



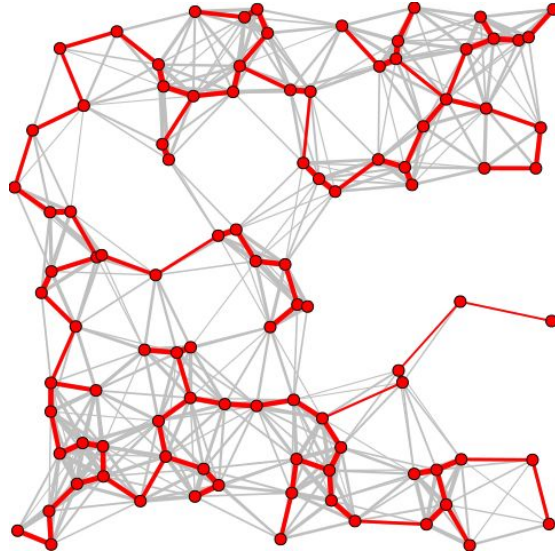
# Grafos: árbol de recubrimiento mínimo

AED II

2020

# árbol de recubrimiento mínimo

Dado un grafo  $G(A,V)$  un **árbol de recubrimiento mínimo** (MST - Minimum Spanning Tree) es un **árbol** (subgrafo conexo y sin ciclos) **que contiene todos los vértices del grafo ponderado no dirigido  $G$**  y además **su peso es mínimo** (menor o igual que el de los otros árboles de recubrimiento posibles)



# árbol de recubrimiento mínimo

Un árbol de recubrimiento mínimo es un subgrafo que contiene todos los vértices del grafo original pero tiene  $V-1$  aristas

La principal aplicación de los árboles de recubrimiento mínimos está en el campo del diseño de redes

# algoritmo de Kruskal

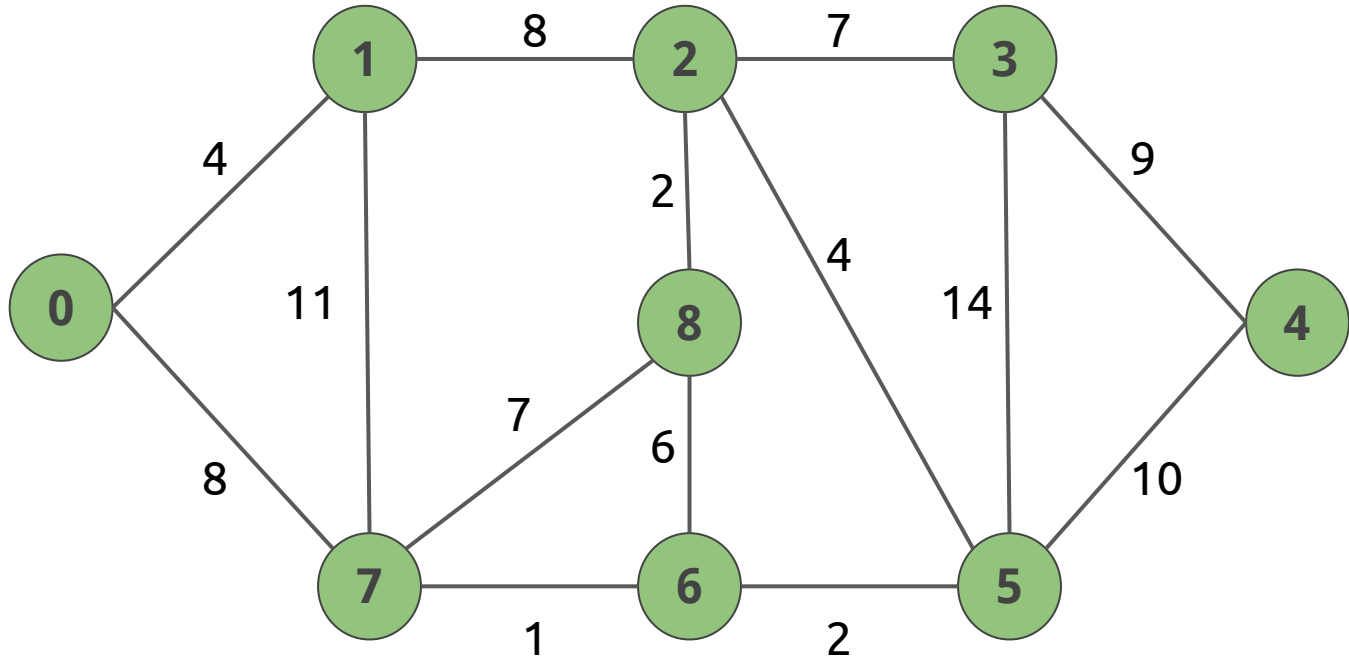


Joseph Kruskal

# algoritmo de Kruskal

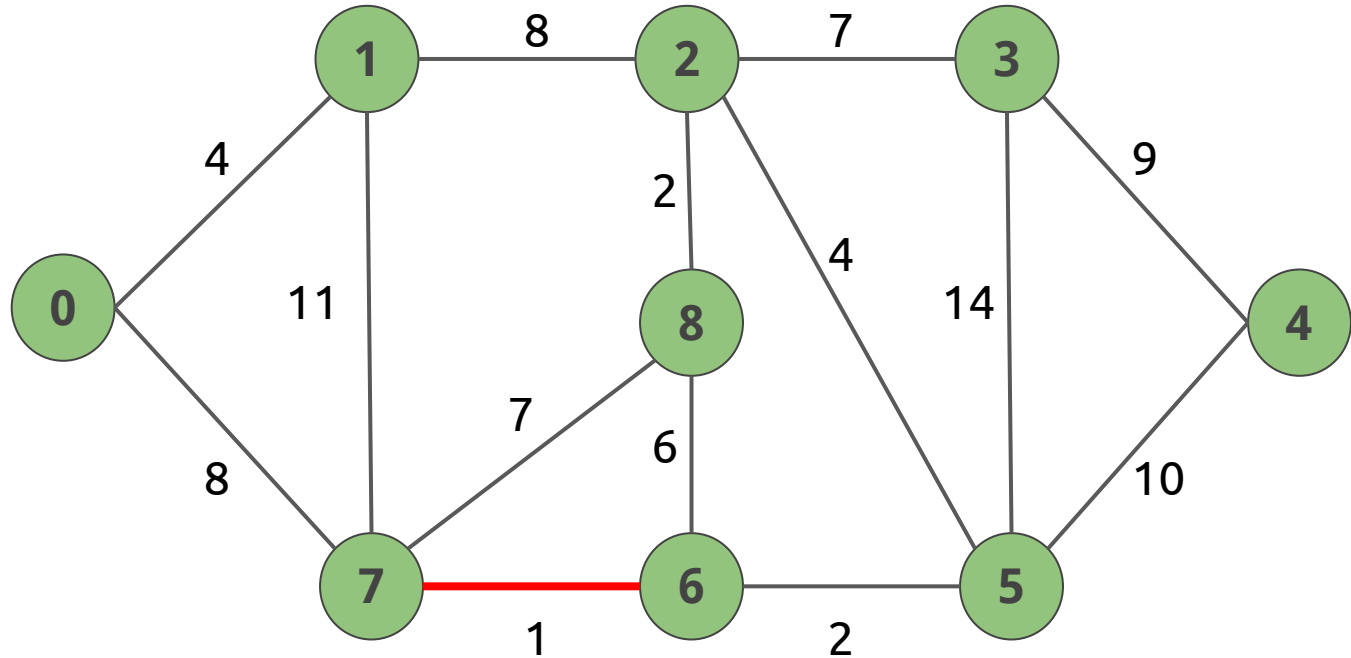
1. Ordenar las aristas en orden de peso creciente
2. Tomar la arista con menor peso y agregarla al árbol, si la arista forma un ciclo se descarta, sino se agrega al árbol
3. Repetir el paso 2 hasta que se hayan agregado  $V-1$  aristas que conforman el árbol de recubrimiento mínimo

# algoritmo de Kruskal



# algoritmo de Kruskal

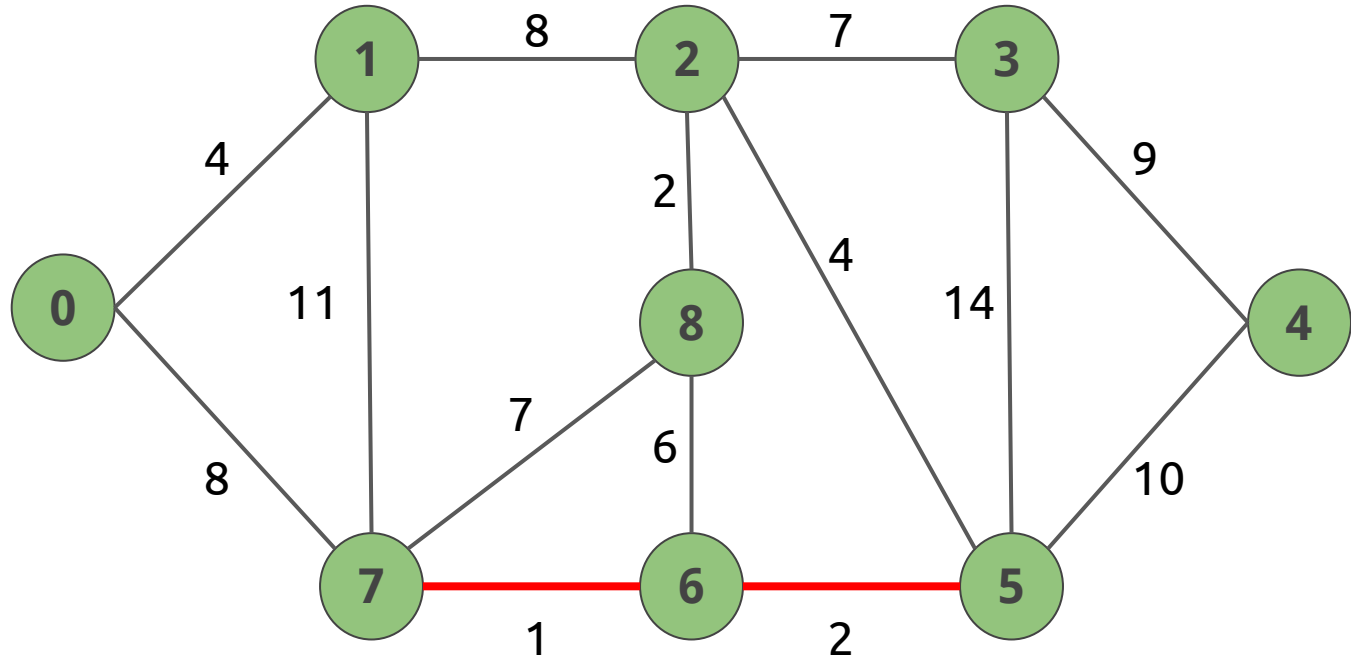
Paso 1: (7,6)



# algoritmo de Kruskal

Paso 1: (7,6)

Paso 2: (6,5)



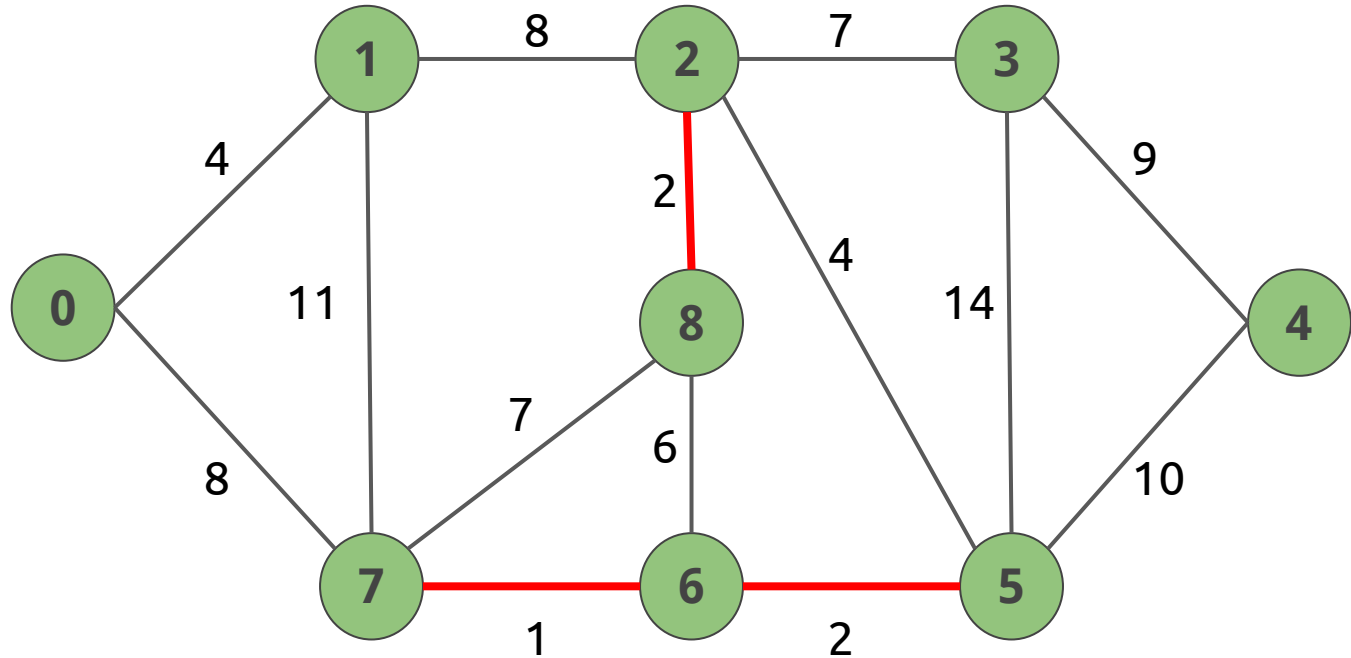


# algoritmo de Kruskal

Paso 1: (7,6)

