UNIVERSIDAD NACIONAL
GUILLERMO BROWN

CLASE 14 - Unidad 10

Problemas NP.
Problema del camino mínimo.

ESTRUCTURAS DE DATOS (271)
Clase N. 14 Unidad 10.

Clase 14: AGENDA

AGENDA

- **Temario:**
  - Problema del camino mínimo: estudio de distintos casos.
  - Su desarrollo para grafos pesados y no pesados; y grafos dirigidos y acíclicos.
  - Algoritmos de Dijkstra y Floyd.
  - Árbol generador mínimo. Algoritmos de Prim y Kruskal.
  - Análisis del tiempo de ejecución de los algoritmos vistos.

- **Ejemplos en Lenguajes Python**
- **Temas relacionados y links de interés**
- **Práctica**
- **Cierre de la clase**

Clase 14: Problema del camino mínimo

Definición:

El ***problema del camino mínimo***: Este problema consiste en hallar la mejor forma de ir desde un punto a otro (o a varios otros) minimizando la distancia recorrida, el tiempo invertido, entre varias posibilidades.

Para poder resolver estos problemas debemos poder representarlos de manera concisa y abstracta. La representación más usual de este tipo de problemas (ya sea para aplicarlo en problemas de camino mínimo o no) es la de **grafos**.

Clase 14: Problema del camino mínimo

Definición:

Sea G=(V,A) un grafo dirigido y pesado, el costo $c(i,j)$ está asociado a la arista $v(i,j)$.

Dado un camino: $v_1, v_2, v_3, \ldots\ldots v_N$

El costo del camino es:

$$C = \sum_{i=1}^{N-1} c(i, i+1)$$

Este valor también se llama longitud del camino pesado.

La longitud del camino no pesado es la cantidad de aristas

Según qué tipo de grafo analicemos, la solución al problema será diferente. Por ejemplo, si el grafo no tuviera pesos (o si todos los pesos fueran iguales, que a efectos prácticos es lo mismo), nos conviene usar otro algoritmo (BFS).

Clase 14: Problema del camino mínimo

Definición camino mínimo:

El camino de costo mínimo desde un vértice v_i a otro vértice v_j es aquel en que la suma de los costos de las aristas es mínima.

Esto significa que:

$$C = \sum_{i=1}^{N-1} c(i, i+1) \quad \text{es mínima}$$

Clase 14: Problema del camino mínimo

Ejemplos:

Ciudades conectadas por
Rutas con *distancias*

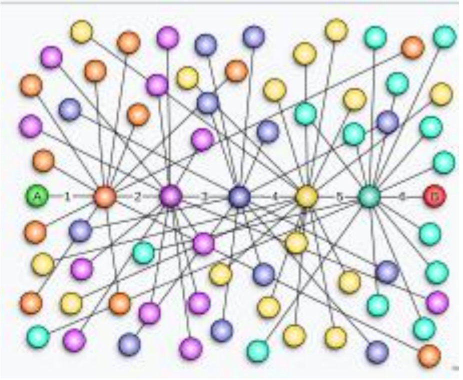
Personas conectadas a
través de las redes sociales

Clase 14: Problema del camino mínimo

Ejemplos:

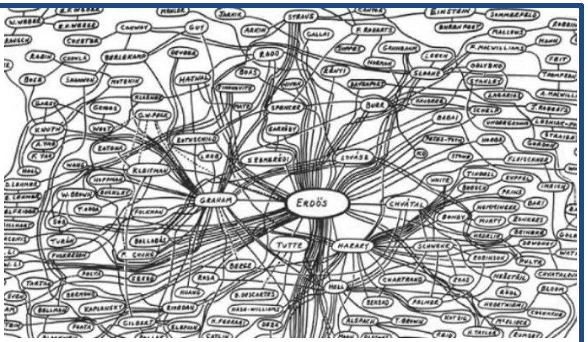
Seis grados de separación

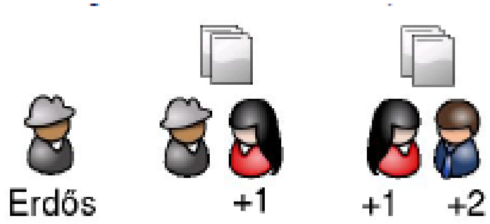
Se le llama *seis grados de separación* a la hipótesis que intenta probar que cualquiera en la Tierra puede estar conectado a cualquier otra persona del planeta a través de una cadena de conocidos que no tiene más de cinco intermediarios (conectando a ambas personas con sólo seis enlaces)



Número de Erdős

Es un modo de describir la distancia colaborativa, en lo relativo a trabajos matemáticos entre un autor y Paul Erdős (matemático húngaro considerado uno de los escritores más prolíficos de trabajos matemáticos).

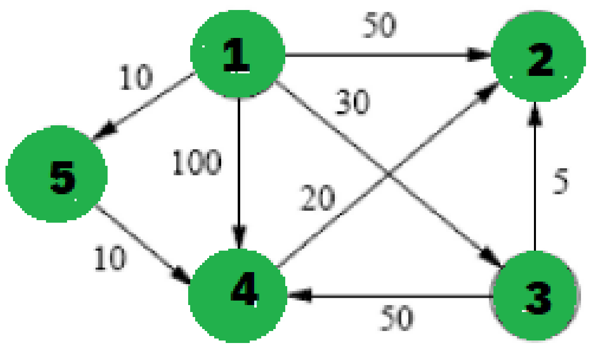




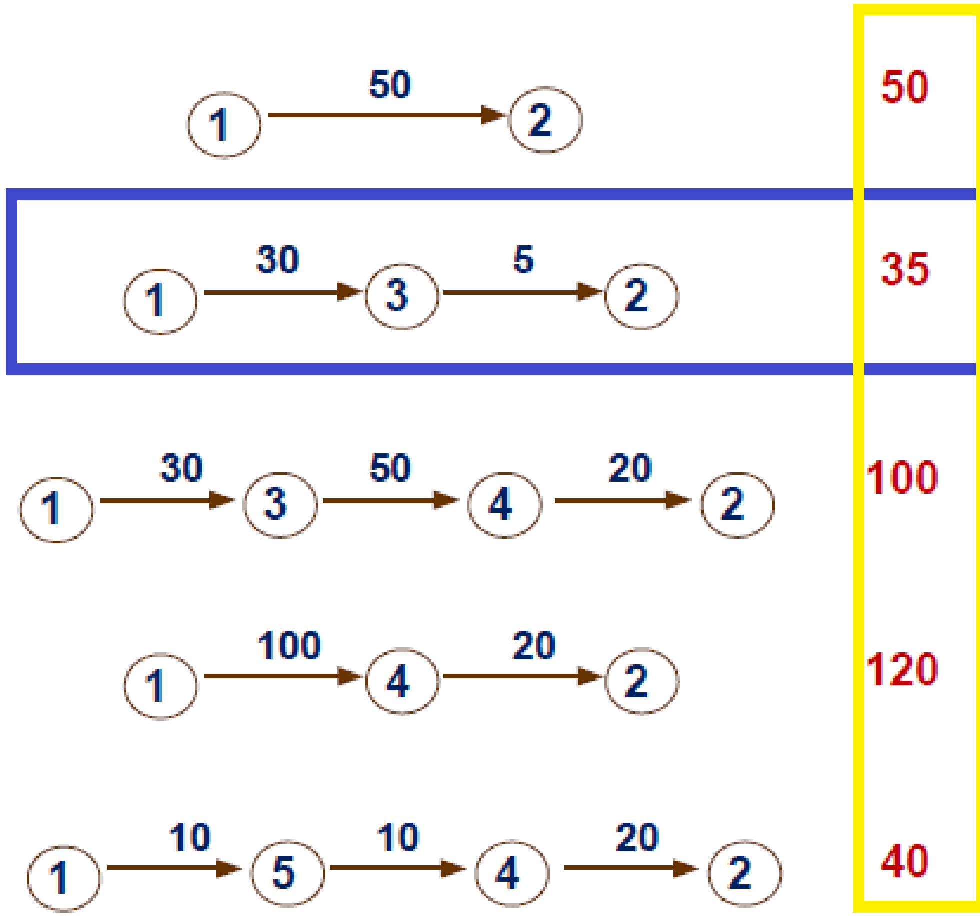
Si la **mujer de rojo** colabora directamente con Erdős en un trabajo, y luego el **hombre de azul** colabora con ella; entonces el hombre de azul tiene un número de Erdős con valor 2, y está "a dos pasos" de Paul Erdős (asumiendo que nunca ha colaborado directamente con éste).

Clase 14: Problema del camino mínimo

Ejemplos:



Caminos posibles desde el vértice 1 al vértice 2

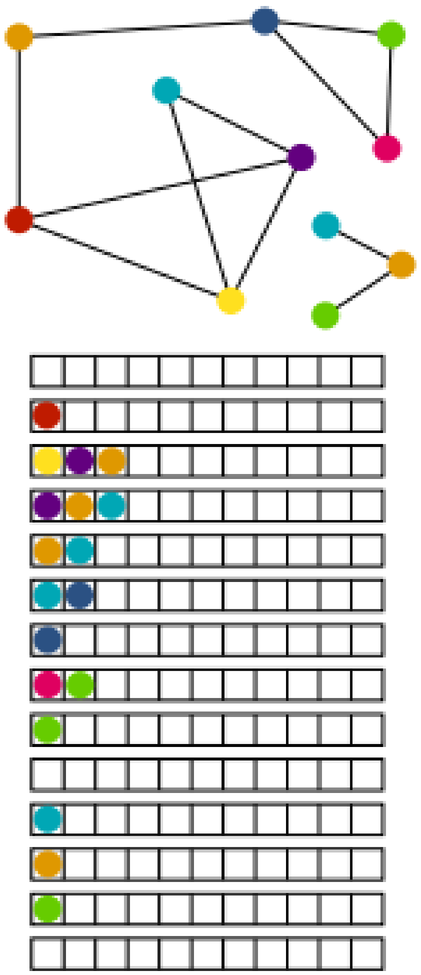


Clase 14: Problema del camino mínimo

Grafos sin pesos:

Recorrido BFS: pseudocódigo

```
cola C; vértices v, w; ord = 0;
Mientras queden vértices sin visitar
    v = elegir un vértice no visitado;
    marcar v;
    C.vaciar();
    C.encolar(v);
    Mientras C no sea vacía
        w = C.primero();
        ord++;
        orden(w)=ord;
        C.desencolar();
        Para cada vecino z de w
            Si z no está visitado
                marcar z;
                C.encolar(z);
```

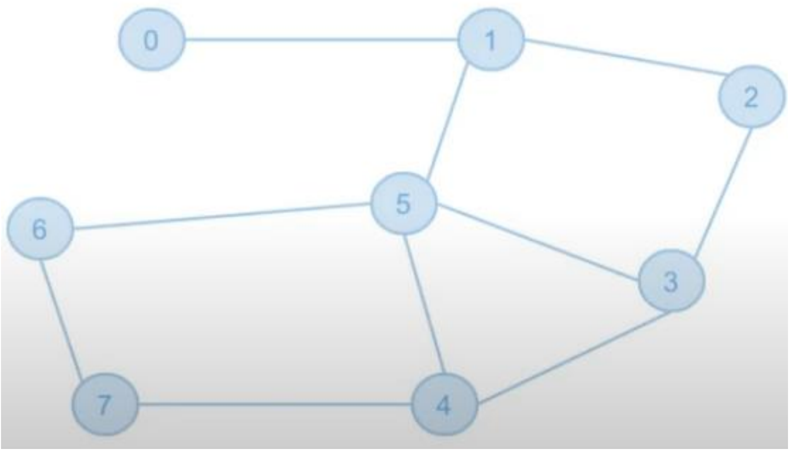


Clase 14: Problema del camino mínimo

Recorrido BFS: propiedades

Si partimos desde un vértice v, modificando levemente el código podemos calcular para cada vértice su distancia a v. Inicialmente todos los vértices tienen rótulo ∞. Rotulamos el vértice v con 0 y luego, al visitar los vecinos de cada vértice w que aún tienen rótulo ∞, los rotulamos con el rótulo de w más 1. (Notemos que los que no sean alcanzados desde v, o sea, no pertenecen a la componente conexa de v, tienen distancia infinita a v.)

