UNIDAD 1

Grafos

Introducción a la teoría de grafos

Árboles de expansión de costo mínimo

Árboles de costo mínimo

Árbol de expansión de costo mínimo

Árbol de expansión

Introduciremos algunos conceptos necesarios para la exposición del problema que se pretende resolver.

Un grafo no dirigido, acíclico y conexo se denomina **árbol libre**.

Un árbol libre cumple las siguientes propiedades:

1. Un árbol libre con n>= 1 nodos contiene exactamente n − 1 aristas.
2. Si se añade una nueva arista a un árbol libre, se crea un ciclo.
3. Cualquier par de vértices están conectados por un único camino.

Árboles de costo mínimo

Definiciones:

Un **árbol de expansión de un grafo no dirigido** G = (V , E), es un árbol libre T = (V´,E´), tal que **V´ = V y E´⊆ E**; es decir T contiene **todos** los vértices de G, y las aristas de T son aristas de G.

Dado un grafo no dirigido y ponderado mediante una función de ponderación w, el **costo de un árbol de expansión** T es la suma de los costos de todas los aristas del árbol.

$$w(T) = \sum_{(u,v) \in T} w(u,v)$$

Árboles de costo mínimo

Un **árbol de expansión de un grafo será de costo mínimo**, si se cumple que su costo es el menor posible respecto al costo de cada uno de los posibles árboles de expansión que contiene el grafo.

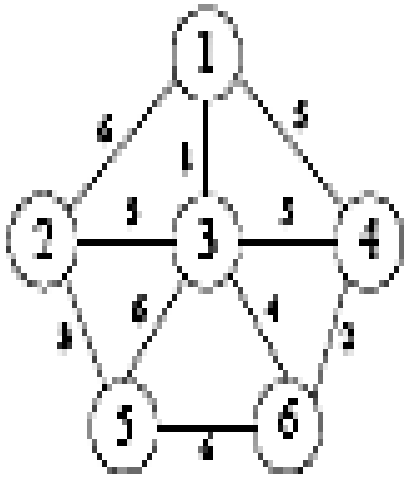
Árboles de costo mínimo

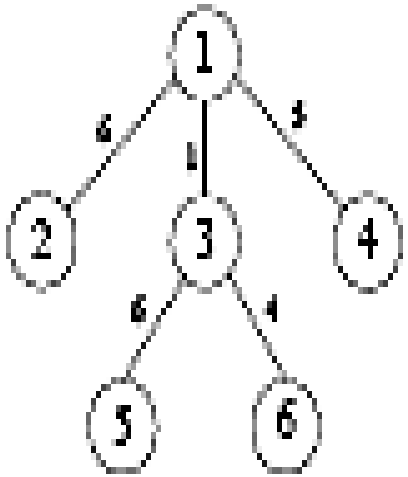
El propósito de obtener un árbol de recubrimiento de costo mínimo es obtener un nuevo grafo que sólo contenga las aristas imprescindibles para una optimización global de las conexiones entre todos los nodos.

"Optimización global": algunos pares de nodos pueden no quedar conectados entre ellos con el mínimo costo posible.

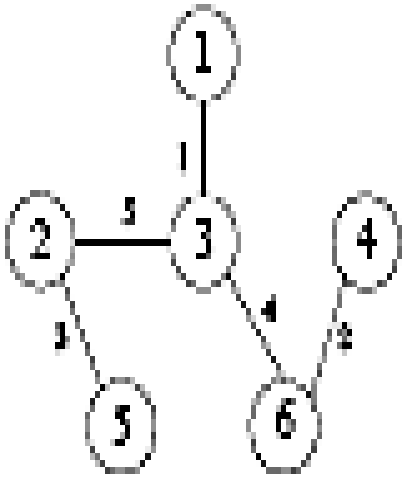
Árboles de costo mínimo

Ejemplos de árboles de expansión





Árbol de expansión
de costo 22



Árbol de expansión
de costo 15

Árboles de costo mínimo

Se puede plantear el problema de obtener el árbol de expansión de costo mínimo para un grafo dado de esta forma:

Problema: Sea G = (V , E) un grafo no dirigido, conexo y ponderado mediante pesos asociados a las aristas; se desea obtener un nuevo grafo G´ = (V ,E´) donde E´$\subseteq$ E tal que G´ sea un árbol de expansión de costo mínimo de G.

