


Algoritmos de Kruskal y Prim

Otro conceptos a tener en cuenta:

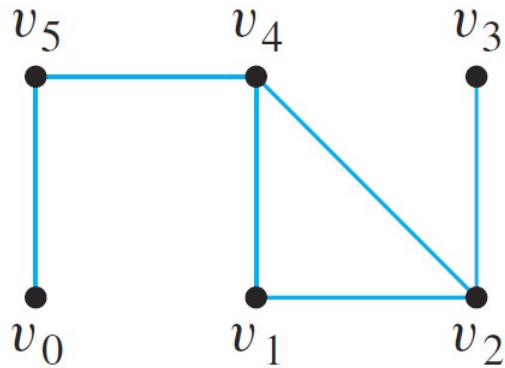
- MST: Arbol Expandido Minimo (Minimum Spanning Tree)
 - Algoritmo Greedy
 - Metodos MakeSet, Find, Union
- 

Arbol Expandido

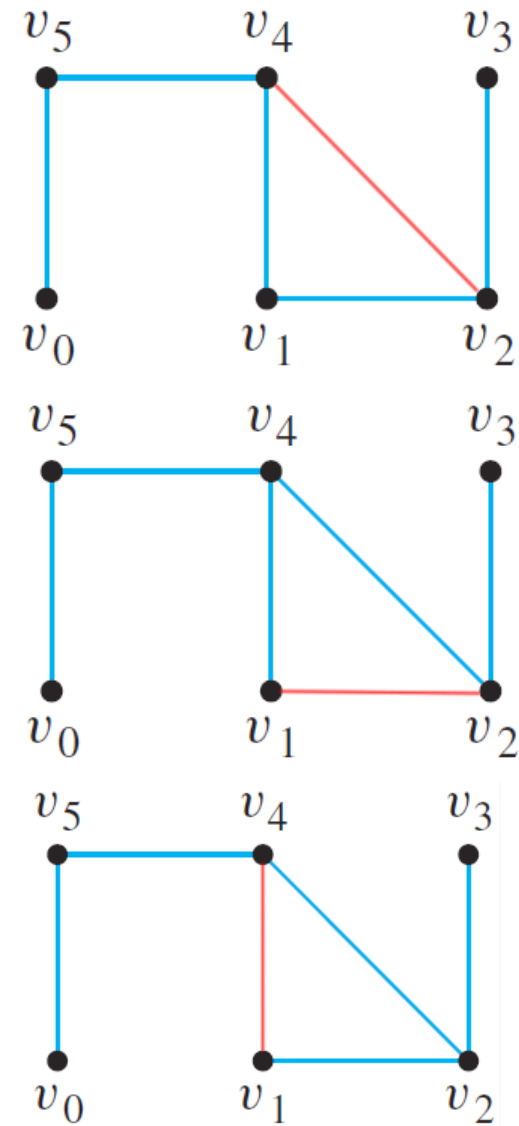
- Un árbol expandido es un subgrafo que contiene TODOS LOS VÉRTICES de un grafo y no forma ningún ciclo
- Observaciones:
 1. Cada grado CONEXO tiene un árbol expandido
 2. Cualesquiera dos arboles expandidos de un mismo grafo tienen el mismo numero de aristas
 3. Teniendo un grafo conexo con un ciclo, se puede quitar una arista para eliminar el ciclo, y este grafo seguirá siendo conexo

Árbol Expandido (Ejemplo)

Veamos el siguiente grafo:



Observemos que se pueden formar diferentes árboles expandidos



Árbol Expandido Mínimo

- Aplican para los grafos ponderados
- Tiene el menor peso total posible, comparando con los demás árboles expandidos posibles

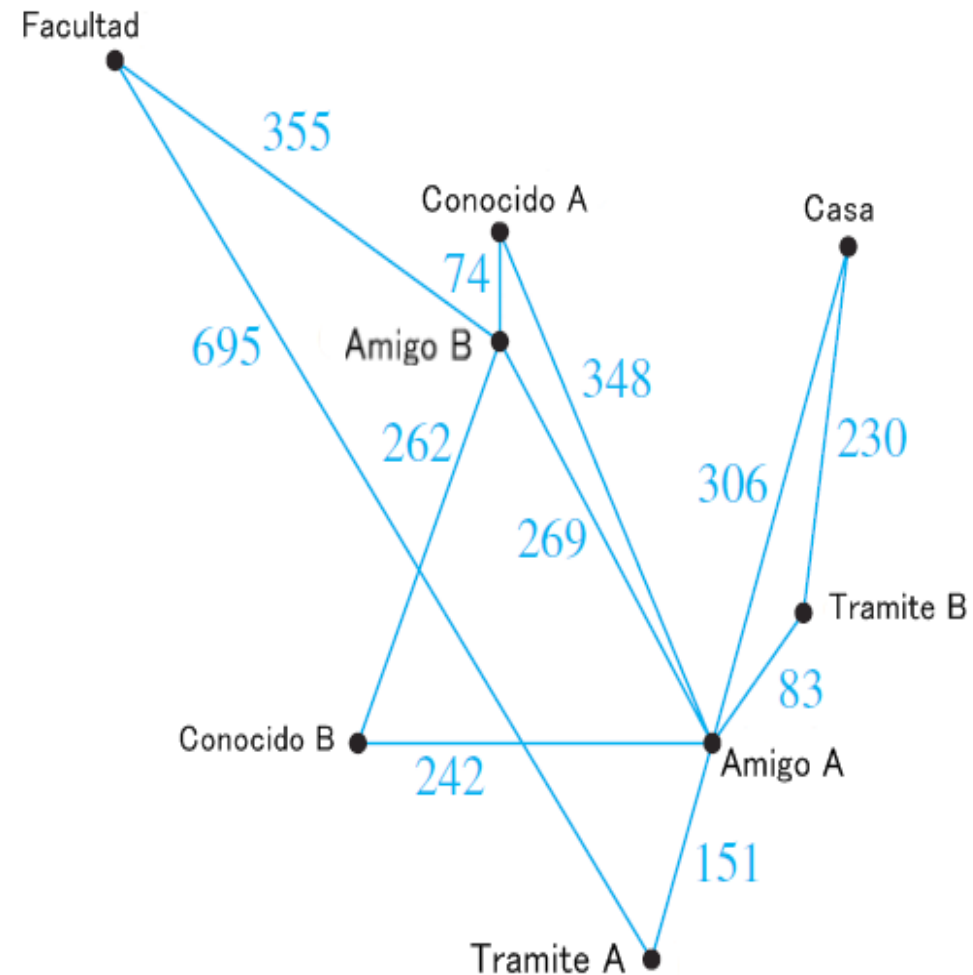
¿Para qué podría servir en la vida diaria?

Cada quien, en lo posible, desea realizar sus tareas de la forma mas eficiente posible para viajar/caminar lo menos posible.

Árbol expandido mínimo (Ejemplo)

Supongamos que el grafo siguiente representa nuestros recorridos usuales.

- ¿Cómo podemos hacer para recorrerlo todo de la forma más eficiente?
- ¿Qué problema surge?



Resolución 1

Nos proponemos encontrar un árbol expandido mínimo, para ello, buscamos todos los arboles expandidos posibles, calculando el peso y nos quedamos con el que tenga el menor

Problema: ¿Cuánto se tardaría en obtener un resultado? ¿Cuántos arboles expandidos posibles hay?

- Ejemplo: Un grafo completo con **n** vértices tiene **n^{n-2}** árboles expandidos

Resolución 2 – Algoritmo Greedy

- Se utiliza generalmente para resolver problemas de optimización (como el ejemplo)
- Se toman decisiones en función de la información que se tiene.
- No se “arrepiente” de sus decisiones.
- PERO, no garantiza la solución optima
- El algoritmo de kruskal utiliza este concepto

Estructura Union-Find

Es una estructura que modela una colección de conjuntos disjuntos

Es una estructura de datos que implementa los métodos `MakeSet()`, `Union()` y `Find()`. Es la estructura que utilizaremos para solucionar un problema aplicando el algoritmo de kruskal

En nuestra implementación, esta estructura vendría a ser un conjunto de arboles

MakeSet(x)

Es un método que inicializa la raíz de cada arbol, para ello, se inicializa como que el padre de cada Nodo es el Nodo mismo

MakeSet(9)



Vértices	0	1	2	3	4	5	6	7	8
Padre	0	1	2	3	4	5	6	7	8

Find(x)

Un método que se encarga de hallar la raíz de cierto elemento x . Para ello, busca el padre del Vértice Ingresado, y mientras el Vértice hallado tenga un padre distinto al mismo vértice, sigue buscando. Cuando el padre del vértice es igual al vértice mismo, quiere decir que ese vértice es la raíz

Union(x,y)

Este método se encarga de unir dos conjuntos que estaban disjuntos, para ello, busca la raíz de ambos conjuntos y le asigna a una de las raíces, como padre, la raíz del otro conjunto

Problema General

Nuestro problema general es hallar el árbol de expansión mínimo de un grafo dado.

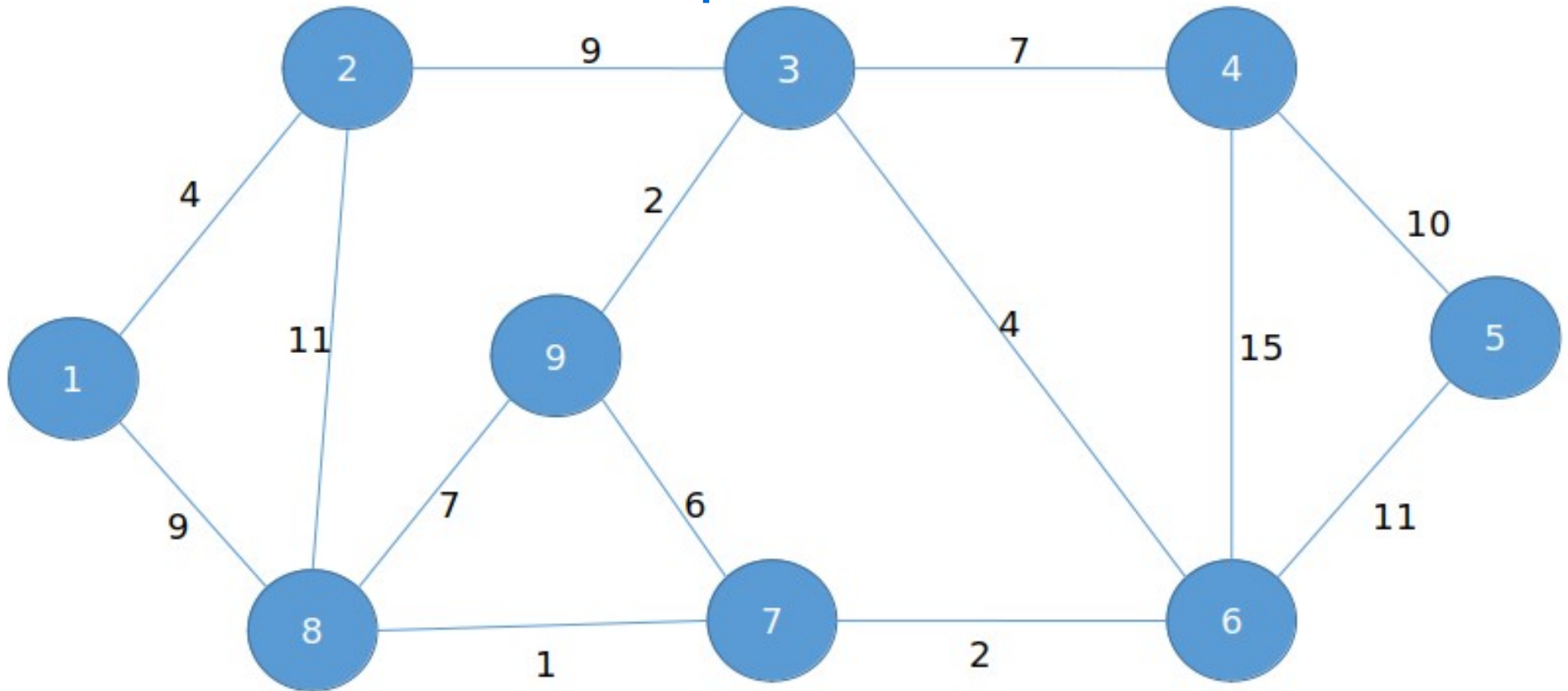
Además, necesitamos de alguna forma obtener las aristas del grafo ordenadas de forma ascendente (esto depende de la implementación del grafo)

Para ello, crearemos un bosque (conjunto de arboles) mediante un arreglo.