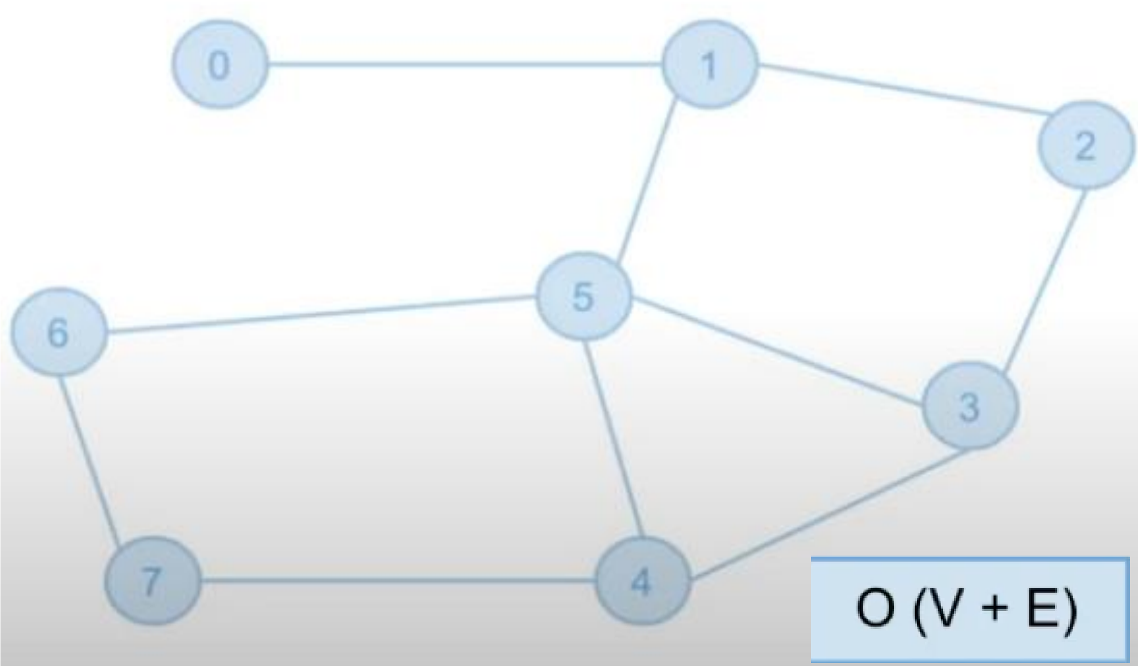
Ejemplo:



Clase 14: Problema del camino mínimo

Grafos sin pesos:

Estrategia: Recorrido en amplitud (BFS)



Padre	Orden	Visitados
1: None	1: 0	1
0: 1	0: 1	0
5: 1	5: 1	5
2: 1	2: 1	2
6: 5	6: 2	6
3: 5	3: 2	3
4: 5	4: 2	4
7: 6	7: 3	7

El camino se construye, Ej., quiero encontrar un camino desde 4 hasta 1:
Voy al nodo **4:5** el padre es 5, busco el nodo **5:1** el padre es 1
(la distancia es 2 **“orden”**)

Clase 14: Problema del camino mínimo

Algoritmo de Dijkstra :

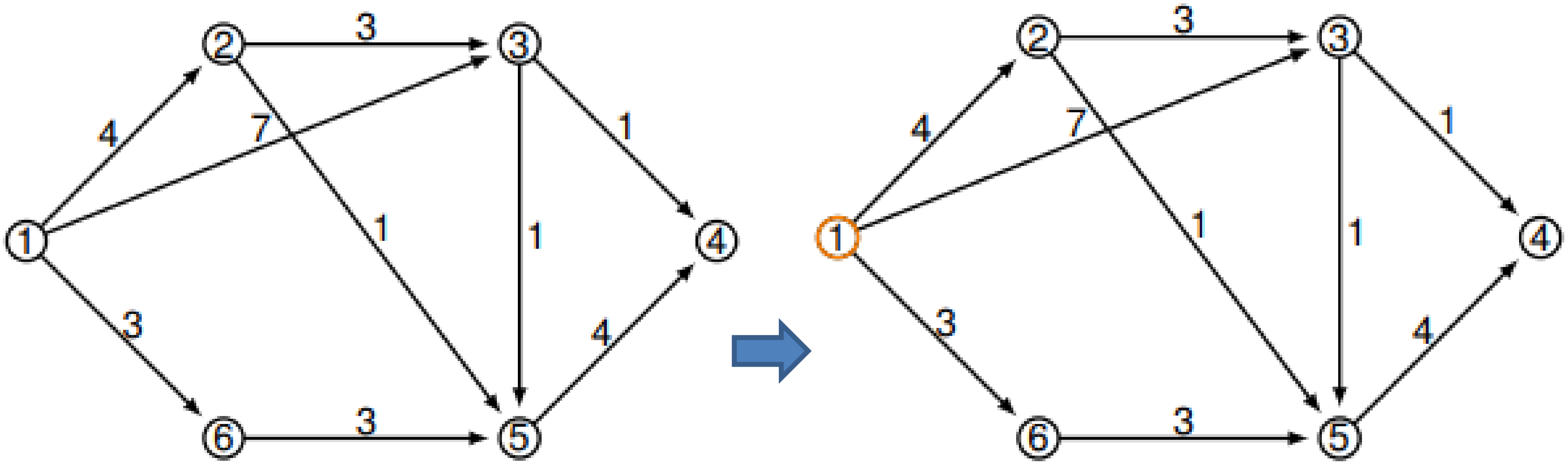
Este algoritmo fue creado por uno de los padres de la computación, Edger W. Dijkstra, en 1956. ***Sirve para cualquier grafo con pesos (dirigido o no) siempre y cuando sus pesos no sean negativos.***

El algoritmo calcula las distancias mínimas desde un nodo inicial a todos los demás. Para hacerlo, en cada paso se toma el nodo más cercano al inicial que aún no fue visitado (le diremos v). Este nodo tiene calculada la menor distancia al nodo inicial . Luego, recalculamos todas los caminos mínimos, teniendo en cuenta a v como camino intermedio. Así, en cada paso tendremos un subconjunto de nodos que ya tienen calculada su mínima distancia y los demás tienen calculada su mínima distancia si solo puedo usar los nodos del conjunto como nodos intermedios. Con cada iteración agregaremos un nodo más a nuestro conjunto, hasta resolver el problema en su totalidad.

Algoritmo de Dijkstra :

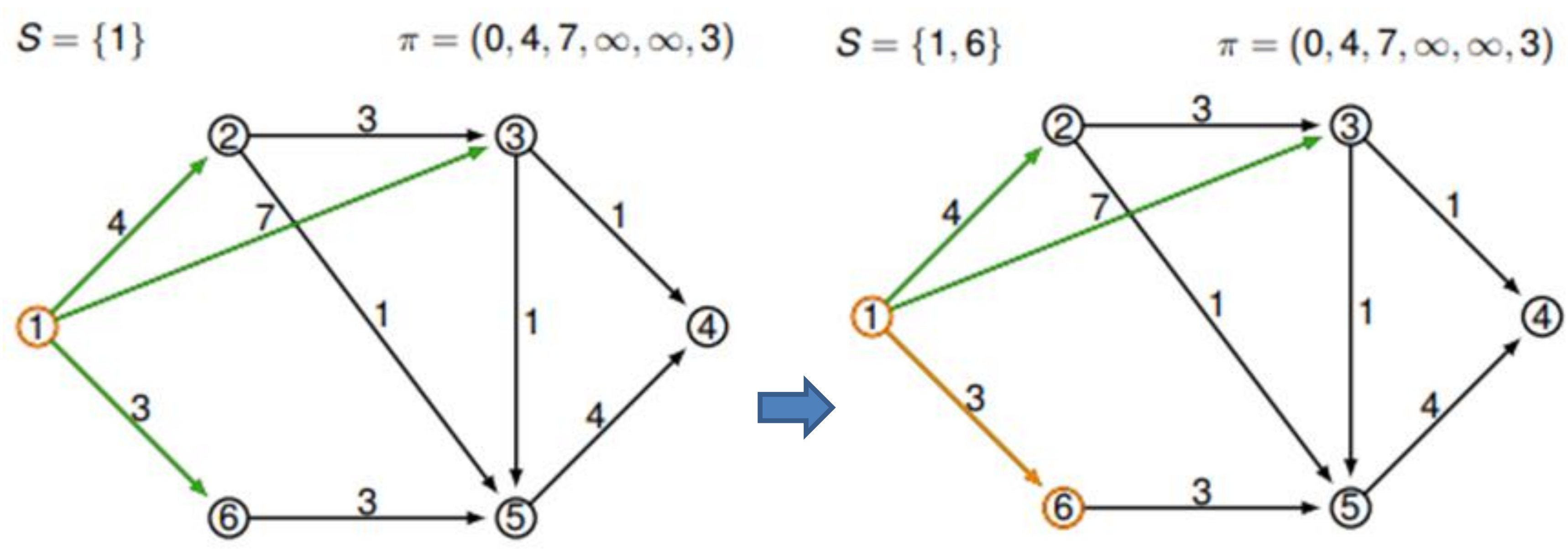
$S = \{1\}$ $\pi = (0, \infty, \infty, \infty, \infty, \infty)$

Ejemplo



Dado el siguiente grafo: partimos del vértice origen(1)

Algoritmo de Dijkstra :

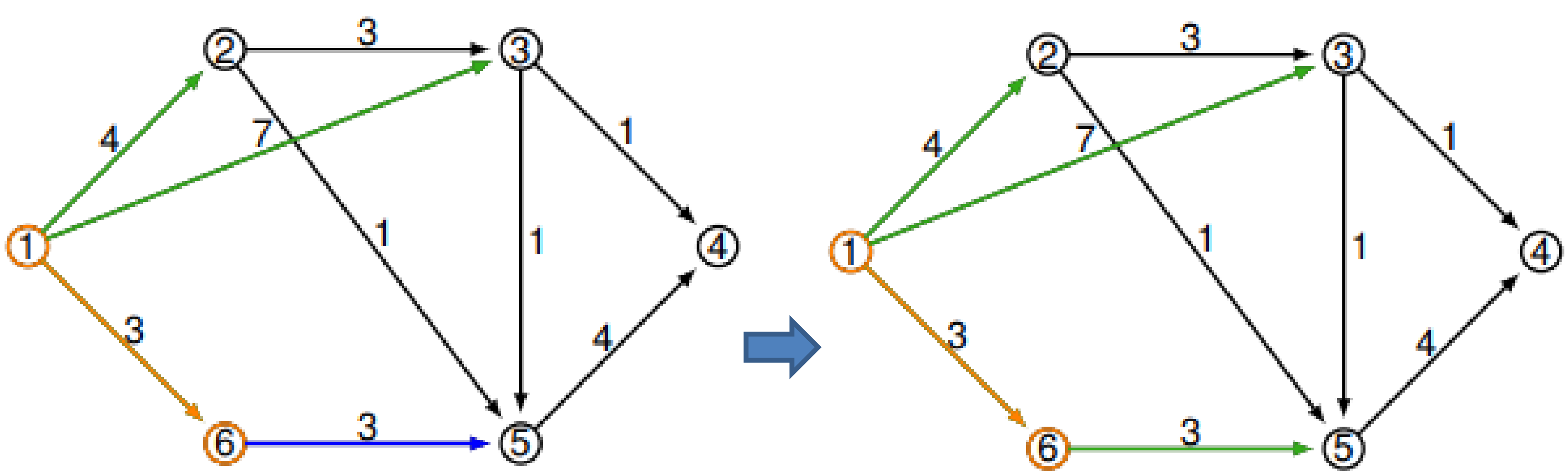


Buscamos los adyacentes a v y los pesos son actualizados en la estructura
Elegimos el vértice que este a menor distancia(v)
Marcamos el vértice v como procesado

Algoritmo de Dijkstra :

