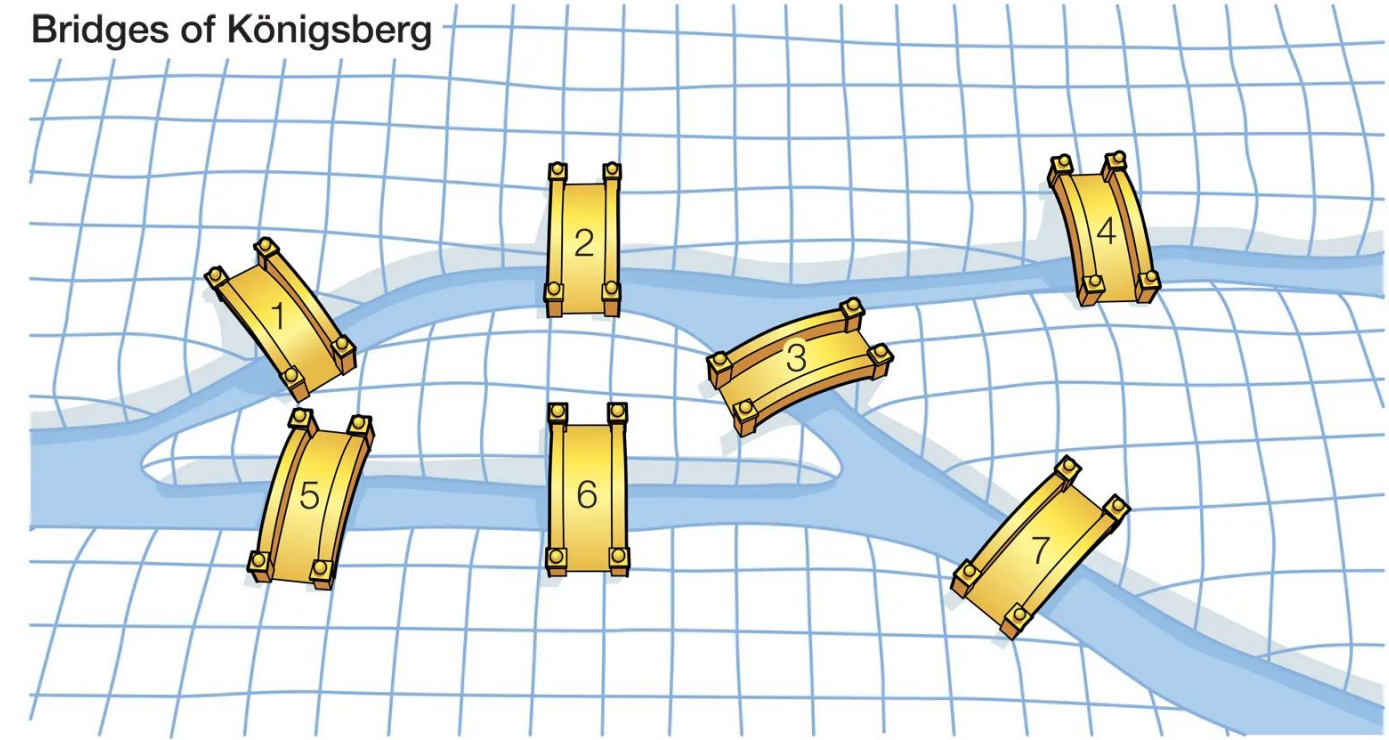
CIENCIAS DE LA COMPUTACIÓN
UNIVERSIDAD DE CHILE

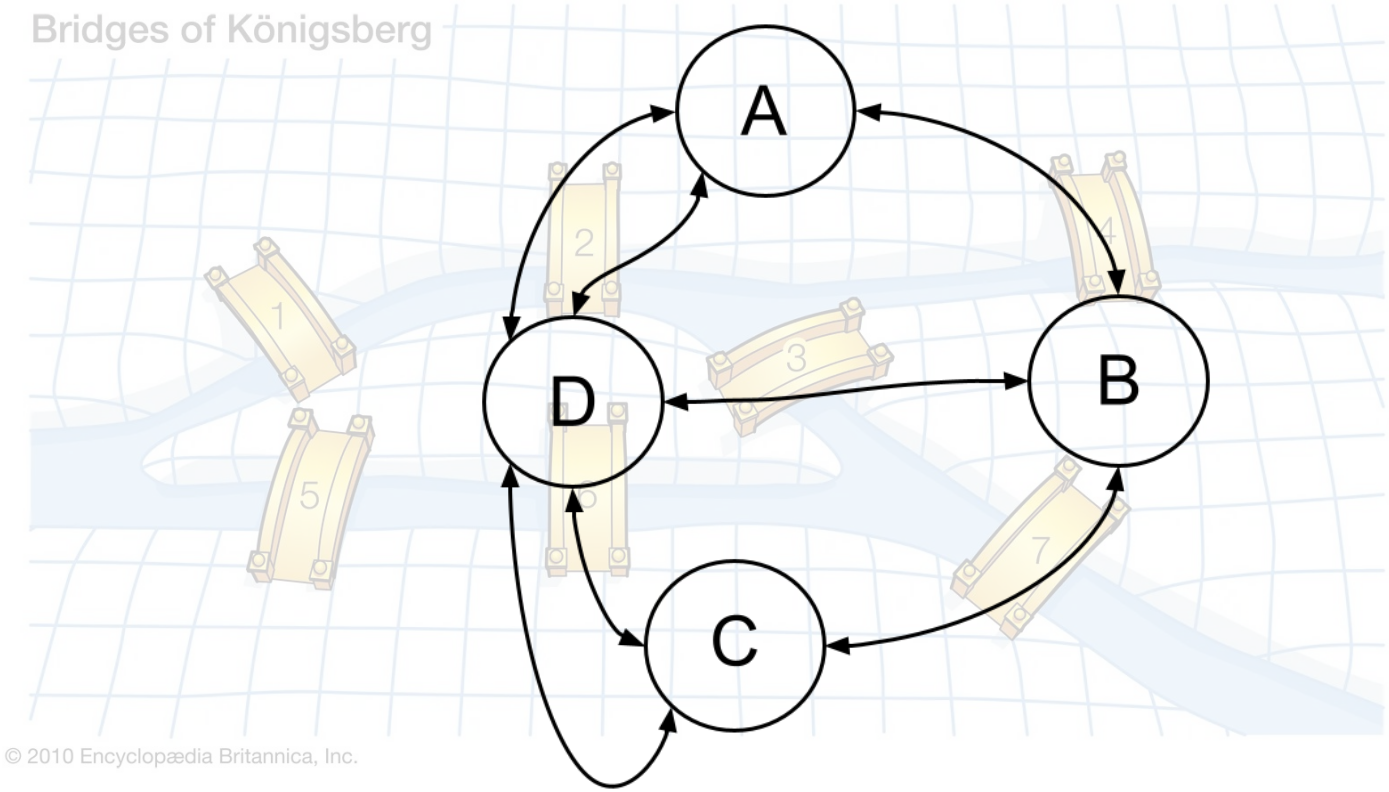
Clase 2: Grafos I
CC4001-CC4005

Otoño 2023

Puentes de Königsberg



Puentes de Königsberg

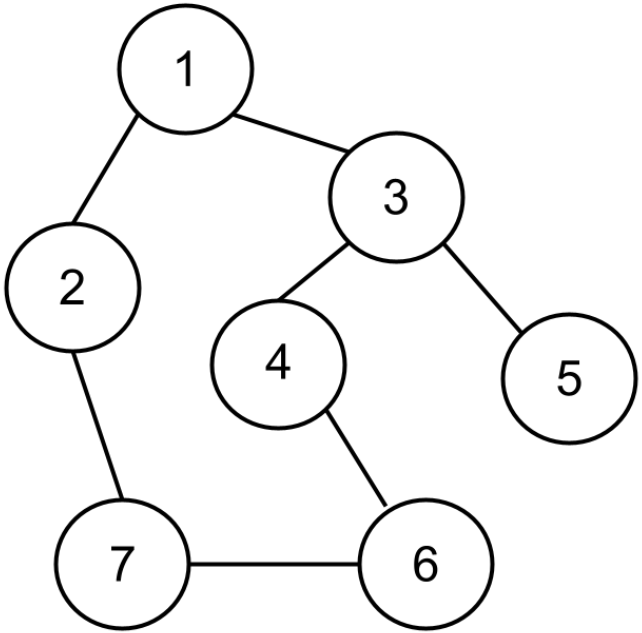


Grafo??

Un **grafo** es una forma de representar un conjunto de objetos *V* (**vértices** o nodos) y como están relacionados entre ellos (**aristas**)

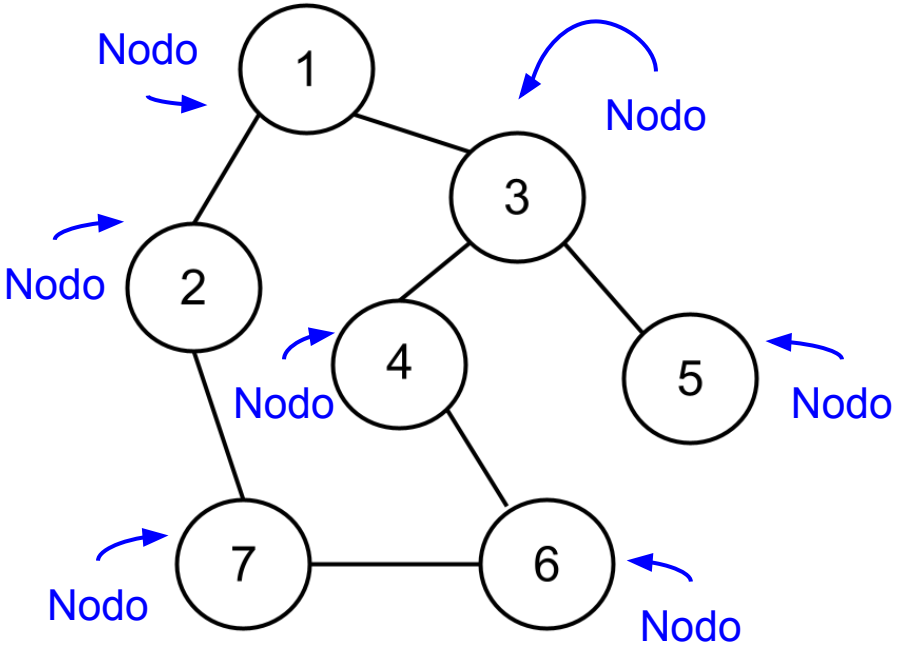
Grafo??

