especial más corto a cada nodo del grafo, de manera que en cada etapa, como añadimos un vértice nuevo a S, todos los valores de D[V] se actualizan.
- Para la actualización vemos si con el nuevo vértice introducido en S podemos llegar al vértice v por un camino de longitud más corta que el que había. Si es posible, se actualiza D[V] .

Algoritmo de Dijkstra

```
FUNCION DIJKSTRA
    C = {2, 3, ..., N }
    PARA I = 2 HASTA N HACER D[I] = L [1, I]
    I = 2, ..., N

    REPETIR N – 2 VECES
        V = algún elemento de C que minimice D[V]
        C = C – (V)
        PARA CADA w ∈ C HACER
            SI D[w] > D[V] + L[v, w] ENTONCES
            D[w] = D[V] + L[v ,w]

    DEVOLVER D
```

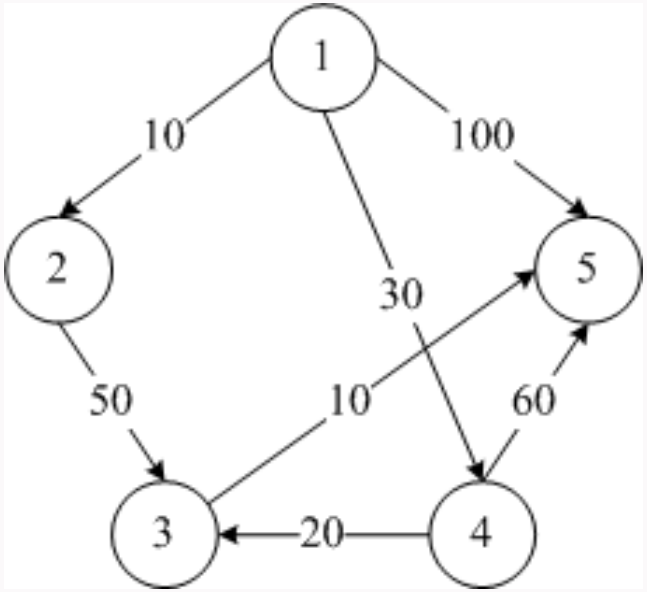
Algoritmo de Dijkstra

```
FUNCION DIJKSTRA
    C = {2, 3, ..., N }
    PARA I = 2 HASTA N HACER D[I] = L [1, I]
    I = 2, ..., N
    P[I] = 1
    REPETIR N – 2 VECES
        w = algún elemento de C que minimice D[V]
        C = C – (w)
        PARA CADA v ∈ C HACER
            SI D[v] > D[w] + L[w,v] ENTONCES
            D[v] = D[w] + L[w,v]
            P[v] = w
    DEVOLVER D
```

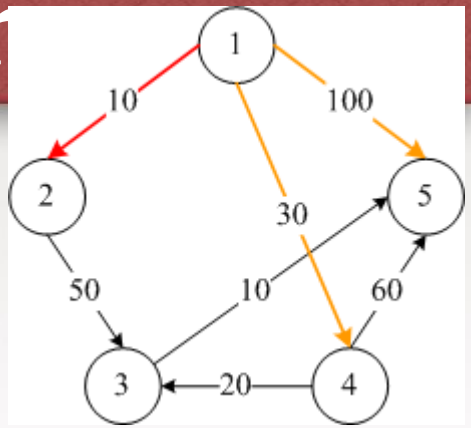
Donde P es un vector que nos permite conocer por donde pasa cada camino de longitud minima desde el origen

Algoritmo de Dijkstra

- Encontrar los caminos más cortos entre el vértice 1 y todos los demás del siguiente grafo dirigido.

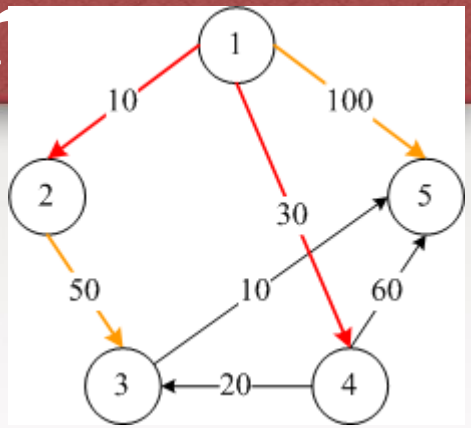


Algoritmo de Dijkstr



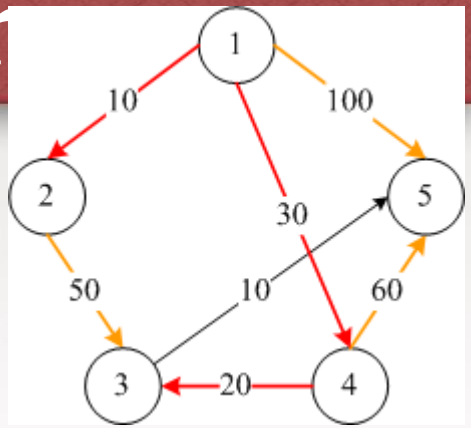
Iteración	S	w	D[2]	D[3]	D[4]	D[5]
Inicial	{1}	—	10	∞	30	100

Algoritmo de Dijkstr



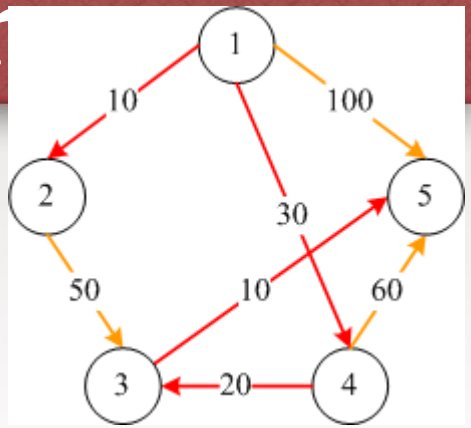
Iteración	S	w	D[2]	D[3]	D[4]	D[5]
Inicial	{1}	—	10	∞	30	100
1	{1,2}	2	10	60	30	100