$S = \{1, 6\}$ $\pi = (0, 4, 7, \infty, \infty, 3)$

$S = \{1, 6\}$ $\pi = (0, 4, 7, \infty, 6, 3)$

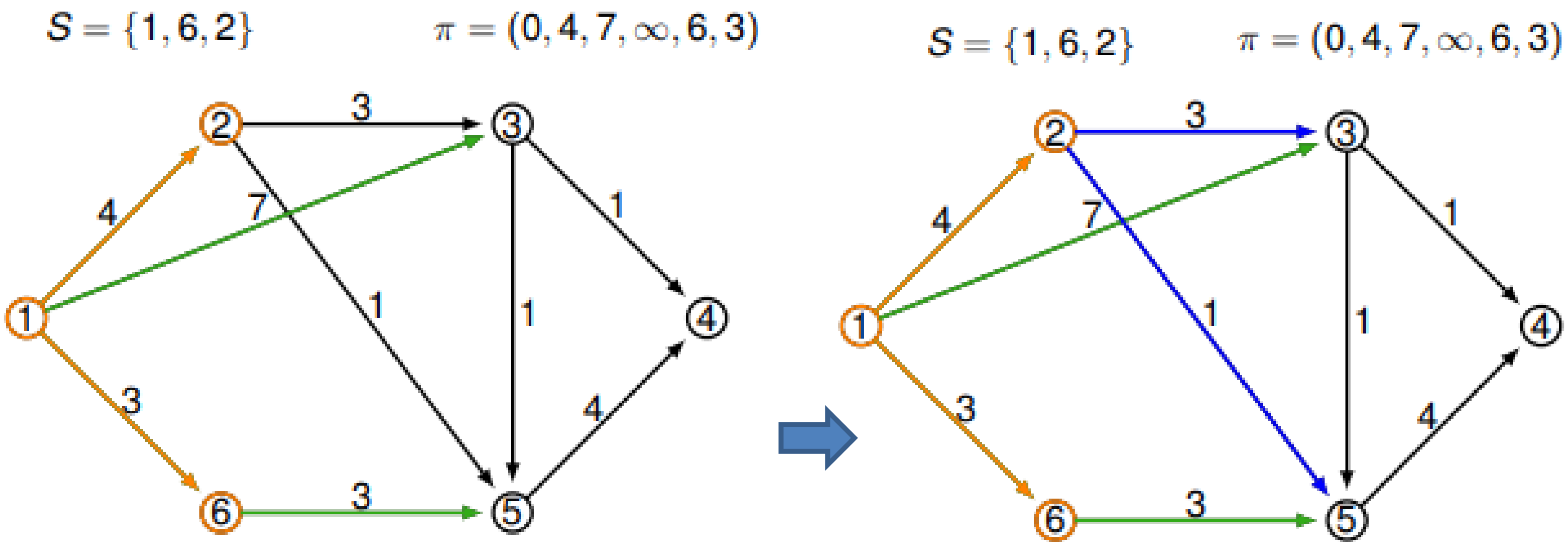


Buscamos los adyacentes a v (6)y los pesos son actualizados
Elegimos el vértice que este a menor distancia(v)
Marcamos el vértice v como procesado

π [6]=3

π [5]=3 + π [6] = 6

Algoritmo de Dijkstra :

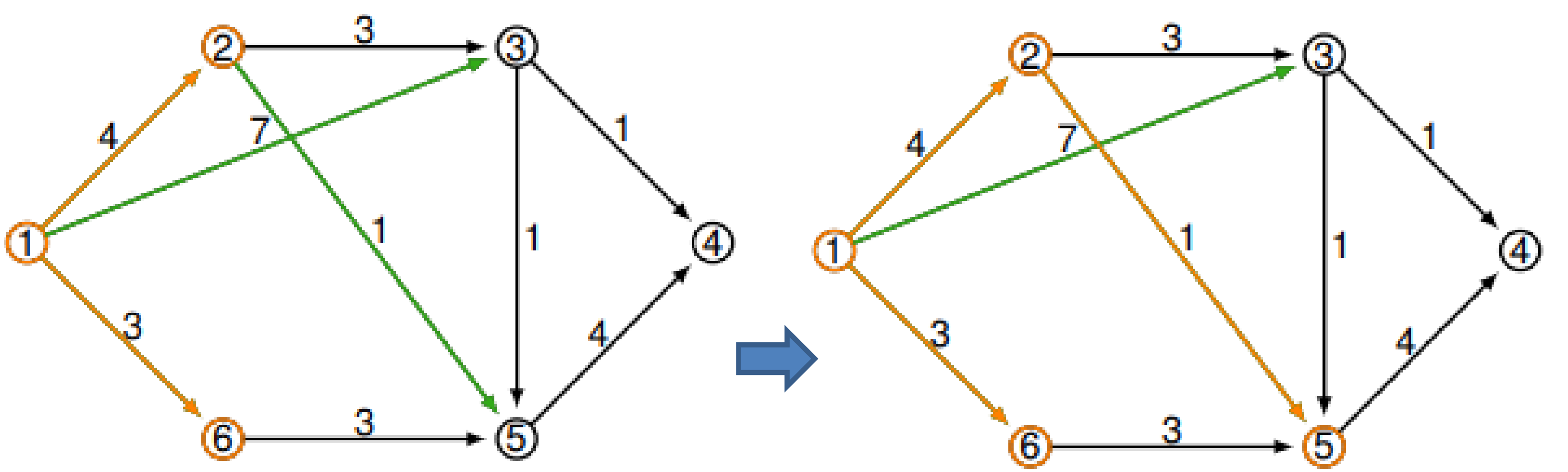


Buscamos los adyacentes a v (2)y los pesos son actualizados
Elegimos el vértice que este a menor distancia(v)
Marcamos el vértice v como procesado

Algoritmo de Dijkstra :

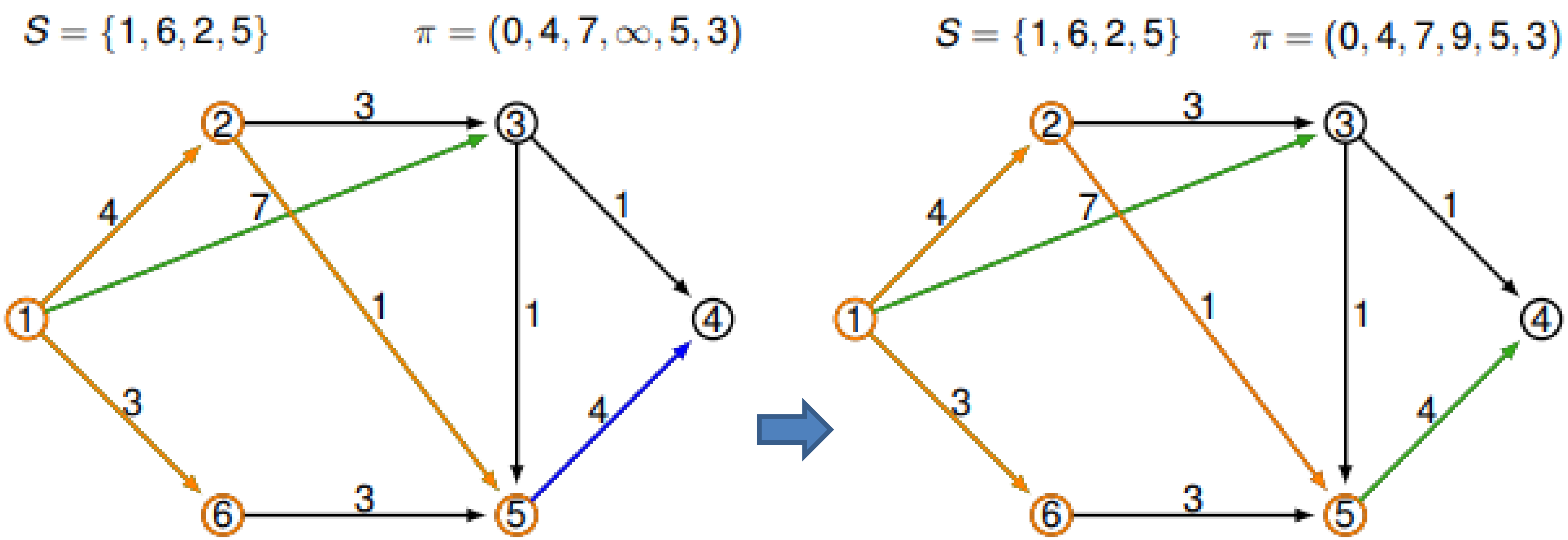
$S = \{1, 6, 2\}$ $\pi = (0, 4, 7, \infty, 5, 3)$

$S = \{1, 6, 2, 5\}$ $\pi = (0, 4, 7, \infty, 5, 3)$



Buscamos los adyacentes a v (5)y los pesos son actualizados
Elegimos el vértice que este a menor distancia(v)
Marcamos el vértice v como procesado

Algoritmo de Dijkstra :

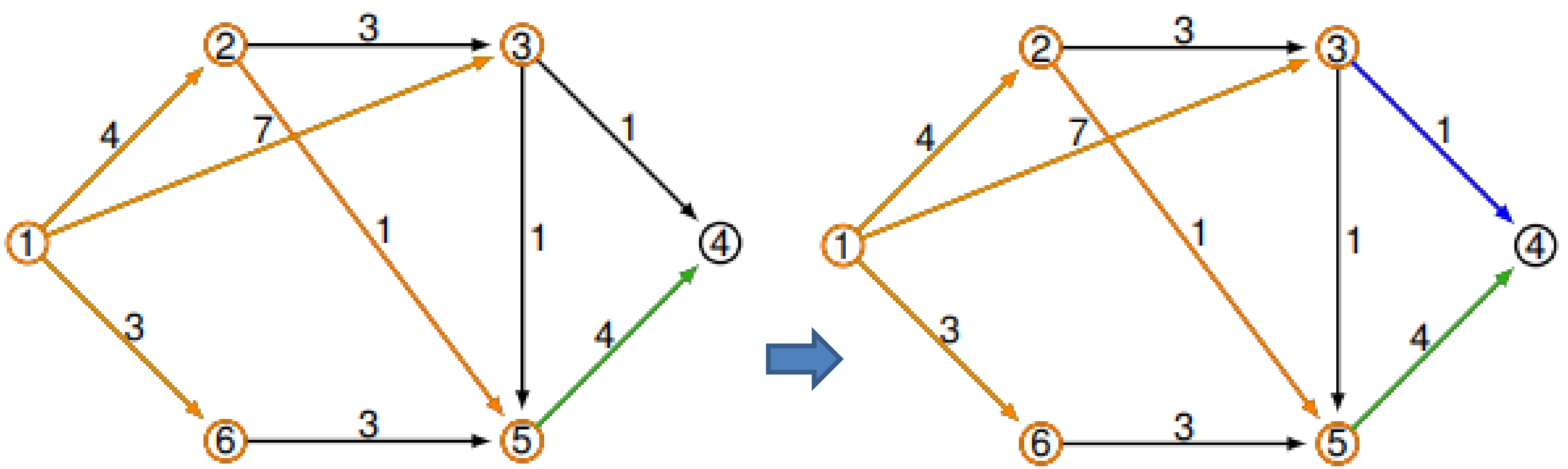


Buscamos los adyacentes a v (4)y los pesos son actualizados
Elegimos el vértice que este a menor distancia(v)
Marcamos el vértice v como procesado

Algoritmo de Dijkstra :

$S = \{1, 6, 2, 5, 3\}$ $\pi = (0, 4, 7, 9, 5, 3)$

$S = \{1, 6, 2, 5, 3\}$ $\pi = (0, 4, 7, 9, 5, 3)$

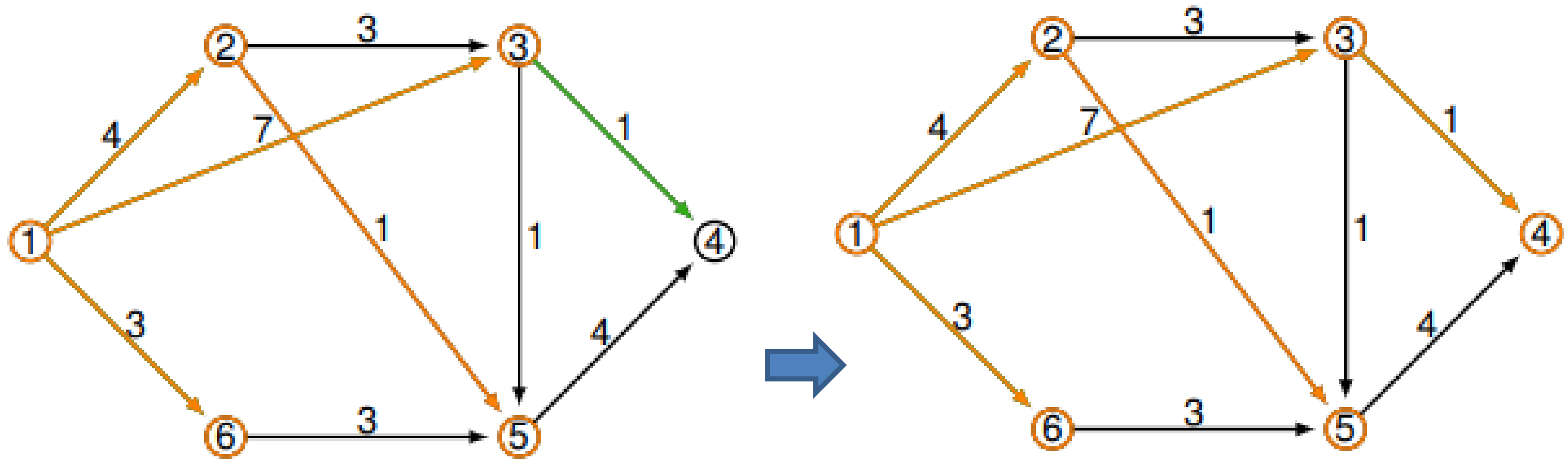


Buscamos los adyacentes a v (3)y los pesos son actualizados
Elegimos el vértice que este a menor distancia(v)
Marcamos el vértice v como procesado