Paso 2: (6,5)

Paso 3: (2,8)



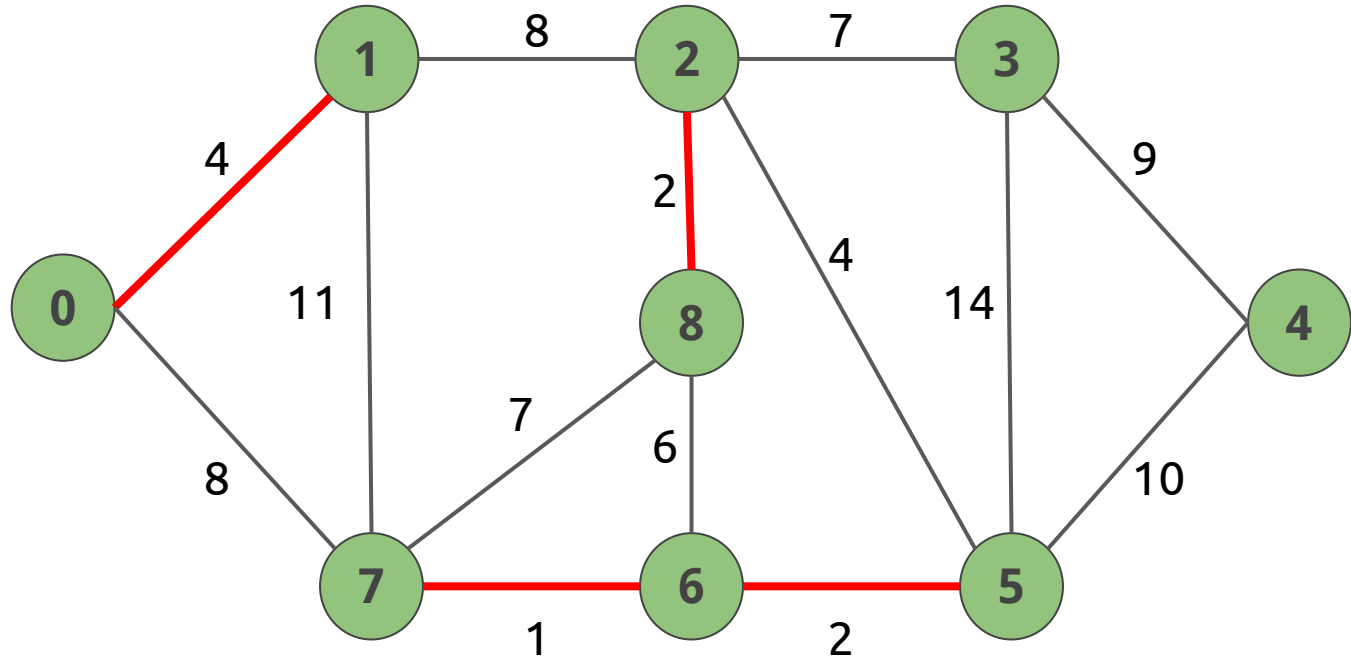
# algoritmo de Kruskal

Paso 1: (7,6)

Paso 2: (6,5)

Paso 3: (2,8)

Paso 4: (0,1)



# algoritmo de Kruskal

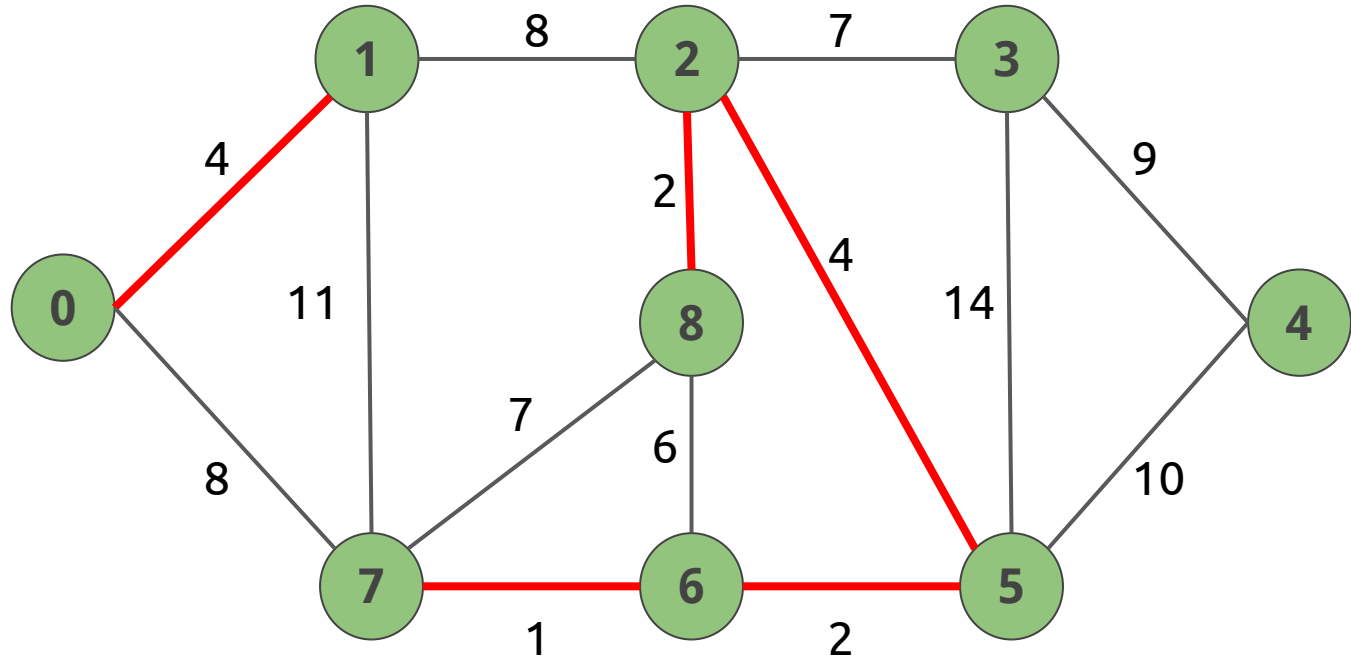
Paso 1: (7,6)

Paso 2: (6,5)

Paso 3: (2,8)

Paso 4: (0,1)

Paso 5: (2,5)



# algoritmo de Kruskal

Paso 1: (7,6)

Paso 2: (6,5)

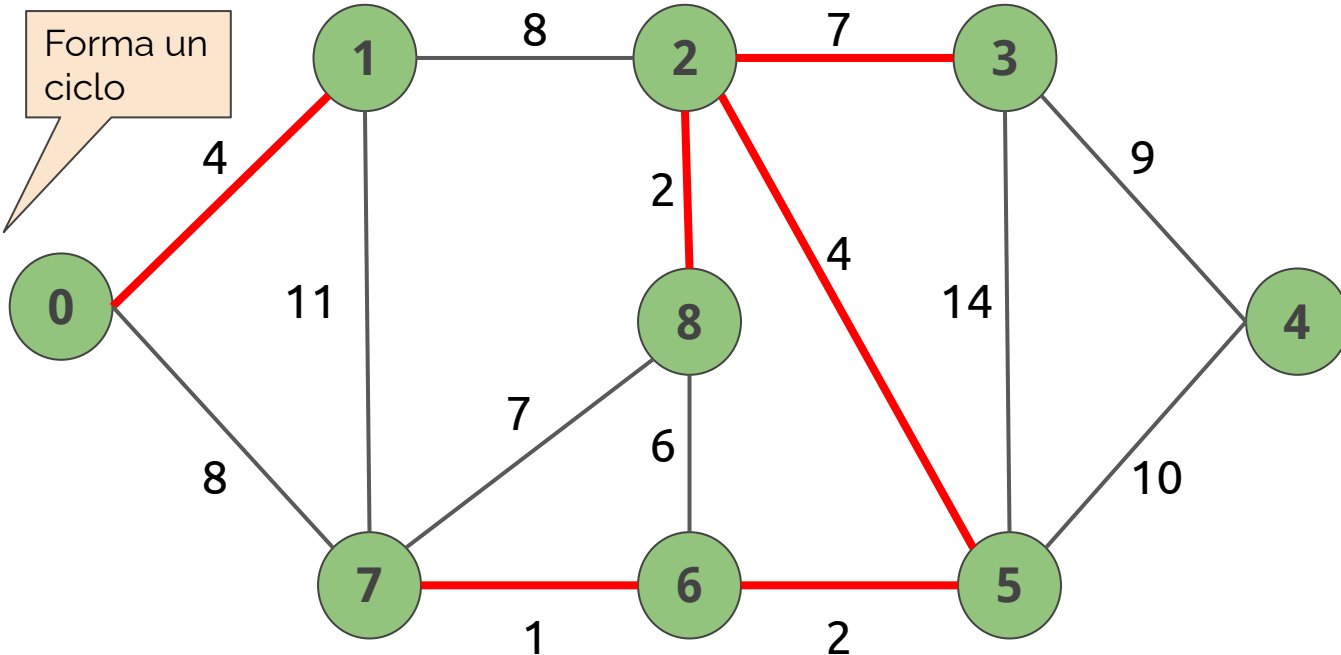
Paso 3: (2,8)

Paso 4: (0,1)

Paso 5: (2,5)

**Paso 6: (6,8) X**

Paso 7: (2,3)



# algoritmo de Kruskal

Paso 1: (7,6)

Paso 2: (6,5)

Paso 3: (2,8)

Paso 4: (0,1)

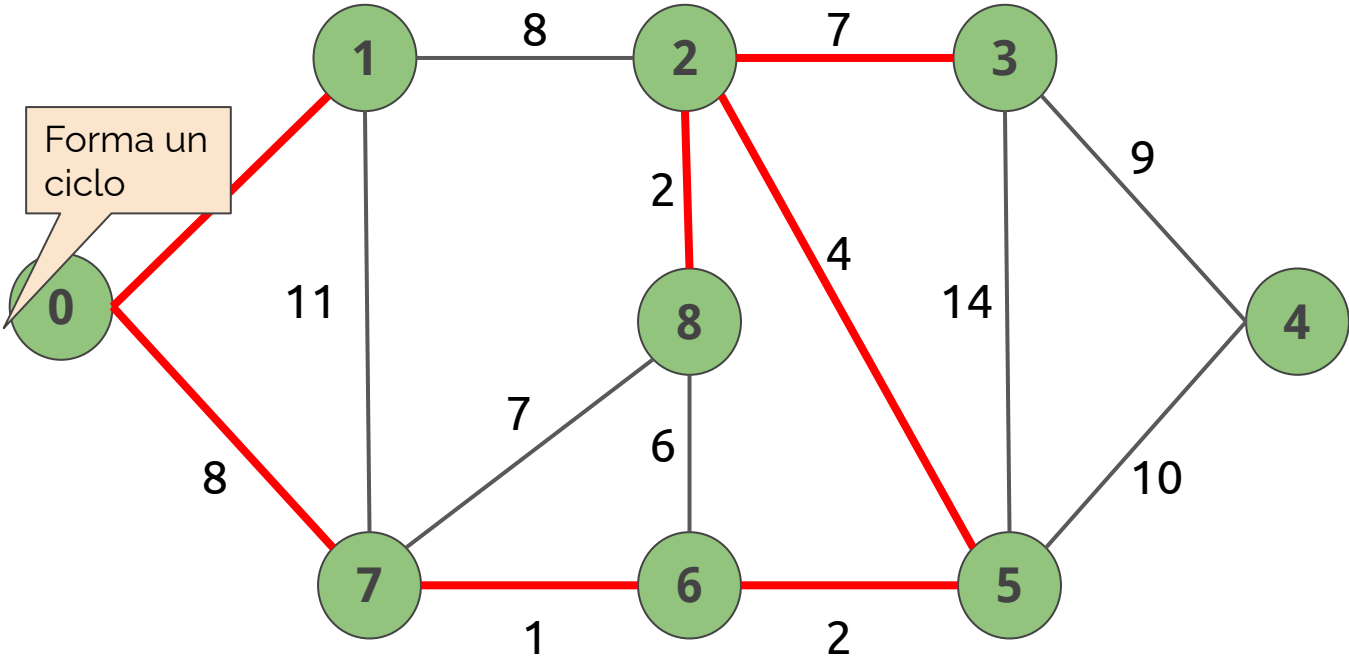
Paso 5: (2,5)