Algoritmo de Dijkstr



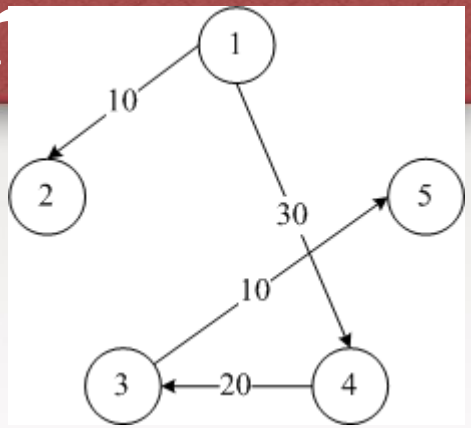
Iteración	*S*	*w*	*D*[2]	*D*[3]	*D*[4]	*D*[5]
Inicial	{1}	—	10	∞	30	100
1	{1,2}	2	10	60	30	100
2	{1,2,4}	4	10	50	30	90

Algoritmo de Dijkstr



Iteración	S	w	D[2]	D[3]	D[4]	D[5]
Inicial	{1}	—	10	∞	30	100
1	{1,2}	2	10	60	30	100
2	{1,2,4}	4	10	50	30	90
3	{1,2,4,3}	3	10	50	30	60

Algoritmo de Dijkstr



Iteración	*S*	*w*	*D*[2]	*D*[3]	*D*[4]	*D*[5]
Inicial	{1}	--	10	∞	30	100
1	{1,2}	2	10	60	30	100
2	{1,2,4}	4	10	50	30	90
3	{1,2,4,3}	3	10	50	30	60
4	{1,2,4,3,5}	5	10	50	30	60

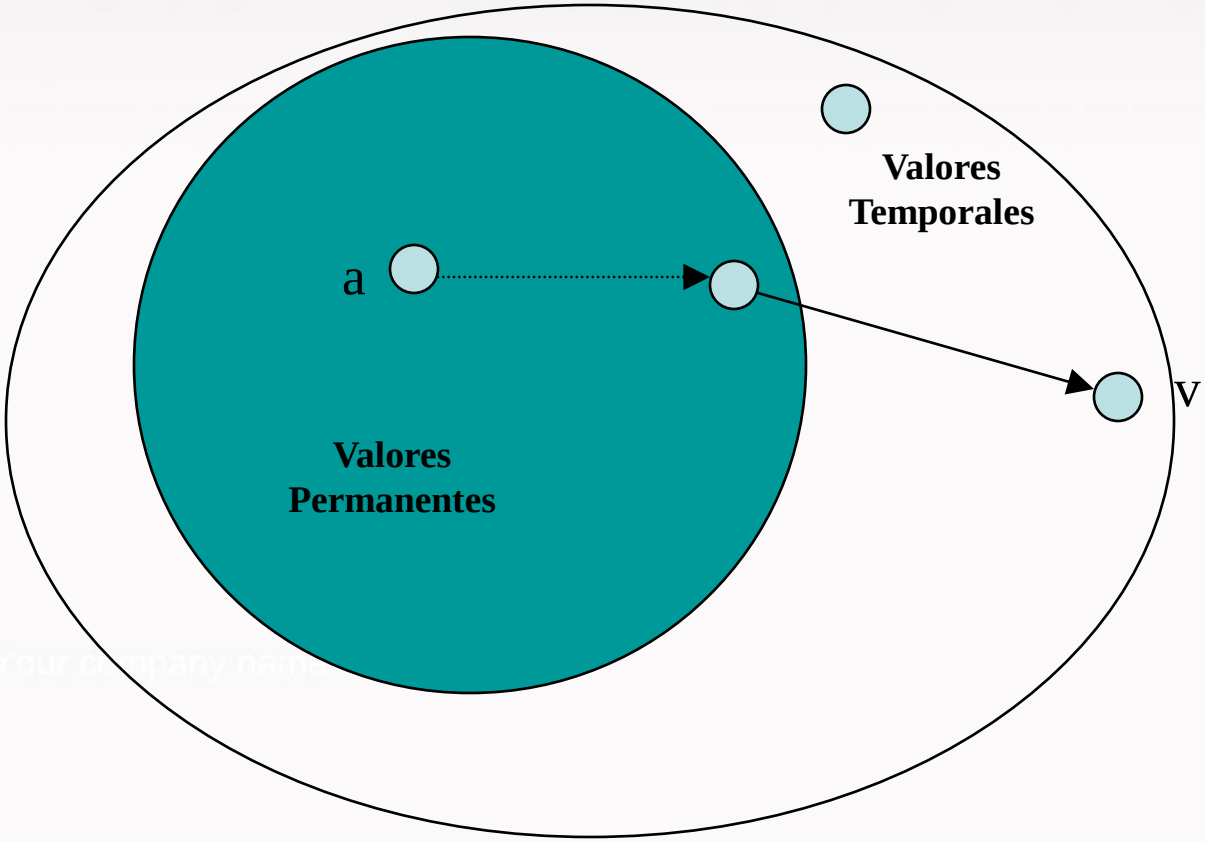
Demostración de la corrección

- **Hipotesis de Inducción T(i):**
  - en la i-ésima iteracion del lazo del algoritmo, el valor D[V] del vértice v se hace permanente
  - D[v] siempre es el valor mínimo entre los valores temporales
    El valor D[v] da el camino mínimo desde el origen hasta el vértice v
- **Base** (i = 1).
  - Inicialmente D[**a**] = 0, y todos los demás D's son mayores que cero, por tanto se elige el origen **a**
  - D[**a**] es el camino mas corto desde **a** a **a**
  - T(1) es cierta.