Clase 14: Problema del camino mínimo

Algoritmo de Dijkstra :

$S = \{1, 6, 2, 5, 3\}$ $\pi = (0, 4, 7, 8, 5, 3)$ $S = \{1, 6, 2, 5, 3, 4\}$ $\pi = (0, 4, 7, 8, 5, 3)$



Buscamos los adyacentes a v (4)y los pesos son actualizados
Elegimos el vértice que este a menor distancia(v)
Marcamos el vértice v como procesado

Clase 14: Problema del camino mínimo

Algoritmo de Dijkstra :

Pseudocódigo de Dijkstra con cola de prioridad

```
Dijkstra (Grafo G, nodo_fuente s)
    para todo u en V[G] hacer
        distancia[u] = INFINITO
        padre[u] = NULL
        visitado[u] = false

    distancia[s] = 0
    adicionar (cola, (s, distancia[s]))

    mientras que cola no sea vacia hacer
        u = extraer_minimo(cola)
        visitado[u] = true
        para todo v en adyacencia[u] hacer
            si no visitado[v] y distancia[v] > distancia[u] + peso (u, v)
              hacer
                distancia[v] = distancia[u] + peso (u, v)
                padre[v] = u
                adicionar(cola,(v, distancia[v]))
```

Clase 14: Problema del camino mínimo

Algoritmo de Dijkstra :

```
def camino_minimo(grafo, origen):
        dist = {}
        padre = {}
        for v in grafo:
                distancia[v] = infinito
        dist[origen] = 0
        padre[origen] = None
        q = heap_crear()
        q.encolar(origen, 0)
        while not q.esta_vacia():
                v = q.desencolar()
                for w in grafo.adyacentes(v):
                        if dist[v] + grafo.peso_union(v, w) < dist[w]:
                                dist[w] = dist[v] + grafo.peso_union(v, w)
                                padre[w] = v
                                q.encolar(w, dist[w]) # o: q.actualizar(w, dist[w])
        return padre, distancia
```

Clase 14: Problema del camino mínimo

Algoritmo de Dijkstra :

```
def camino_minimo(grafo, origen):
    distancia = {}
    padre = {}
    for v in grafo:
        distancia[v] = inf
    distancia[origen] = 0
    padre[origen] = None
    q = heap_crear()
    q.encolar(origen, 0)
    while not q.esta_vacia():
        v = q.desencolar()
        for w in grafo.adyacentes(v):
            if (distancia[v] + grafo.peso(v,w) < distancia[w]):
                distancia[w] = distancia[v] + grafo.peso(v,w)
                padre[w] = v
                q.encolar(w, distancia[w])
    return padre, distancia
```

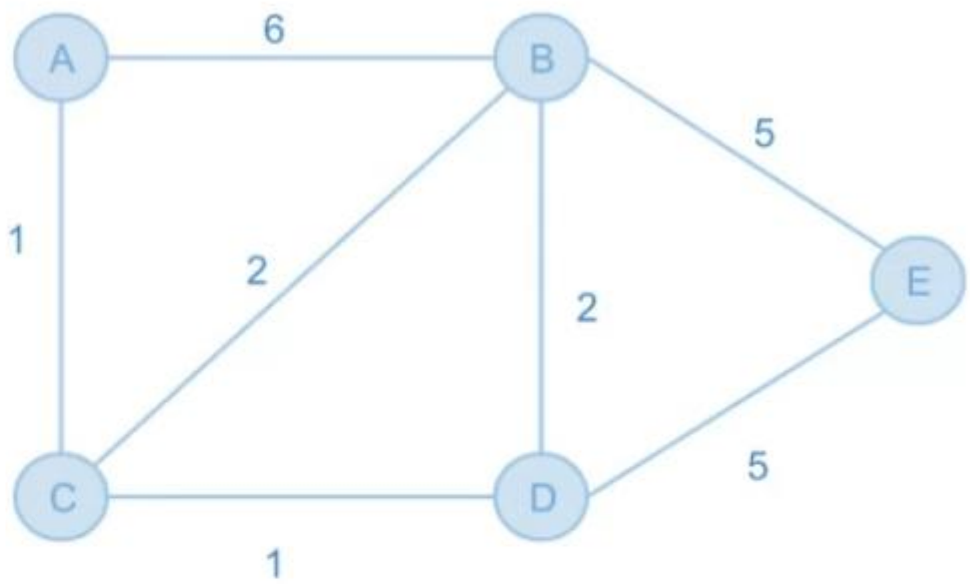
$O(m \log n)$

n la cantidad de nodos del problema y m la cantidad de aristas.

se basa en que las estructuras que usamos para
la cola de prioridad extraen e insertan un elemento en tiempo logarítmico.

Algoritmo de Dijkstra - Ejemplo :

Encuentra el camino mínimo de un vértice a todos los vértices

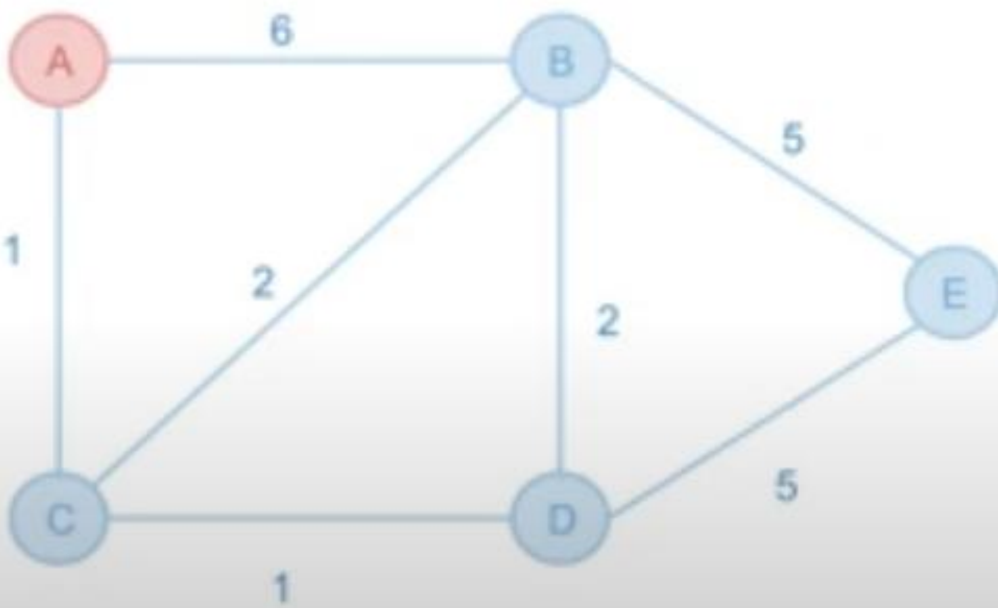


Vértice	Distancia	Padre

Heap:

Algoritmo de Dijkstra :

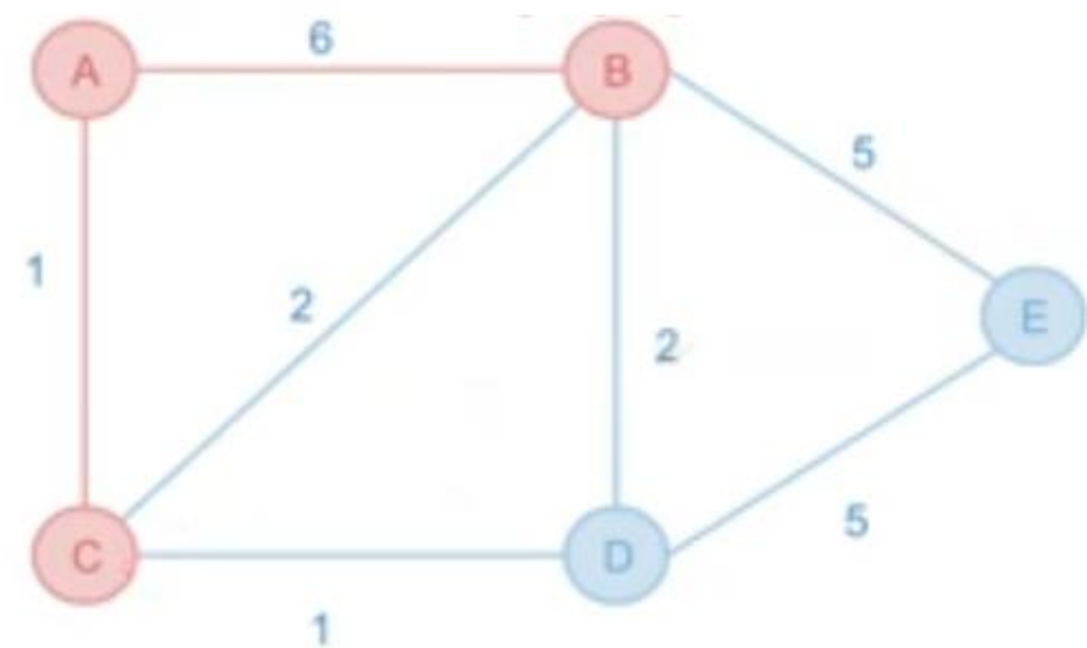
Visitar el vértice con la menor distancia desde el origen



Vértice	Distancia	Padre
A	0	
B	∞	
C	∞	
D	∞	
E	∞	

Heap: (A, 0)

Algoritmo de Dijkstra :



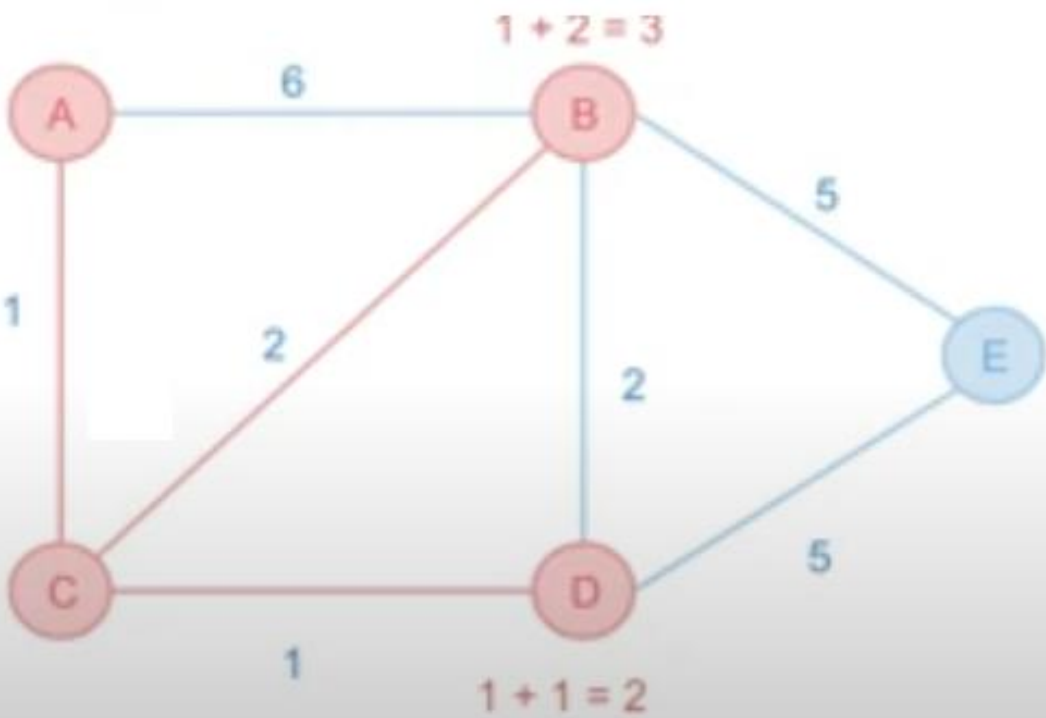
Vértice	Distancia	Padre
A	0	
B	∞	
C	∞	
D	∞	
E	∞	

Heap:

Clase 14: Problema del camino mínimo

Algoritmo de Dijkstra :

Calcular la distancia de cada adyacente desde el origen
Si su distancia actual es menor a la anterior, actualizar

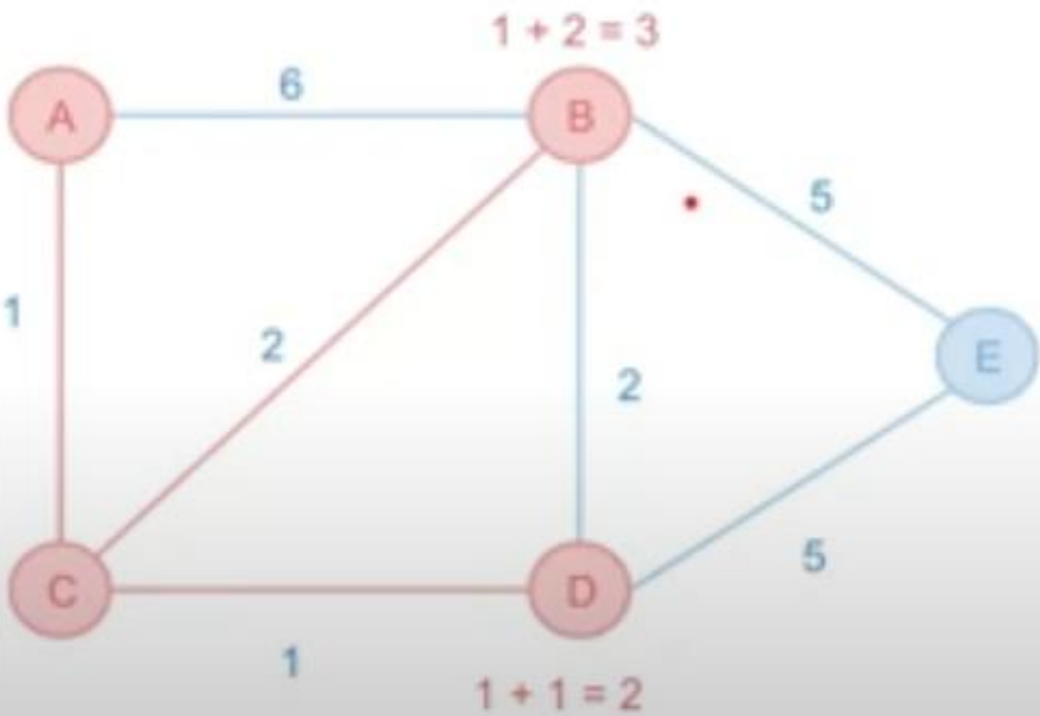


Vértice	Distancia	Padre
A	0	
B	6	A
C	1	A
D	∞	
E	∞	

Heap: (B, 6)

Algoritmo de Dijkstra :

Calcular la distancia de cada adyacente desde el origen
Si su distancia actual es menor a la anterior, actualizar

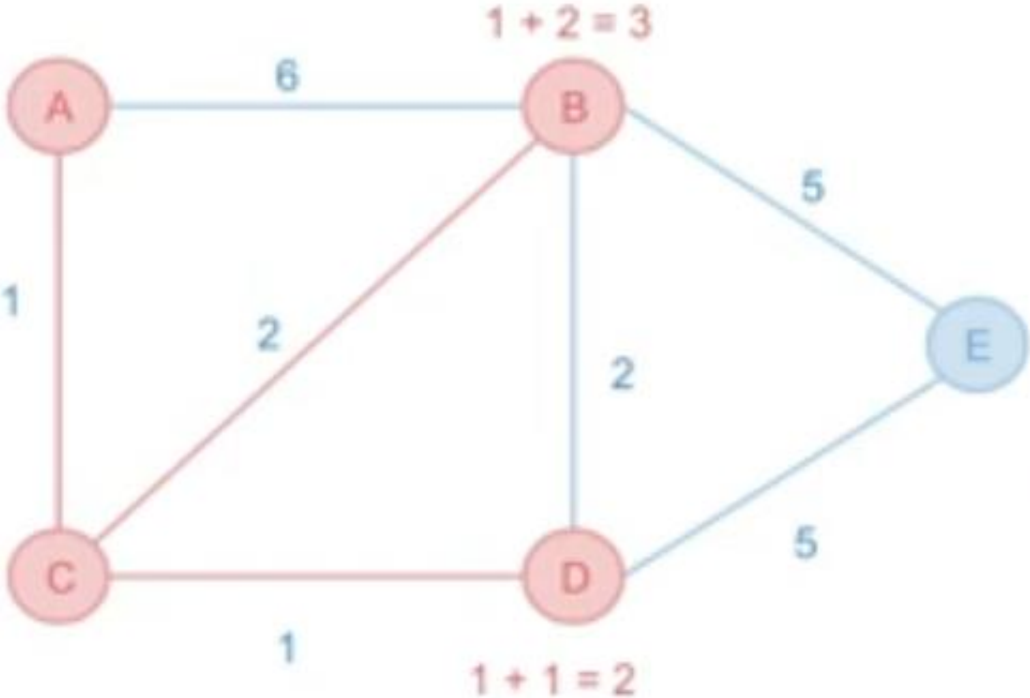


Vértice	Distancia	Padre
A	0	
B	6	A
C	1	A
D	∞	
E	∞	

Heap: (B, 6)

Algoritmo de Dijkstra :

Calcular la distancia de cada adyacente desde el origen
Si su distancia actual es menor a la anterior, actualizar

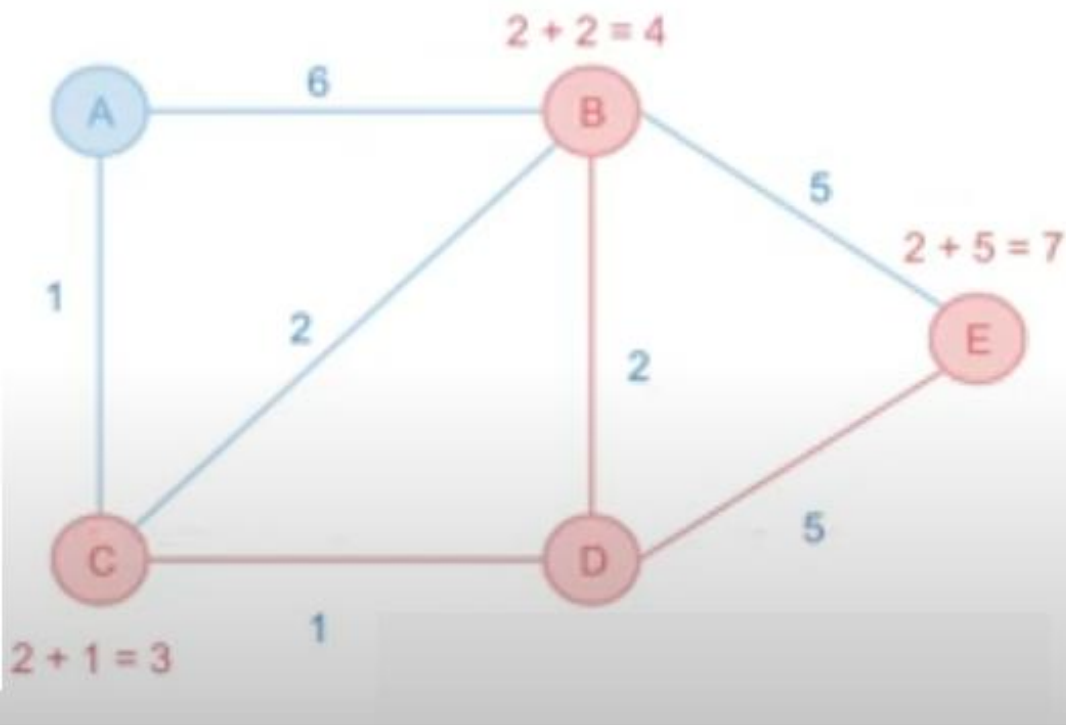


Vértice	Distancia	Padre
A	0	
B	3	C
C	1	A
D	2	C
E	∞	

Heap: (D, 2), (B, 3)

Clase 14: Problema del camino mínimo

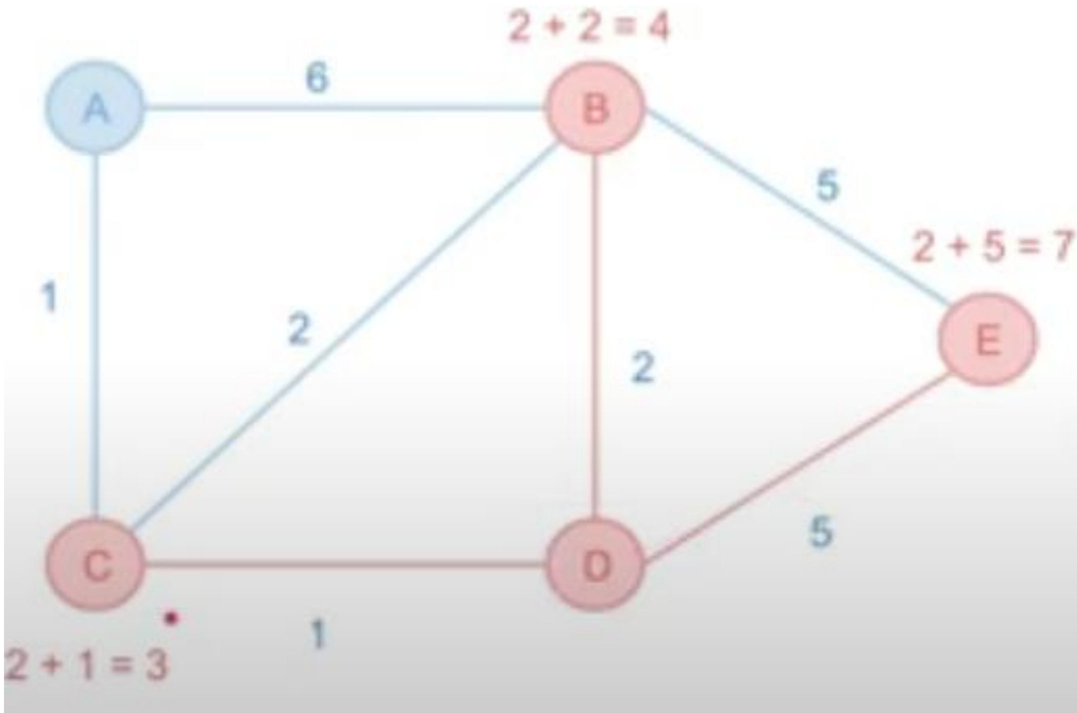
Algoritmo de Dijkstra :



Vértice	Distancia	Padre
A	0	
B	3	C
C	1	A
D	2	C
E	7	D

Heap: (B, 3), (E, 7)

Algoritmo de Dijkstra :



Vértice	Distancia	Padre
A	0	
B	3	C
C	1	A
D	2	C
E	7	D

Heap: (B, 3), (E, 7)

Clase 14: Problema del camino mínimo

Algoritmo de Bellman Ford :

Grafos con pesos negativos

```
def camino_minimo_bf(grafo, origen):
    dist = {}
    padre = {}
    for v in grafo:
        distancia[v] = infinito
    dist[origen] = 0
    padre[origen] = None
    aristas = obtener_aristas(grafo)
    for i in range(len(grafo)):
        for v, w, peso in aristas:
            if dist[v] + peso < dist[w]:
                padre[w] = v
                dist[w] = dist[v] + peso

    for v, w, peso in aristas:
        if dist[v] + peso < dist[w]:
            return None # Hay un ciclo negativo (lanzar excep)
    return padre, dist
```