**Paso 6: (6,8) X**

Paso 7: (2,3)

**Paso 8: (7,8) X**

Paso 9: (0,7)



# algoritmo de Kruskal

Paso 1: (7,6)

Paso 2: (6,5)

Paso 3: (2,8)

Paso 4: (0,1)

Paso 5: (2,5)

**Paso 6: (6,8) X**

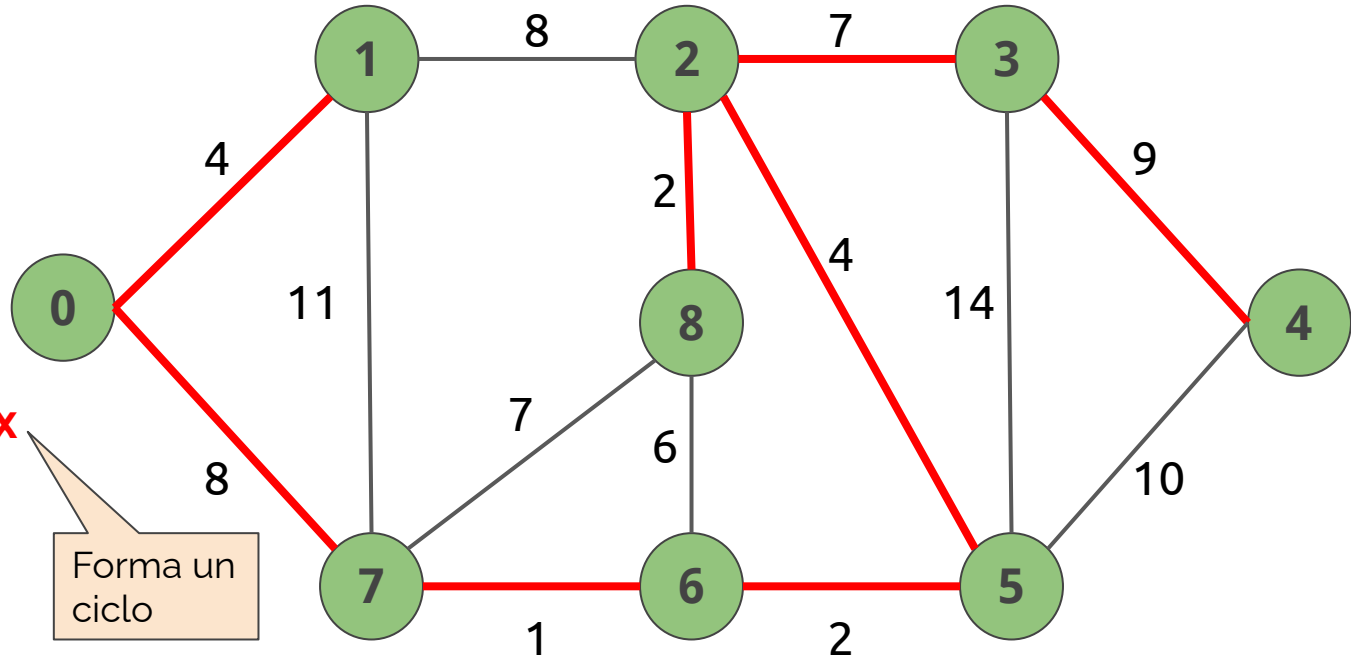
Paso 7: (2,3)

**Paso 8: (7,8) X**

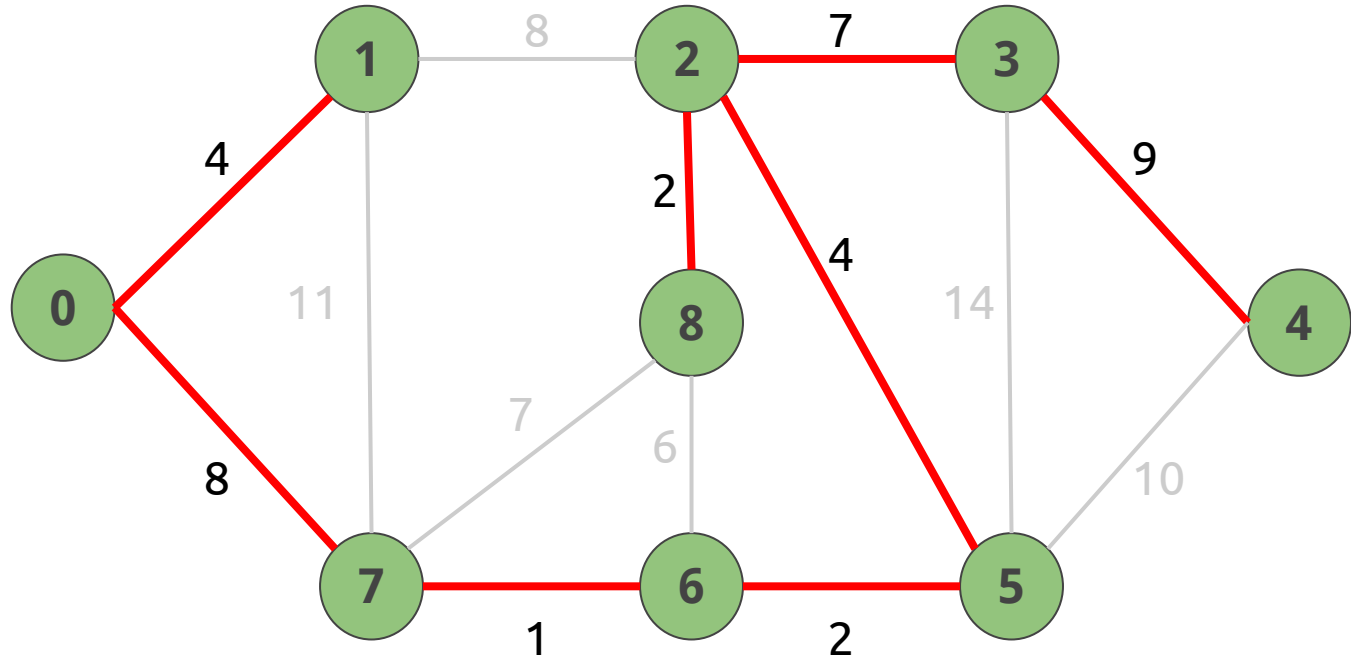
Paso 9: (0,7)

**Paso 10: (1,2) X**

Paso 11: (3,4)



# algoritmo de Kruskal



# algoritmo de Kruskal: eficiencia

La complejidad es de  $O(a \log v)$

Siendo  $v$  el número de vértices y  $a$  el número de aristas del grafo.





## Ejercicio 1.20.

Implementá el algoritmo de Kruscal para hallar un árbol de recubrimiento mínimo en un grafo. Partiendo una representación como lista de aristas.



## Ejercicio 1.20.

```
#include<algorithm>
```

```
struct Arista{  
    int origen;  
    int destino;  
    int peso;  
    bool operator < (const Arista& a1) const {  
        return (peso < a1.peso);  
    }  
};
```

```
sort(aristas, aristas+a);
```



## Ejercicio 1.20.

```
class Grafo {  
    private:  
        int v;  
        int a;  
        int na;  
        Arista* aristas;  
        int Find(int* padre, int i);  
        void Union(int* padre, int x, int y);  
    public:  
        Grafo(int,int);  
        ~Grafo();  
        void AgregarArista(int,int,int);  
        void Mostrar();  
        void Kruskal();  
};
```



## Ejercicio 1.20.

```
int compararArista(const void* a, const void* b)
{
    Arista* pa = (Arista*)a;
    Arista* pb = (Arista*)b;
    return pa->peso > pb->peso;
}
```



## Ejercicio 1.20.

```
void Grafo::Kruskal(){
    Arista resultado[v];
    int r = 0; // indice resultado
    int o = 0; // indice para las aristas ordenadas

    int* padre = new int[a];
    for (int i = 0; i < a; i++){
        padre[i] = -1;
    }
    sort(aristas, aristas+a);
    while (r < v - 1){
        Arista aristaCandidata = aristas[o++];
        int x = Find(padre, aristaCandidata.origen);
        int y = Find(padre, aristaCandidata.destino);
        if (x != y){
            resultado[r++] = aristaCandidata;
            Union(padre, x, y);
        }
    }
}
```



## Ejercicio 1.20.

```
    cout << "Arbol de recubrimiento mínimo" << endl;
    for (int i = 0; i < r; ++i)
        cout << resultado[i].origen << " --- " << resultado[i].destino << " = " << resultado[i].peso <<
endl;
    return;
}
```

# algoritmo de Prim

- Como el algoritmo de Kruskal también es un algoritmo voraz
- Comienza con un árbol de recubrimiento mínimo vacío
- Mantiene dos conjuntos de vértices: uno con aquellos incluidos en el árbol de recubrimiento mínimo y otro con aquellos que todavía no han sido incluidos
- En cada paso se consideran todas las aristas que unen ambos conjuntos y se elige la de menor peso. Una vez que se selecciona la arista se agrega el vértice opuesto al árbol de recubrimiento mínimo

# algoritmo de Prim



Robert C. Prim



Vojtěch Jarník



# algoritmo de Prim

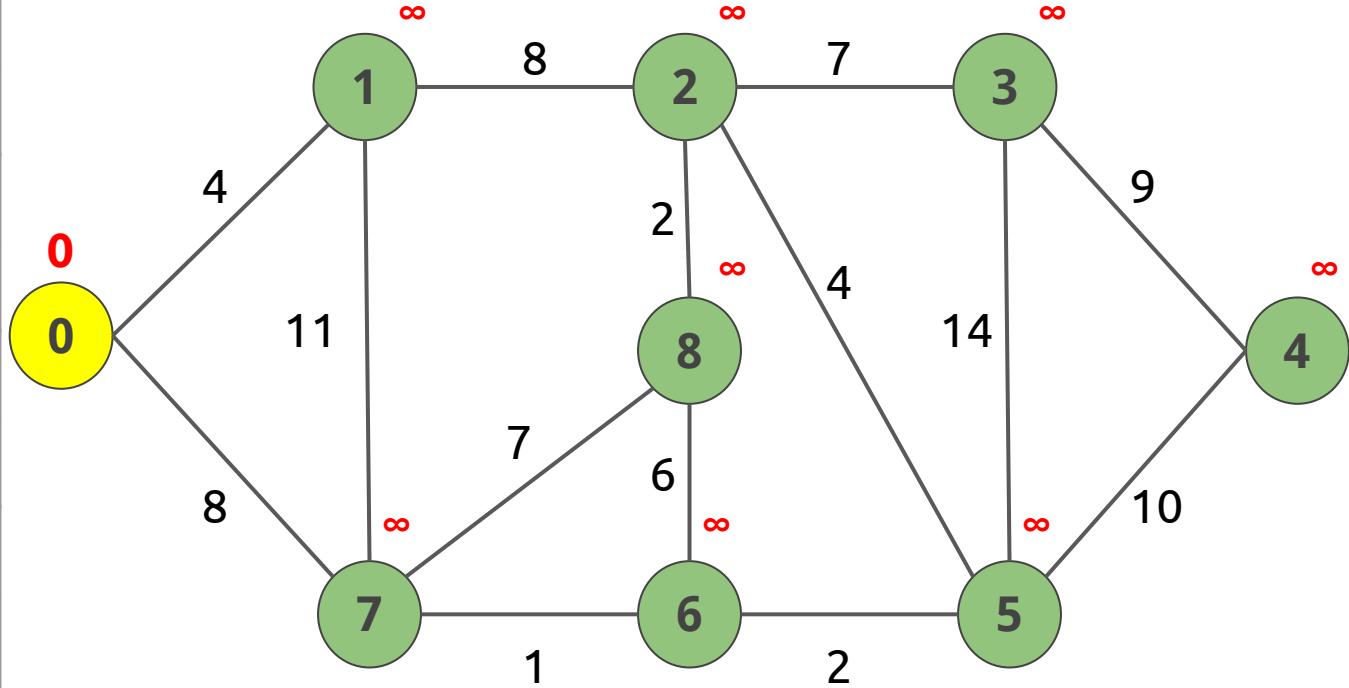
1. Crear un conjunto **mst** que mantiene la información de los vértices que han sido ya agregados al árbol.
2. Inicializar con **INFINITO** todos los vértices del grafo. Inicializar con **0** el vértice inicial así es seleccionado primero
3. Mientras **mst** no contenga todos los vértices:
  - a. Seleccionar el vértice **u** que todavía no está en **mst** y tiene el mínimo valor
  - b. Incluir **u** en **mst**.
  - c. Actualizar el valor de los vértices adyacentes a **u**: para cada vértice **v** adyacente a **u**, si el peso de  $(u,v)$  es menor que el valor asignado actualmente a **v**, actualizarlo con el peso de  $(u,v)$