Un **grafo** es una forma de representar un conjunto de objetos *V* (**vértices** o nodos) y como están relacionados entre ellos (**aristas**)



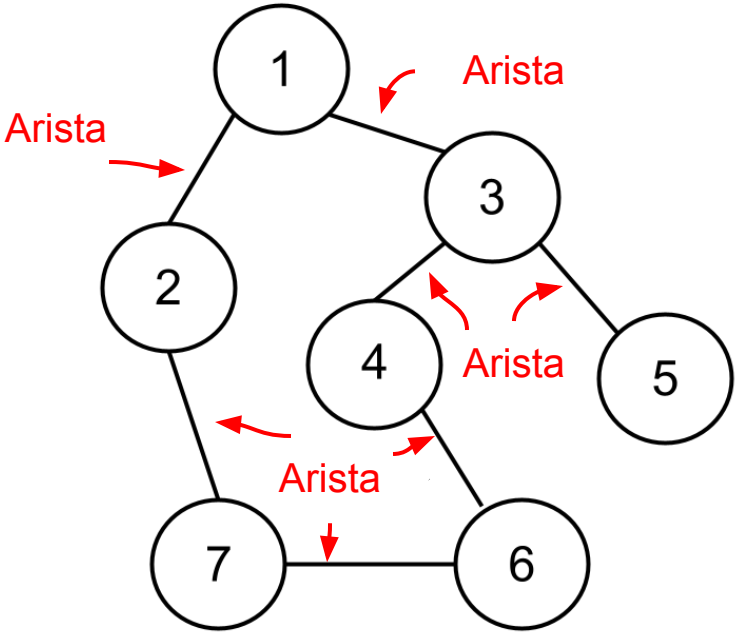
Grafo??

Un **grafo** es una forma de representar un conjunto de objetos *V* (**vértices** o nodos) y como están relacionados entre ellos (**aristas**)



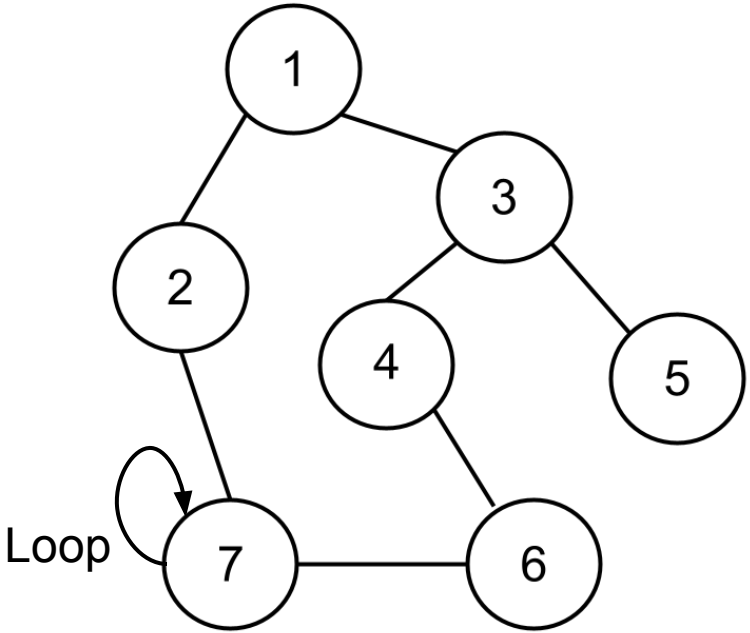
Grafo??

Un **grafo** es una forma de representar un conjunto de objetos *V* (**vértices** o nodos) y como están relacionados entre ellos (**aristas**)



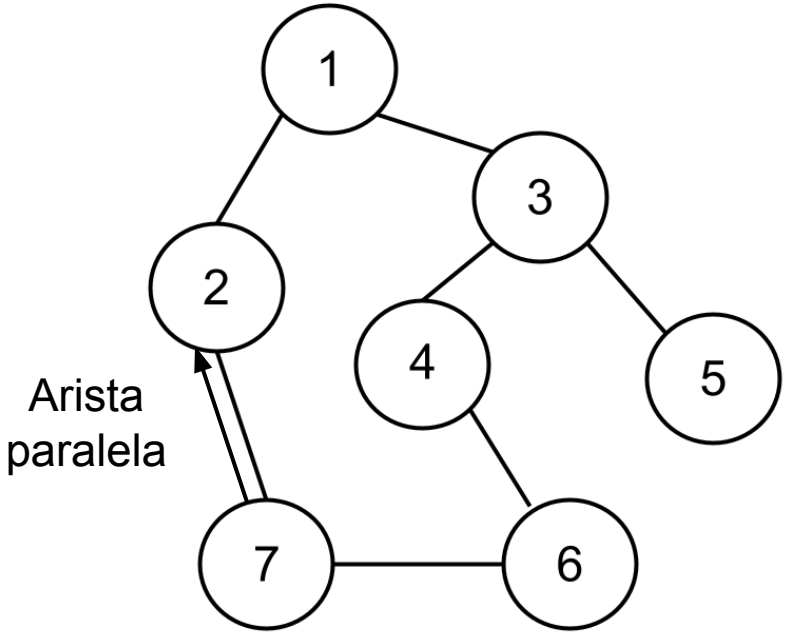
Grafo??

Un **grafo** es una forma de representar un conjunto de objetos *V* (**vértices** o nodos) y como están relacionados entre ellos (**aristas**)



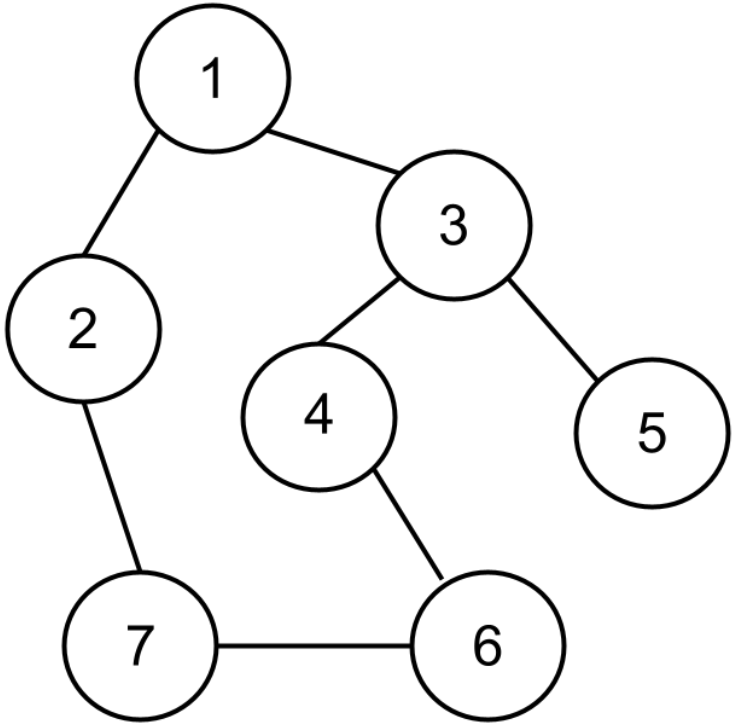
Grafo??

Un **grafo** es una forma de representar un conjunto de objetos *V* (**vértices** o nodos) y como están relacionados entre ellos (**aristas**)



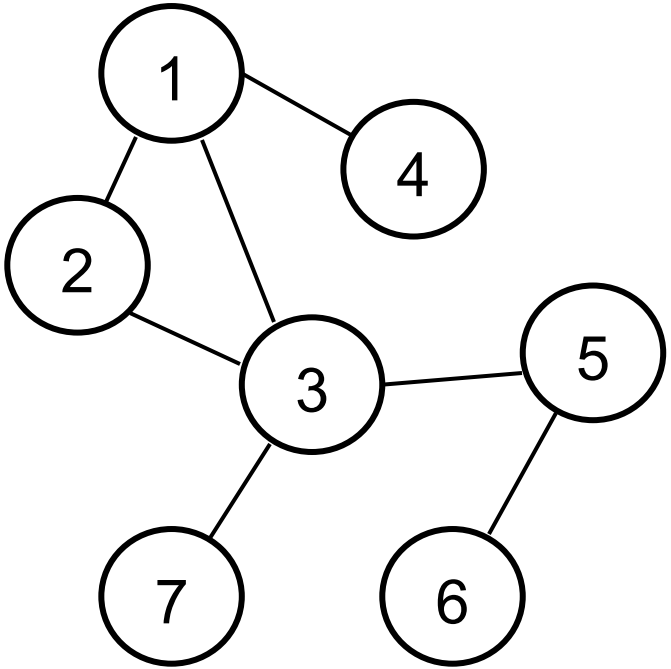
Grafo Simple

Si el grafo no tiene loops ni aristas paralelas, diremos que es un grafo **simple**