$O(V * E)$

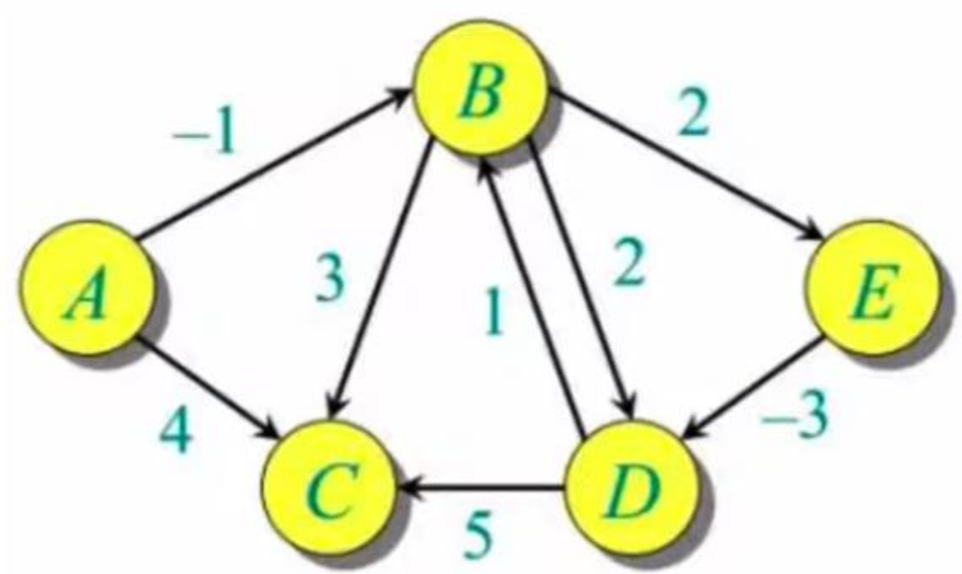
Clase 14: Problema del camino mínimo

Grafos con pesos negativos

Algoritmo de Bellman Ford :

Algoritmo de Bellman-Ford

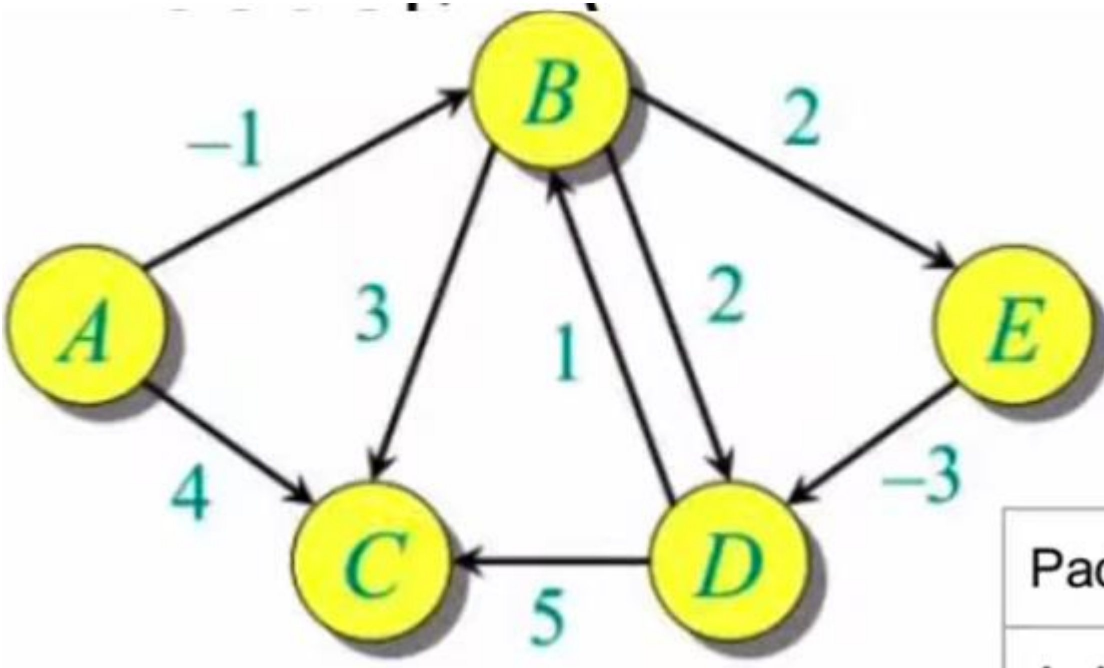
Ejemplo (sin ciclo negativo)



Padre	Distancia
A: None	A: 0 B: inf C: inf D: inf E: inf

Algoritmo de Bellman Ford :

Grafos con pesos negativos

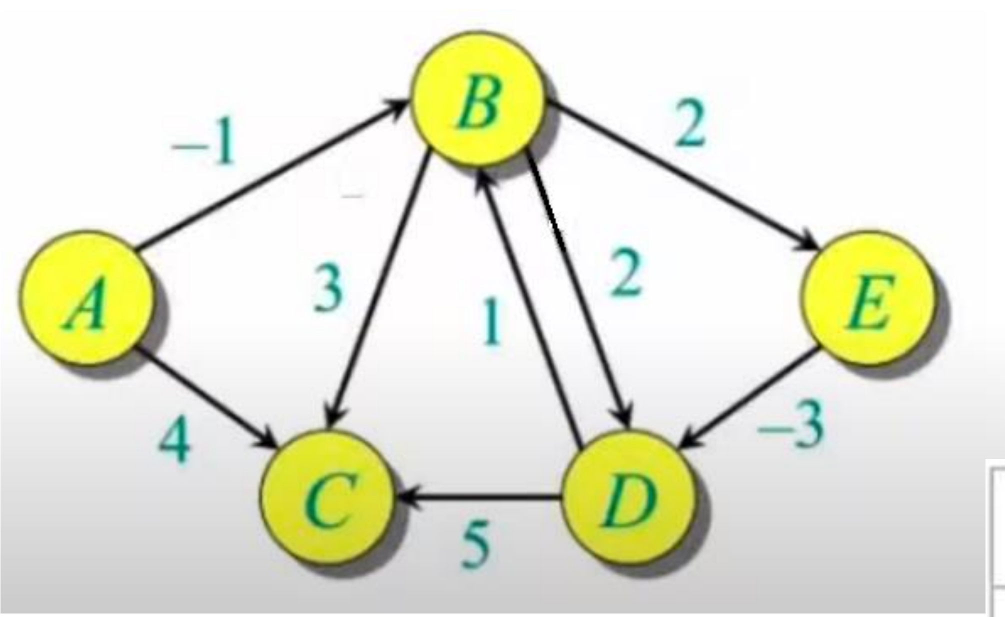


Padre	Distancia
A: None B: A C: A	A: 0 B: -1 C: 4 D: inf E: inf

Clase 14: Problema del camino mínimo

Algoritmo de Bellman Ford :

Grafos con pesos negativos

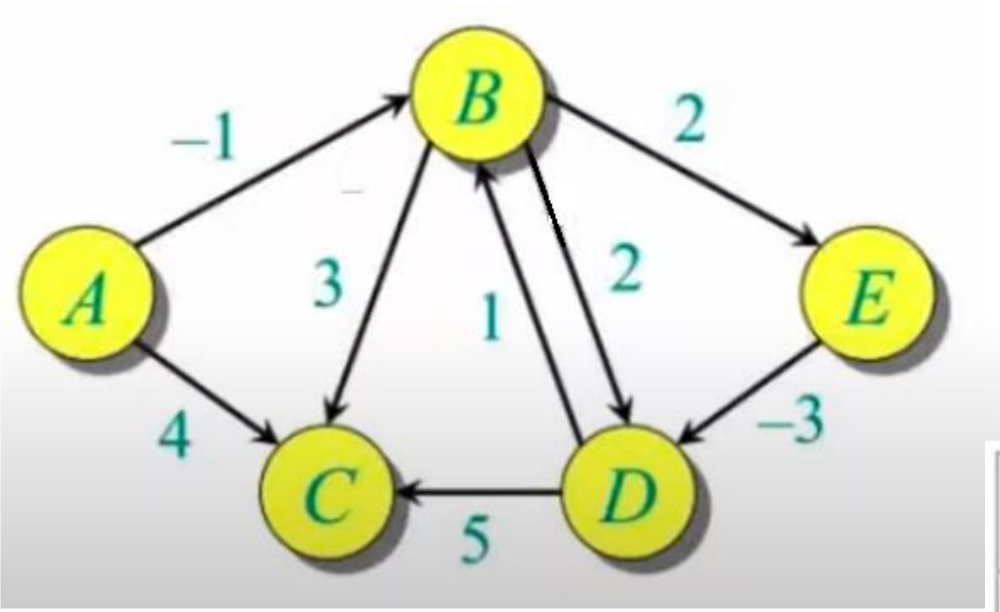


Padre	Distancia
A: None B: A C: B E: B D: B	A: 0 B: -1 C: 2 D: 1 E: 1

Clase 14: Problema del camino mínimo

Algoritmo de Bellman Ford :

Grafos con pesos negativos



Padre	Distancia
A: None B: A C: B E: B D: E	A: 0 B: -1 C: 2 D: -2 E: 1

Algoritmo de Floyd :

Algorithm Floyd-Warshall (A, n):

    for $k \leftarrow 1$ **to** n **do**

        for $i \leftarrow 1$ **to** n **do**

            for $j \leftarrow 1$ **to** n **do**

$$A[i][j] \leftarrow min(A[i][j], A[i][k] + A[k][j])$$

Pseudocódigo del algoritmo Floyd-Warshall

Estrategia: Algoritmo de Floyd

Lleva dos matrices D y P, ambas de |V| x |V|

Matriz de costos mínimos

Matriz de vértices intermedios

Para ver si hay ciclos negativos, hay que ver si hay números negativos en la diagonal.

Estimación del tiempo de ejecución

Ahora, calculamos el tiempo de ejecución $t(n)$ del algoritmo Floyd-Warshall sumando el costo obtenido de las operaciones primitivas.

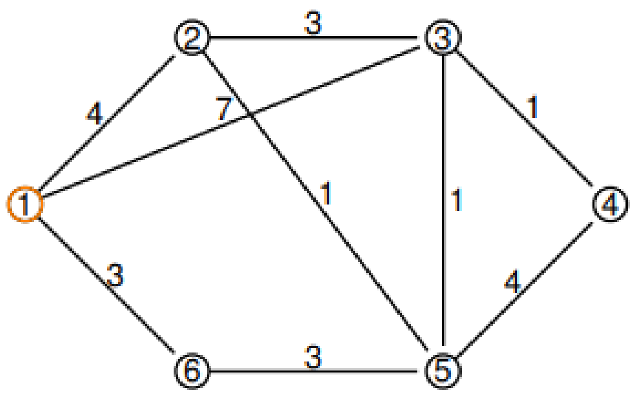
En conclusión, el tiempo de ejecución del algoritmo Floyd-Warshall es es $O(n^3)$.

Árbol generador mínimo:

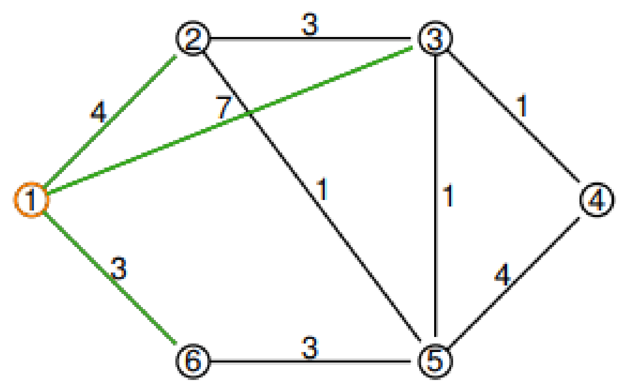
- El **algoritmo de Prim** mantiene un conjunto de nodos C que son los nodos que ya fueron conectados entre sí.
- En cada paso, elige la arista más barata entre algún nodo de C y un nodo no agregado todavía. Al elegir la arista, se agrega el nodo nuevo a C.
- De esta forma, en cada paso hay un nodo nuevo sumándose al conjunto de nodos ya conectados entre sí.
- Luego de $n - 1$ pasos, habremos agregado todos los nodos de la forma más barata posible: obtendremos un árbol generador mínimo.