Demostración de la corrección

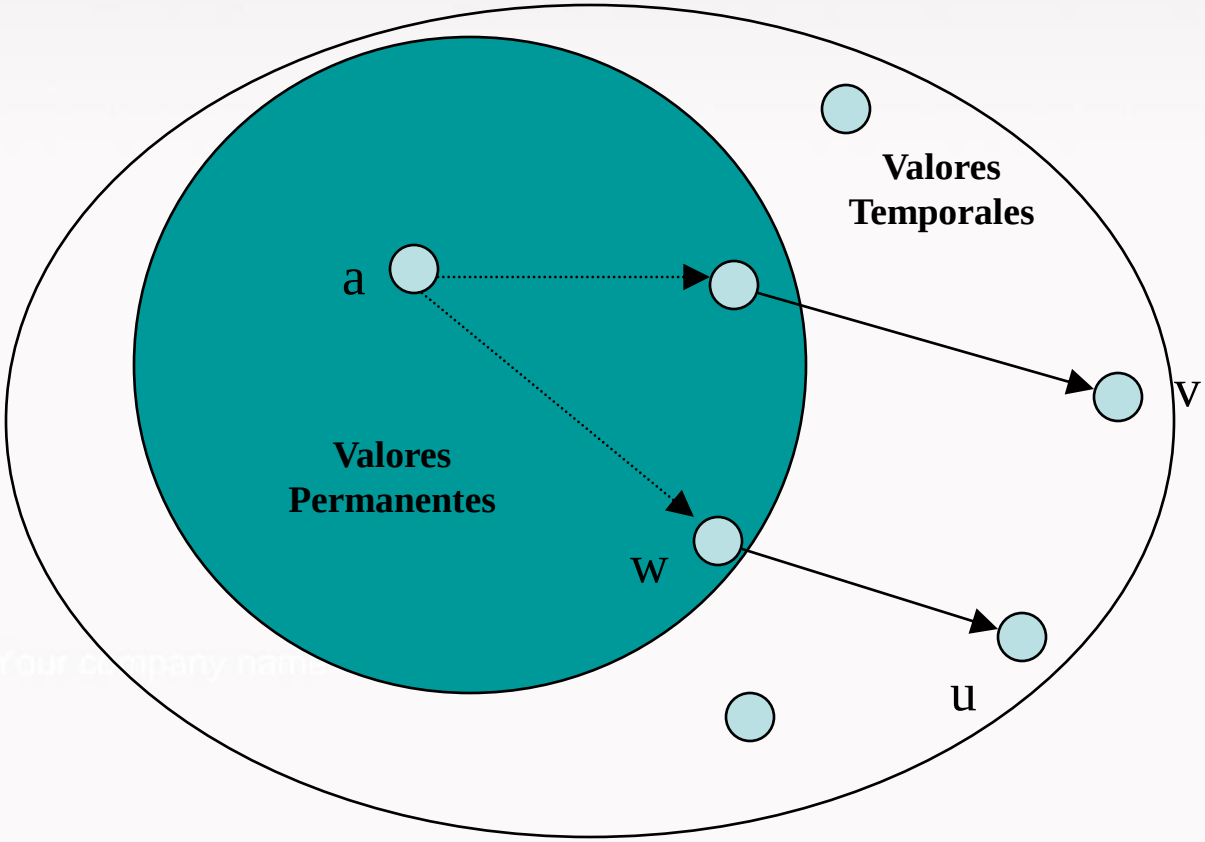
- Supongamos que T(k) es cierto para todo k < i.
  - En al iteración k+1, seleccionamos el nodo v.

    ¿ Es D(**v**) el camino mas corto desde **a** a **v**?



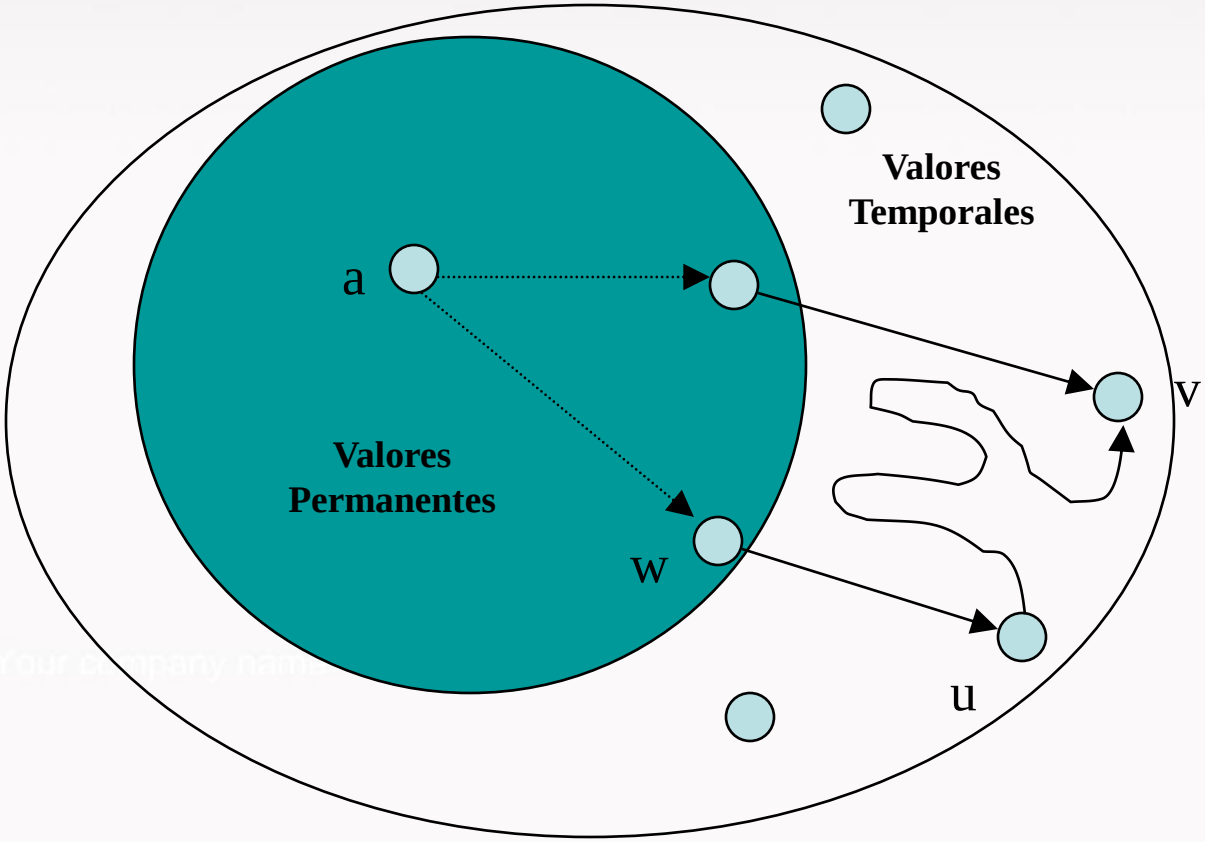
Demostración de la corrección

Supongamos que hay **otro vértice u** tal que el camino hasta **v** que pasa por **u** es mas corto que el anterior (el que nos daba D(**v**)).