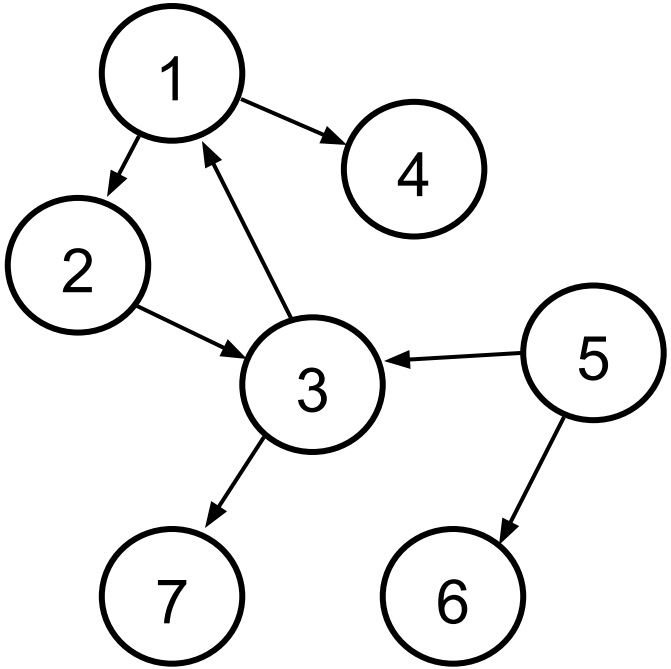


Grafo no dirigido y dirigido

No dirigido

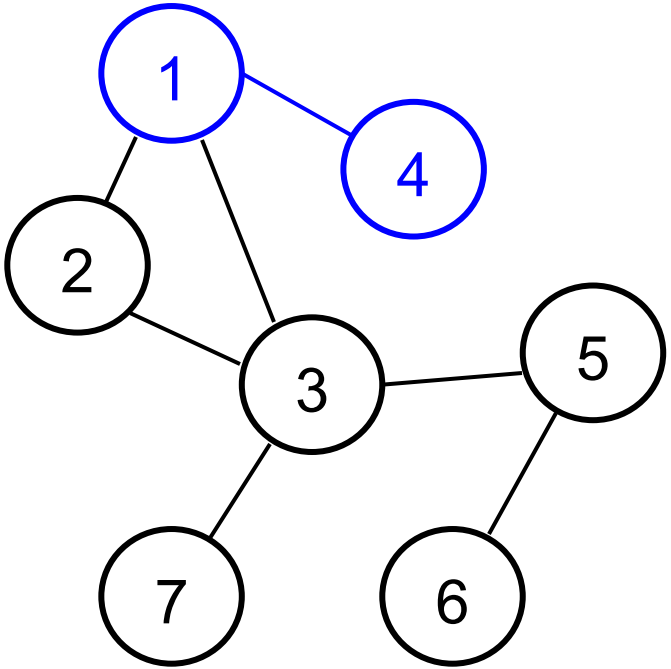


Dirigido



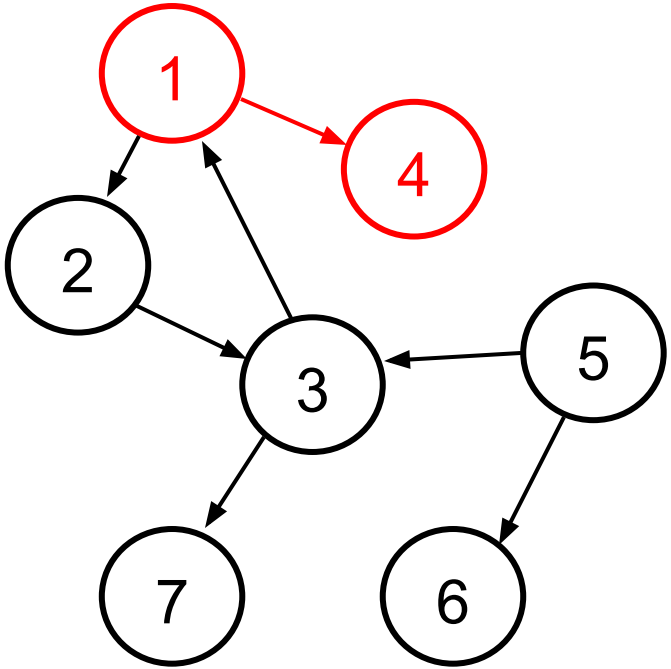
1 está conectado con 4
4 está conectado con 1