Algoritmo de Prim - Ejemplo:

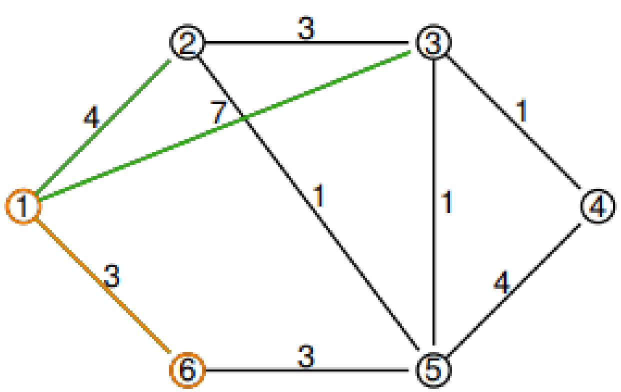
$S = \{1\}$ $\pi = (0, \infty, \infty, \infty, \infty, \infty)$



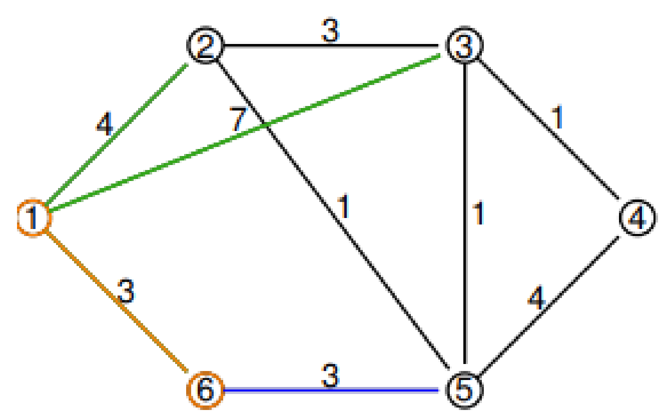
$S = \{1\}$ $\pi = (0, 4, 7, \infty, \infty, 3)$



$S = \{1, 6\}$ $\pi = (0, 4, 7, \infty, \infty, 3)$

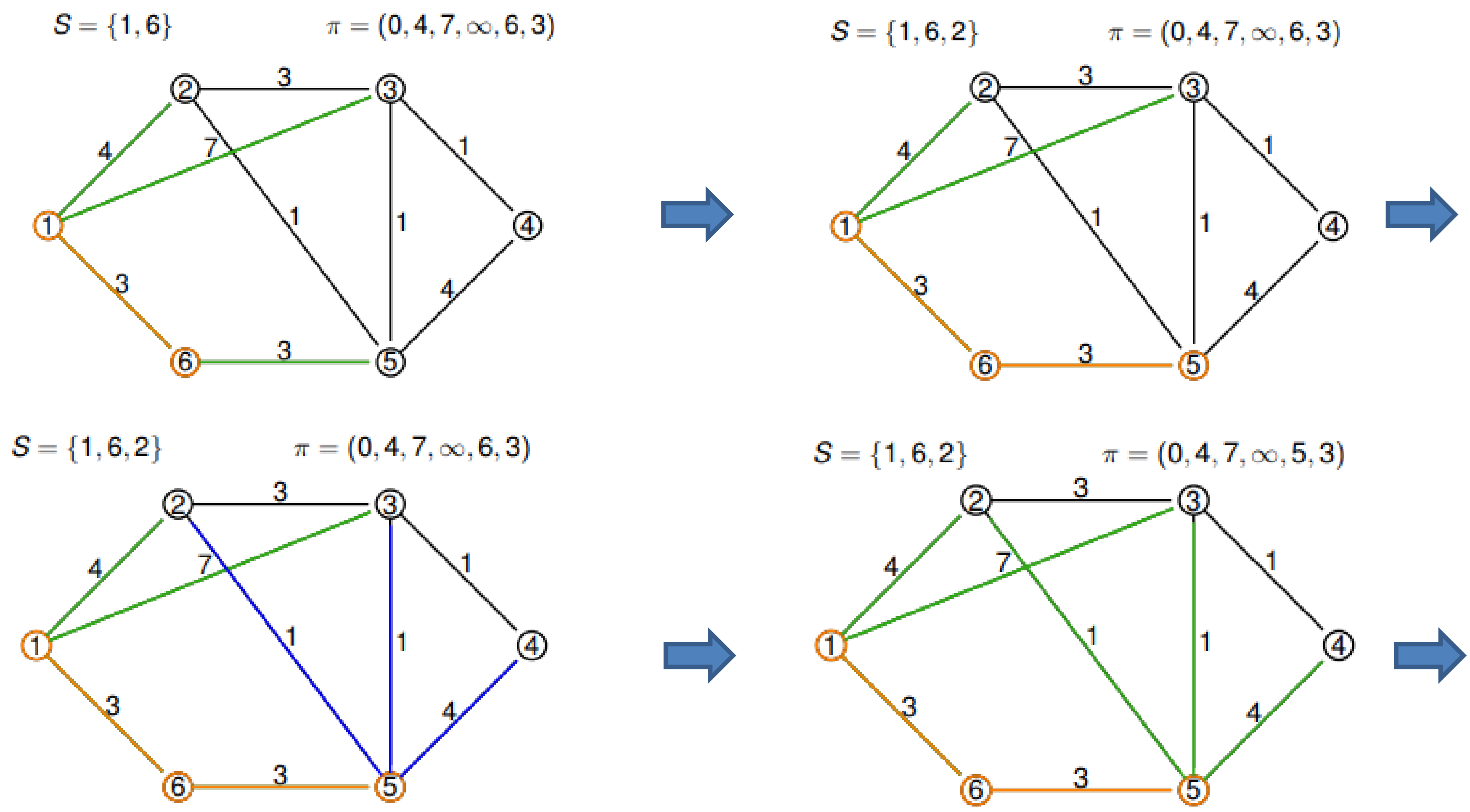


$S = \{1, 6\}$ $\pi = (0, 4, 7, \infty, \infty, 3)$



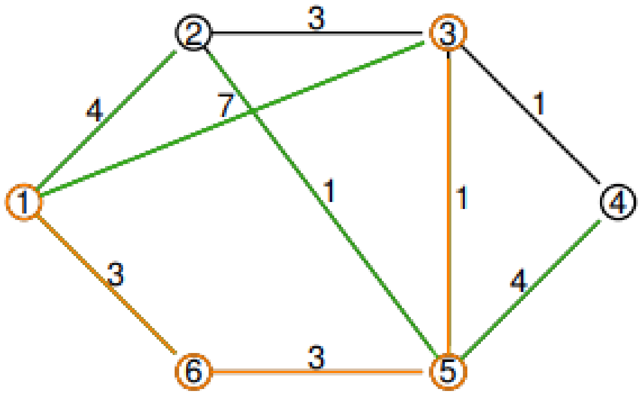
Clase 14: Problema del camino mínimo

Algoritmo de Prim - Ejemplo:

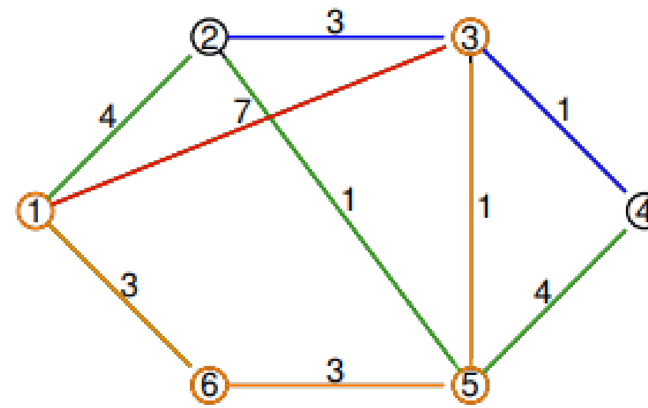


Algoritmo de Prim - Ejemplo:

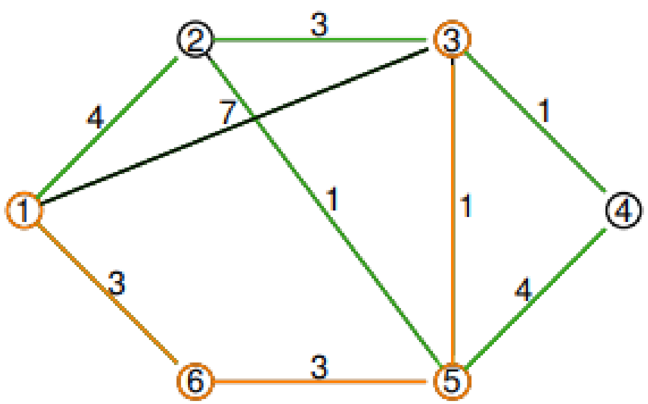
$S = \{1, 6, 2, 5\}$ $\pi = (0, 4, 7, \infty, 5, 3)$



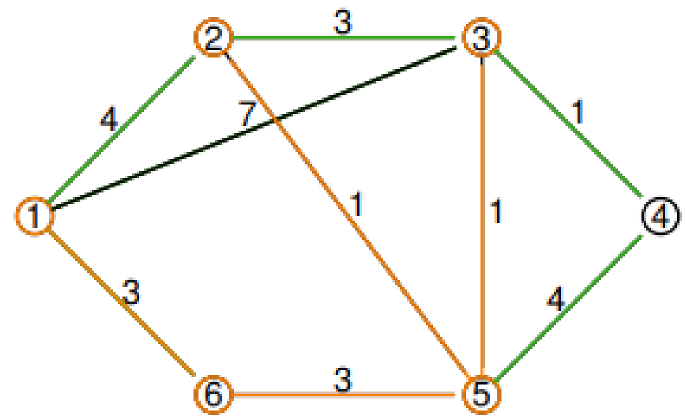
$S = \{1, 6, 2, 5\}$ $\pi = (0, 4, 7, \infty, 5, 3)$



$S = \{1, 6, 2, 5\}$ $\pi = (0, 4, 7, 9, 5, 3)$

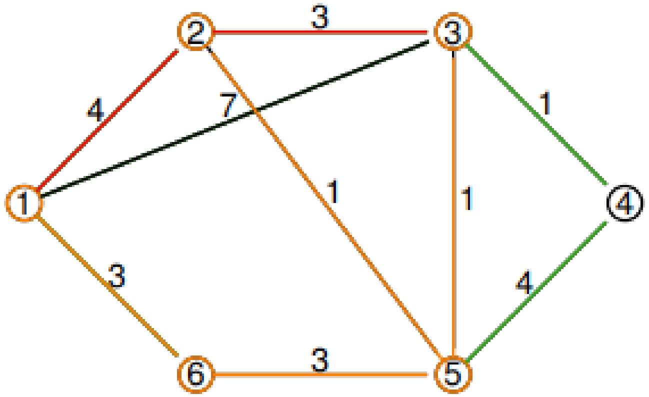


$S = \{1, 6, 2, 5, 3\}$ $\pi = (0, 4, 7, 9, 5, 3)$

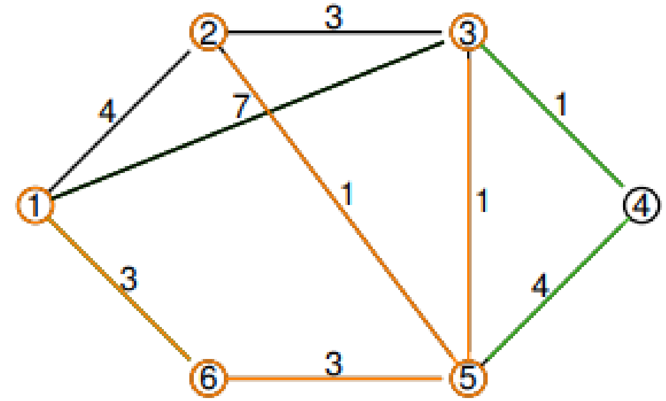


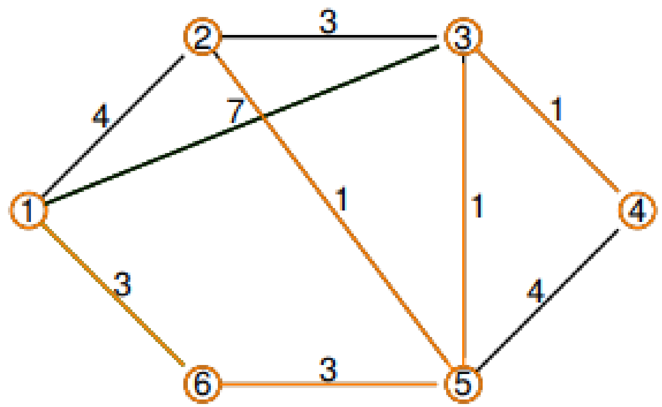
Algoritmo de Prim - Ejemplo:

$S = \{1, 6, 2, 5, 3\}$ $\pi = (0, 4, 7, 9, 5, 3)$



$S = \{1, 6, 2, 5, 3\}$





Clase 14: Problema del camino mínimo

Algoritmo de Prim - Ejemplo:

Pseudocódigo de Prim sin cola de prioridad

```
Prim (Grafo G, nodo inicial s)
   visitado[n] = {false, ..., false} // guarda si un nodo ya fue visitado
   distancia[n] = {Infinito, ..., Infinito} // guarda las distancias de cada
         nodo al conjunto visitado

   para cada w en V[G] hacer
      si existe arista entre s y w entonces
         distancia[w] = peso (s, w)

   distancia[s] = 0
   visitado[s] = true
   mientras que no esten visitados todos hacer
      v = nodo de menor distancia del conjunto que no fue visitado aun (itero
            el arreglo)
      visitado[v] = true
      para cada w en los vecinos de v hacer
         si distancia[w] > peso (v, w) entonces
            distancia[w] = peso (v, w)
            padre[w] = v
```