Existen dos algoritmos clásicos que resuelven este problema: el algoritmo de PRIM y el algoritmo de KRUSKAL

Algoritmo de Prim

El algoritmo de Prim hace la selección en forma local, comenzando por cualquier vértice para construir incrementalmente un árbol de recubrimiento.

Algoritmo de Prim. Características

- el árbol toma como raíz un vértice cualquiera del grafo, $u \in V$ y a partir de este se extiende por todos los vértices del grafo,
- en cada paso se selecciona la arista de menor peso que une un vértice presente en un conjunto S con uno de V que no lo esté,
- si se añade (u,v) al conjunto de aristas obtenido hasta el momento no se crea ningún ciclo,
- se asegura que produzca el menor incremento de peso posible,
- los arcos del árbol de expansión parcial forman un único árbol,

Algoritmo de Prim

PRIM utiliza las siguientes estructuras durante la ejecución del algoritmo:

- Un conjunto A en el que se guardan aquellas aristas que ya forman parte del árbol de expansión de costo mínimo
- Un conjunto S inicialmente vacío al que se añaden los vértices de V conforme se vayan recorriendo para formar el árbol de expansión

Algoritmo de Prim

Prim (G) {

A= $\varnothing$

u= elemento arbitrario de V

S={u}

mientras S $\neq$ V

(u,v) =arco_min_peso (x,y) $_{(x,y) \in S \times (V-S)}$

A=A $\cup$ {(u,v)}

S=S $\cup$ {v}

retorna (A)

Algoritmo de Prim

```
arco_min_peso (x,y) (x,y) ∈ S x (V-S)
m= infinito
para x ∈ S
   para y ∈ V-S
      si (costo (x,y) < m
         m=costo (x,y)
         (u,v)= (x,y)
      fin
   fin
fin
```

vértice que no pertenece a la solución

vértice que pertenece a la solución

Algoritmo de Prim

Ejemplo + Seguimiento: Pizarrón

PRIM. Complejidad temporal

Bucle principal (mientras) se realiza $|V|-1$ iteraciones, debido a que en la primera iteración el conjunto S solo tiene un vértice y a cada iteración solo se añade un nuevo vértice.

La operación del cálculo de la arista factible de menor peso, tiene costo $O(|V|^2)$

Luego el costo es $O(|V|^3)$

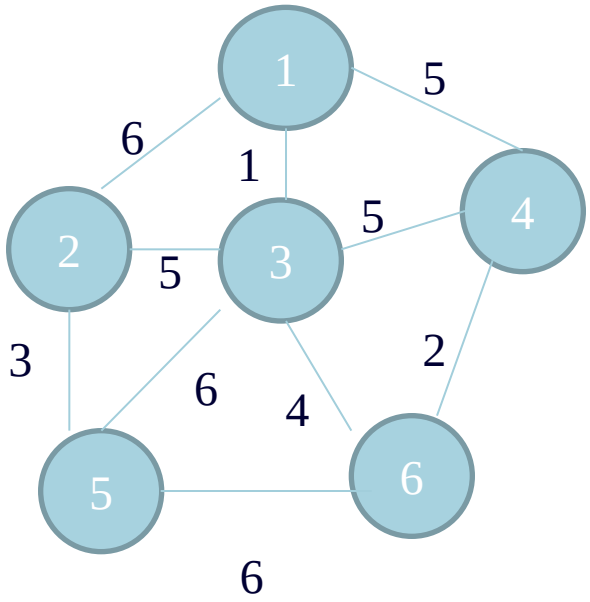
PRIM. Complejidad temporal

Sin embargo, existen optimizaciones para el algoritmo del cálculo de:

arco_min_peso $(x,y)_{(x,y) \in S \times (V-S)}$

Se consigue bajar a un costo de $O(|V|)$ y por lo tanto, el costo de PRIM se reduce a $O(|V|^2)$.