No dirigido



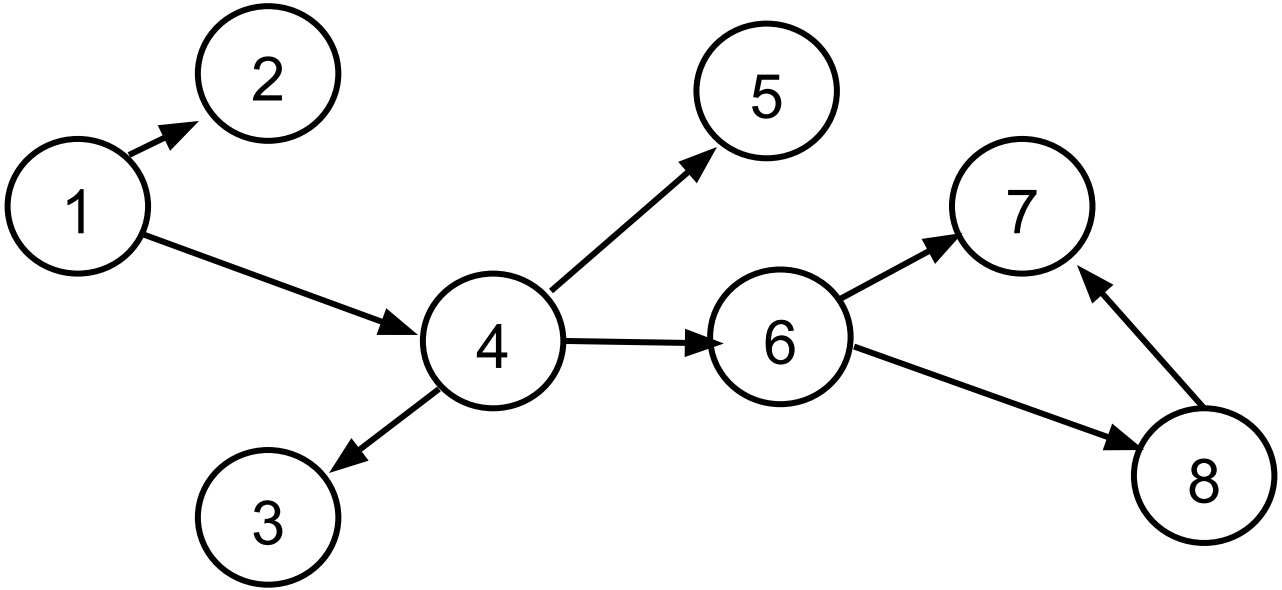
Solo 1 está conectado con 4

Dirigido



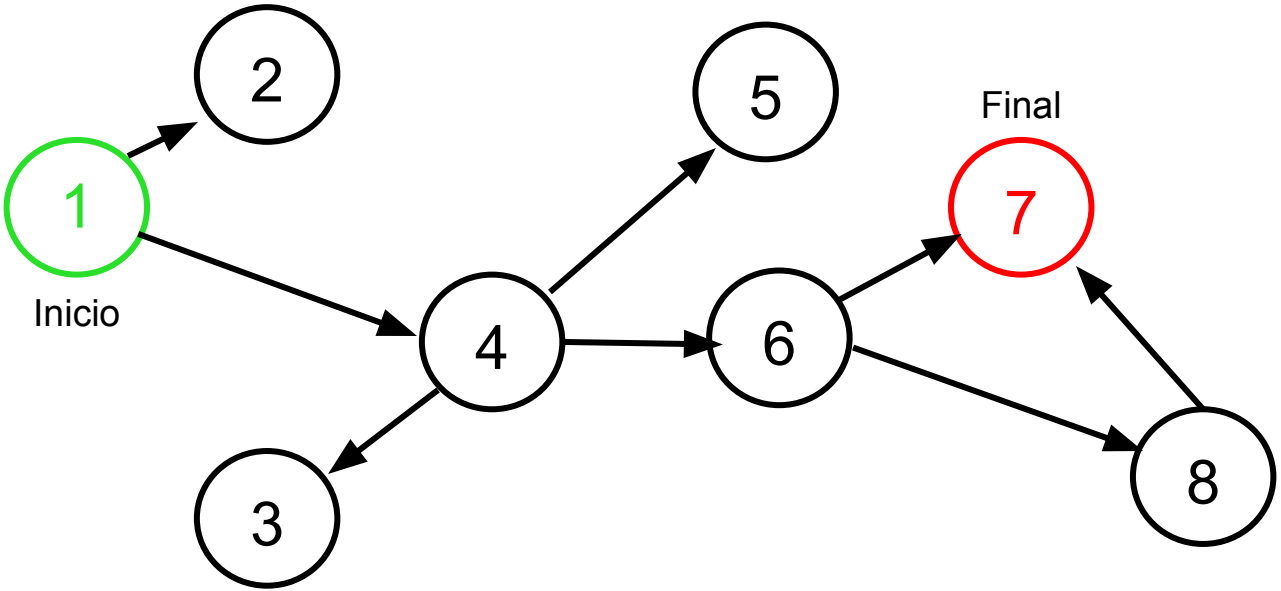
Camino

Secuencia de nodos unidos por aristas



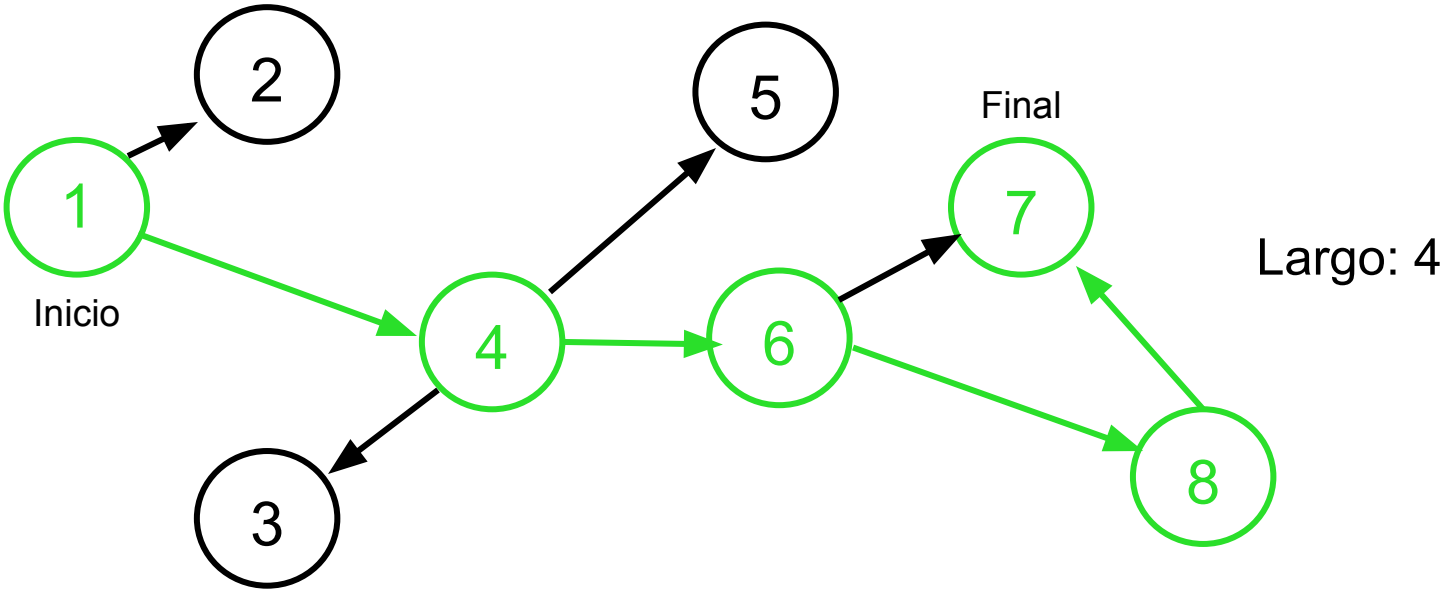
Camino

Secuencia de nodos distintos unidos por aristas



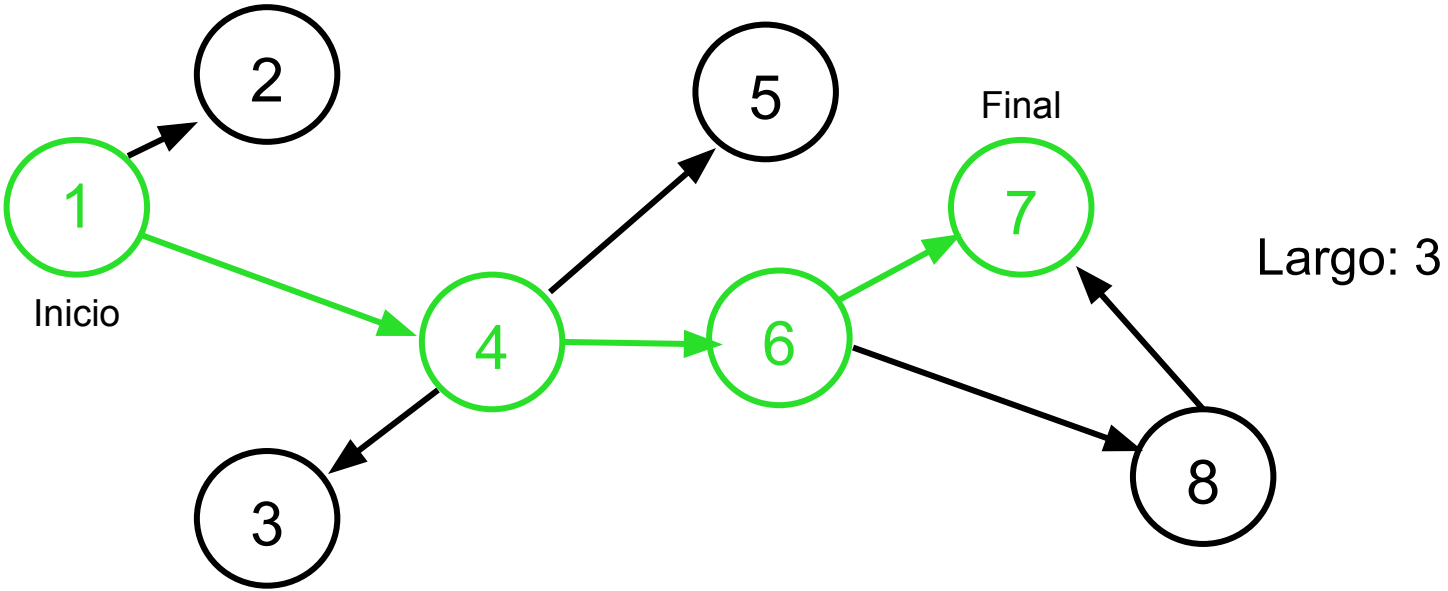
Camino

Secuencia de nodos distintos unidos por aristas



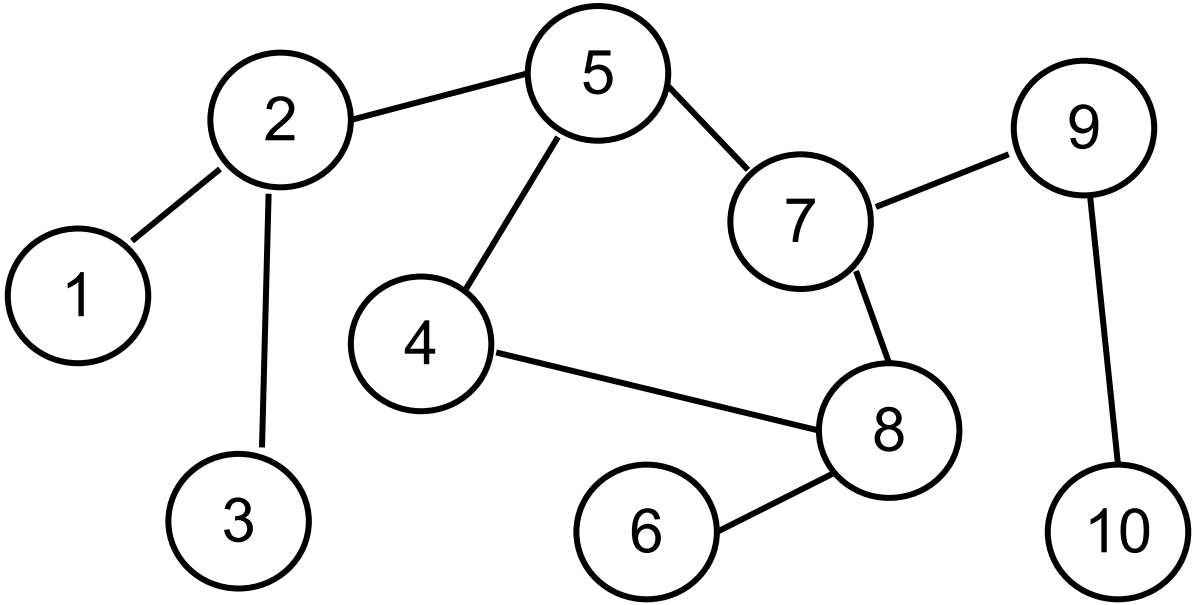
Camino

Secuencia de nodos distintos unidos por aristas



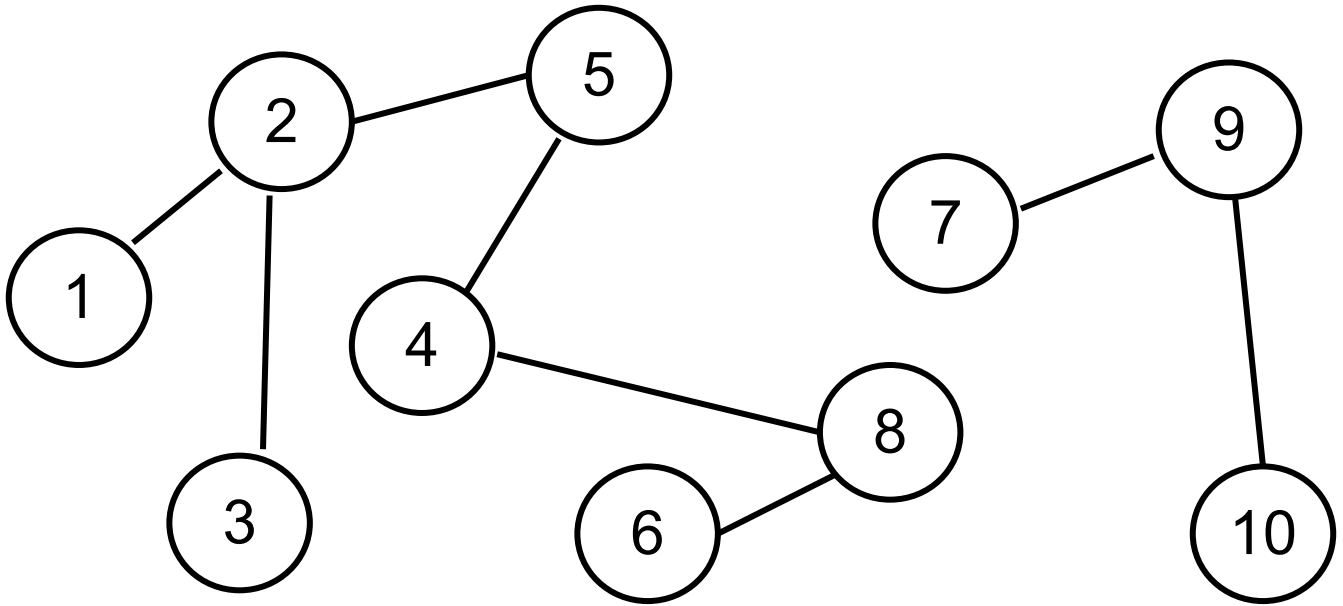
Grafo conexo

Decimos que un grafo **no dirigido** es conexo si existe un camino entre cualquier par de nodos

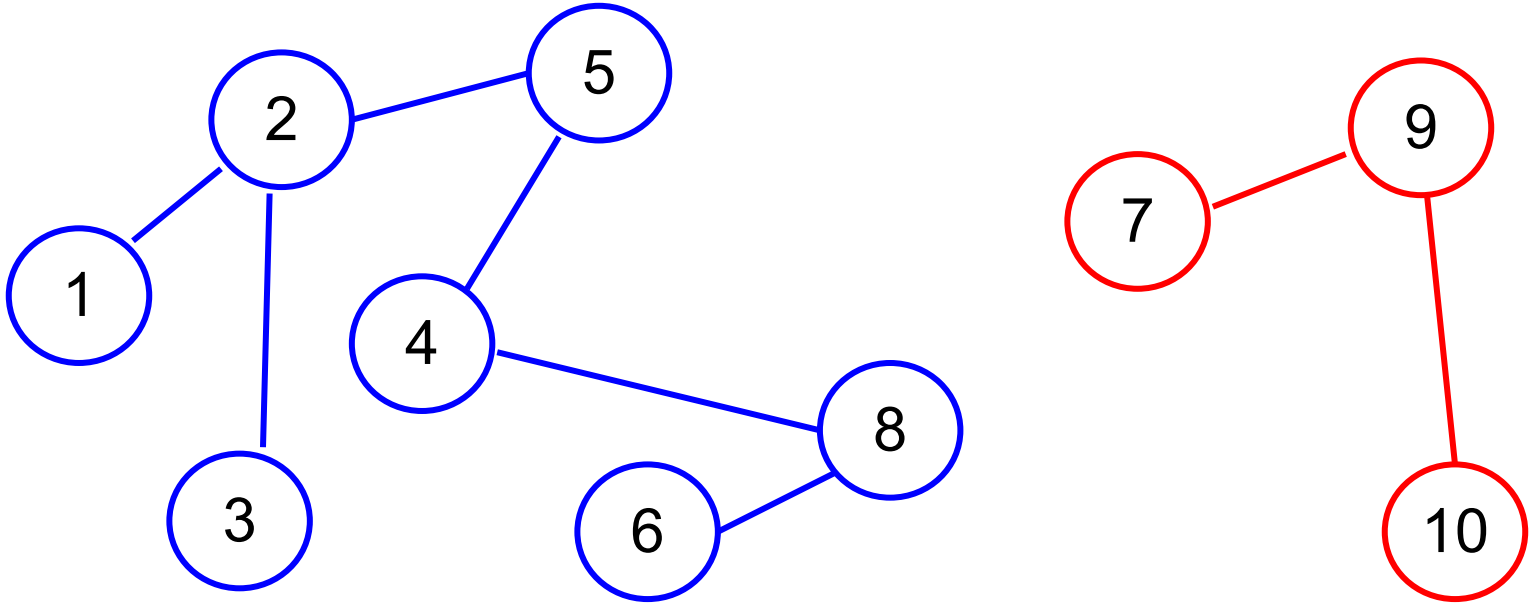


Grafo conexo

Decimos que un grafo **no dirigido** es conexo si existe un camino entre cualquier par de nodos