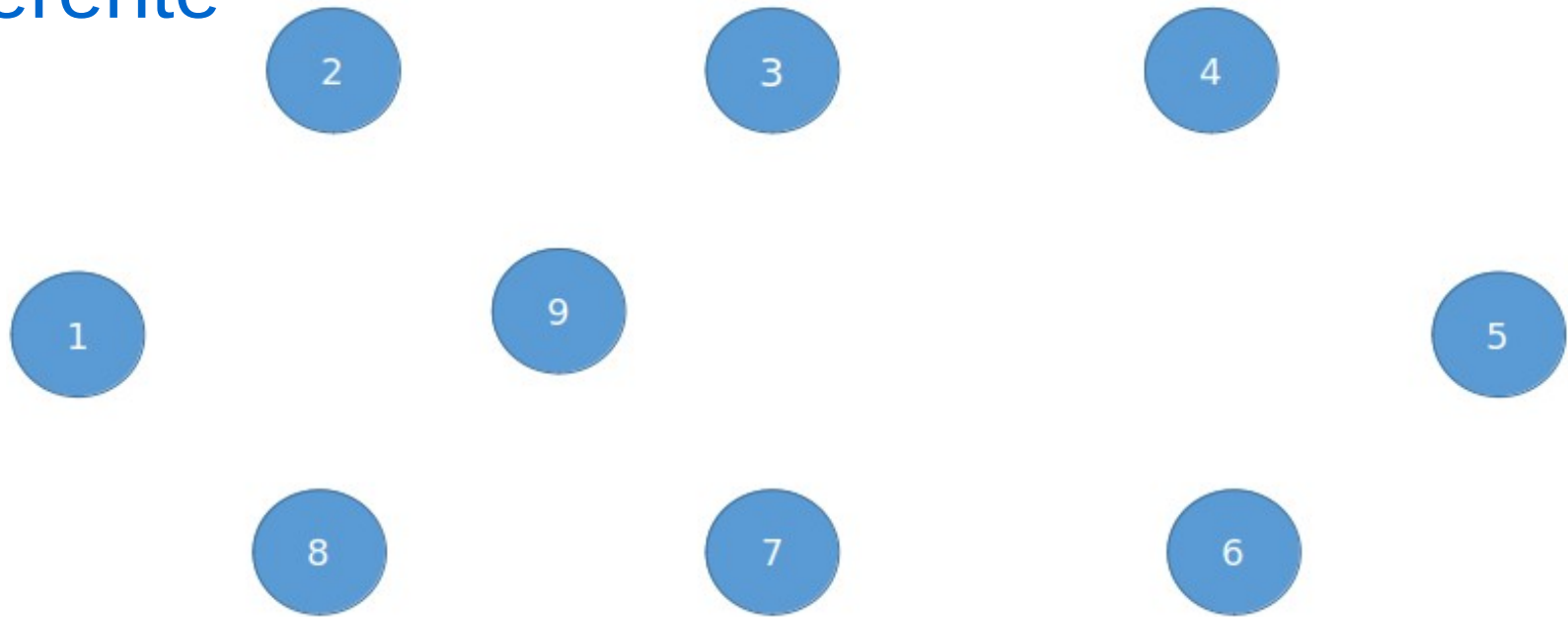
Nuestro bosque estará representado en un arreglo donde el i -ésimo elemento del arreglo representa el padre del i -ésimo Vértice/Nodo

Algoritmo de Kruskal

Veamos el siguiente grafo, tratemos de hallar el árbol expandido mínimo:

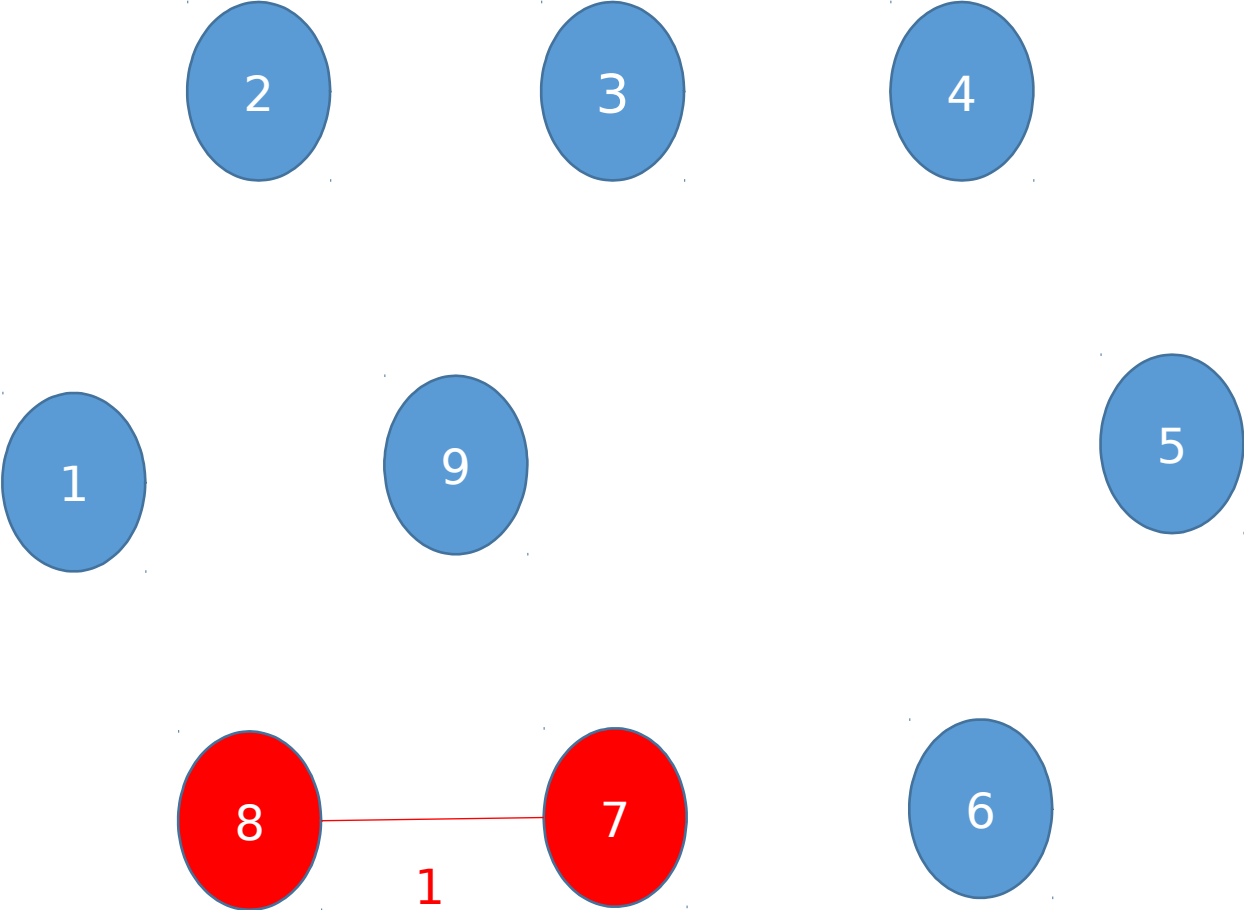


Para ello, nos creamos el siguiente bosque, hay que pensar cada Nodo como un árbol diferente



Vértice	1	2	3	4	5	6	7	8	9
padre	1	2	3	4	5	6	7	8	9

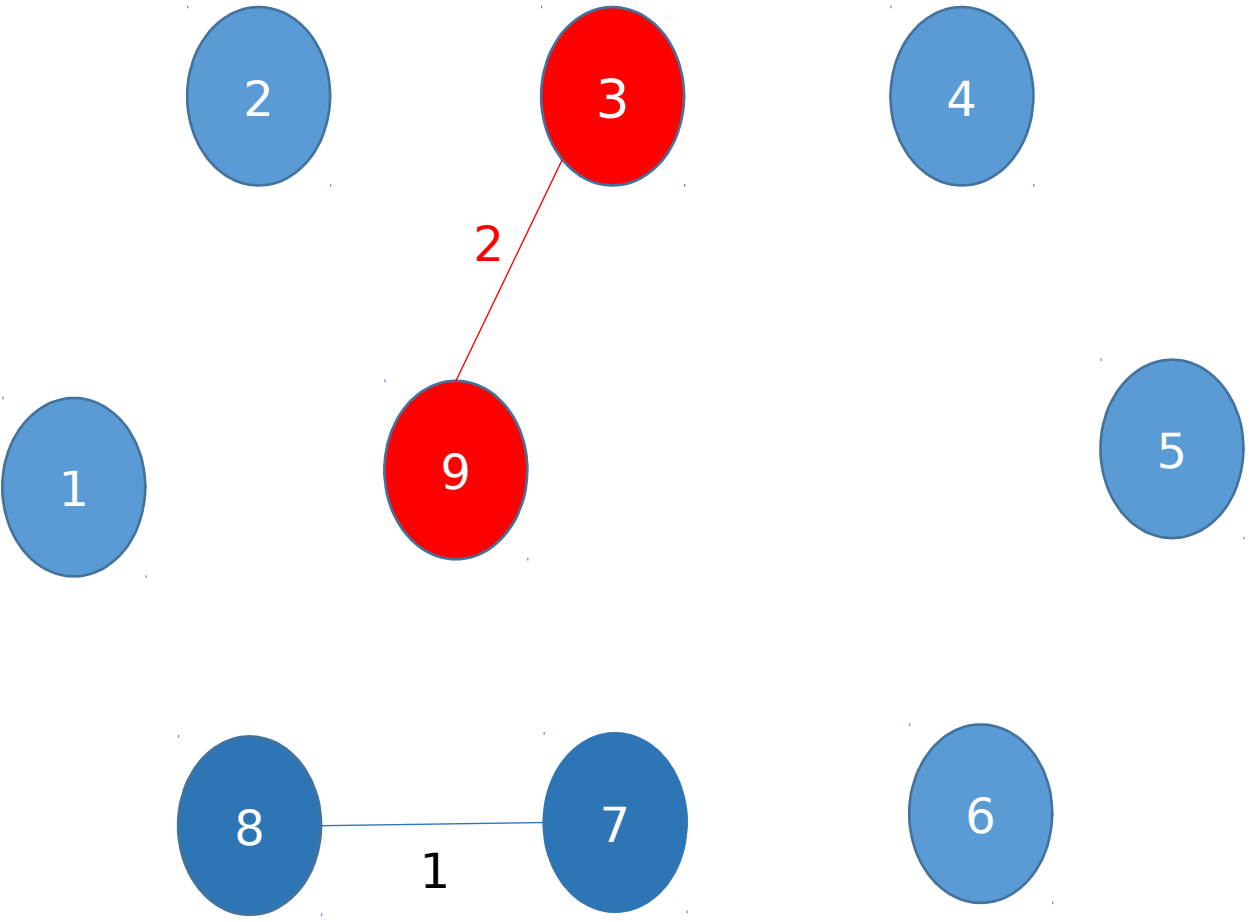
Vértices de las Aristas	Peso de la Arista
8-7	1
3 - 9	2
6 - 7	2
1 - 2	4
3 - 6	4
7 - 9	6
3 - 4	7
8 - 9	7
1 - 8	9
2 - 3	9
4 - 5	10
2 - 8	11
5 - 6	11
4 - 6	15