# algoritmo de Prim

padre

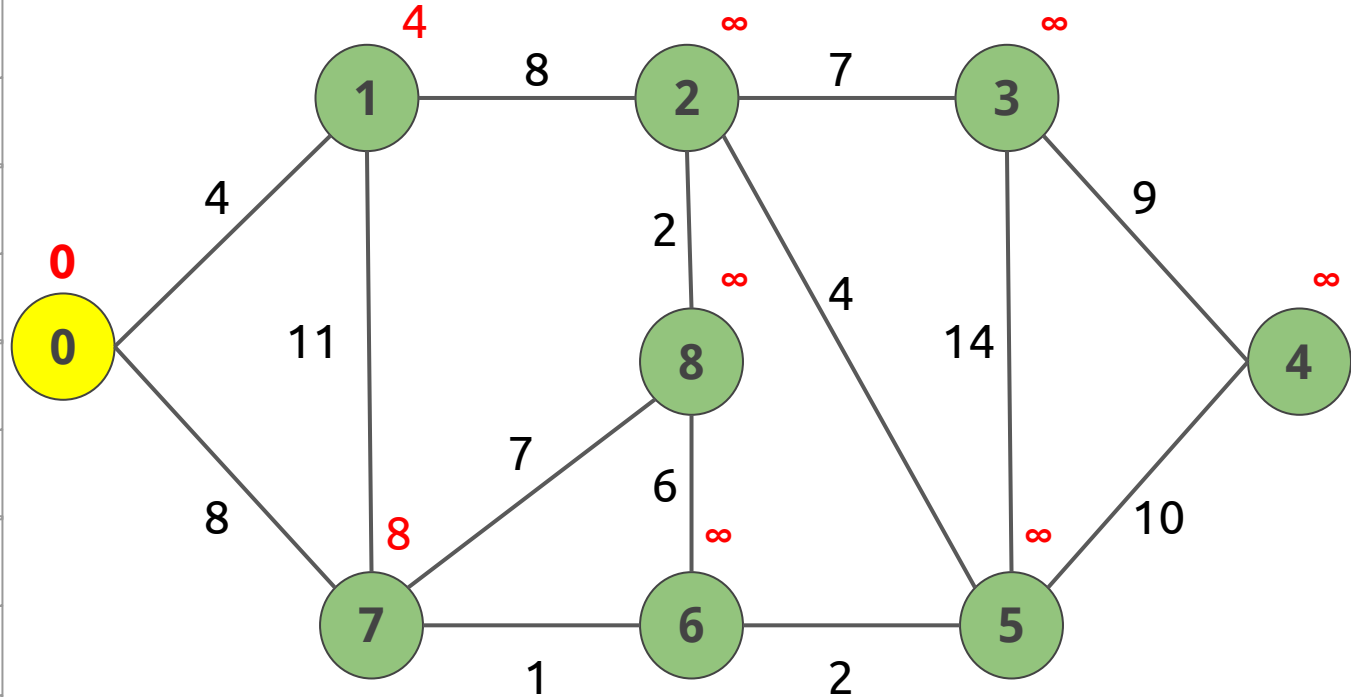
valor

mst	p	v
0	1	-
1	0	-
2	0	-
3	0	-
4	0	-
5	0	-
6	0	-
7	0	-
8	0	-



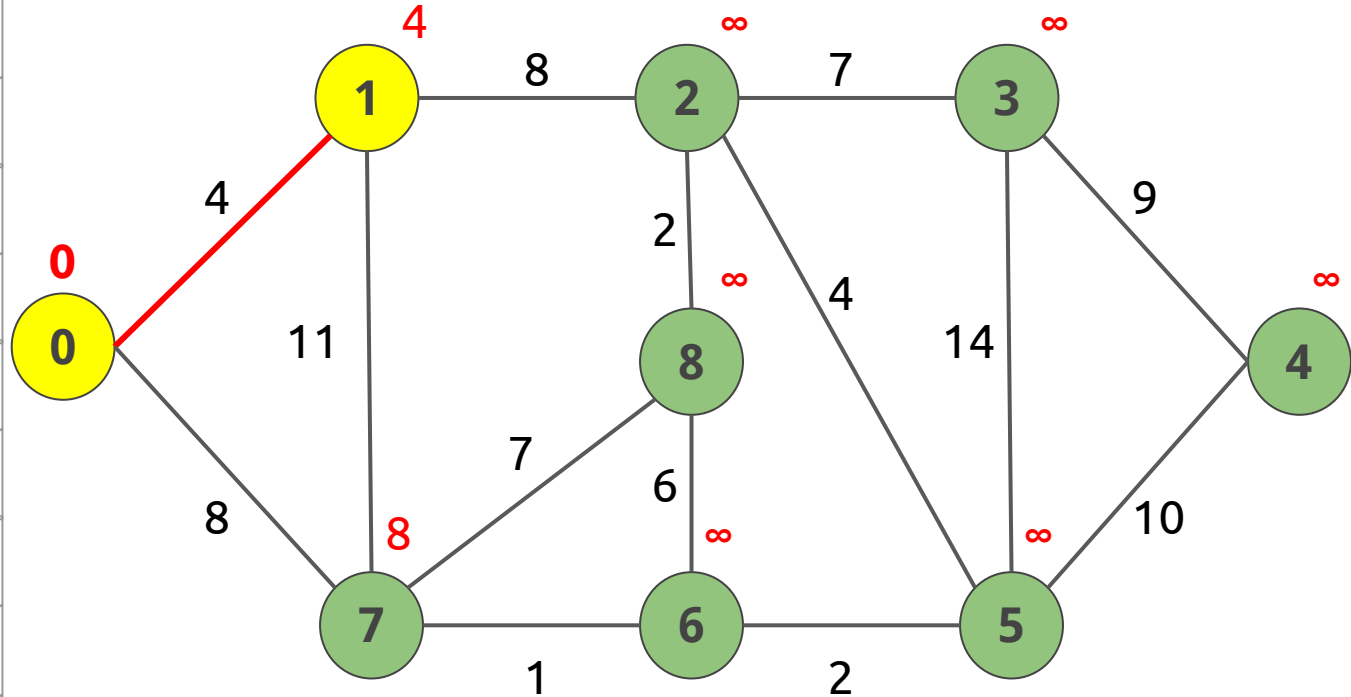
# algoritmo de Prim

	mst	p	v
0	1	-	0
1	0	0	4
2	0	-	$\infty$
3	0	-	$\infty$
4	0	-	$\infty$
5	0	-	$\infty$
6	0	-	$\infty$
7	0	0	8
8	0	-	$\infty$



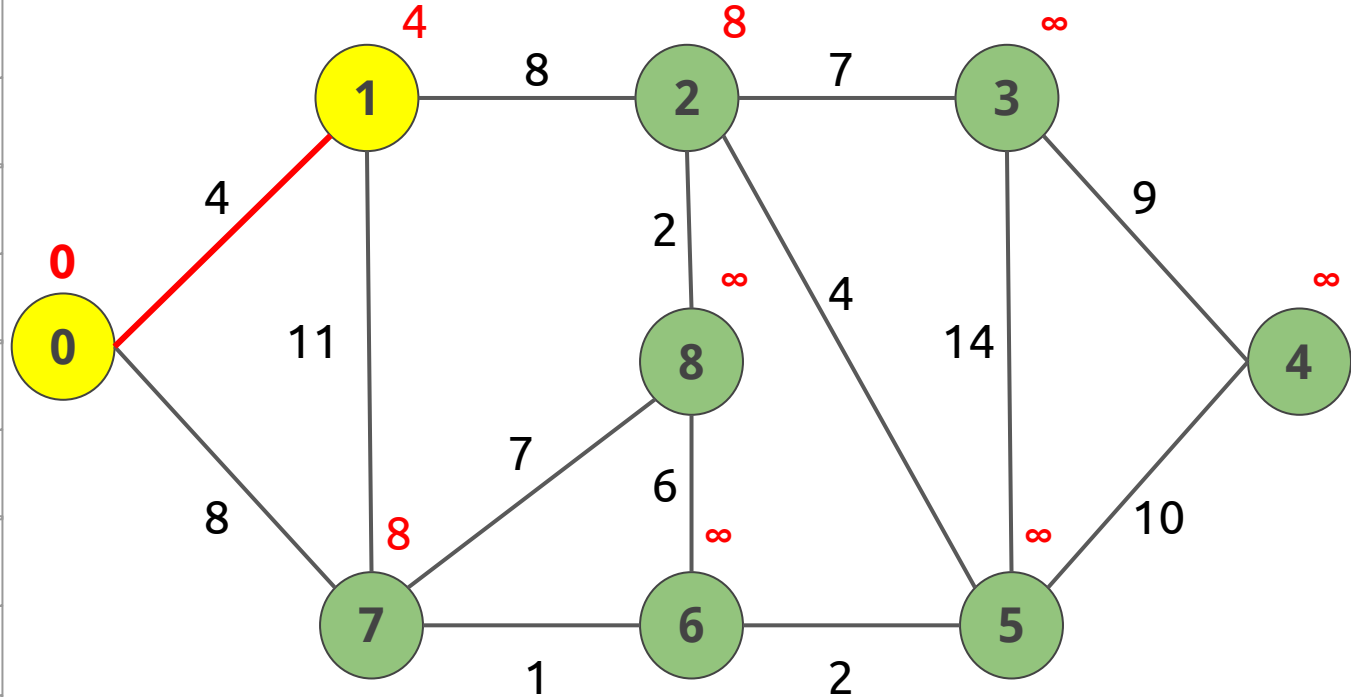
# algoritmo de Prim

	mst	p	v
0	1	-	0
1	1	0	4
2	0	-	$\infty$
3	0	-	$\infty$
4	0	-	$\infty$
5	0	-	$\infty$
6	0	-	$\infty$
7	0	0	8
8	0	-	$\infty$



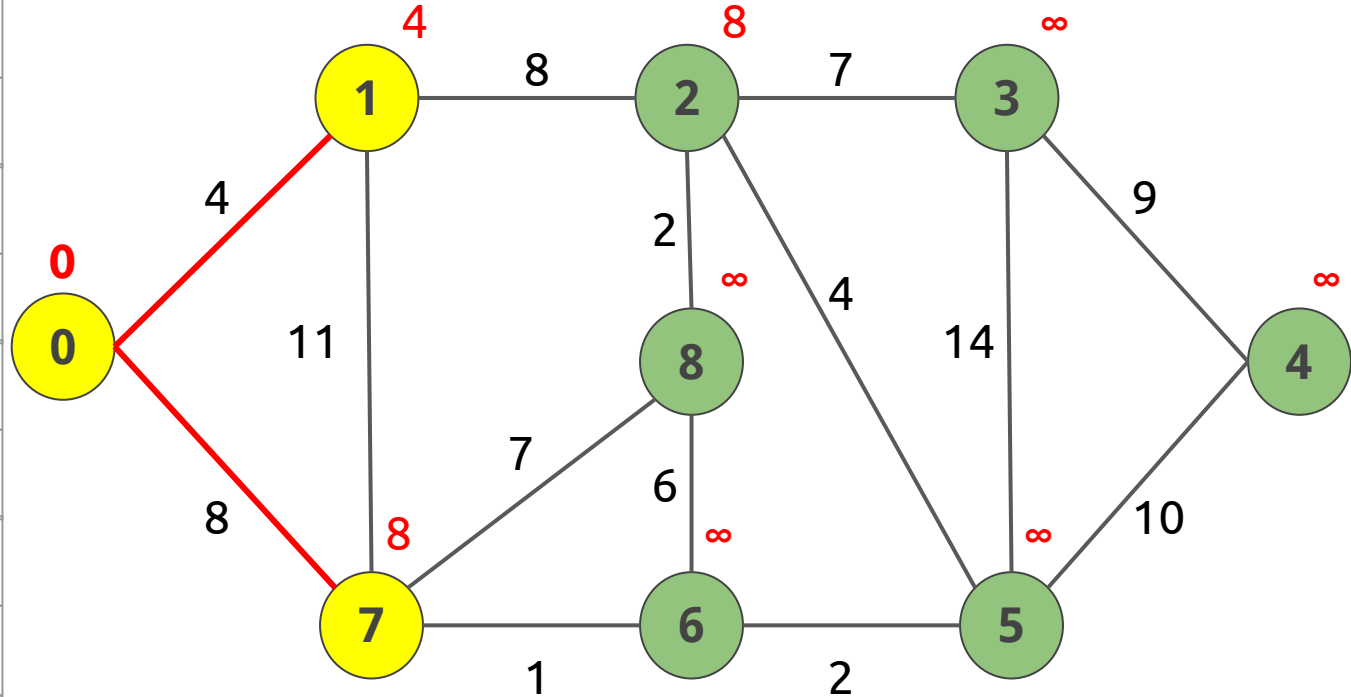
# algoritmo de Prim

	mst	p	v
0	1	-	0
1	1	0	4
2	0	1	8
3	0	-	$\infty$
4	0	-	$\infty$
5	0	-	$\infty$
6	0	-	$\infty$
7	0	0	8
8	0	-	$\infty$