Demostración de la corrección

Obligatoriamente el camino que pasa a través de **u** tiene que ser como el de la figura



Demostración de la corrección

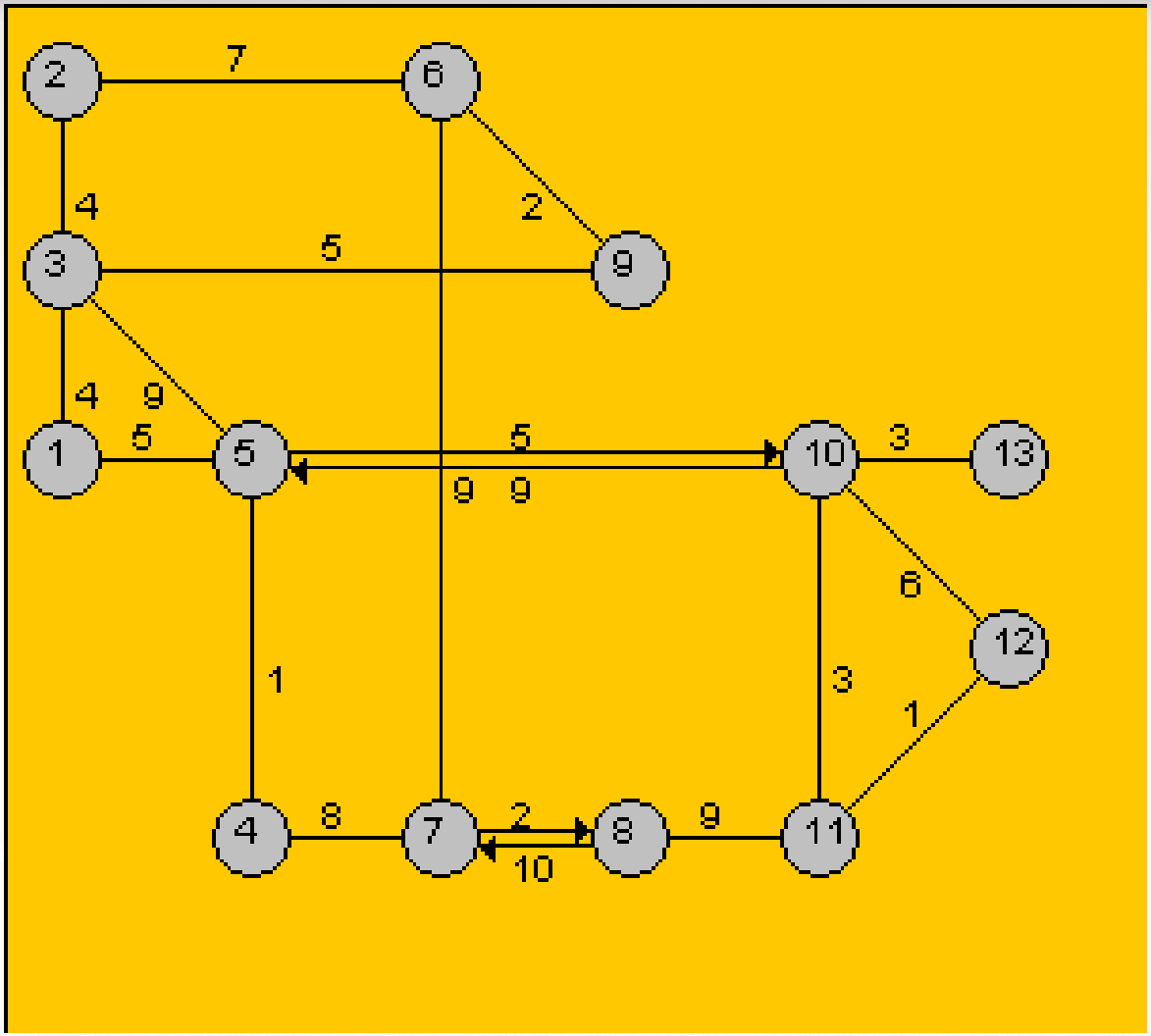
- La hipótesis de inducción era:
  - En la iteracion k+1 seleccionar el nodo v. ¿Es D(v) el camino mas corto de **a** a **v**?
  - La respuesta es SI.
  - Por tanto, T(k+1) es cierto.

- Como la base y la hipótesis de inducción son ciertas, T(i) es cierta para todo i.