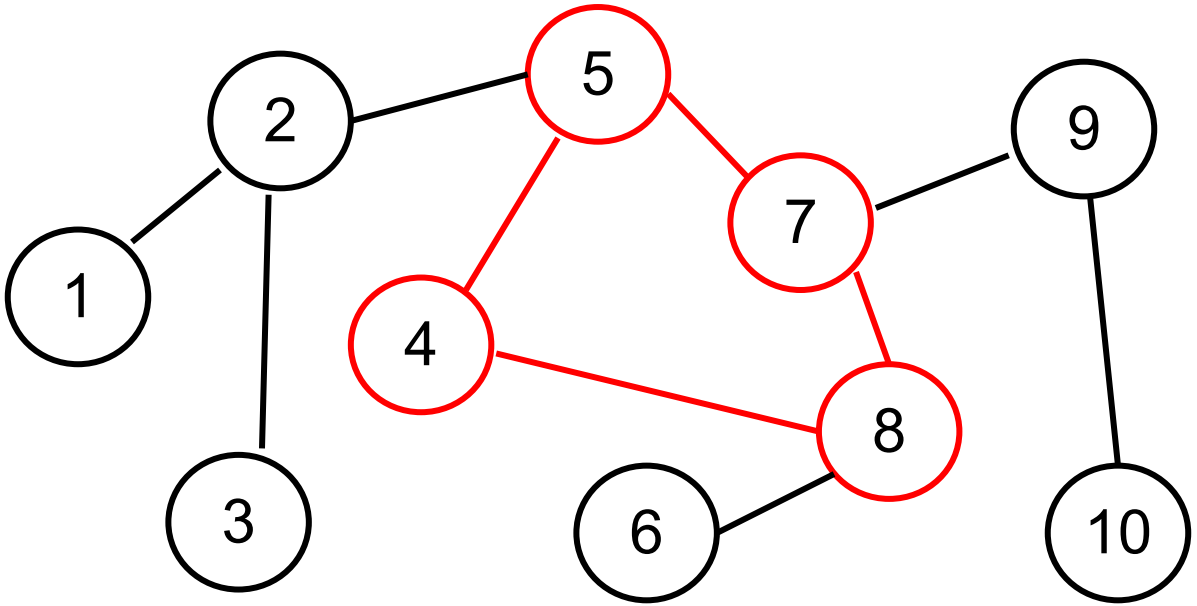


Componente conexa

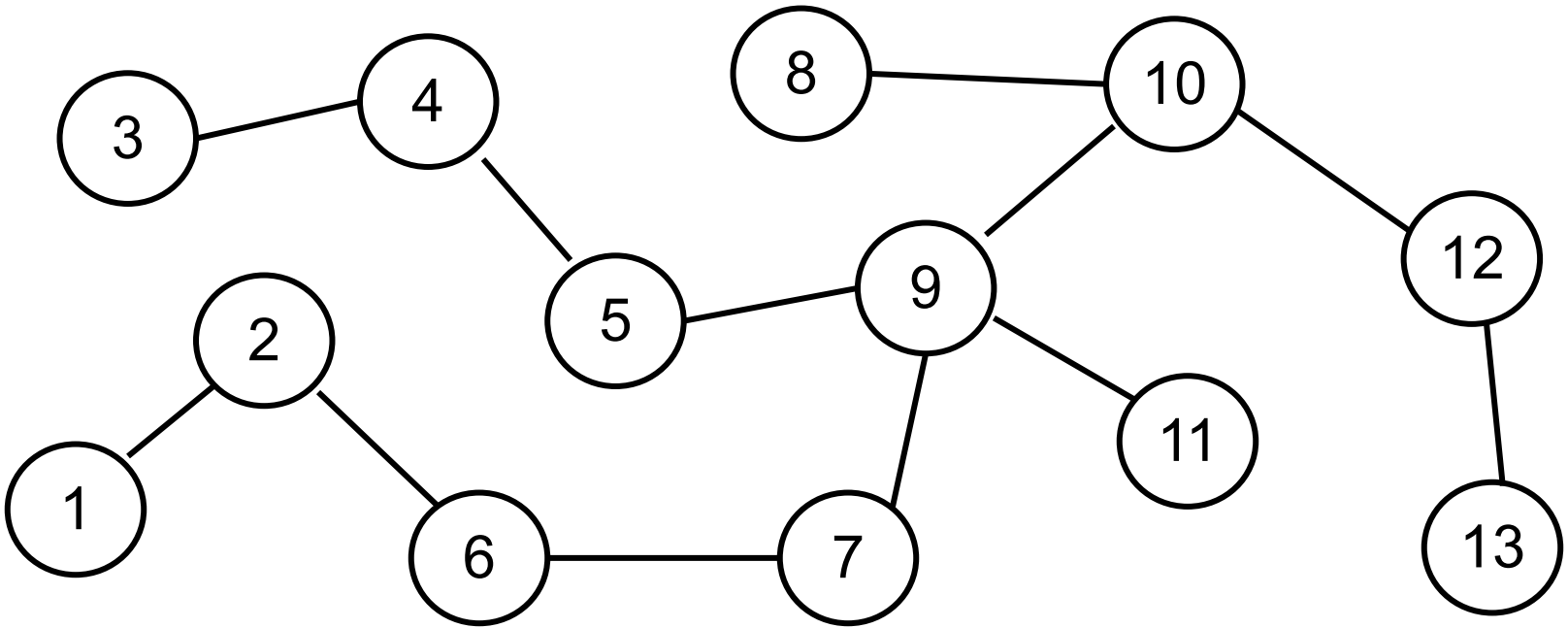


Ciclo

Camino que inicia y termina en el mismo nodo

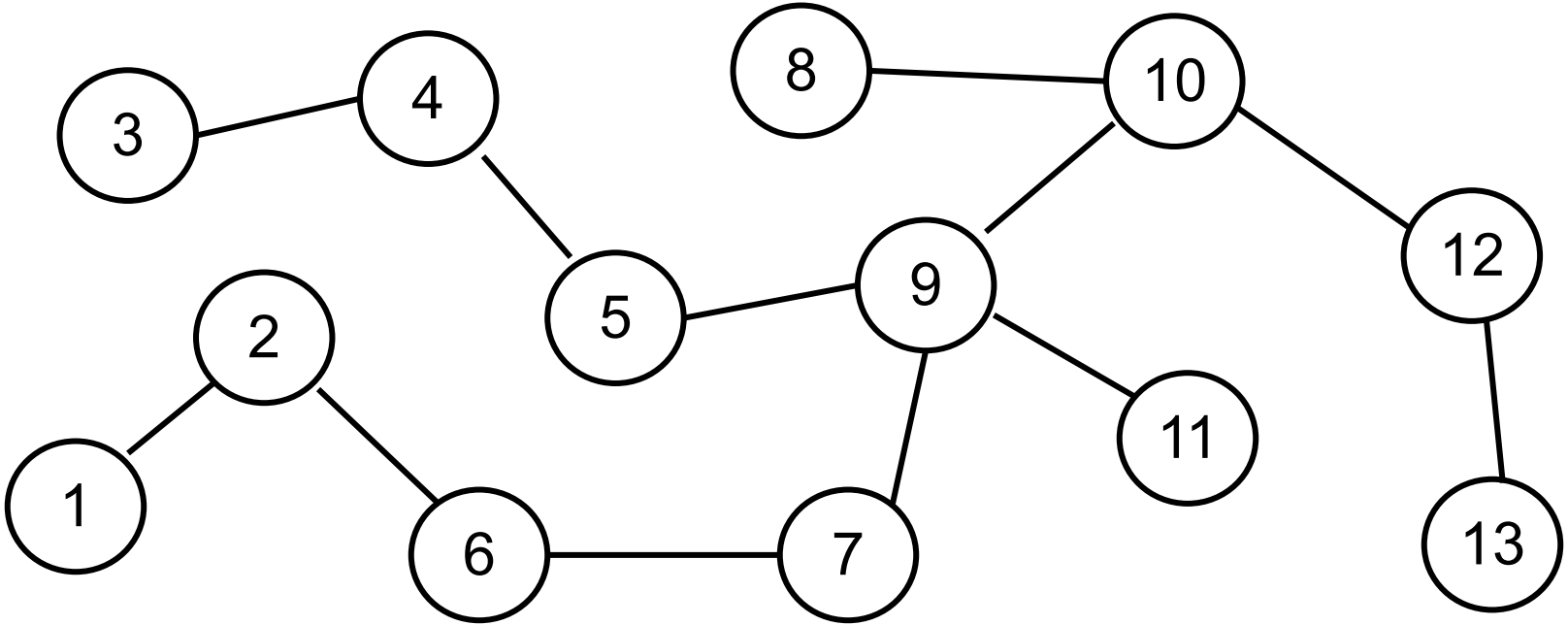


Árbol



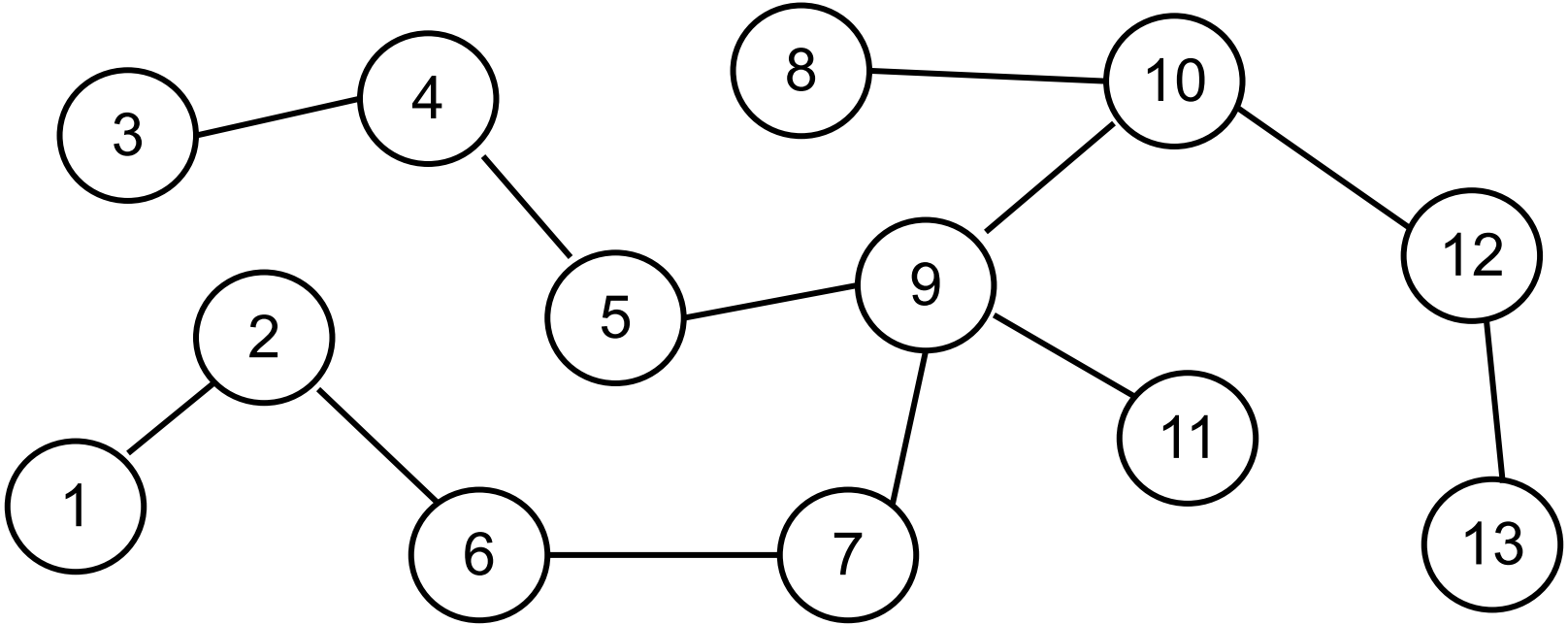
Árbol

No tiene ciclos