Vértices de las Aristas	Peso de la Arista
8-7	1
3 - 9	2
6 - 7	2
1 - 2	4
3 - 6	4
7 - 9	6
3 - 4	7
8 - 9	7
1 - 8	9
2 - 3	9
4 - 5	10
2 - 8	11
5 - 6	11
4 - 6	15

Vértice	1	2	3	4	5	6	7	8	9
Padre	1	2	3	4	5	6	7	7	9

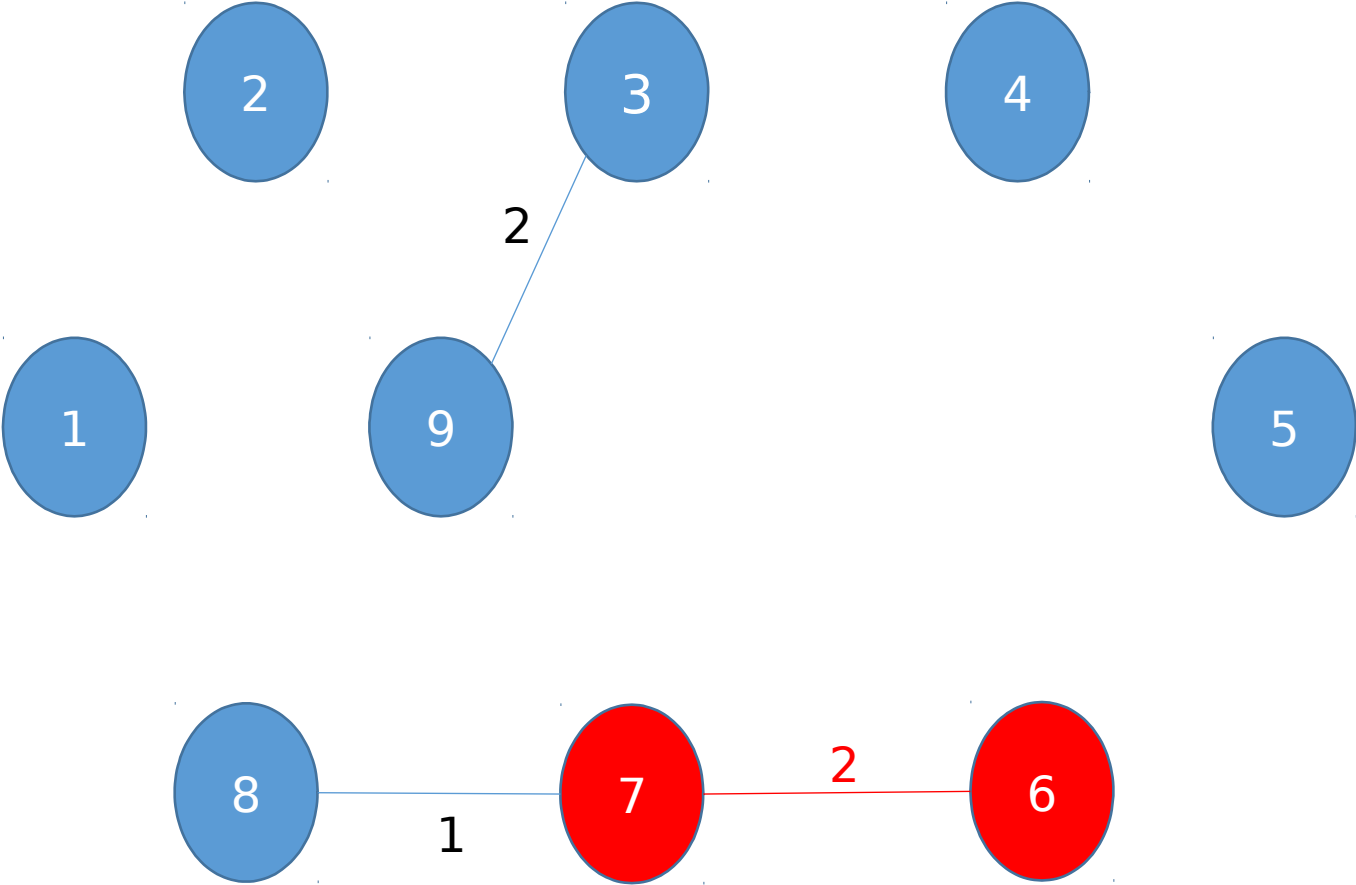
Raiz(8) = 8;
Raiz(7) = 7;



Vértices de las Aristas	Peso de la Arista
8-7	1
3 - 9	2
6 - 7	2
1 - 2	4
3 - 6	4
7 - 9	6
3 - 4	7
8 - 9	7
1 - 8	9
2 - 3	9
4 - 5	10
2 - 8	11
5 - 6	11
4 - 6	15

Vértice	1	2	3	4	5	6	7	8	9
Padre	1	2	9	4	5	6	7	7	9

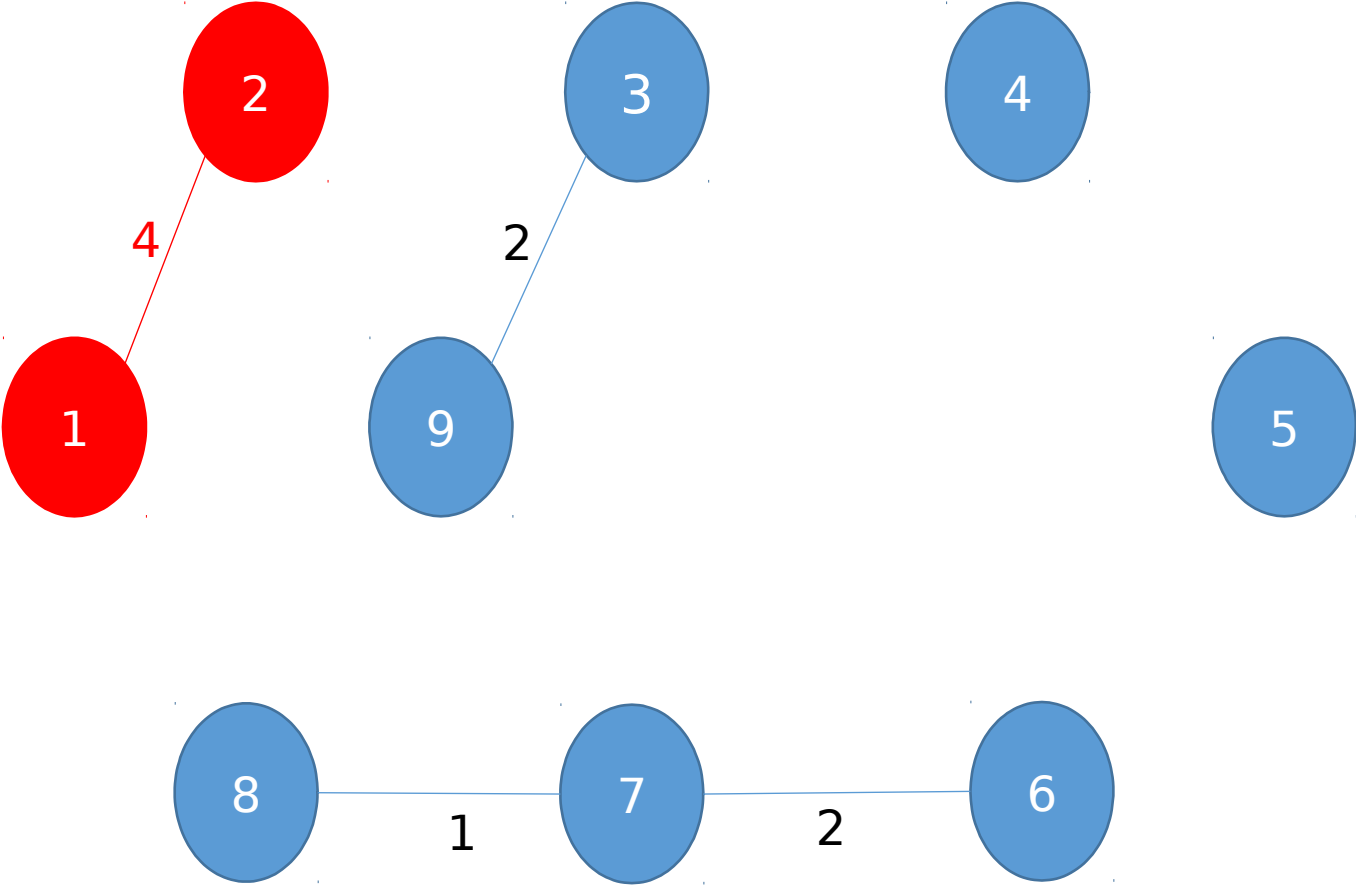
Raiz(3) = 3;
Raiz(9) = 9;



Vértice	1	2	3	4	5	6	7	8	9
Padre	1	2	9	4	5	7	7	7	9

Vértices de las Aristas	Peso de la Arista
8-7	1
3 - 9	2
6 - 7	2
1 - 2	4
3 - 6	4
7 - 9	6
3 - 4	7
8 - 9	7
1 - 8	9
2 - 3	9
4 - 5	10
2 - 8	11
5 - 6	11
4 - 6	15