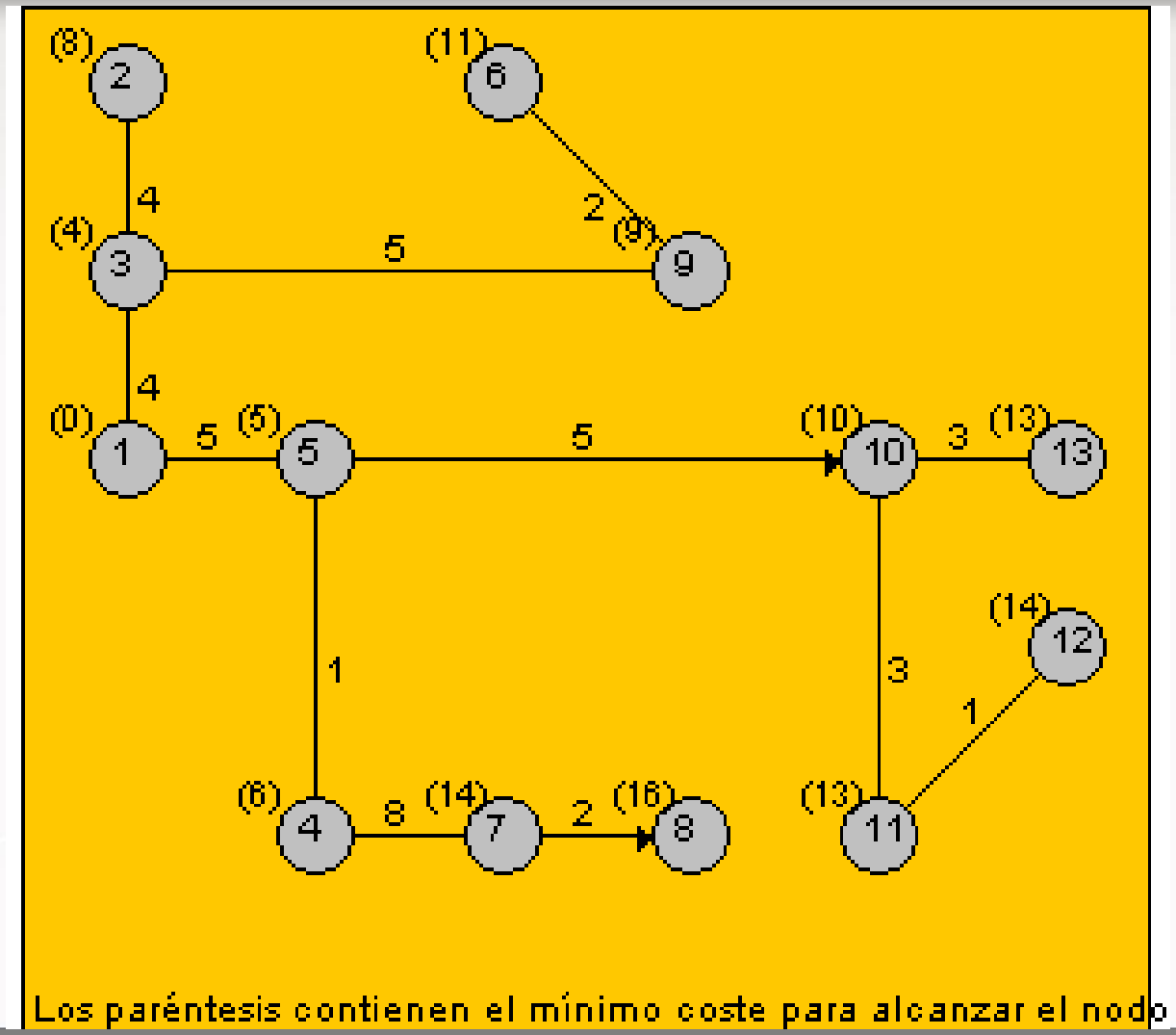
Análisis de la eficiencia

- El algoritmo es obvio que consume un tiempo en $O(n^2)$ en el peor caso
- La hipótesis de que las distancias sean no negativas es esencial
  - Si hay alguna distancia negativa el algoritmo no funciona correctamente: La hipótesis de inducción no puede demostrarse porque no se verifica la propiedad triangular.
- El algoritmo puede extenderse fácilmente para que de la distancia entre todos los pares de vértices: $O(n^3)$.