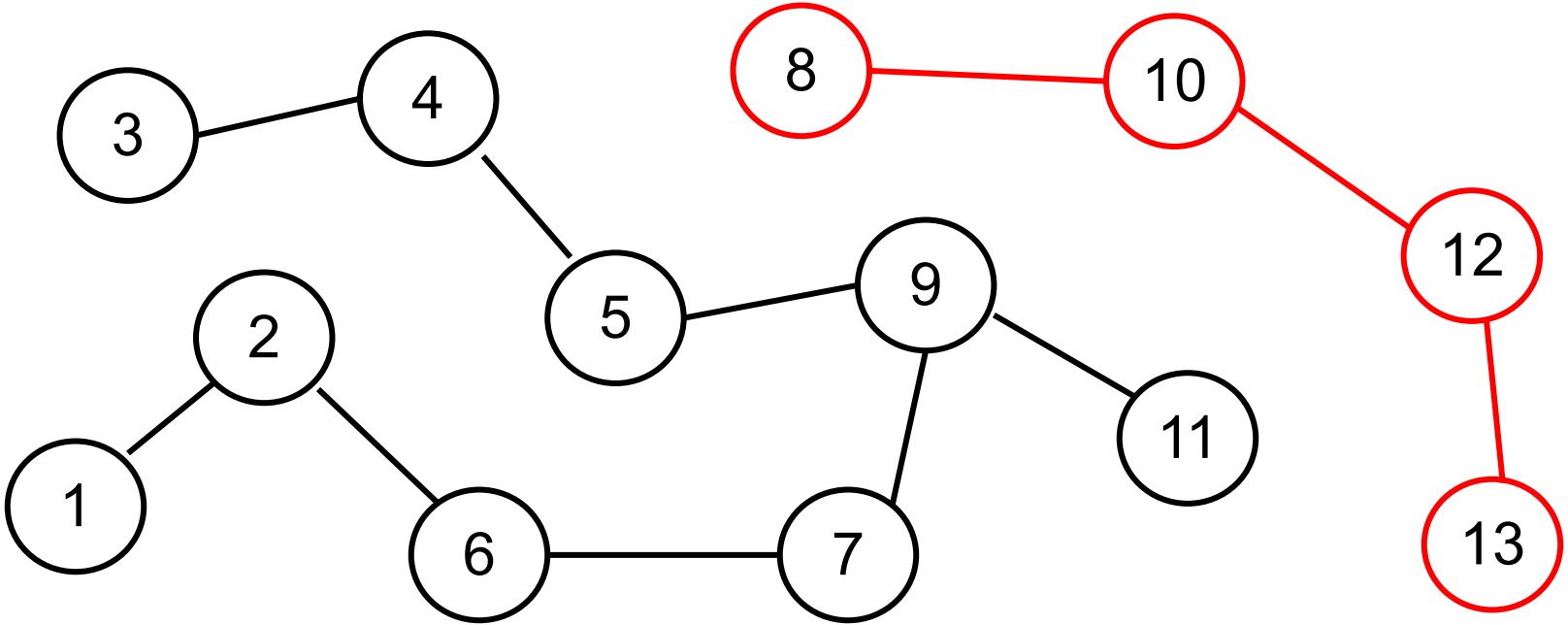
Árbol

Si se le **quita** una arista deja de ser conexo



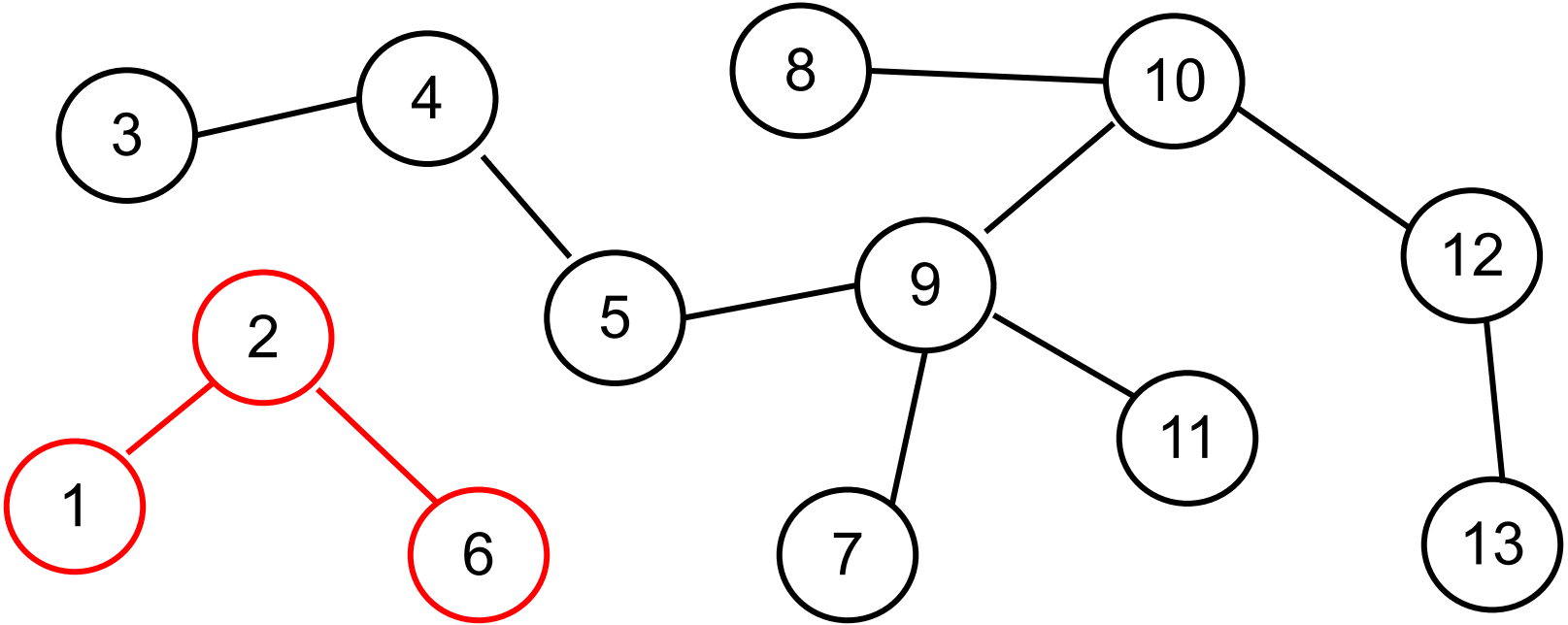
Árbol

Si se le **quita** una arista deja de ser conexo



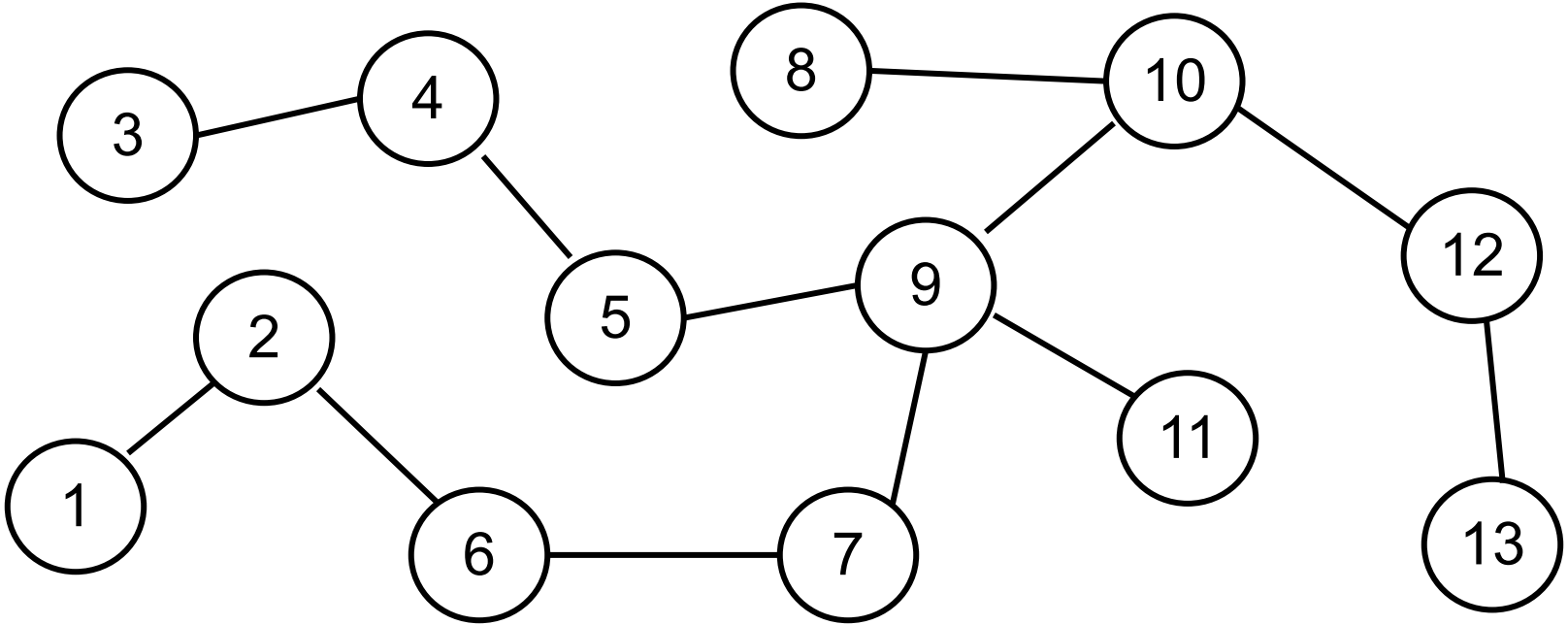
Árbol

Si se le **quita** una arista deja de ser conexo