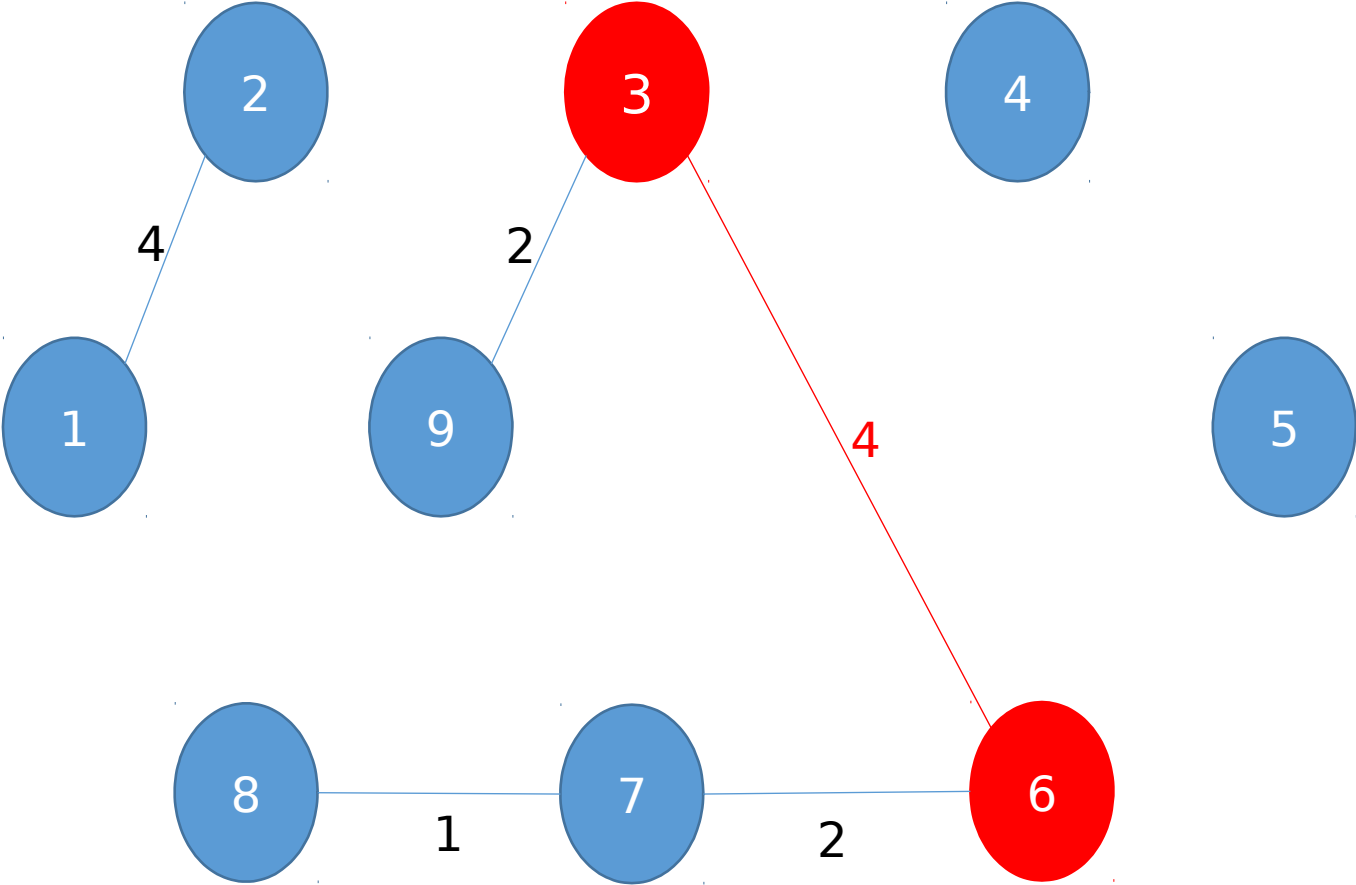
Raiz(6) = 6;
Raiz(7) = 7;



Vértice	1	2	3	4	5	6	7	8	9
Raíz	2	2	9	4	5	7	7	7	9

Vértices de las Aristas	Peso de la Arista
8-7	1
3 - 9	2
6 - 7	2
1 - 2	4
3 - 6	4
7 - 9	6
3 - 4	7
8 - 9	7
1 - 8	9
2 - 3	9
4 - 5	10
2 - 8	11
5 - 6	11
4 - 6	15

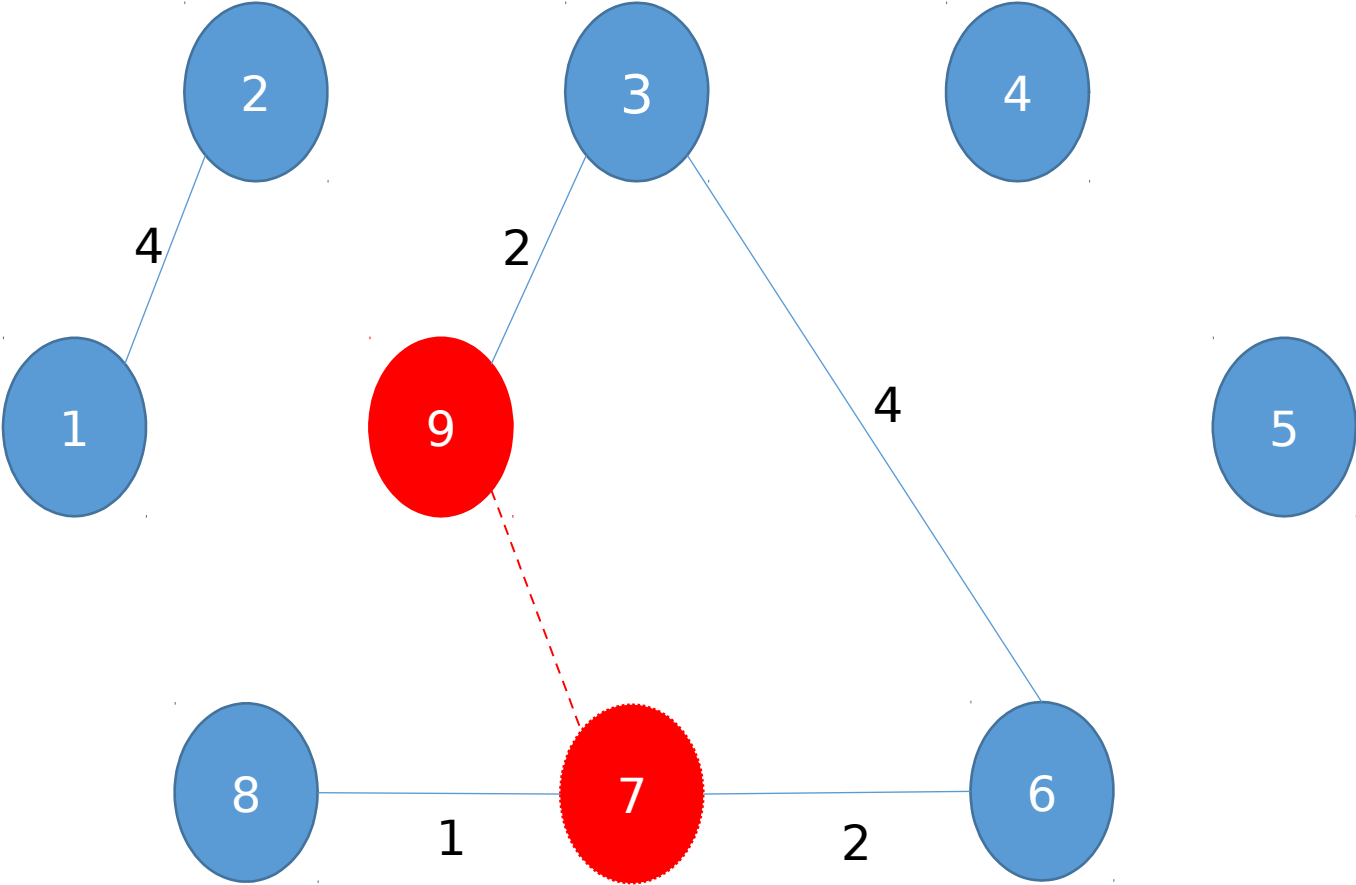
Raiz(1) = 1;
Raiz(2) = 2;



Vértice	1	2	3	4	5	6	7	8	9
Padre	2	2	9	4	5	7	7	7	7

Vértices de las Aristas	Peso de la Arista
8 - 7	1
3 - 9	2
6 - 7	2
1 - 2	4
3 - 6	4
7 - 9	6
3 - 4	7
8 - 9	7
1 - 8	9
2 - 3	9
4 - 5	10
2 - 8	11
5 - 6	11
4 - 6	15

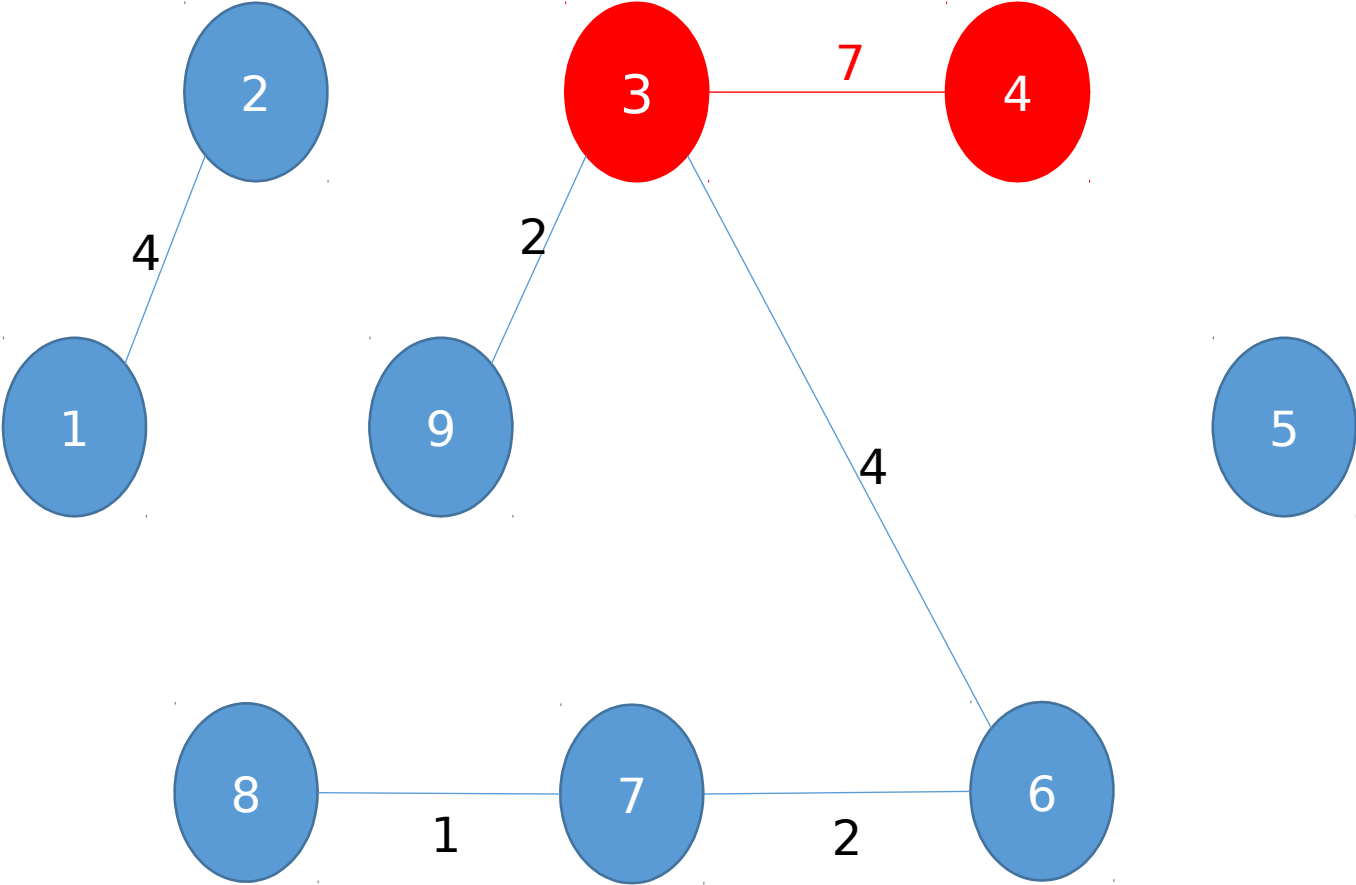
Raiz(3) = 9;
Raiz(6) = 7;



Vértice	1	2	3	4	5	6	7	8	9
Padre	2	2	9	4	5	7	7	7	7

Vértices de las Aristas	Peso de la Arista
8-7	1
3 - 9	2
6 - 7	2
1 - 2	4
3 - 6	4
7 - 9	6
3 - 4	7
8 - 9	7
1 - 8	9
2 - 3	9
4 - 5	10
2 - 8	11
5 - 6	11
4 - 6	15