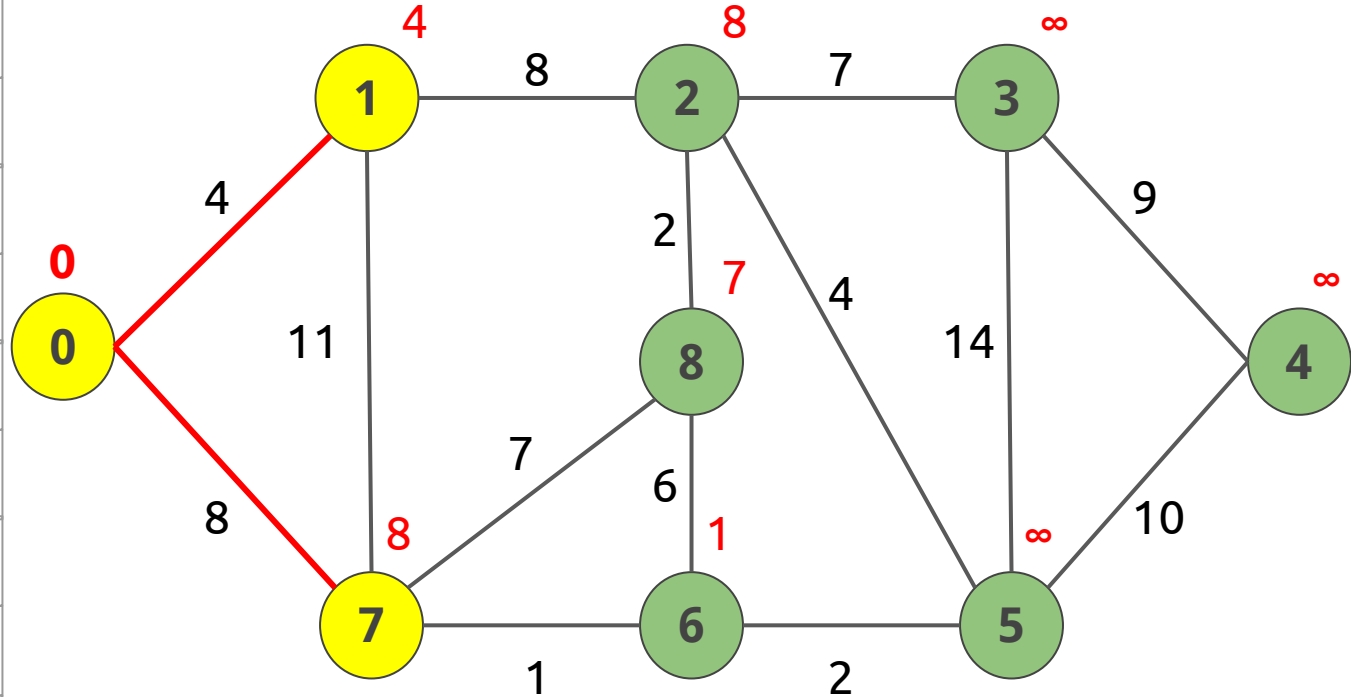
# algoritmo de Prim

	mst	p	v
0	1	-	0
1	1	0	4
2	0	1	8
3	0	-	$\infty$
4	0	-	$\infty$
5	0	-	$\infty$
6	0	-	$\infty$
7	1	0	8
8	0	-	$\infty$



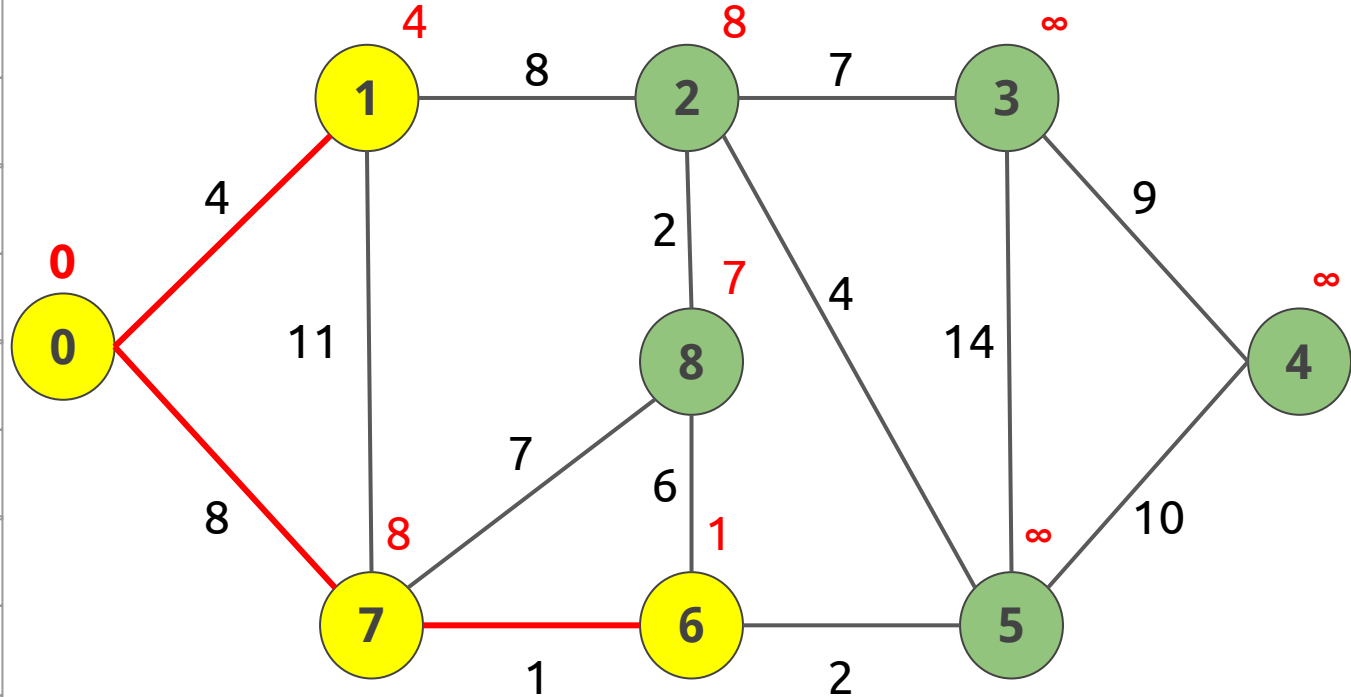
# algoritmo de Prim

	mst	p	v
0	1	-	0
1	1	0	4
2	0	1	8
3	0	-	$\infty$
4	0	-	$\infty$
5	0	-	$\infty$
6	0	7	1
7	1	0	8
8	0	7	7



# algoritmo de Prim

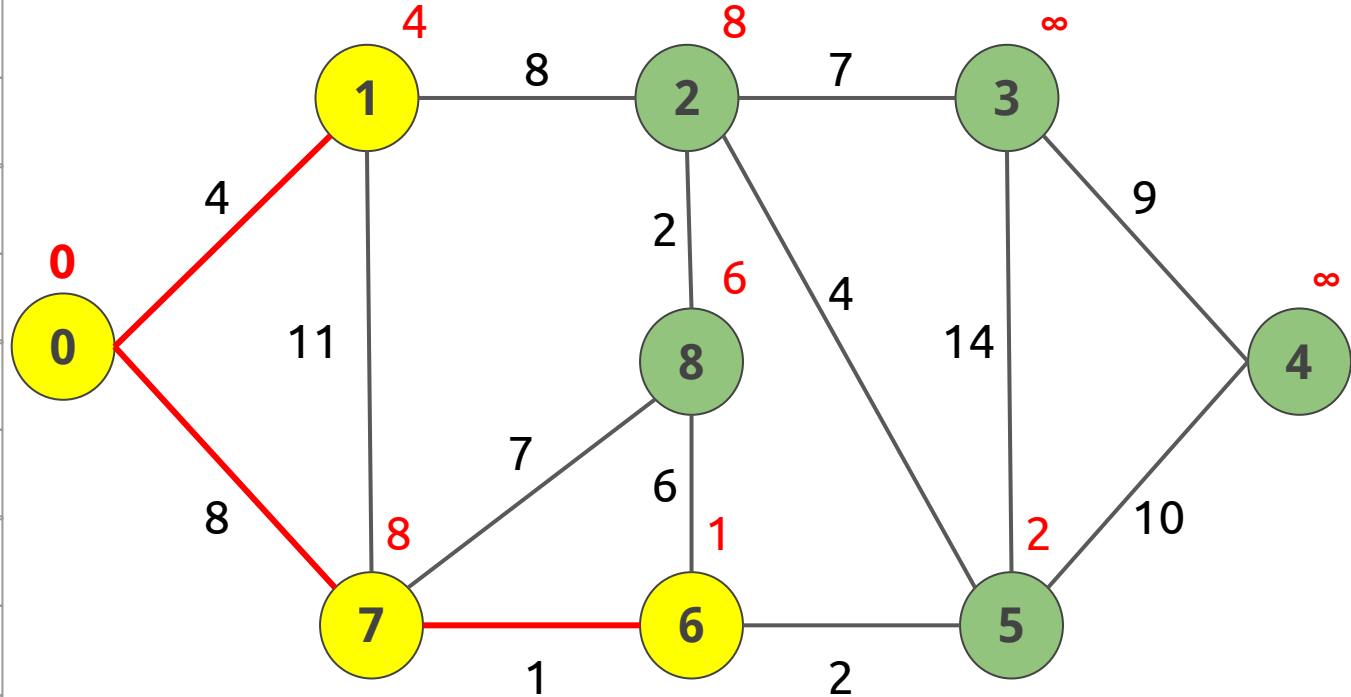
	mst	p	v
0	1	-	0
1	1	0	4
2	0	1	8
3	0	-	$\infty$
4	0	-	$\infty$
5	0	-	$\infty$
6	1	7	1
7	1	0	8
8	0	7	7





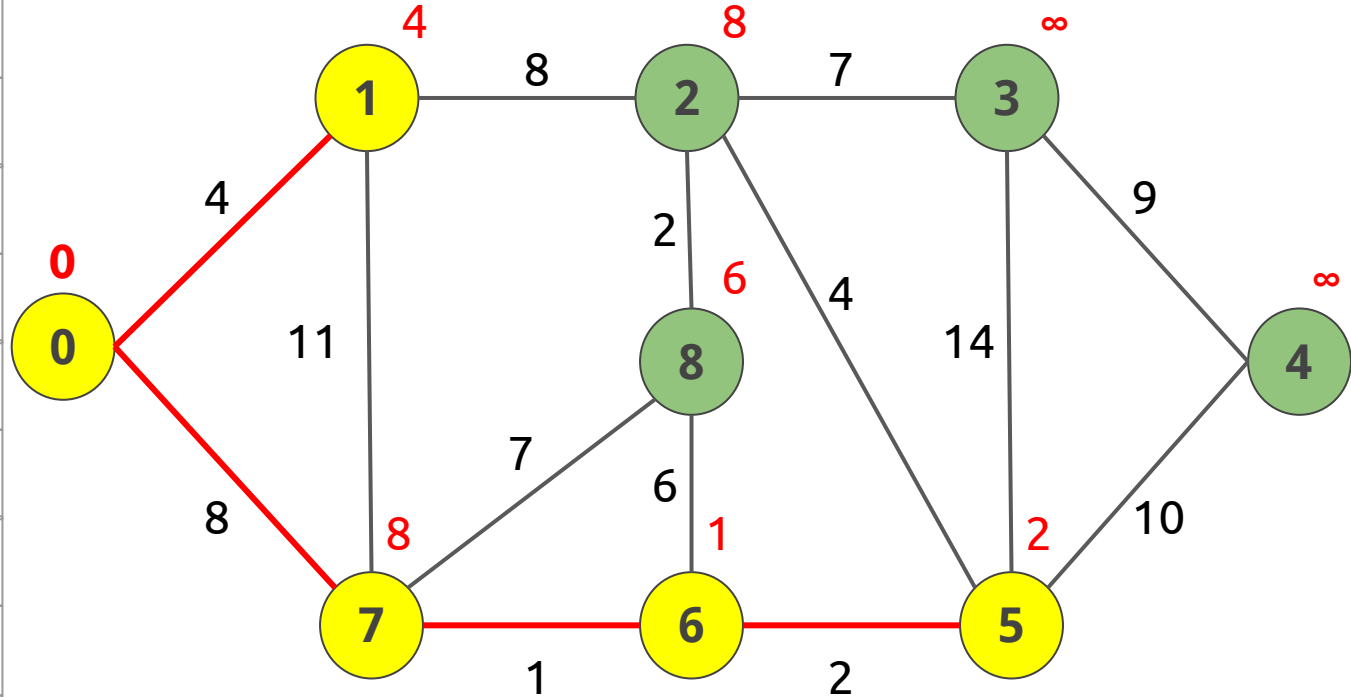
# algoritmo de Prim

	mst	p	v
0	1	-	0
1	1	0	4
2	0	1	8
3	0	-	$\infty$
4	0	-	$\infty$
5	0	6	2
6	1	7	1
7	1	0	8
8	0	6	6