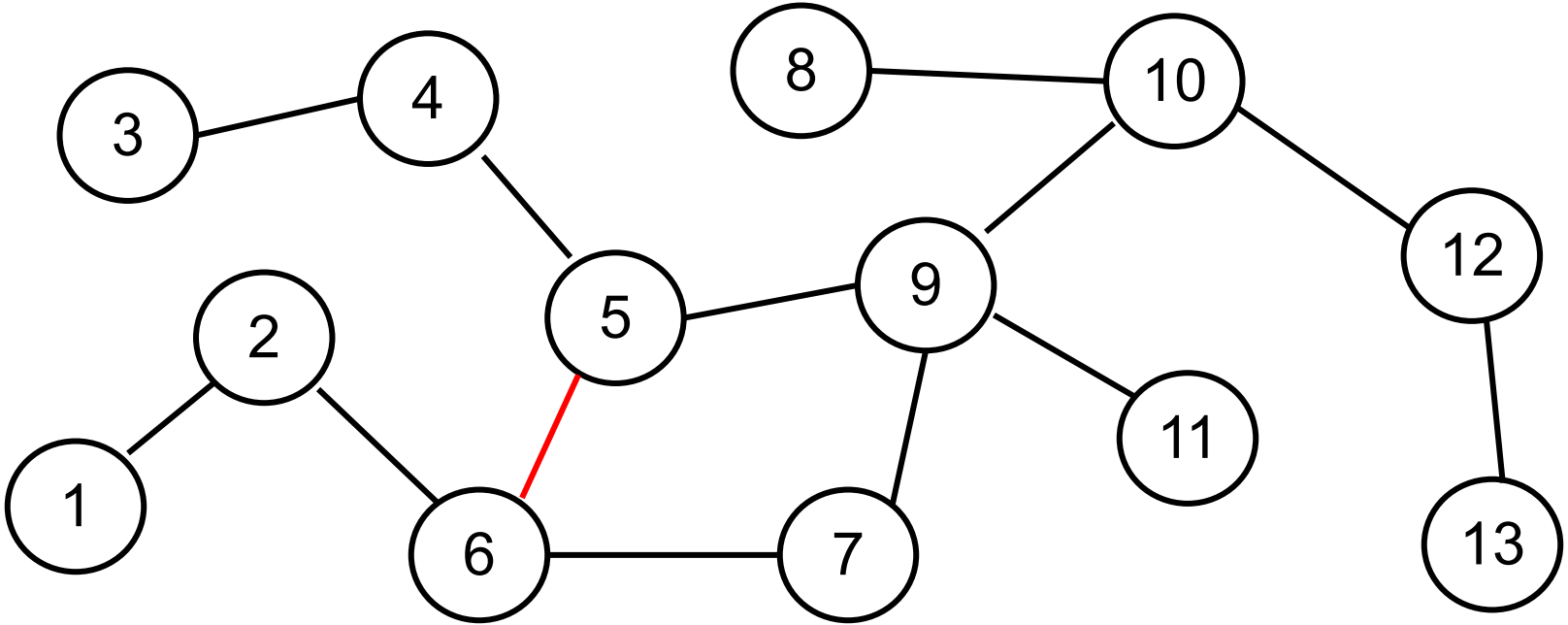
Árbol

Si se le **agrega** una arista se crean ciclos



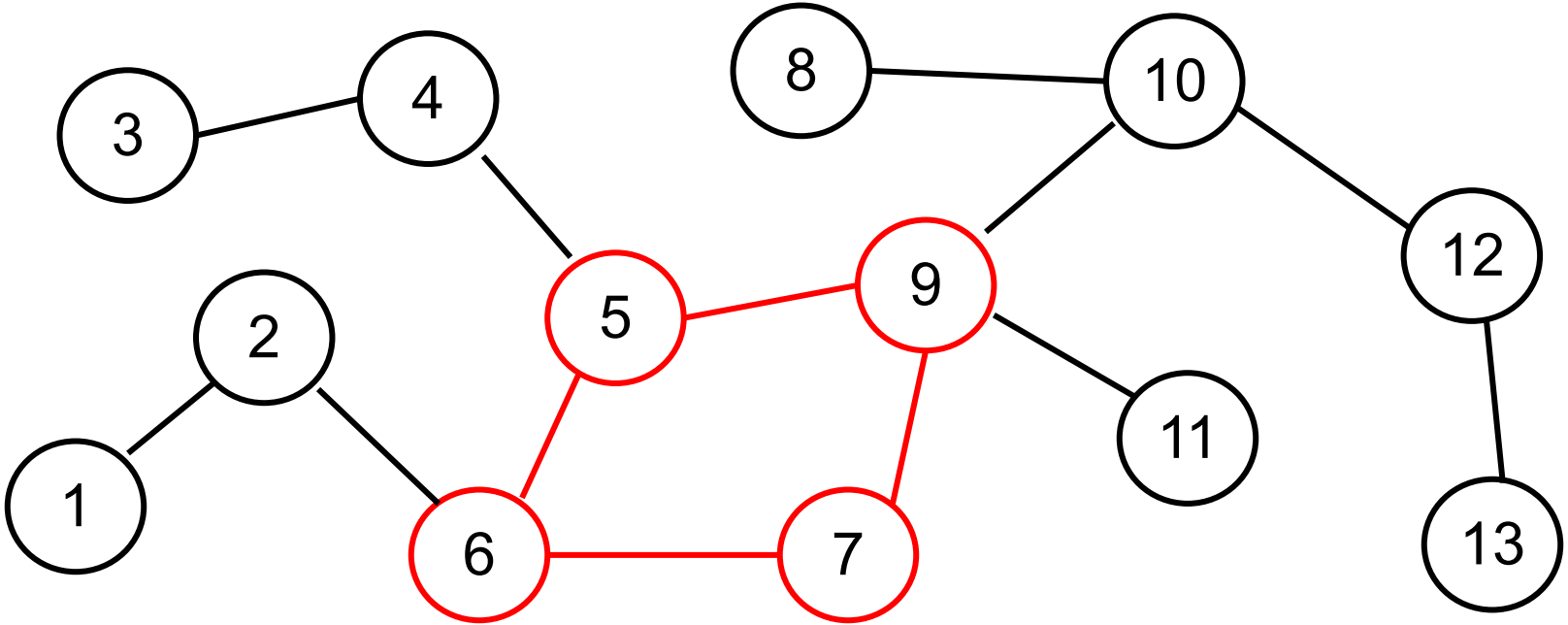
Árbol

Si se le **agrega** una arista se crean ciclos



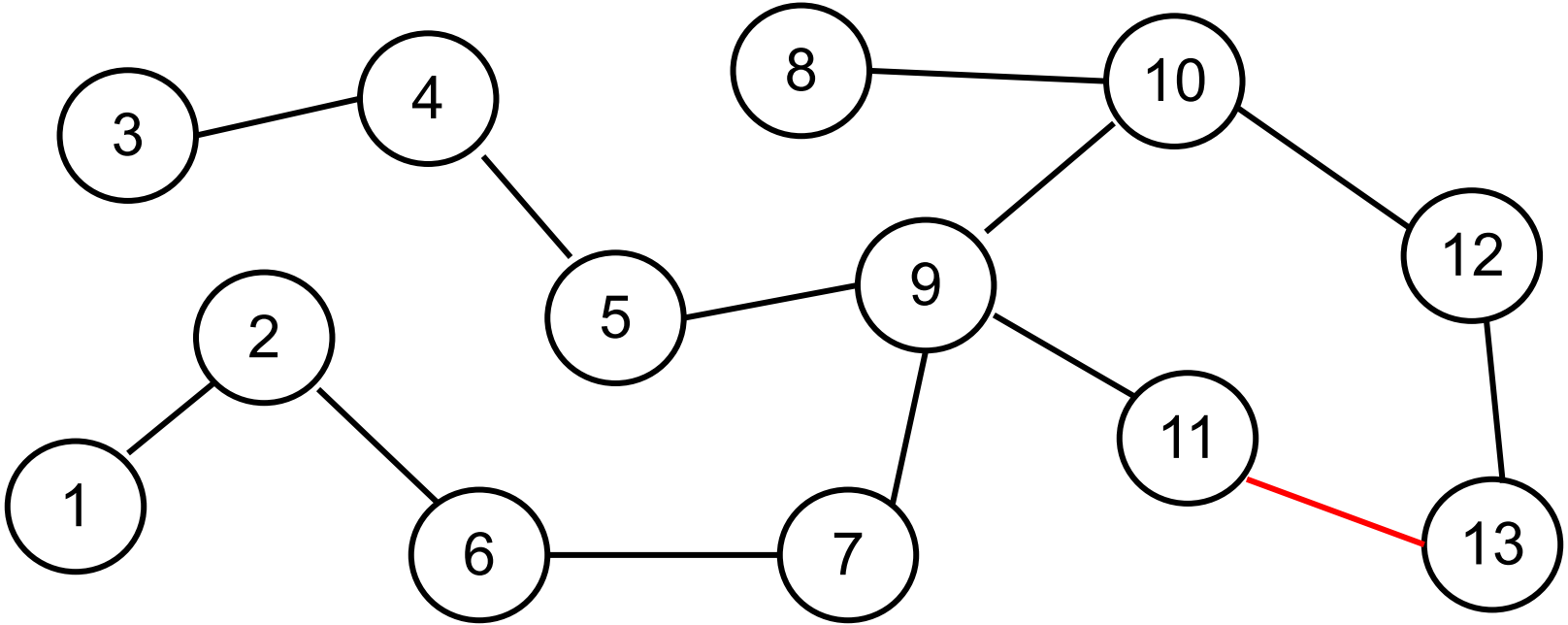
Árbol

Si se le **agrega** una arista se crean ciclos