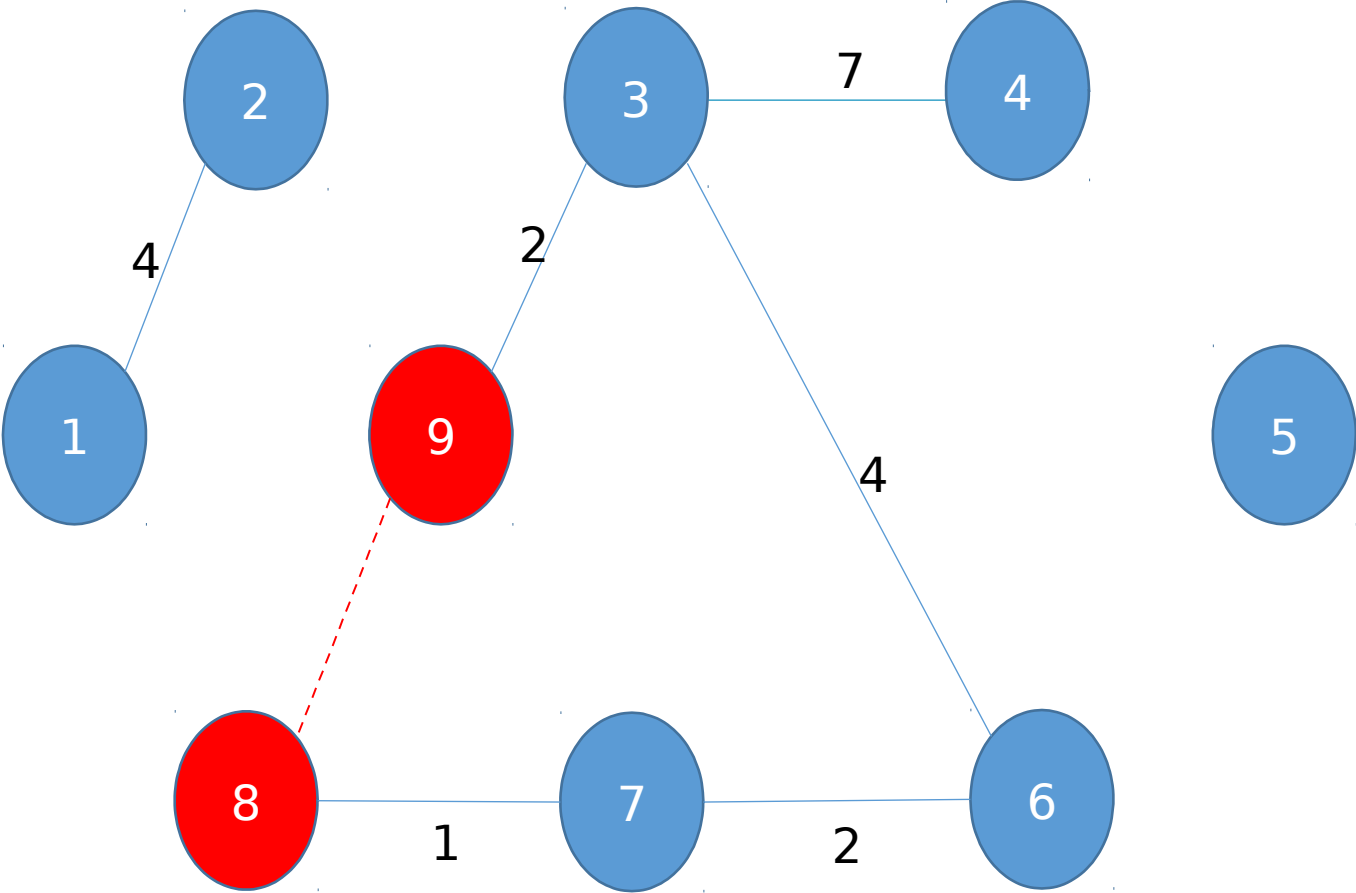
Raiz(7) = 7;
Raiz(9) = 7;



Vértice	1	2	3	4	5	6	7	8	9
Padre	2	2	9	4	5	7	4	7	7

Vértices de las Aristas	Peso de la Arista
8-7	1
3 - 9	2
6 - 7	2
1 - 2	4
3 - 6	4
7 - 9	6
3 - 4	7
8 - 9	7
1 - 8	9
2 - 3	9
4 - 5	10
2 - 8	11
5 - 6	11
4 - 6	15

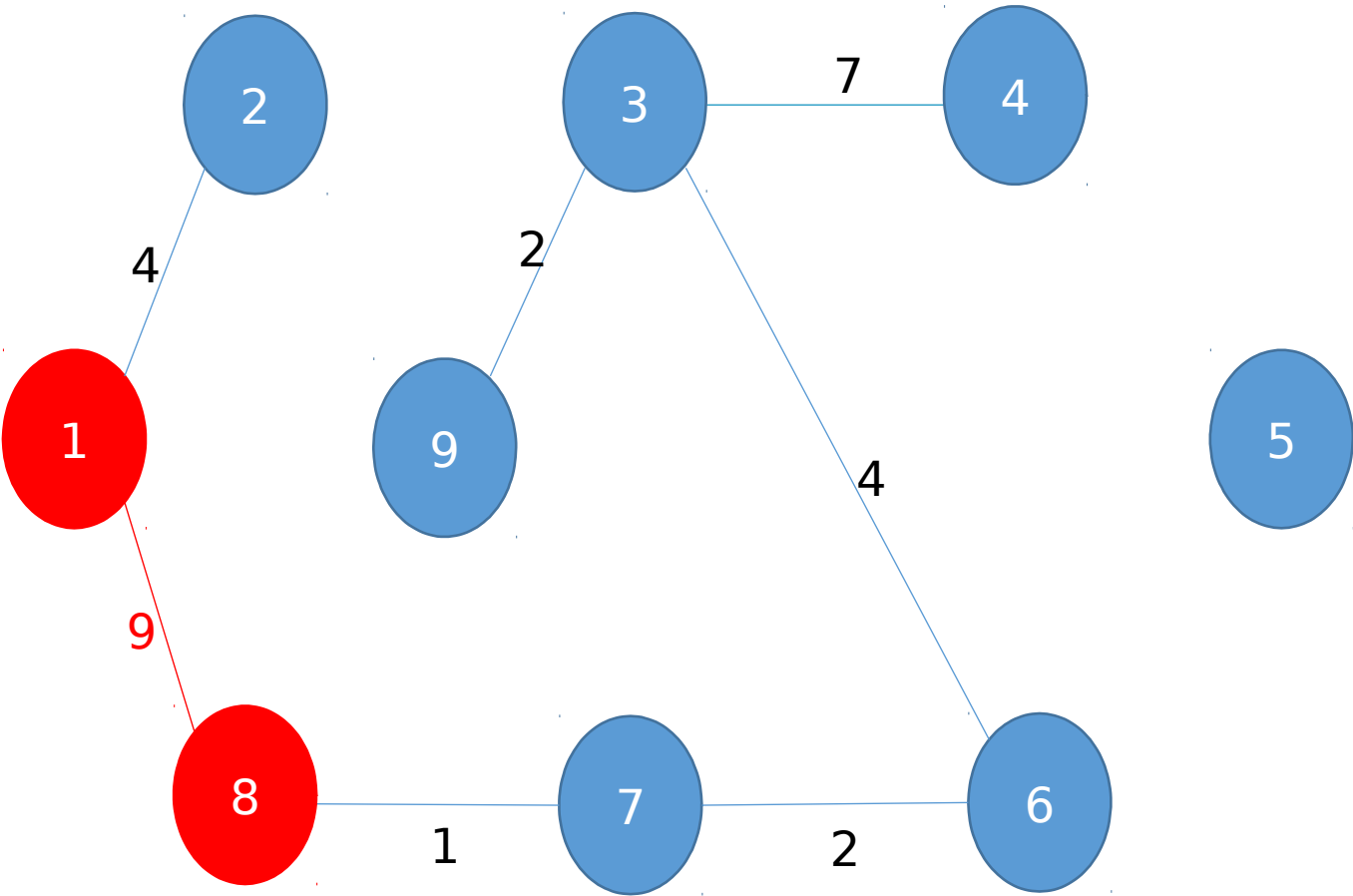
Raiz(3) = 7;
Raiz(4) = 4;



Vértice	1	2	3	4	5	6	7	8	9
Padre	2	2	9	4	5	7	4	7	7

Vértices de las Aristas	Peso de la Arista
8-7	1
3 - 9	2
6 - 7	2
1 - 2	4
3 - 6	4
7 - 9	6
3 - 4	7
8 - 9	7
1 - 8	9
2 - 3	9
4 - 5	10
2 - 8	11
5 - 6	11
4 - 6	15

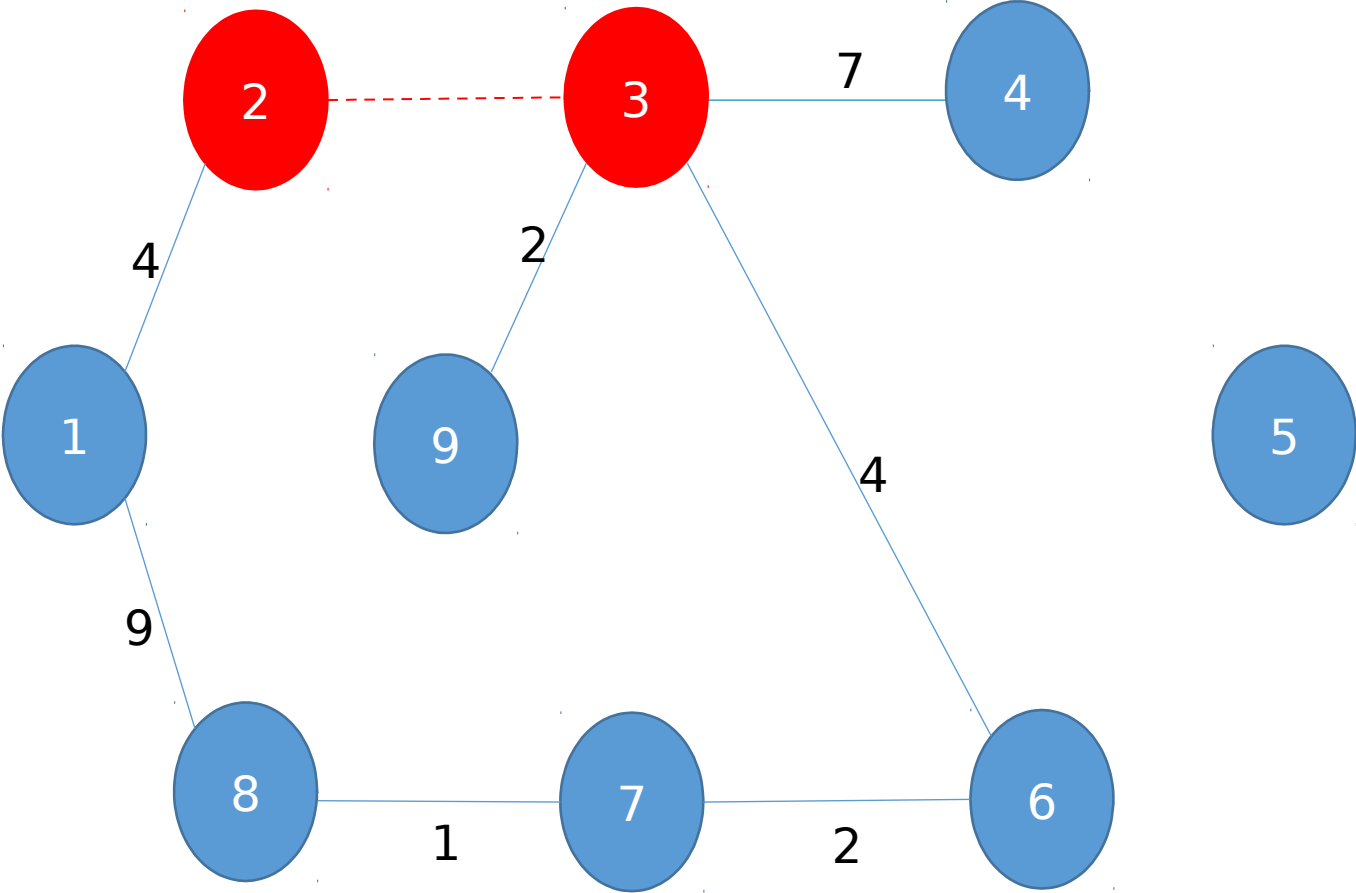
Raiz(8) = 4;
Raiz(9) = 4;