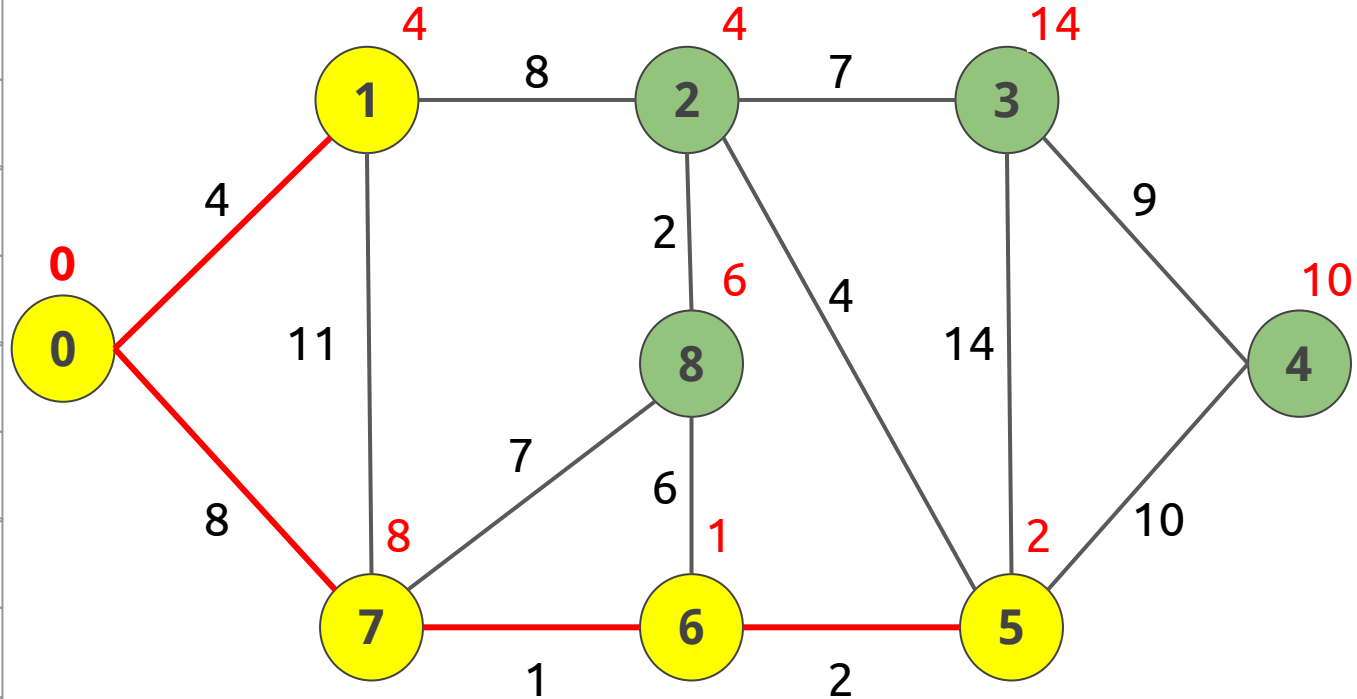
# algoritmo de Prim

	mst	p	v
0	1	-	0
1	1	0	4
2	0	1	8
3	0	-	$\infty$
4	0	-	$\infty$
5	1	6	2
6	1	7	1
7	1	0	8
8	0	6	6



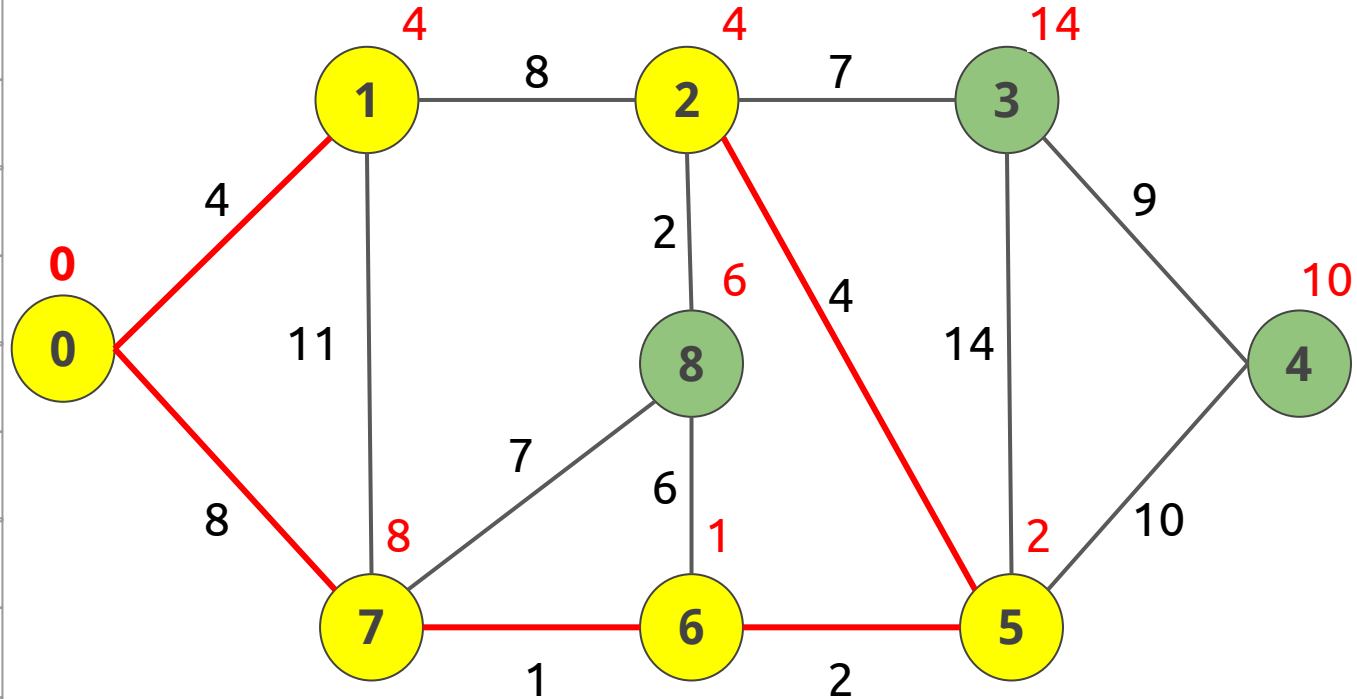
# algoritmo de Prim

	mst	p	v
0	1	-	0
1	1	0	4
2	0	5	4
3	0	5	14
4	0	5	10
5	1	6	2
6	1	7	1
7	1	0	8
8	0	6	6



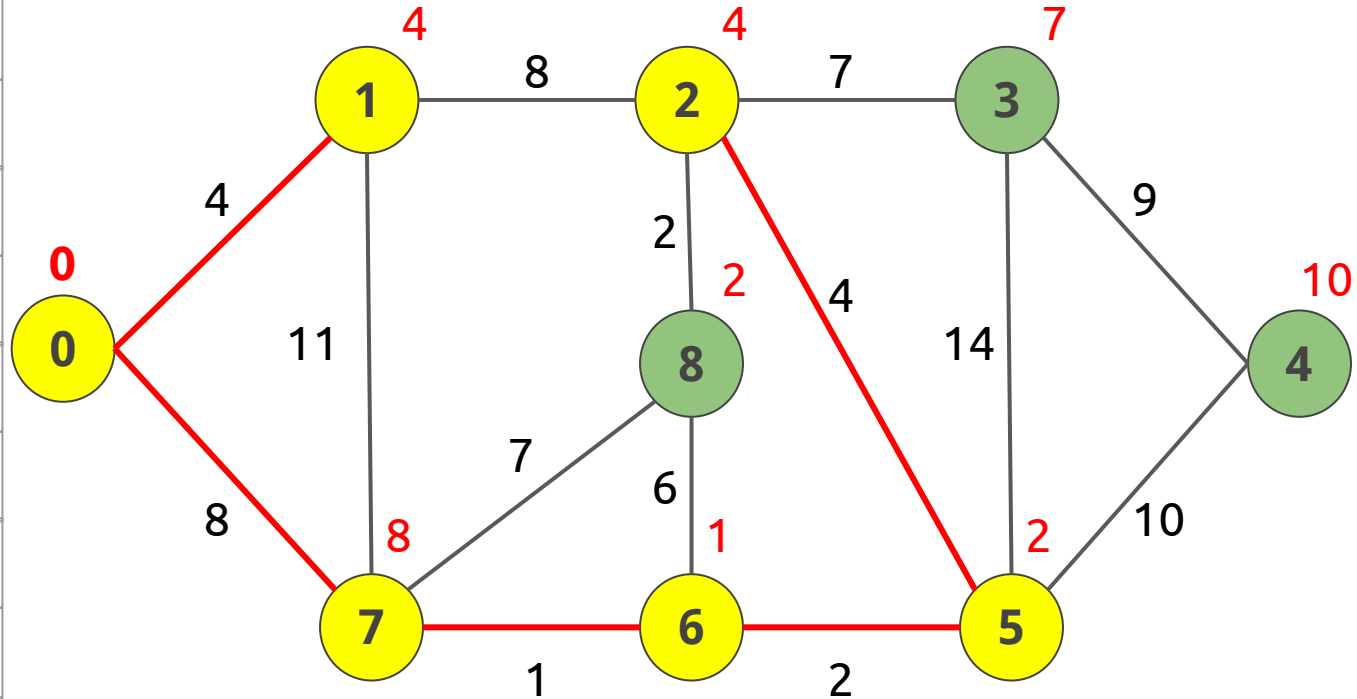
# algoritmo de Prim

	mst	p	v
0	1	-	0
1	1	0	4
2	1	5	4
3	0	5	14
4	0	5	10
5	1	6	2
6	1	7	1
7	1	0	8
8	0	6	6