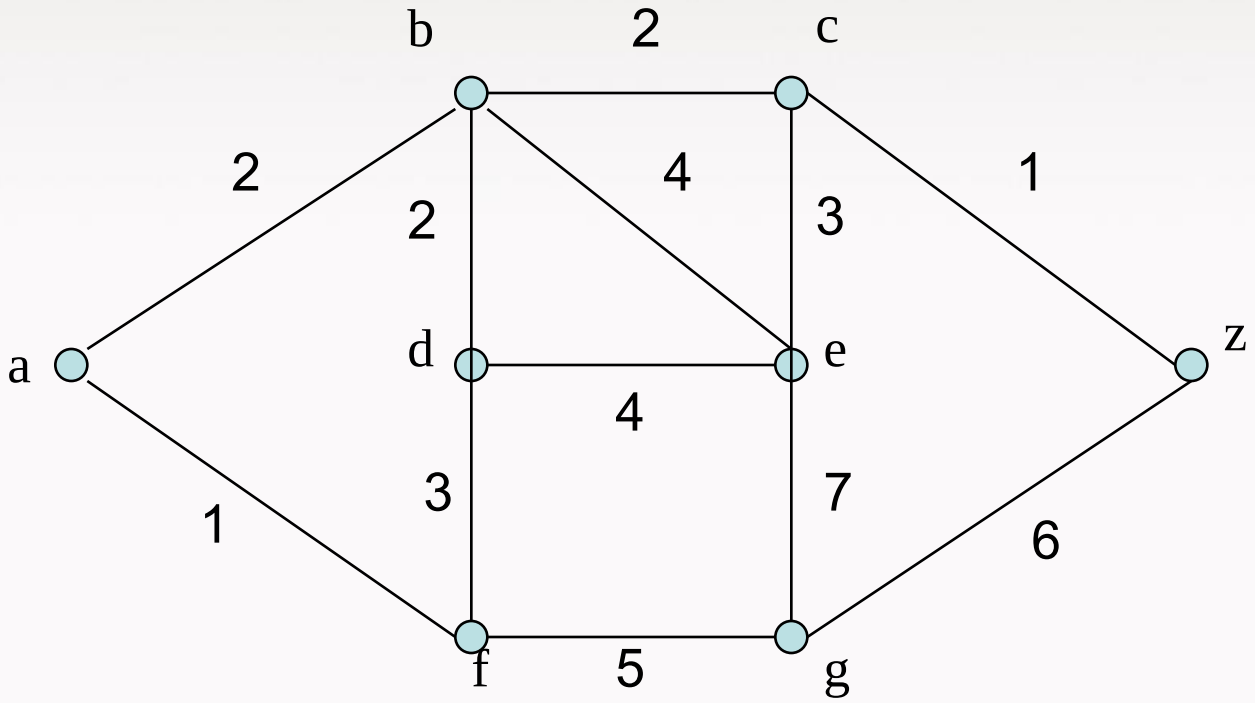
Ejemplo-1



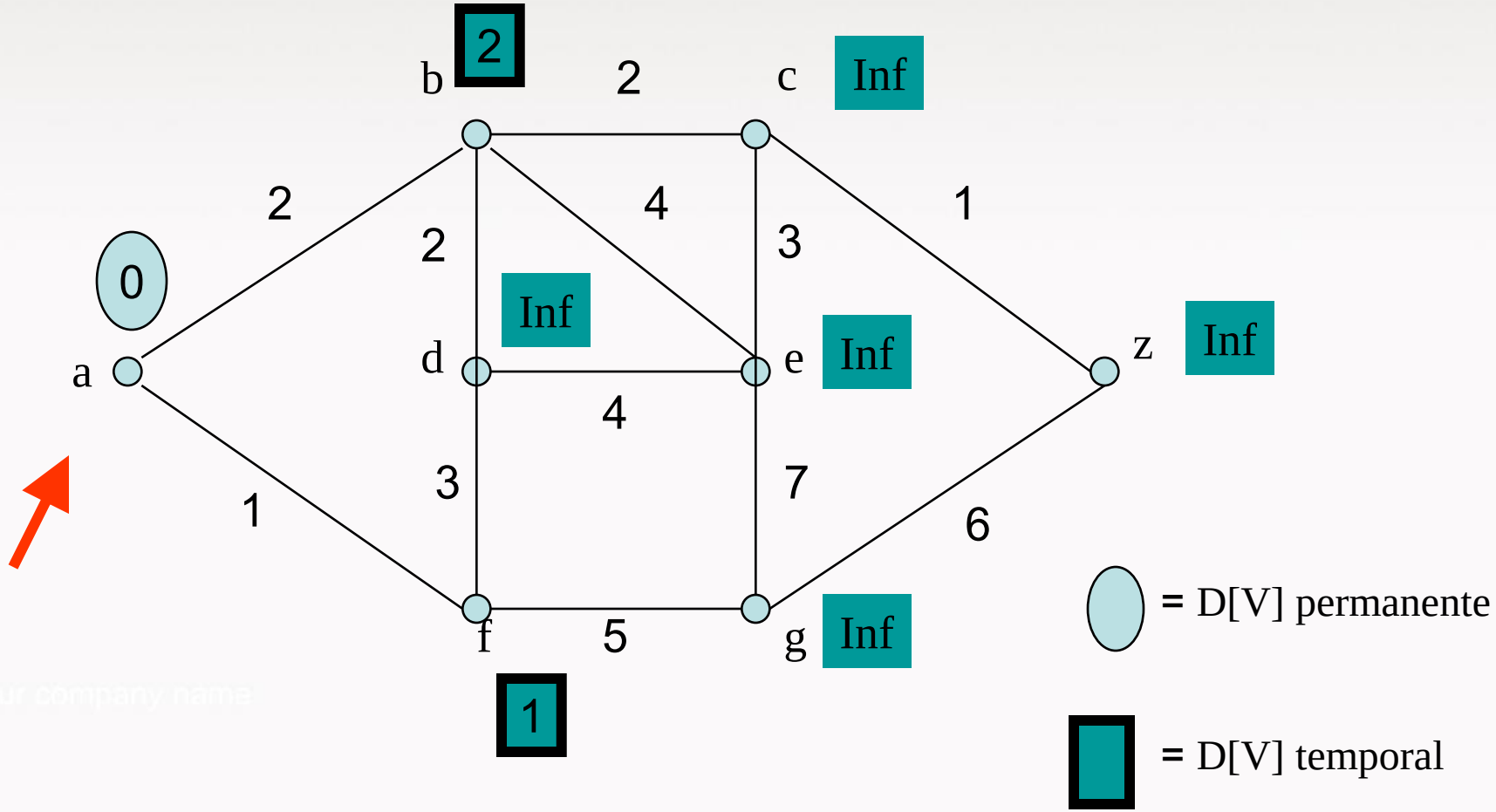
Ejemplo-1


(8)
2
(11)
6
4
2
(4)
3
5
(8)
9
4
(0)
1
5
(5)
5
(10)
10
3
(13)
13
1
3
(14)
12
1
(6)
4
8
(14)
7
2
(16)
8
(13)
11
Los paréntesis contienen el mínimo coste para alcanzar el nodo

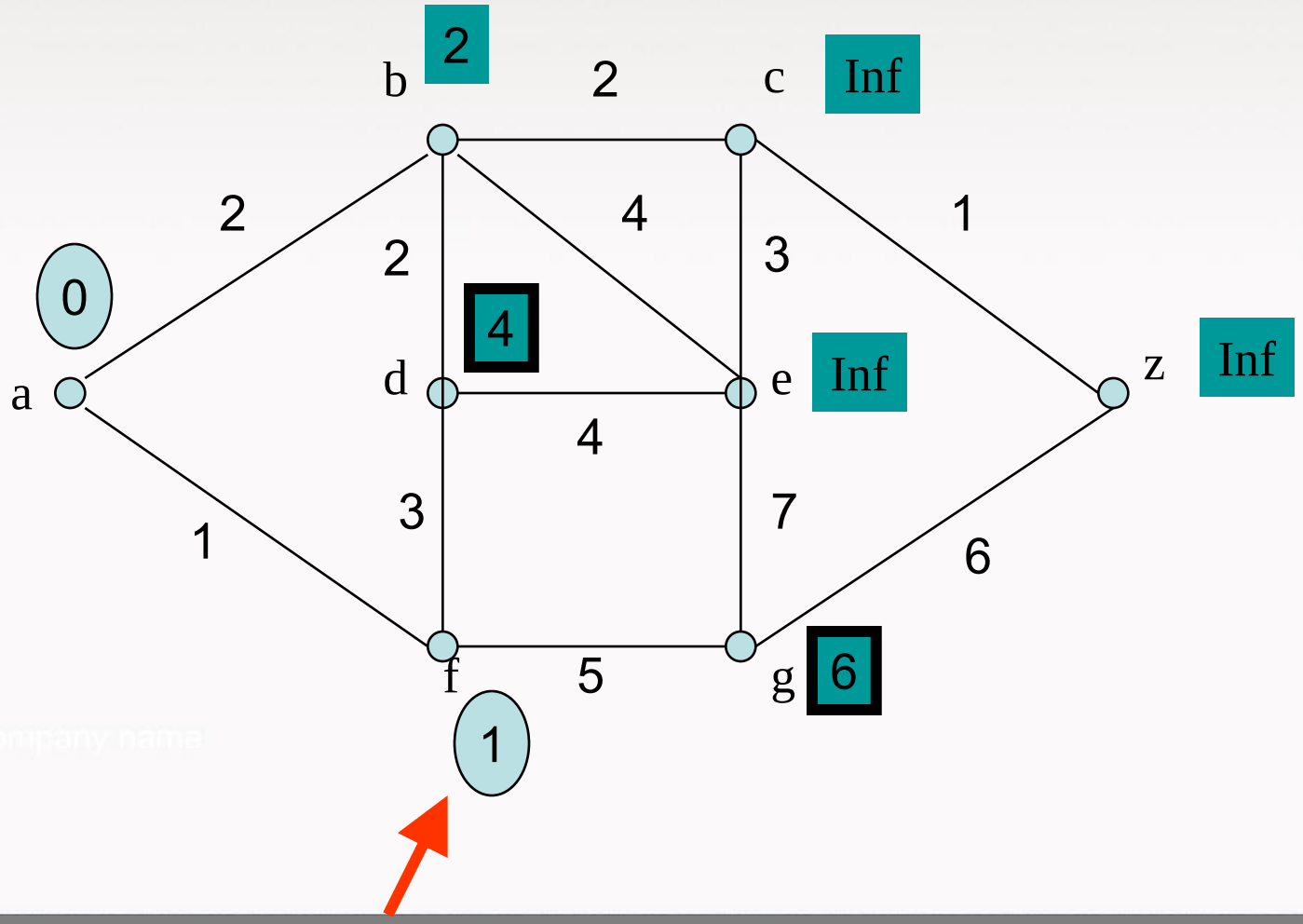
Ejemplo (Solo entre a y z)