Vértice	1	2	3	4	5	6	7	8	9
Padre	2	4	9	4	5	7	4	7	7

Vértices de las Aristas	Peso de la Arista
8-7	1
3 - 9	2
6 - 7	2
1 - 2	4
3 - 6	4
7 - 9	6
3 - 4	7
8 - 9	7
1 - 8	9
2 - 3	9
4 - 5	10
2 - 8	11
5 - 6	11
4 - 6	15

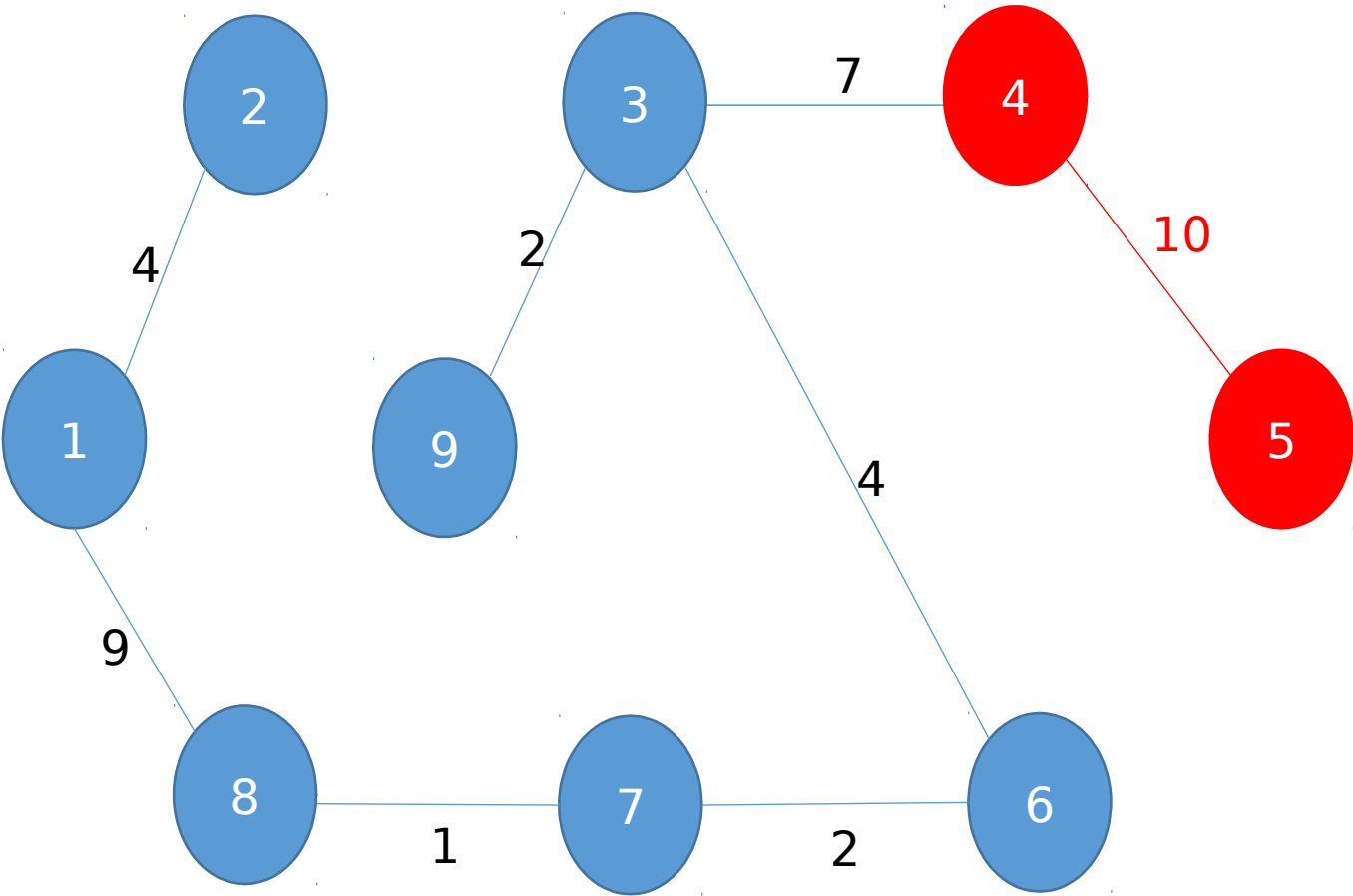
Raiz(1) = 2;
Raiz(8) = 4;



Vértice	1	2	3	4	5	6	7	8	9
Padre	2	4	9	4	5	7	4	7	7

Vértices de las Aristas	Peso de la Arista
8 - 7	1
3 - 9	2
6 - 7	2
1 - 2	4
3 - 6	4
7 - 9	6
3 - 4	7
8 - 9	7
1 - 8	9
2 - 3	9
4 - 5	10
2 - 8	11
5 - 6	11
4 - 6	15

Raiz(2) = 4;
Raiz(3) = 4;

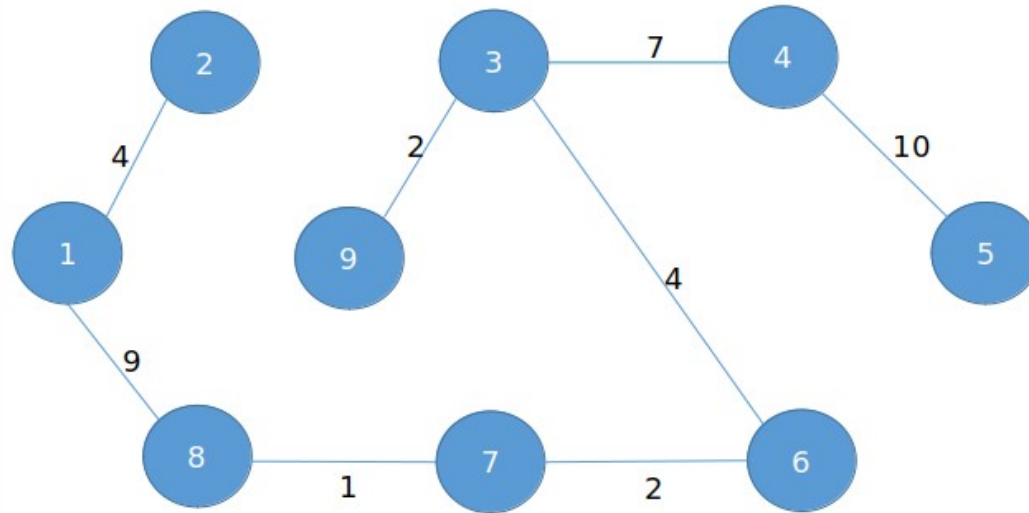


Vértice	1	2	3	4	5	6	7	8	9
Raíz	2	4	9	5	5	7	4	7	7

Vértices de las Aristas	Peso de la Arista
8 - 7	1
3 - 9	2
6 - 7	2
1 - 2	4
3 - 6	4
7 - 9	6
3 - 4	7
8 - 9	7
1 - 8	9
2 - 3	9
4 - 5	10
2 - 8	11
5 - 6	11
4 - 6	15

Raiz(4) = 4;
Raiz(5) = 5;

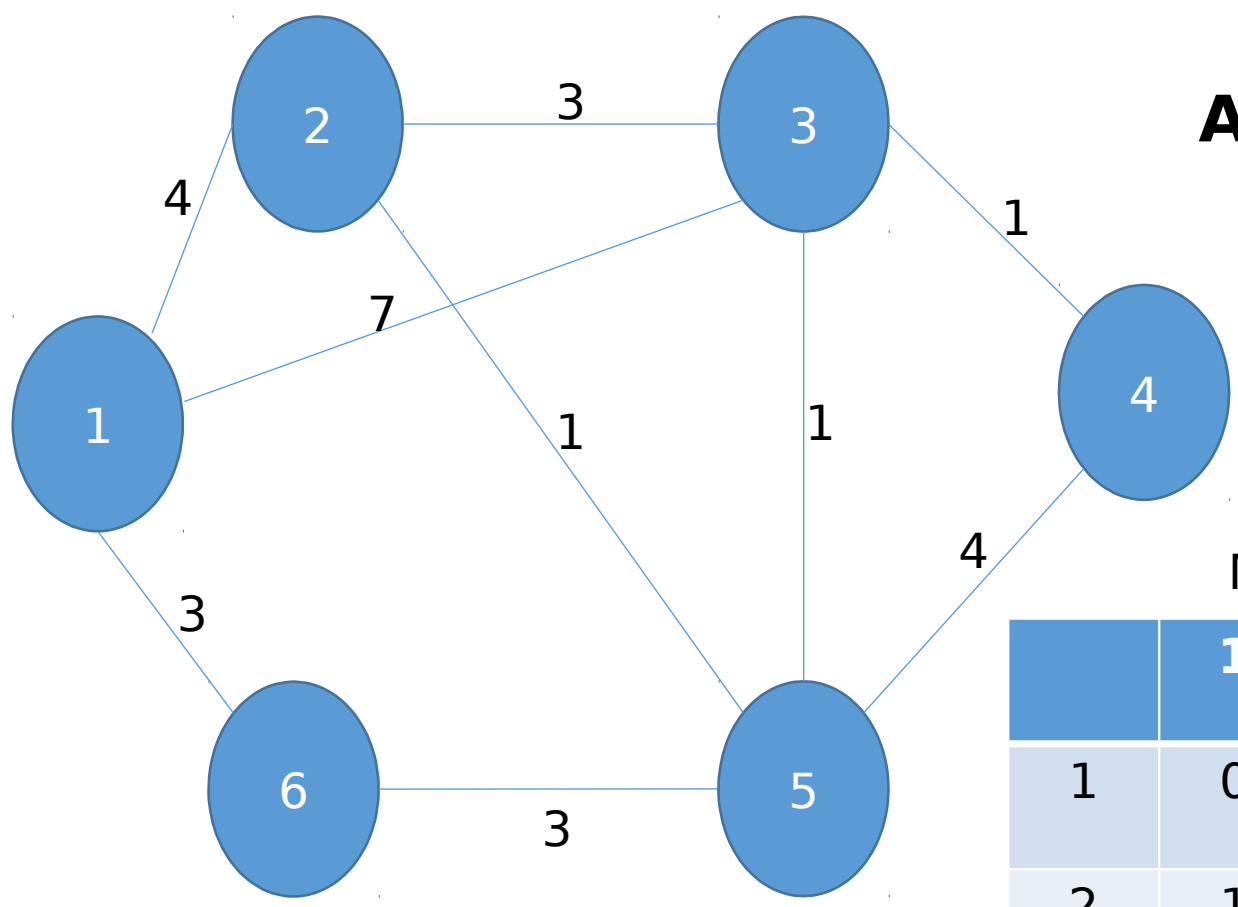
Como podemos observar ya están todos los vértices del grafo conectados así que al momento de continuar viendo las demás aristas ordenadas siempre tendremos el caso de que ya están en la misma componente conexa por lo tanto el Árbol de Expansión Mínima para el grafo es el siguiente:



Supongamos ahora esta implementación:

```
Int v = 9;  
Int a = 14;  
Struct Nodo{  
    Int origen;           //vértice origen  
    Int destino;        //vértice destino  
    Int peso;           //peso entre el vértice origen y destino  
};  
Nodo arista[a];  
Int padre[v];           // este arreglo contiene los padres de los vértices
```