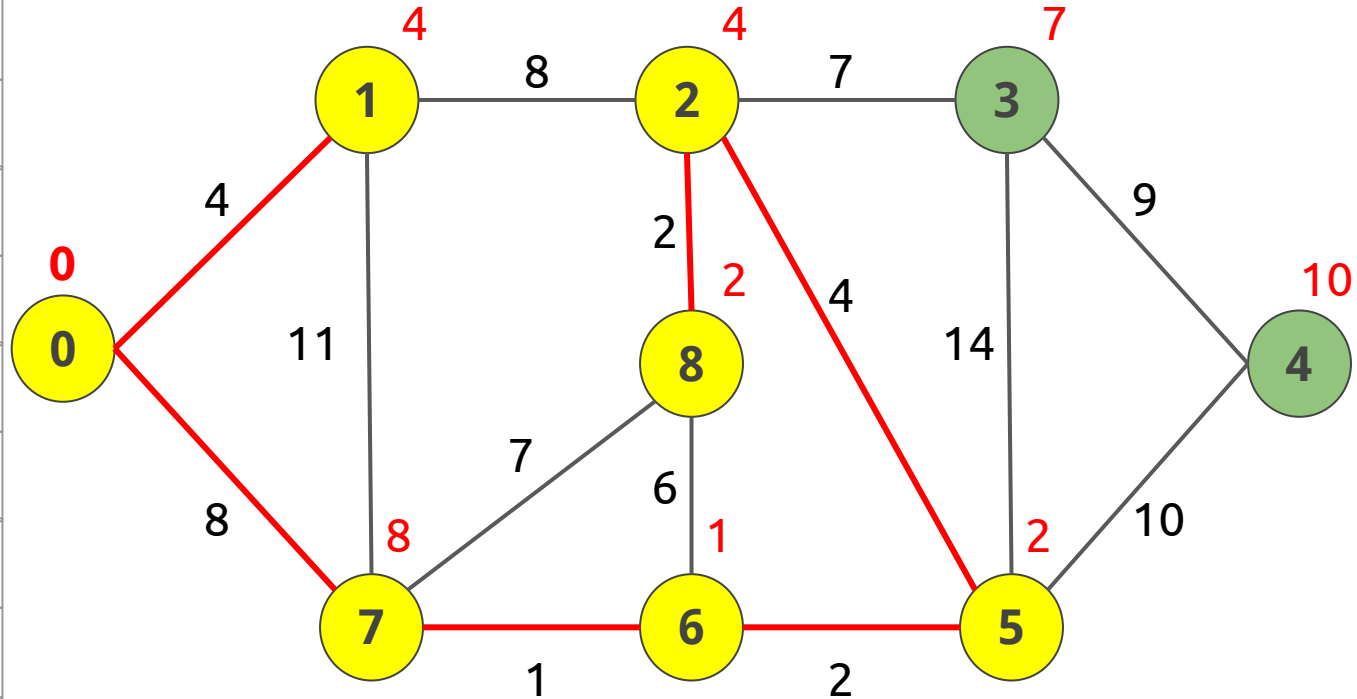
# algoritmo de Prim

	mst	p	v
0	1	-	0
1	1	0	4
2	1	5	4
3	0	2	7
4	0	5	10
5	1	6	2
6	1	7	1
7	1	0	8
8	0	2	2



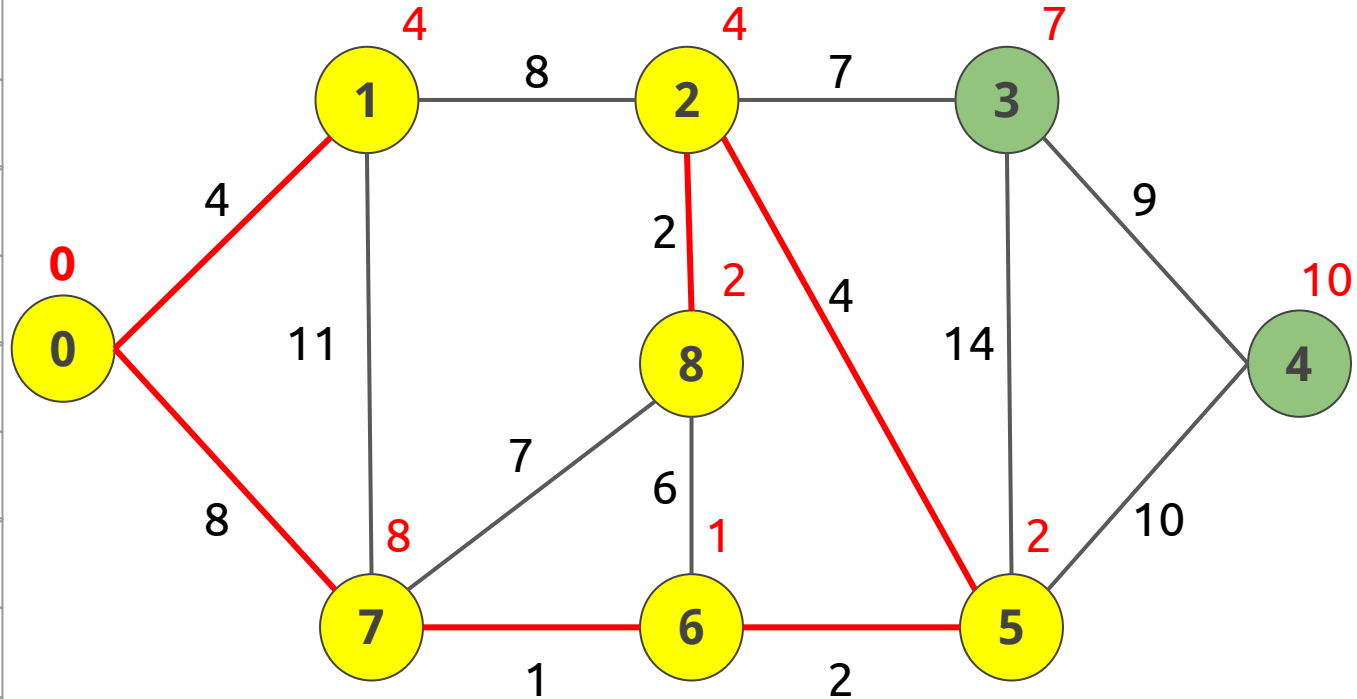
# algoritmo de Prim

	mst	p	v
0	1	-	0
1	1	0	4
2	1	5	4
3	0	2	7
4	0	5	10
5	1	6	2
6	1	7	1
7	1	0	8
8	1	2	2



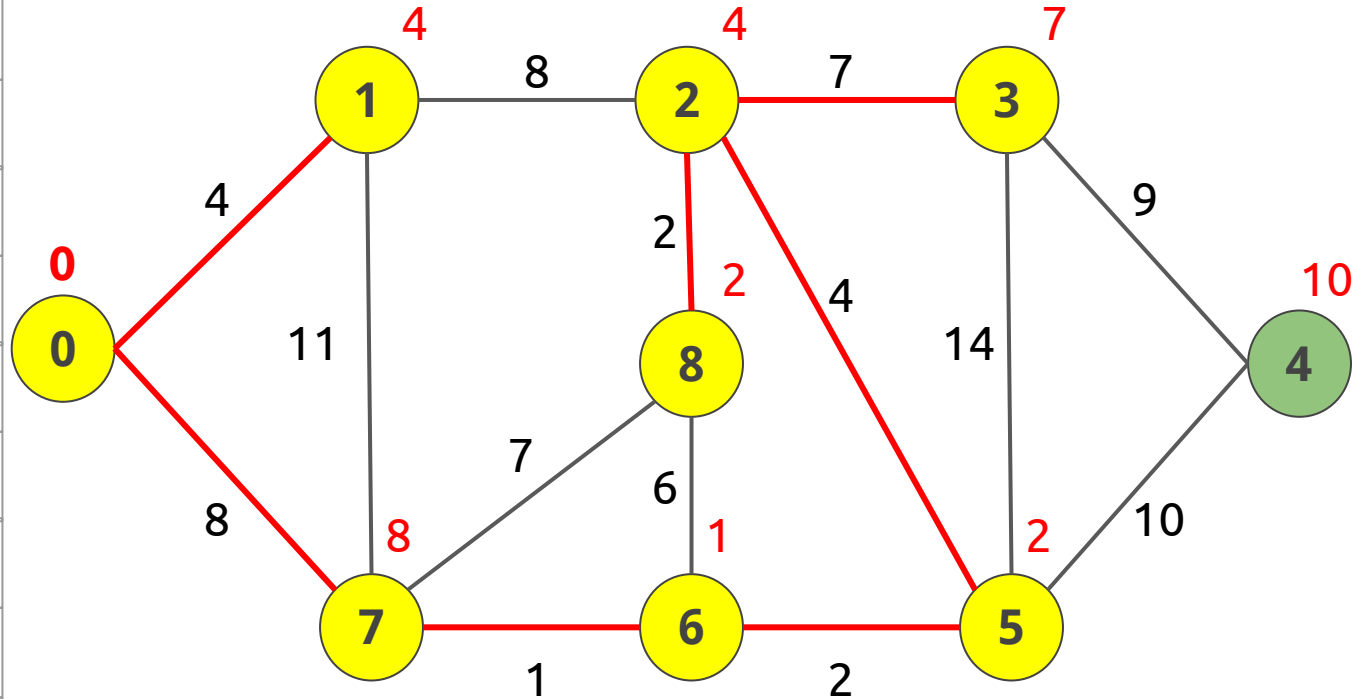
# algoritmo de Prim

	mst	p	v
0	1	-	0
1	1	0	4
2	1	5	4
3	0	2	7
4	0	5	10
5	1	6	2
6	1	7	1
7	1	0	8
8	1	2	2



# algoritmo de Prim

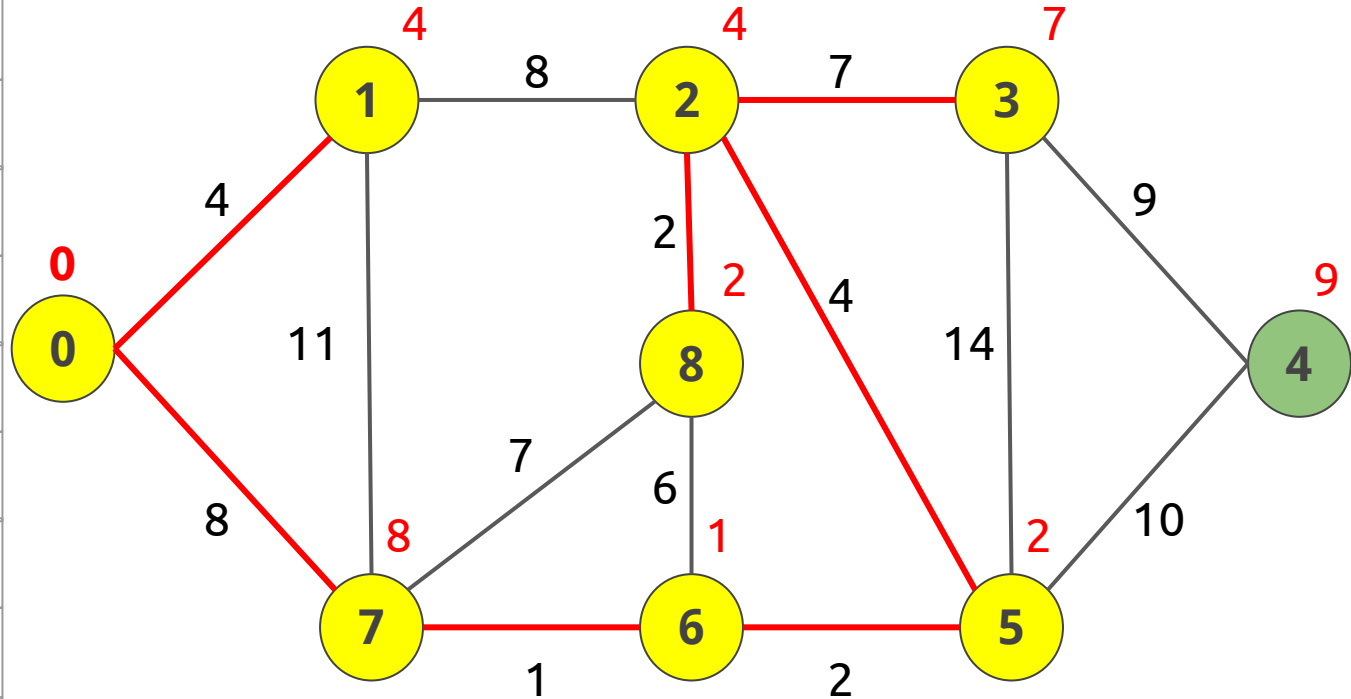
	mst	p	v
0	1	-	0
1	1	0	4
2	1	5	4
3	1	2	7
4	0	5	10
5	1	6	2
6	1	7	1
7	1	0	8
8	1	2	2