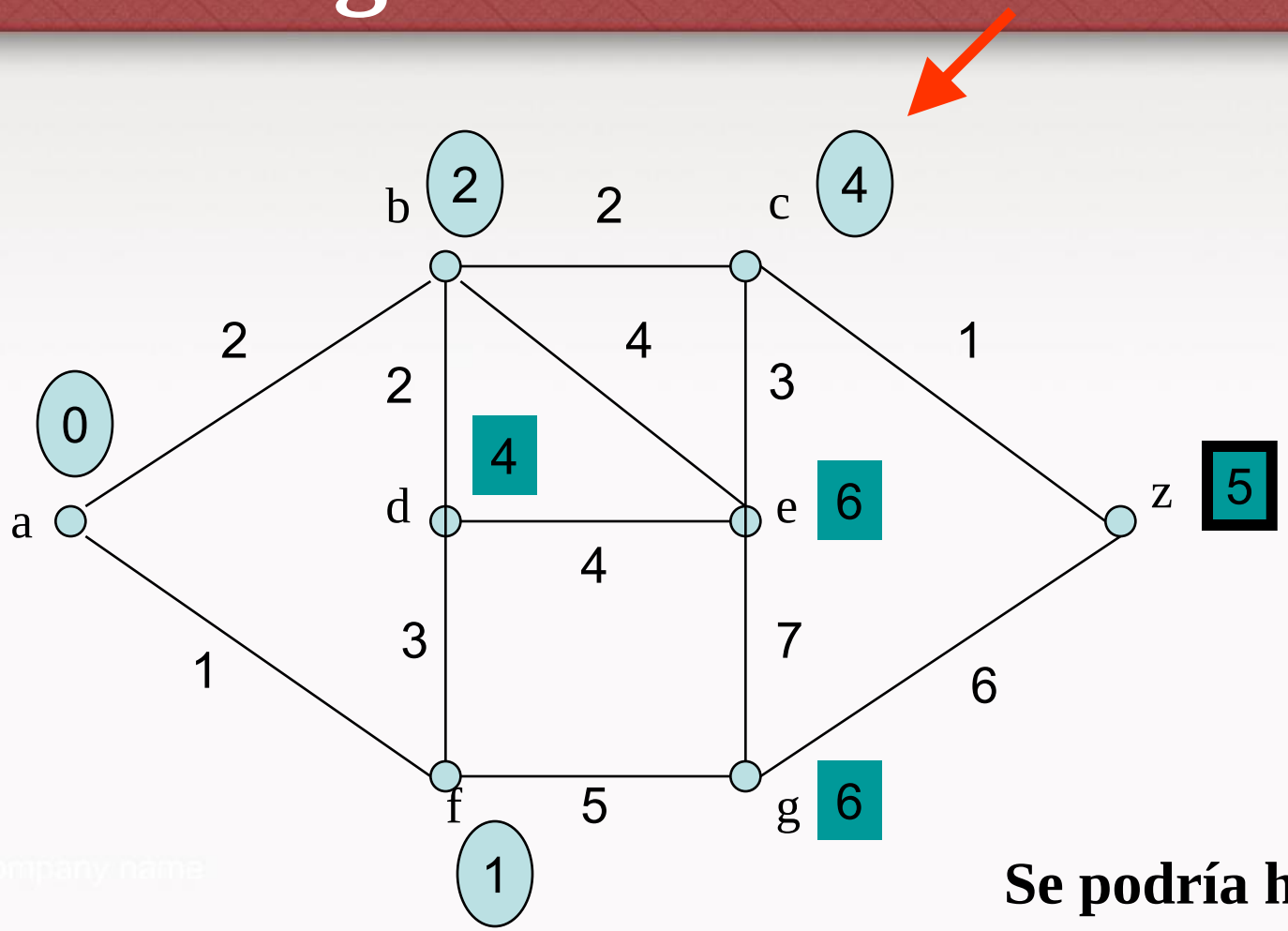
Inicialización



Primera Iteración

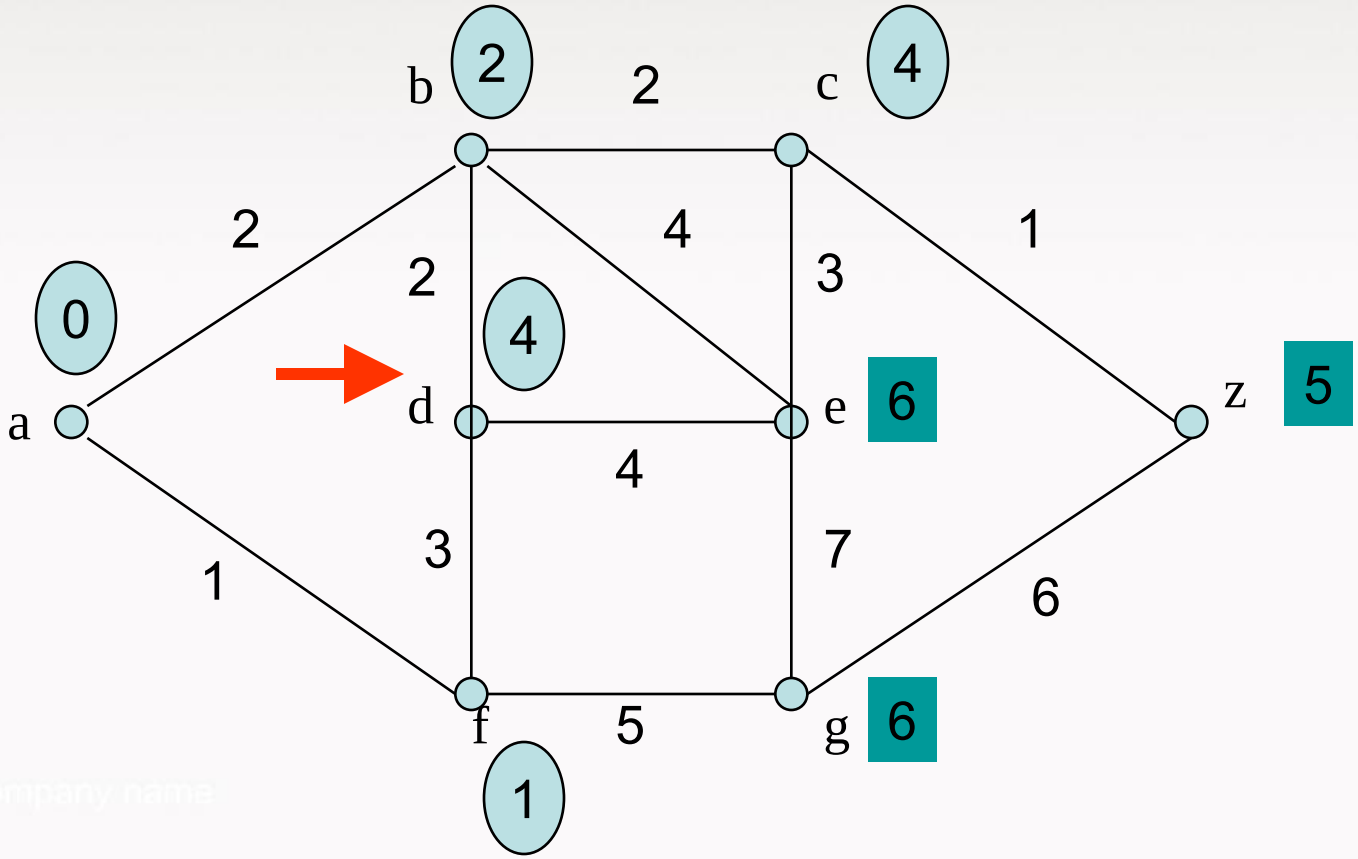


Segunda Iteración