Algoritmo de Prim



Matriz de adyacencia

	1	2	3	4	5	6
1	0	1	1	0	0	1
2	1	0	1	0	1	0
3	1	1	0	1	1	0
4	0	0	1	0	1	0
5	0	1	1	1	0	1
6	1	0	0	0	1	0

```
Nodo inicial s           // s = vértice 1
Crear_cola()           // es una cola de prioridad, donde la prioridad es la
                           distancia/peso
Para todo U en V[G] hacer // u es un nodo/vértice del grafo
    visitado[u] = false           //guarda si un nodo ya fue visitado
    distancia[u] = infinito;      // guarda la distancias de cada nodo al
    conjunto visitado
    padre[u] = NULL
```

```
distancia[s] = 0
encolar(s)
```

Vértice	1	2	3	4	5	6
Visitado	false	false	false	false	false	false
Vértice	1	2	3	4	5	6
Distancia	0	∞	∞	∞	∞	∞
Vértice	1	2	3	4	5	6
Padre	NULL	NULL	NULL	NULL	NULL	NULL

COLA
1

mientras que cola no sea vacia hacer

u = desencolar(col)

visitado[u] = true

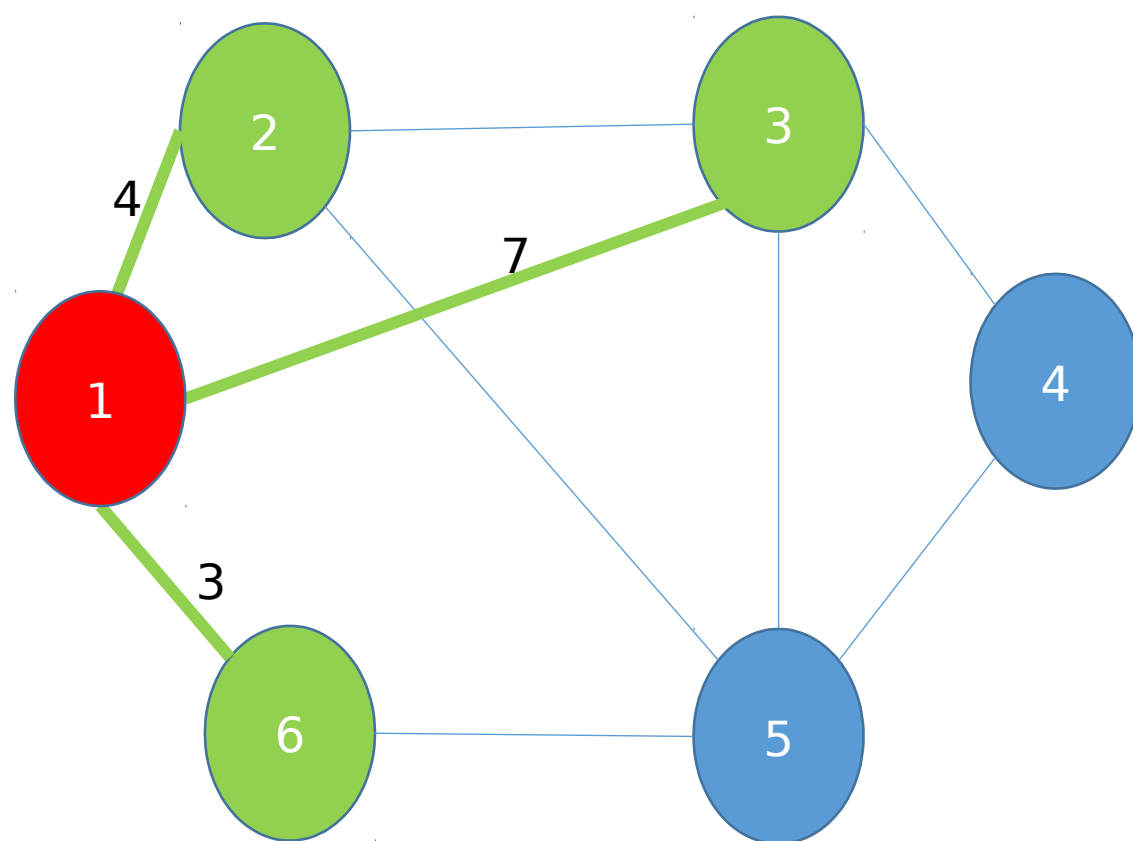
para todo v en vecinos de u hacer

si no visitado[v] y distancia[v] > peso(u, v) hacer

distancia[v] = peso(u, v)

padre[v] = u

encolar(v)



Desencolar(cola) = 1

Visitado[1] = true

Distancia[2] = peso(2,1)

Padre[2] = 1;

Encolar(2)

Distancia[3] = peso(3,1)

Padre[3] = 1

Encolar(3)

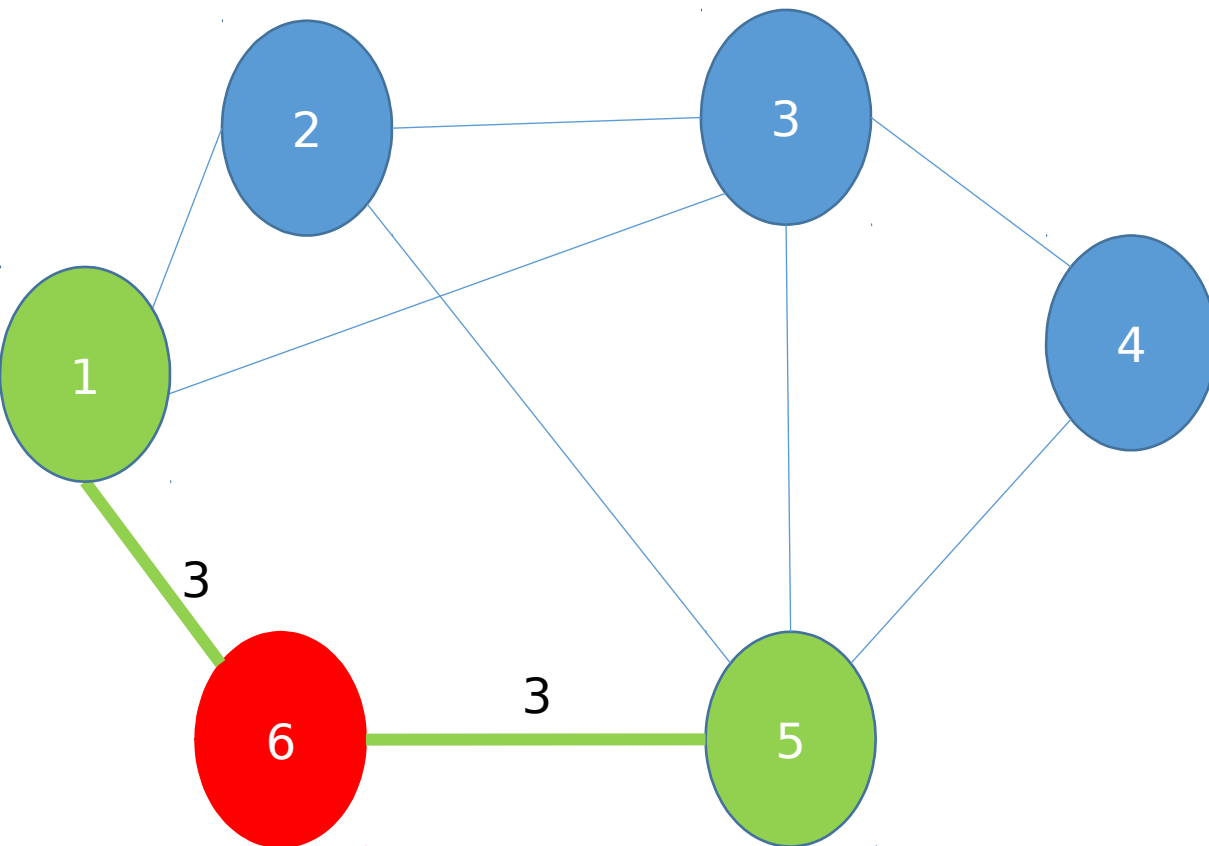
Distancia[6] = peso(6,1)

Padre[6] = 1

Encolar(6)

Vértice	1	2	3	4	5	6
Visitado	true	false	false	false	false	false
Vértice	1	2	3	4	5	6
Distancia	0	∞ 4	∞ 7	∞	∞	∞ 3
Vértice	1	2	3	4	5	6
Padre	NULL	NULL 1	NULL 1	NULL	NULL	NULL 1

cola
6
2
3

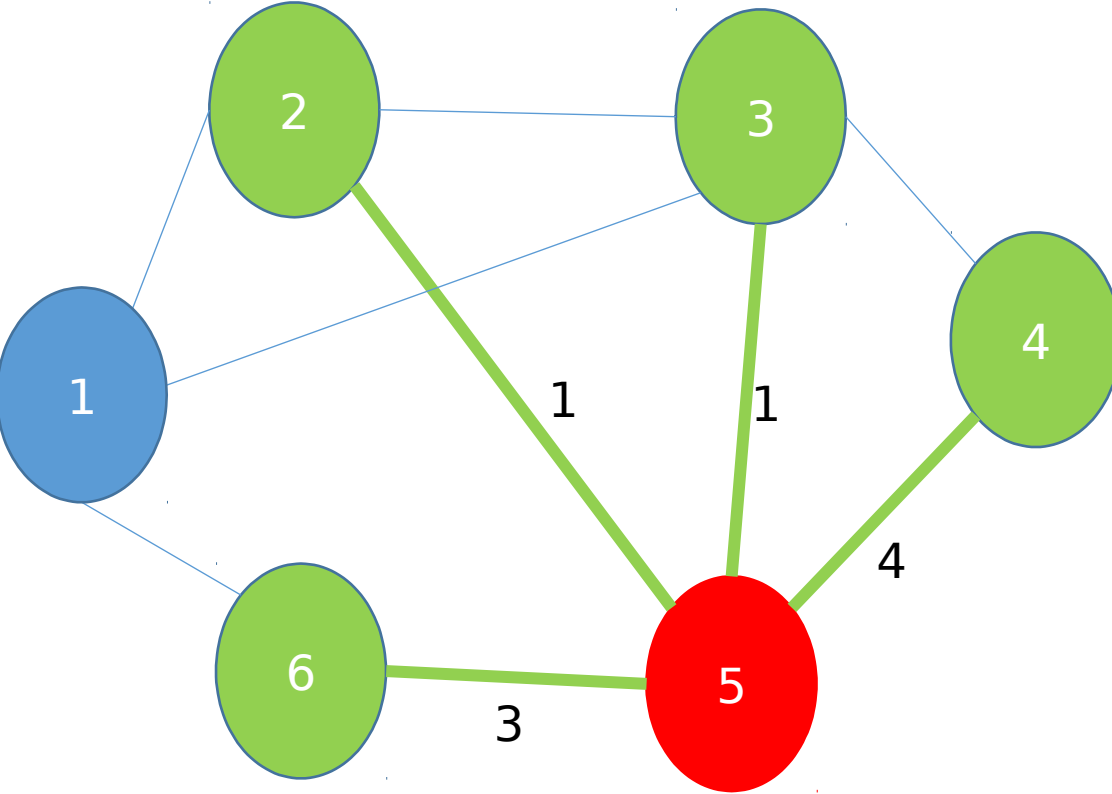


Desencolar(
cola
) = 6
Visitado[6] =
true

Distancia[5] =
peso(5,6)
Padre[5] = 6
Encolar(5)