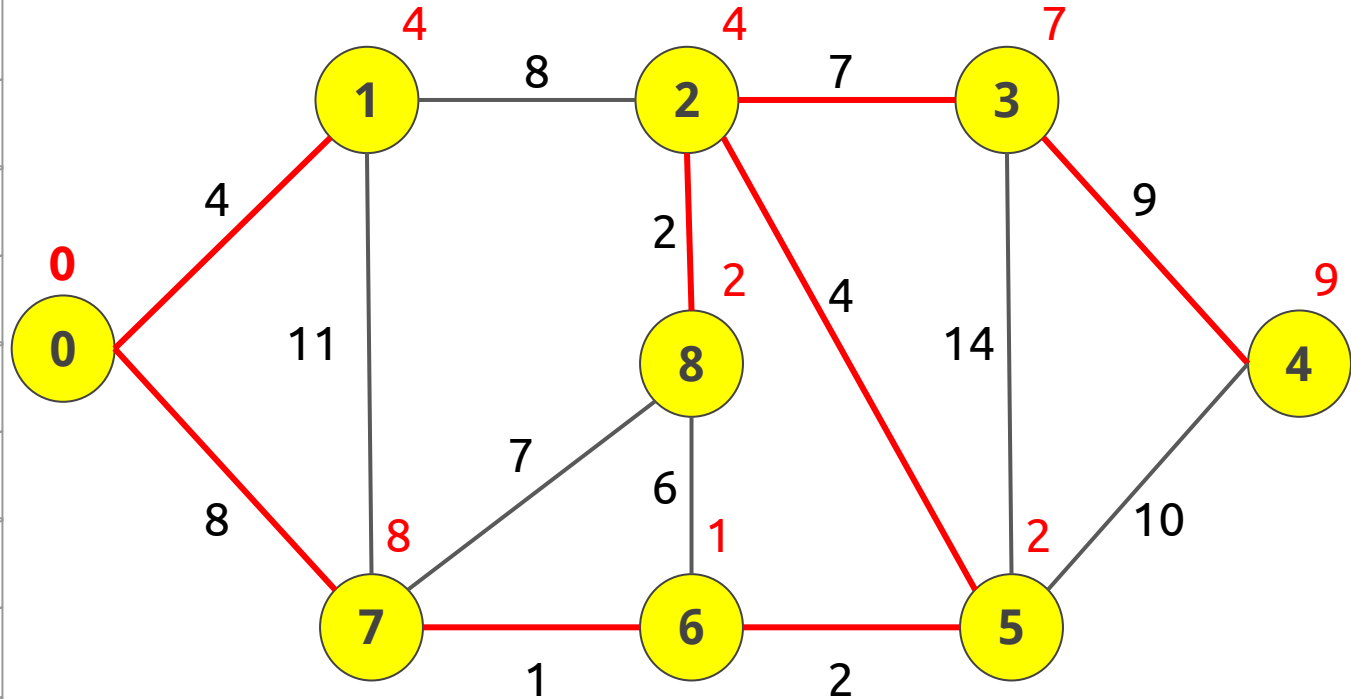
# algoritmo de Prim

	mst	p	v
0	1	-	0
1	1	0	4
2	1	5	4
3	1	2	7
4	0	3	9
5	1	6	2
6	1	7	1
7	1	0	8
8	1	2	2



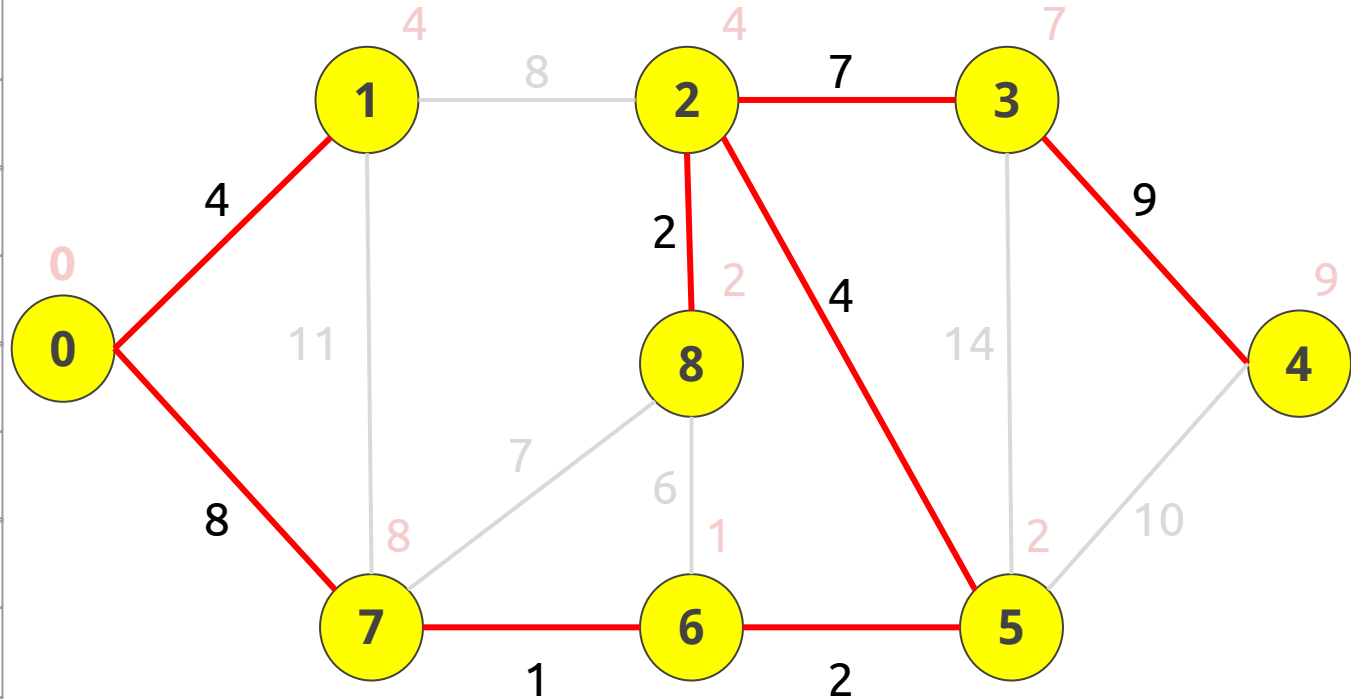
# algoritmo de Prim

	mst	p	v
0	1	-	0
1	1	0	4
2	1	5	4
3	1	2	7
4	1	3	9
5	1	6	2
6	1	7	1
7	1	0	8
8	1	2	2



# algoritmo de Prim

	mst	p	v
0	1	-	0
1	1	0	4
2	1	5	4
3	1	2	7
4	1	3	9
5	1	6	2
6	1	7	1
7	1	0	8
8	1	2	2



# algoritmo de Prim: eficiencia

La complejidad es de  $O(v^2)$

Siendo  $v$  el número de vértices del grafo.

Como  $n-1 \leq a \leq \frac{n(n-1)}{2}$  se cumple, entonces:

- Si  $a \approx n$  conviene utilizar Kruskal
- Si  $a \approx n^2$  conviene utilizar Prim



## Ejercicio 1.21.

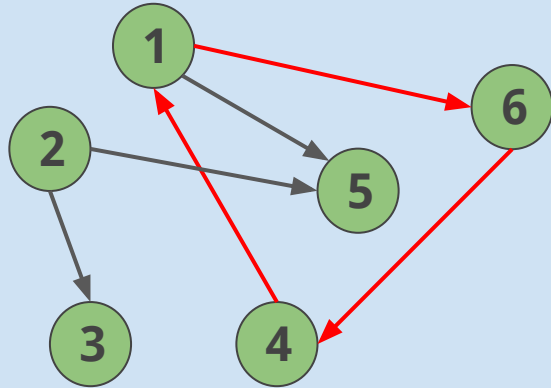
Implementá el algoritmo de Prim para hallar un árbol de recubrimiento mínimo en un grafo. Partiendo una representación mediante matriz de adyacencia.

# Ejercicios adicionales



## Ejercicio 1.24.

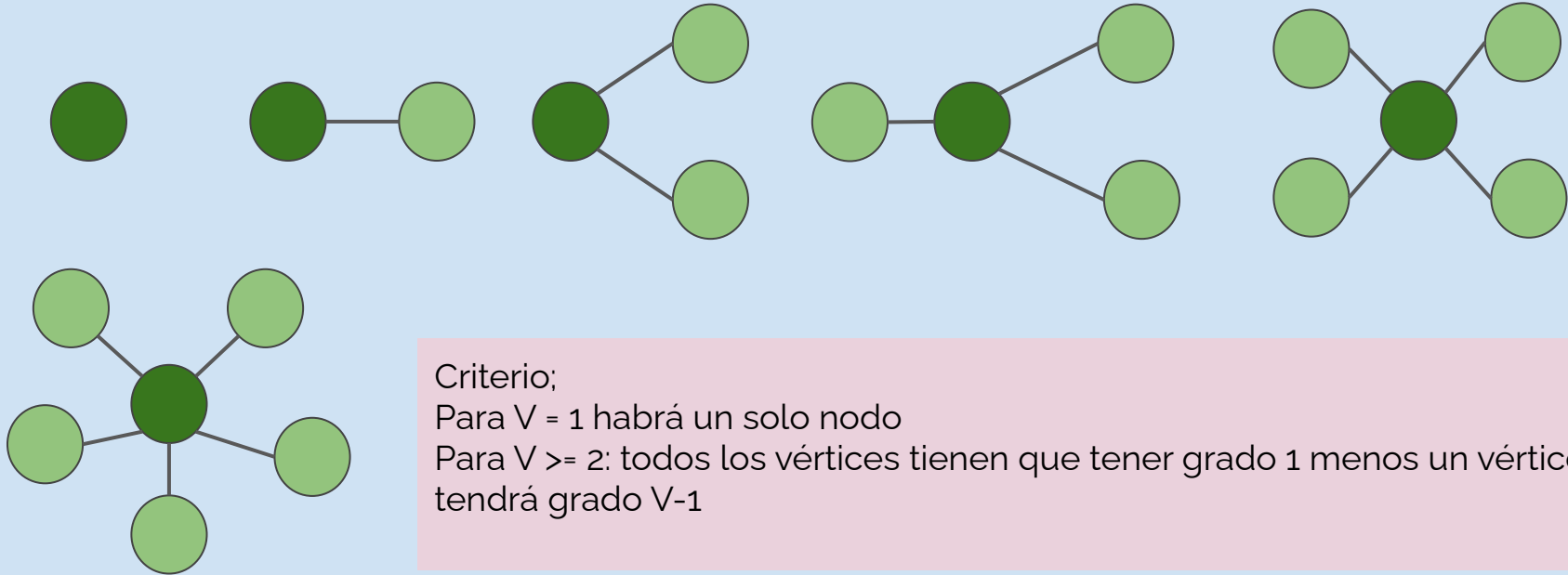
Escribí una función que determine si en un grafo dirigido hay un ciclo. Esto puede ser determinado mediante una variante de la búsqueda en profundidad (DFS).





## Ejercicio 1.25.

Escribí una función que determine si un grafo es un grafo estrella



Criterio;

Para  $V = 1$  habrá un solo nodo

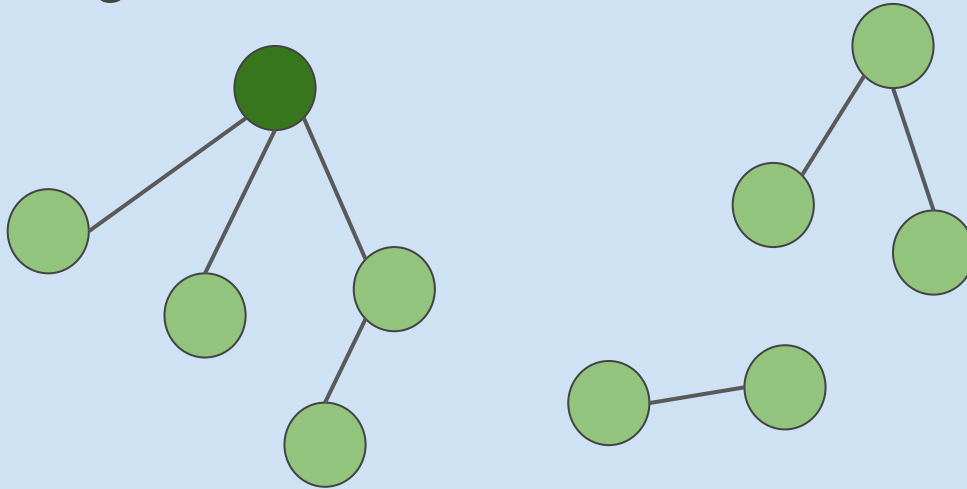
Para  $V \geq 2$ : todos los vértices tienen que tener grado 1 menos un vértice que tendrá grado  $V-1$





## Ejercicio 1.28.

Escribí una función que cuente a cuántas ciudades no se puede llegar desde una ciudad principal usando rutas asfaltadas. El mapa de rutas está representado por un grafo no dirigido

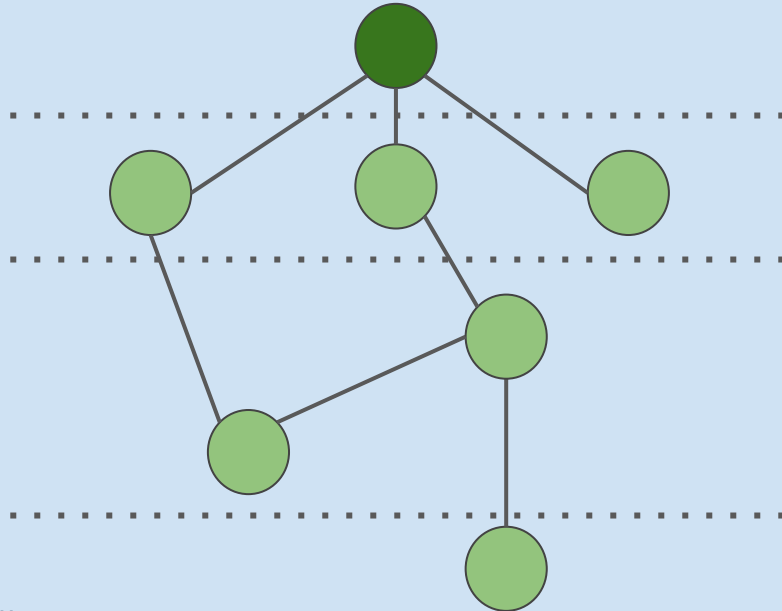


Criterio;  
Usar búsqueda en profundidad - DFS



## Ejercicio 1.29.

Escribí una función que imprima el nivel en que se encuentra cada nodo desde un nodo determinado



Criterio;  
Usar búsqueda en anchura - BFS



## Ejercicio 1.30.

Utilizá el algoritmo de Dijkstra para hallar manualmente el camino más corto desde el vértice 5 a los demás vértices