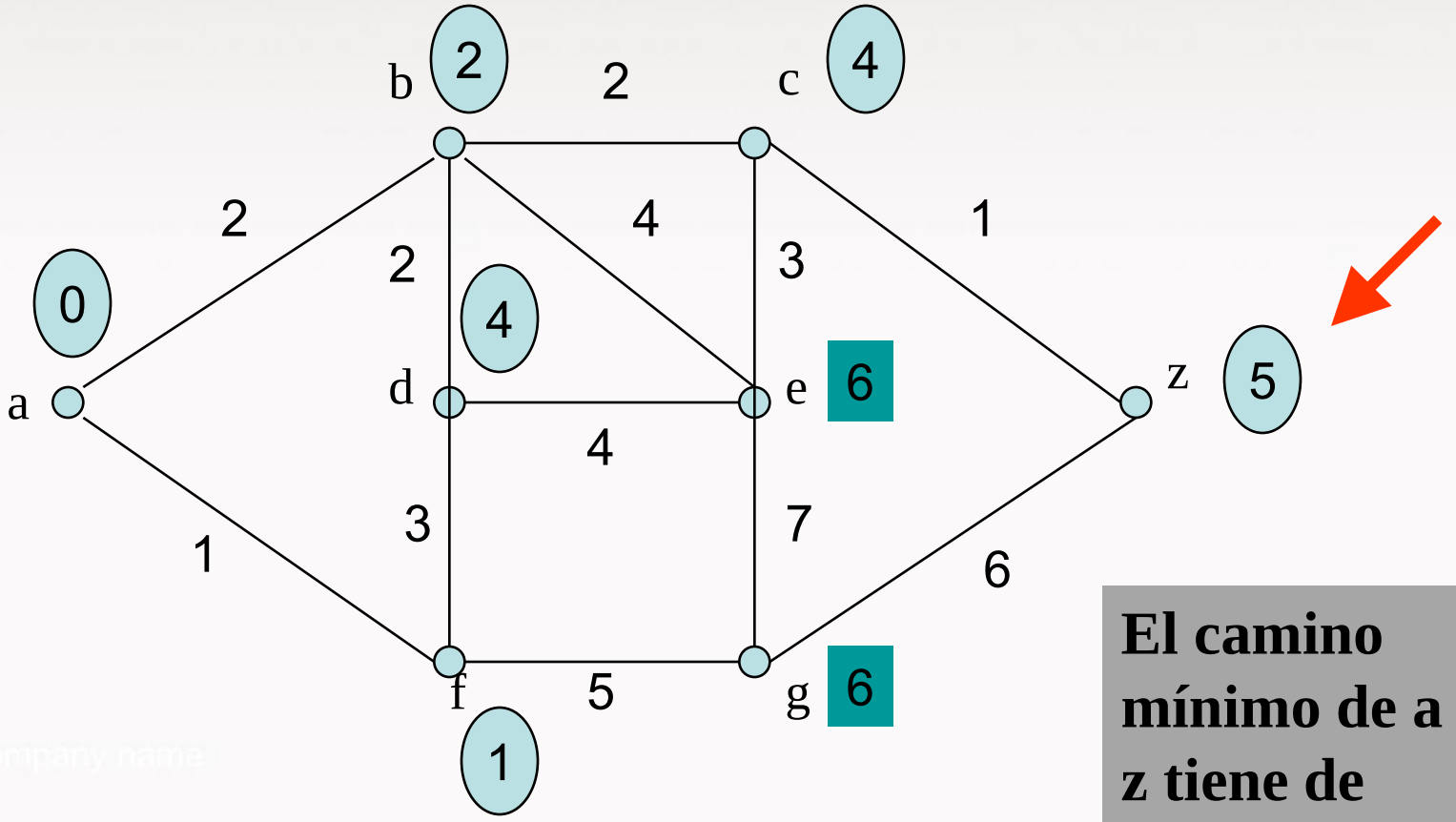


Se podría haber elegido tambien ‘d’

Tercera Iteración



Cuarta (y última) Iteración



El camino mínimo de a a z tiene de longitud 5