Vértice	1	2	3	4	5	6
Visitado	true	false	false	false	false	true
Vértice	1	2	3	4	5	6
Distancia	0	4	7	∞	∞ 3	3
Vértice	1	2	3	4	5	6
Padre	NULL	1	1	NULL	NULL 6	1

cola
5
2
3

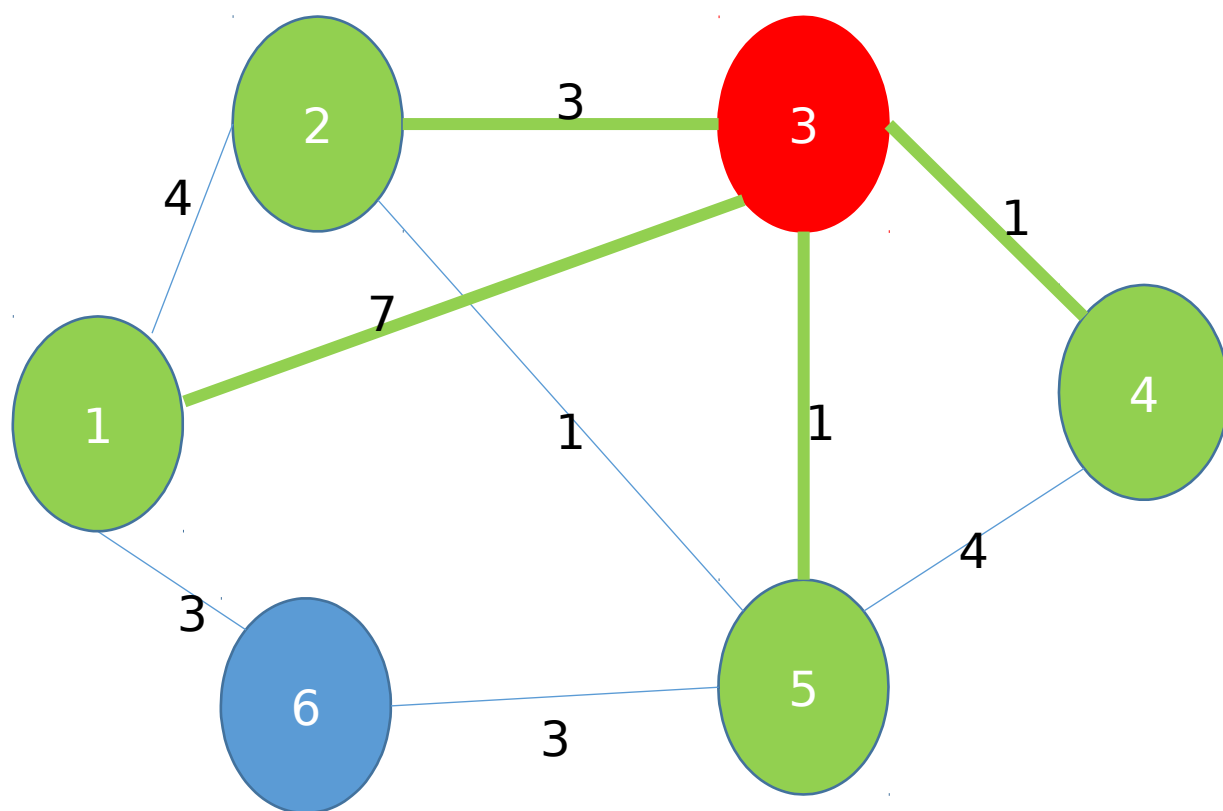


Desencolar(cola) = 5
 Visitado[5] = true

Distancia[2] = peso(2,5)
 Padre[2] = 5
 Encolar(2)
 Distancia[3] = peso(3,5)
 Padre[3] = 5
 Encolar(3)
 Distancia[4] = peso(4,5)
 Padre[4] = 5
 Encolar(4)

Vértice	1	2	3	4	5	6
Visitado	true	false	false	false	true	true
Vértice	1	2	3	4	5	6
Distancia	0	4 ¹	7 ¹	∞ ⁴	3	3
Vértice	1	2	3	4	5	6
Padre	NULL	1 ⁵	1 ⁵	NULL ⁵	6	1

cola
3
2
4

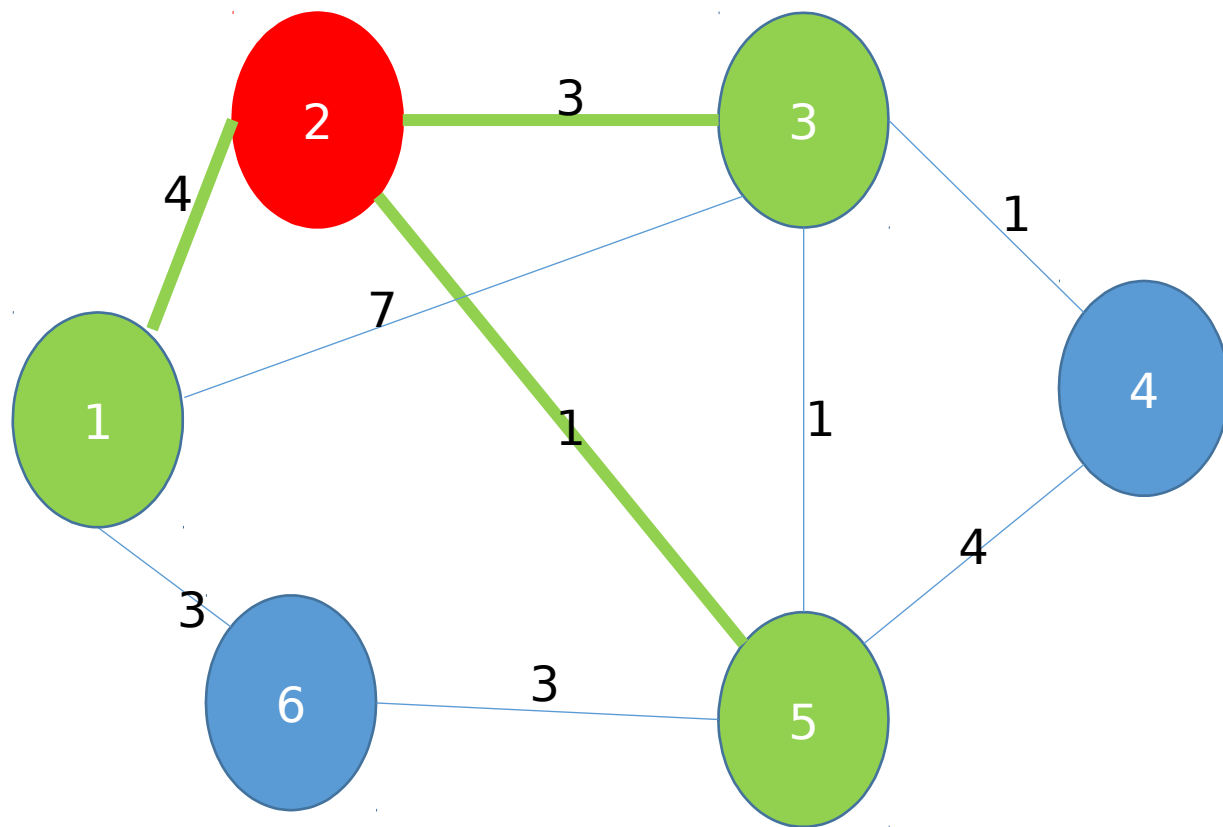


Desencolar(
cola
) = 3
Visitado[3] =
true

Distancia[4] =
peso(4,3)
Padre[4] = 3
Encolar(4)

Vértice	1	2	3	4	5	6
Visitado	true	false	true	false	true	true
Vértice	1	2	3	4	5	6
Distancia	0	1	1	4 1	3	3
Vértice	1	2	3	4	5	6
Padre	NULL	5	5	5 3	6	1

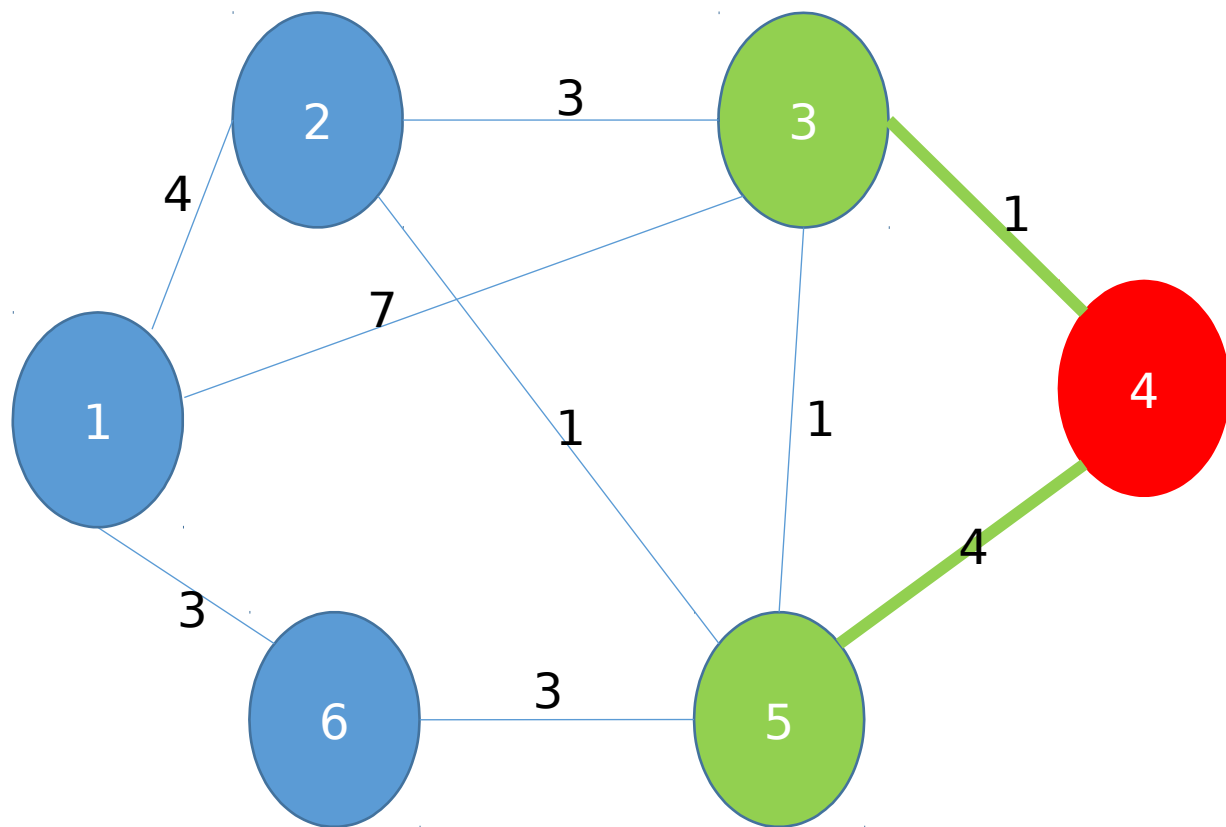
cola
2
4



Desencolar(
cola
) = 2
Visitado[2] =
true

Vértice	1	2	3	4	5	6
Visitado	true	true	true	false	true	true
Vértice	1	2	3	4	5	6
Distancia	0	1	1	1	3	3
Vértice	1	2	3	4	5	6
Padre	NULL	5	5	3	6	1

cola
4



Desencolar(cola
) = 4
Visitado[4] =
true

Vértice	1	2	3	4	5	6
Visitado	true	true	true	true	true	true
Vértice	1	2	3	4	5	6
Distancia	0	1	1	1	3	3
Vértice	1	2	3	4	5	6
Padre	NULL	5	5	3	6	1

cola