PRIM. Implementación con arreglos



estado inicial

1	0	0	0	0	0	S

0	6	1	5	∞	∞	Menor_costo

1	1	1	1	1	1	Mas_cercano

PRIM. Implementación con arreglos

```
Prim (costo [n][n])
   S[1]=1;
   for (i=2;i<=n;i++)
      {Menor_costo[i]=costo[1][i];
       Mas_cercano[i]=1;
       S[i]=0;}
    for (i=2;i<=n;i++)
      elegir w/Menor_costo[w] sea min .and. ∉ solucion
      S[w]=1;
      for(j=2;j<=n;j++) //actualiz de costos
         if (costo[w][j]<Menor_costo[j] && S[j]=0)
          { Menor_costo[j]=costo[w][j];
           Mas_cercano[j]=w;}
```

Complejidad temporal: $O(n^2)$

PRIM. Implementación con arreglos

1	0	0	0	0	0

S

ESTADO INICIAL

0	6	1	5	∞	∞

Menor_costo

1	1	1	1	1	1

Mas_cercano

1	0	1	0	0	0

S

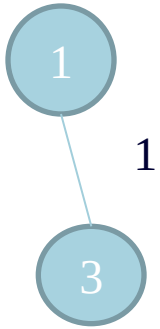
SE SELECCIONA VERTICE 3

	5	1	5	6	4

Menor_costo

	3	1	1	3	3

Mas_cercano



PRIM. Implementación con arreglos

Se actualizan los arreglos:

1	0	1	0	0	1

S

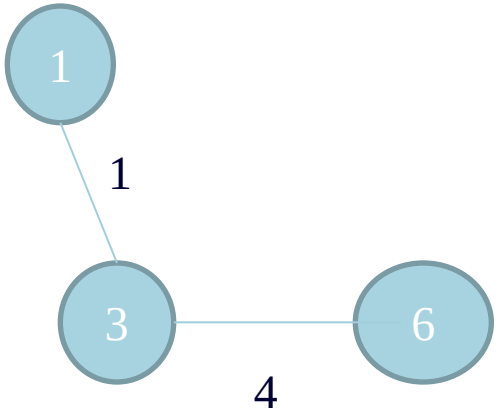
	5	1	2	6	4

Menor_costo

	3	1	6	3	3

Mas_cercano

SE SELECCIONA VERTICE 6



PRIM. Implementación con arreglos

Se actualizan los arreglos:

1	0	1	1	0	1

S

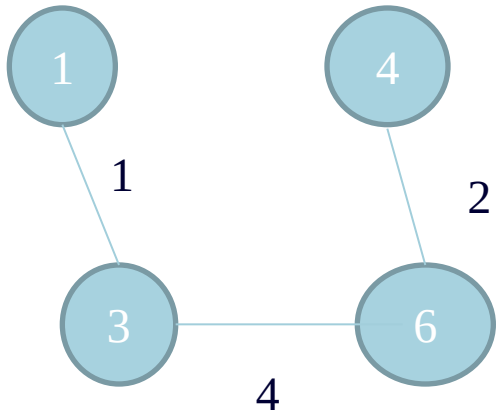
	5	1	2	6	4

Menor_costo

	3	1	6	3	3

Mas_cercano

SE SELECCIONA VERTICE 4



PRIM. Implementación con arreglos

Se actualizan los arreglos:

1	1	1	1	0	1

S

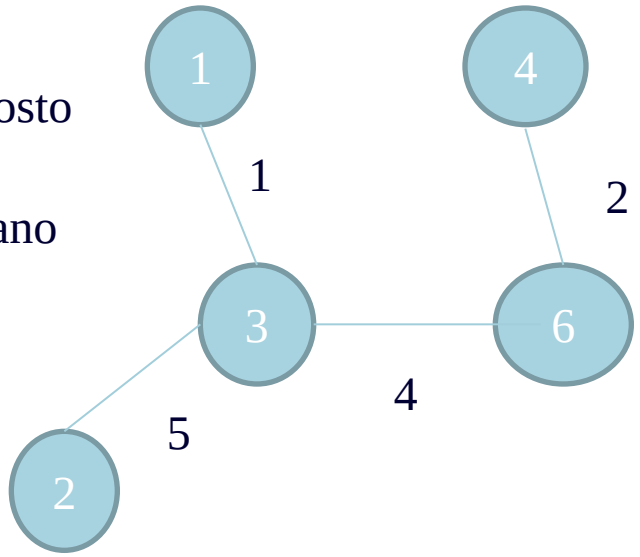
	5	1	2	3	4

Menor_costo

	3	1	6	2	3

Mas_cercano

SE SELECCIONA VERTICE 2



PRIM. Implementación con arreglos

Se actualizan los arreglos:

1	1	1	1	1	1

S

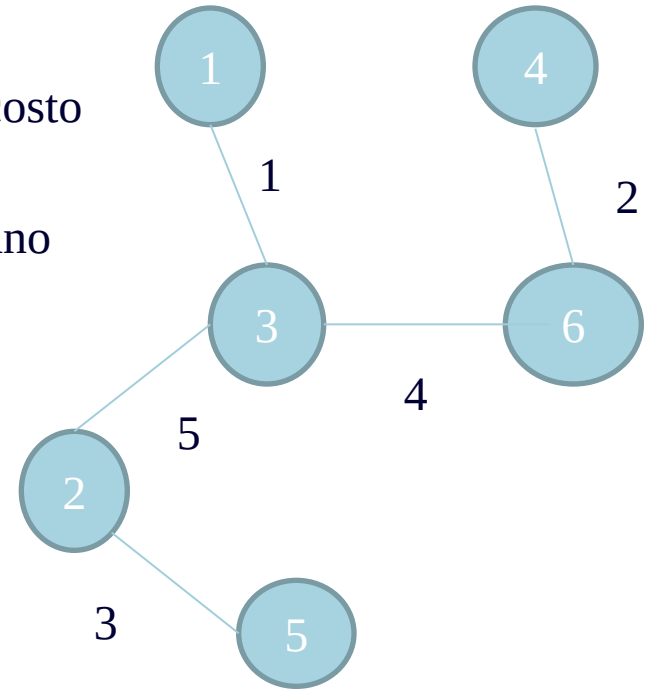
	3	1	2	3	4

Menor_costo

	5	1	6	2	3

Mas_cercano

SE SELECCIONA VERTICE 5



PRIM. Implementación con Heap

```
Prim (G)
  heap *h; S[1]=1;
  for (i=2;i<=n;i++)
    {h->insertar(costo[1][i],i);          -> O(logn)
     Mas_cercano[i]=1; S[i]=0;}
   for (i=2;i<=n;i++)
     h->eliminar (w)                                  -> O(logn)
     S[w]=1; T<- T ∪ {w, Mas_cercano[w]};
     for(v ∈ adyacentes (w) .and. S[v] =0)
       {if (h->consultar(v) > costo[w][v])
          h ->modificar(v, costo[w][v]);              -> O(logn)
         Mas_cercano[j]=w;}}
```

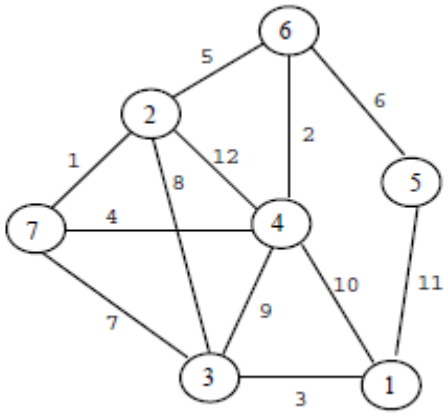
PRIM. Complejidad temporal

Cómo bajar la complejidad de Prim ?