Algoritmo de Prim - Ejemplo:

Pseudocódigo de Prim con cola de prioridad

```
Prim (Grafo G, nodo_fuente s)
    para todo u en V[G] hacer
        distancia[u] = INFINITO
        padre[u] = NULL
        visitado[u] = false

    distancia[s] = 0
    adicionar (cola, (distancia[s], s))

    mientras que cola no sea vacia hacer
        u = extraer_minimo(cola)
        visitado[u] = true
        para todo v en vecinos de u hacer
            si no visitado[v] y distancia[v] > peso(u, v) hacer
                distancia[v] = peso(u, v)
                padre[v] = u
                adicionar(cola, (distancia[v], v))
```

Algoritmo de Prim:

Tiempo de Ejecución

Se hacen las mismas consideraciones que para el algoritmo de Dijkstra:

- Si se implementa con una tabla :

  El costo total del algoritmo es $O(|V|^2)$

- Si se implementa con heap:

  El costo total del algoritmo es $O(|E| \log|V|)$

Clase 14: Problema del camino mínimo

Algoritmo de Kruskal:

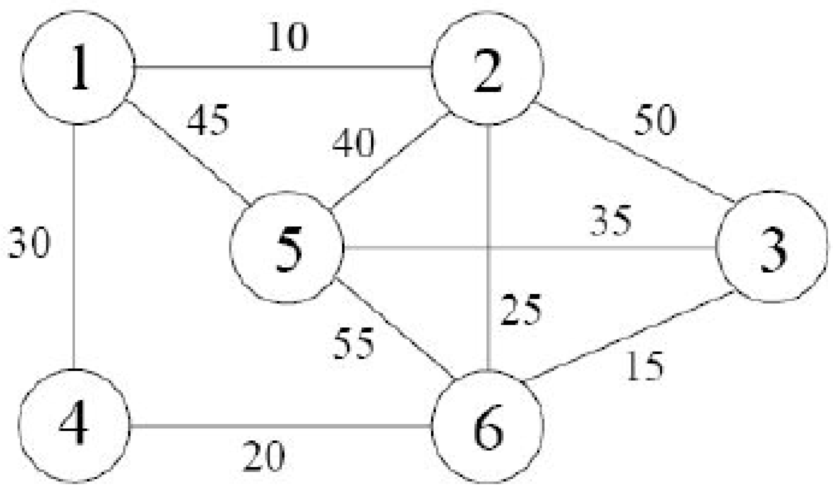
- Selecciona las aristas en orden creciente según su peso y las acepta si no originan un ciclo.
- El invariante que usa me indica que en cada punto del proceso, dos vértices pertenecen al mismo conjunto si y sólo sí están conectados.
- Si dos vértices **u** y **v** están en el mismo conjunto, la arista (**u**,**v**) es rechazada porque al aceptarla forma un ciclo.
- Inicialmente cada vértice pertenece a su propio conjunto

  |V| conjuntos con un único elemento
- Al aceptar una arista se realiza la Unión de dos conjuntos.
- Las aristas se organizan en una heap, para ir recuperando la de mínimo costo en cada paso.

Clase 14: Problema del camino mínimo

Algoritmo de Kruskal:

Ejemplo:



Aristas ordenadas por su costo de menor a mayor:

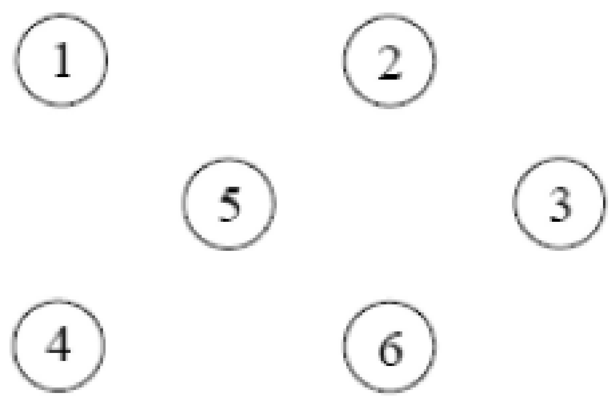
(1,2)	10
(3,6)	15
(4,6)	20
(2,6)	25
(1,4)	30
(5,3)	35
(5,2)	40
(1,5)	45
(2,3)	50
(5,6)	55

- Ordenar las aristas, usando un algoritmo de ordenación
- Construir una min-heap **más eficiente**

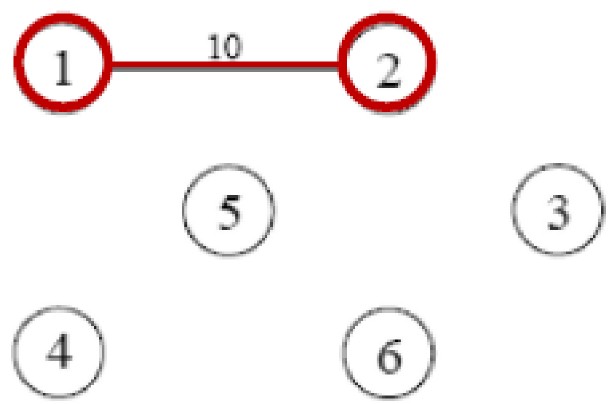
Clase 14: Problema del camino mínimo

Algoritmo de Kruskal:

Inicialmente cada vértice está en su propio conjunto

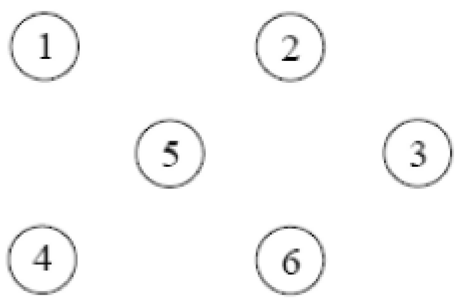


Se agrega la arista (1,2)

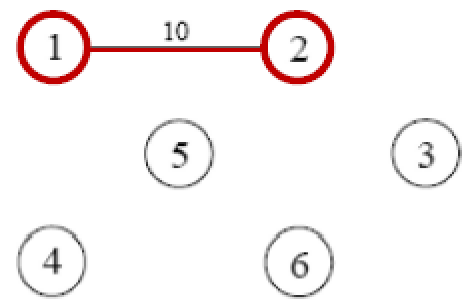


Algoritmo de Kruskal:

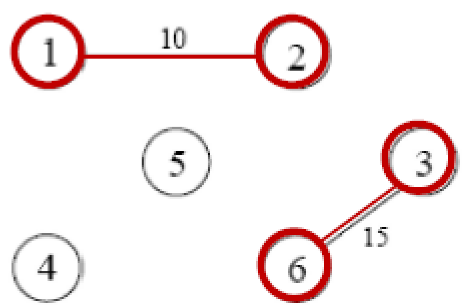
Inicialmente cada vértice está en su propio conjunto

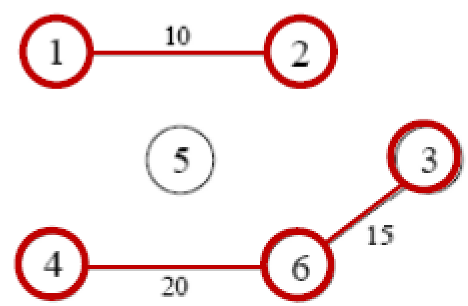


Se agrega la arista (1,2)



Se agrega la arista (3,6)

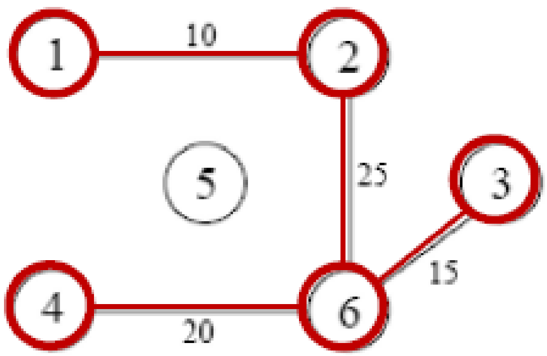




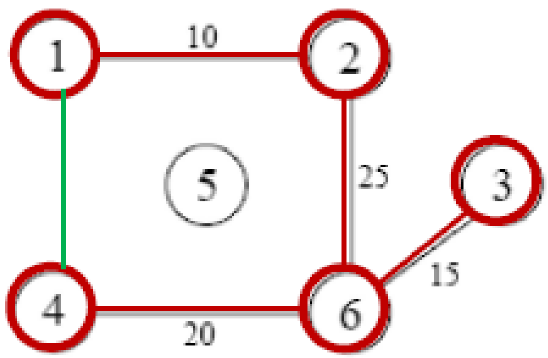
Se agrega la arista (4,6)

Algoritmo de Kruskal:

Se agrega la arista (2,6)



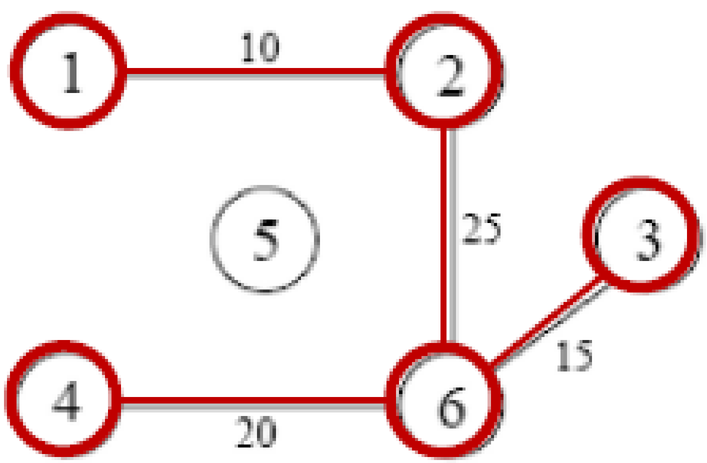
¿Se agrega la arista (1,4) con costo 30?



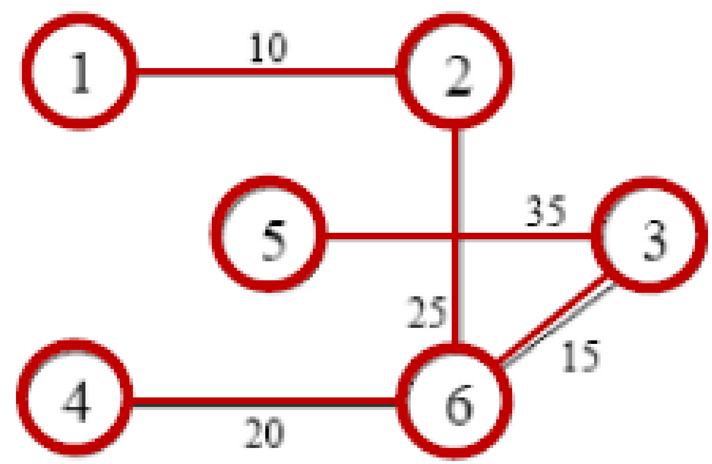
No, porque forma ciclo, ya que pertenece a la misma componente conexa

Algoritmo de Kruskal:

Se agrega la arista (2,6)



Se agrega la arista (3,5)



Algoritmo de Kruskal:

Tiempo de Ejecución

Se organizan las aristas en una heap, para optimizar la recuperación de la arista de mínimo costo

El tamaño de la heap es $|E|$, y extraer cada arista lleva $O(\log |E|)$

El tiempo de ejecución es $O(|E|\log|E|)$

Dado que $|E| \leq |V|^2$, $\log |E| \leq 2 \log |V|$,

el costo total del algoritmo es $O(|E| \log|V|)$

Clase 14: Problema del camino mínimo

Conclusión:

Existen muchos algoritmos “clásicos” para resolver diferentes problemas sobre grafos para resolver los problemas de camino más corto vimos:

- La búsqueda de amplitud y la búsqueda de profundidad, se refieren a diferentes órdenes de búsqueda.
- El algoritmo de Dijkstra resuelve el problema del camino más corto de un origen si todos los pesos de las aristas son mayores o iguales a cero.
- El algoritmo de Bellman-Ford también resuelve el problema de los caminos más cortos de una sola fuente, pero a diferencia del algoritmo de Dijkstra, los pesos de las aristas pueden ser negativos.
- El algoritmo de Floyd-Warshall resuelve el problema de las rutas más cortas de todos los pares.

Nuestra tarea consiste en modelar los problemas de interés usando grafos y encontrar el algoritmo adecuado para la aplicación que se requiera.

Consultas