Grafos poco densos: implementación con Heap

PRIM. Ejercicio:

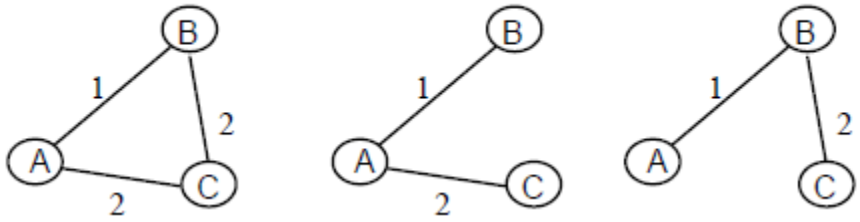
Construya el árbol de recubrimiento siguiendo Prim



PRIM. Ejercicio:

Supóngase que un grafo tiene exactamente 2 aristas que tienen el mismo peso.

(a) ¿Construye el algoritmo de *Prim* el mismo árbol de expansión mínimo, independientemente de cuál de esas aristas sea seleccionada antes?



PRIM. Ejercicios

- Implemente PRIM
- Implemente eficientemente la función del cálculo de la arista factible

Algoritmo de Kruskal

Estrategia Kruskal:

- Comienza con un grafo T=(V,∅) (n vértices, 0 arcos),
- cada vértice ∈ a una componente conectado por sí mismo,
- para construir componentes mas largas, se examina progresivamente los arcos desde E (se hace para cada arista),
- si el arco conecta 2 vértices en 2 componentes conectadas diferentes, entonces agregamos el arco a T,
- cuando todos los vértices están en una sola componente, T es el árbol expandido de costo mínimo para G.

Algoritmo de Kruskal

El algoritmo mantiene las siguientes estructuras:

- Un conjunto en el que se mantiene, en todo momento, aquellas aristas que ya han sido seleccionadas como pertenecientes al árbol de expansión de costo mínimo.

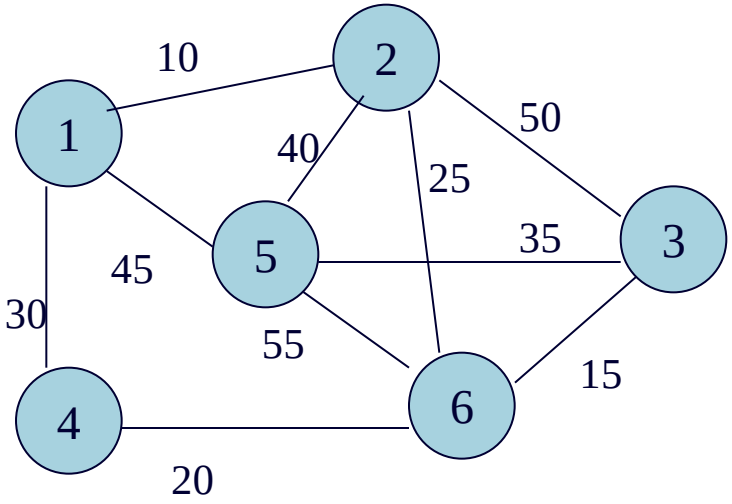
- Cuando el algoritmo finalice, el conjunto contendrá el subconjunto de aristas $E' \subseteq E$ que forman parte del árbol de expansión de costo mínimo.

Algoritmo de Kruskal

```
Kruskal(G) {
A=∅
para (cada vertice v ∈ V)
   crear_Subconjunto (v)
fin
//Ordenar las aristas pertenecientes a E, según su peso, en orden creciente
para (cada arista (u,v) ∈E, siguiendo orden creciente)
    si (Buscar(u) ≠ Buscar(v))
       A=A∪ {(u,v)}
       Union (Buscar(u), Buscar(v)
    fin
fin
Devuelve (A)
}
```

Algoritmo de Kruskal

Ejemplo



Lista de aristas ordenadas según costo (orden creciente):

(1,2)	10
(6,3)	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

Algoritmo de Kruskal

Ejemplo + Seguimiento: Pizarrón

Algoritmo de Kruskal

- La implementación de este algoritmo se hace usando conjuntos y operaciones sobre conjuntos
- Costo temporal del algoritmo:

  Para analizar el costo temporal del algoritmo asumiremos que en las operaciones sobre la EDD, Unión y Buscar, se aplican los algoritmos heurísticas union_por_rango y compresión _ de caminos