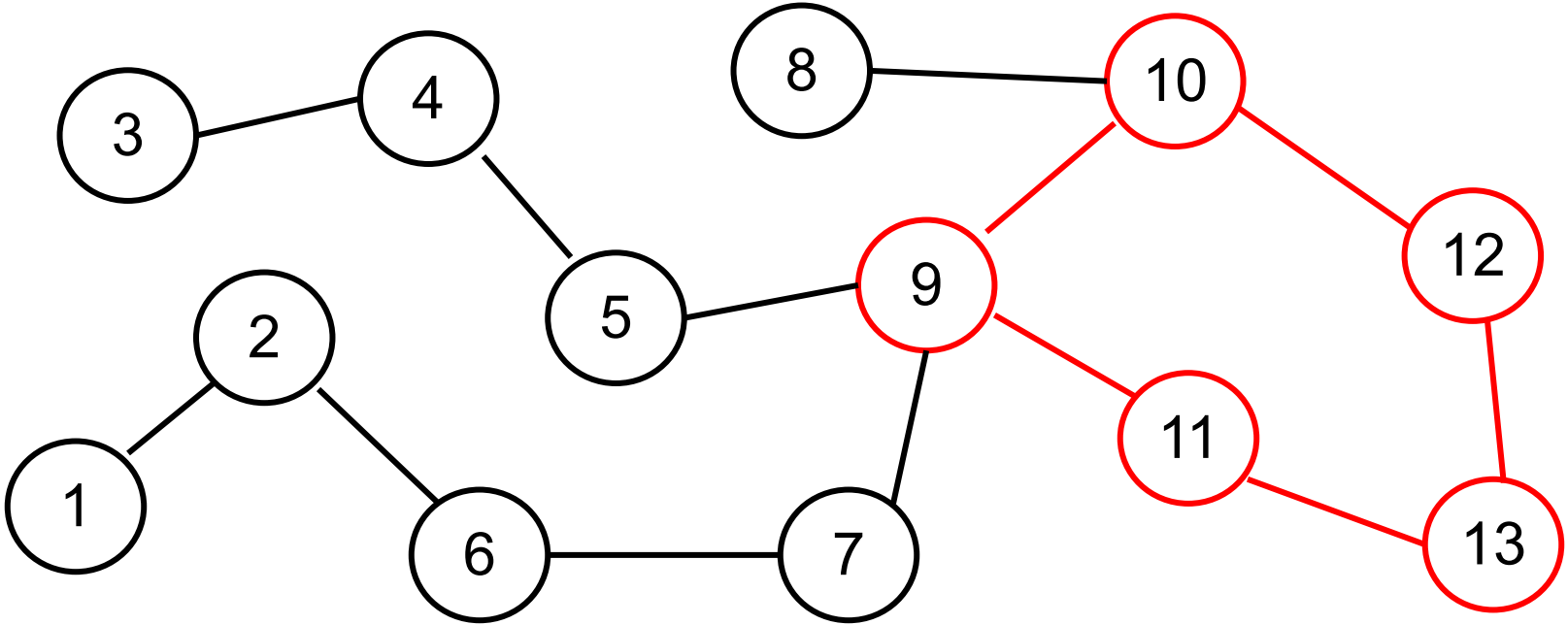
Árbol

Si se le **agrega** una arista se crean ciclos



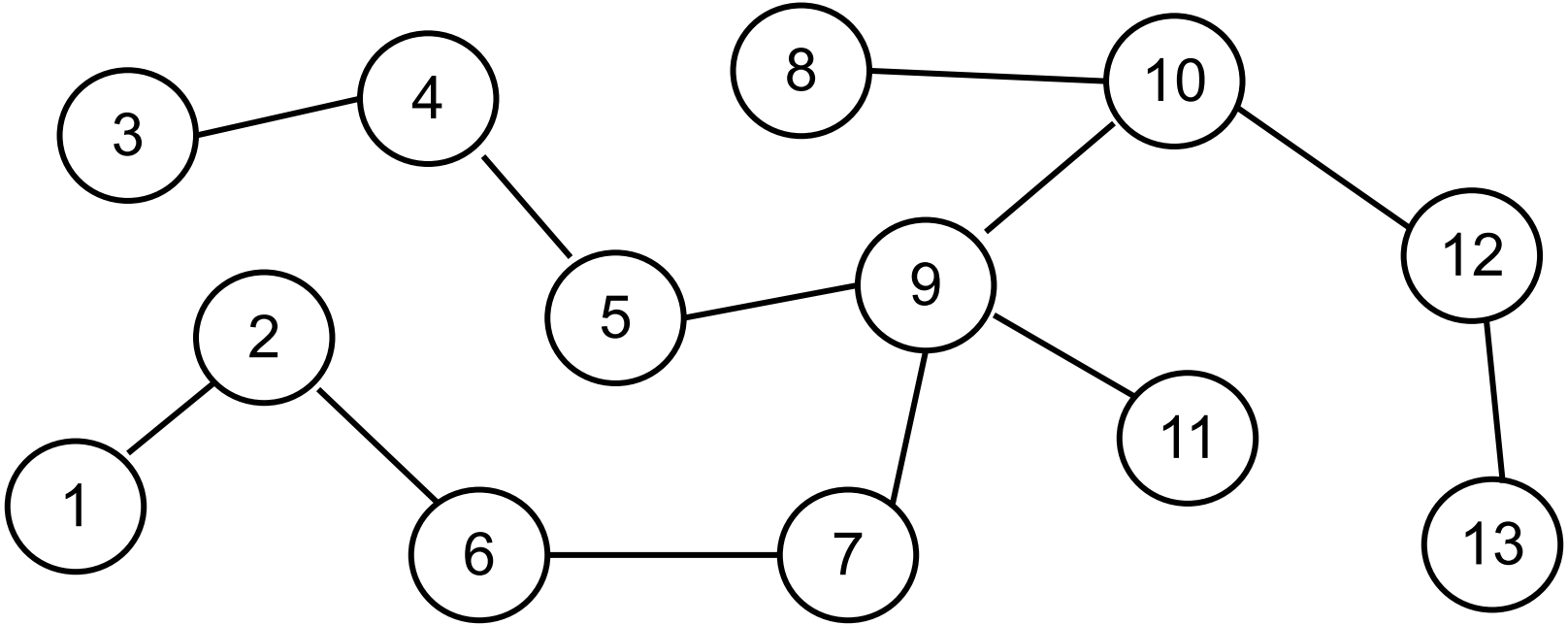
Árbol

Si se le **agrega** una arista se crean ciclos



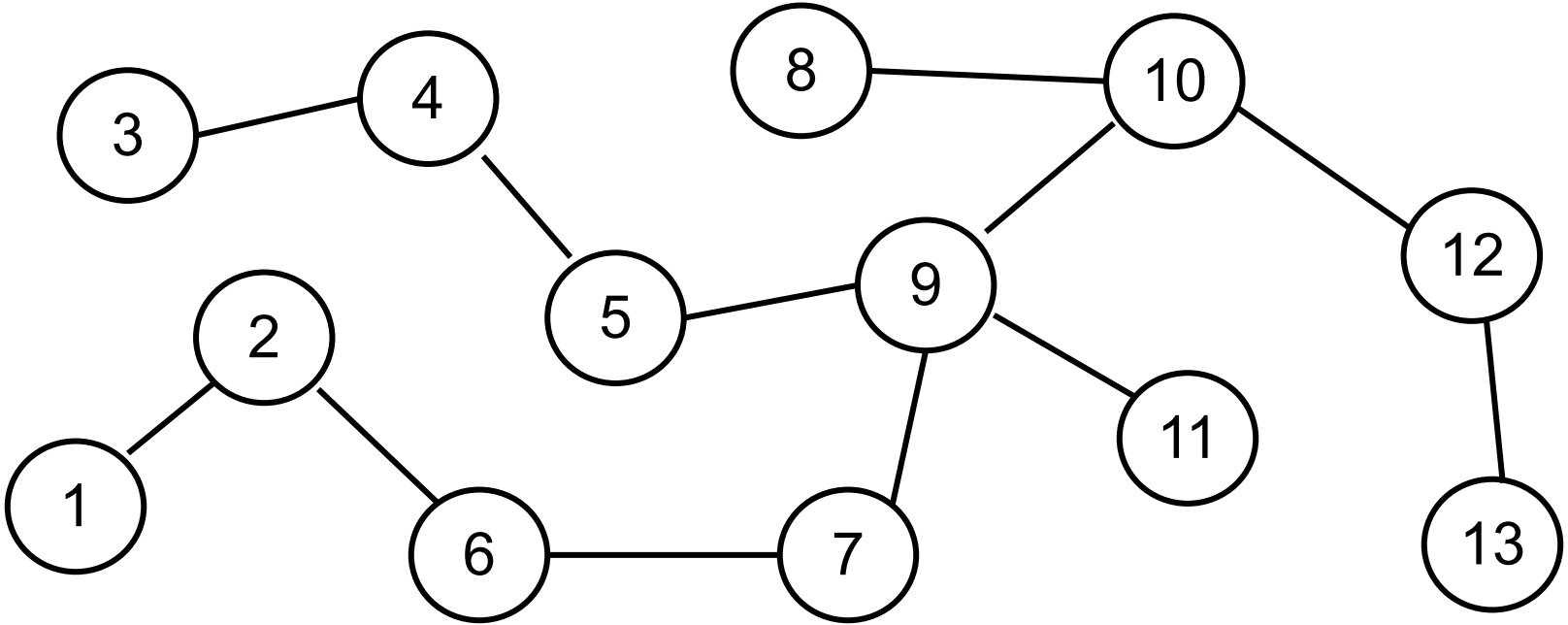
Árbol

Existe solo un camino entre cada par de nodos



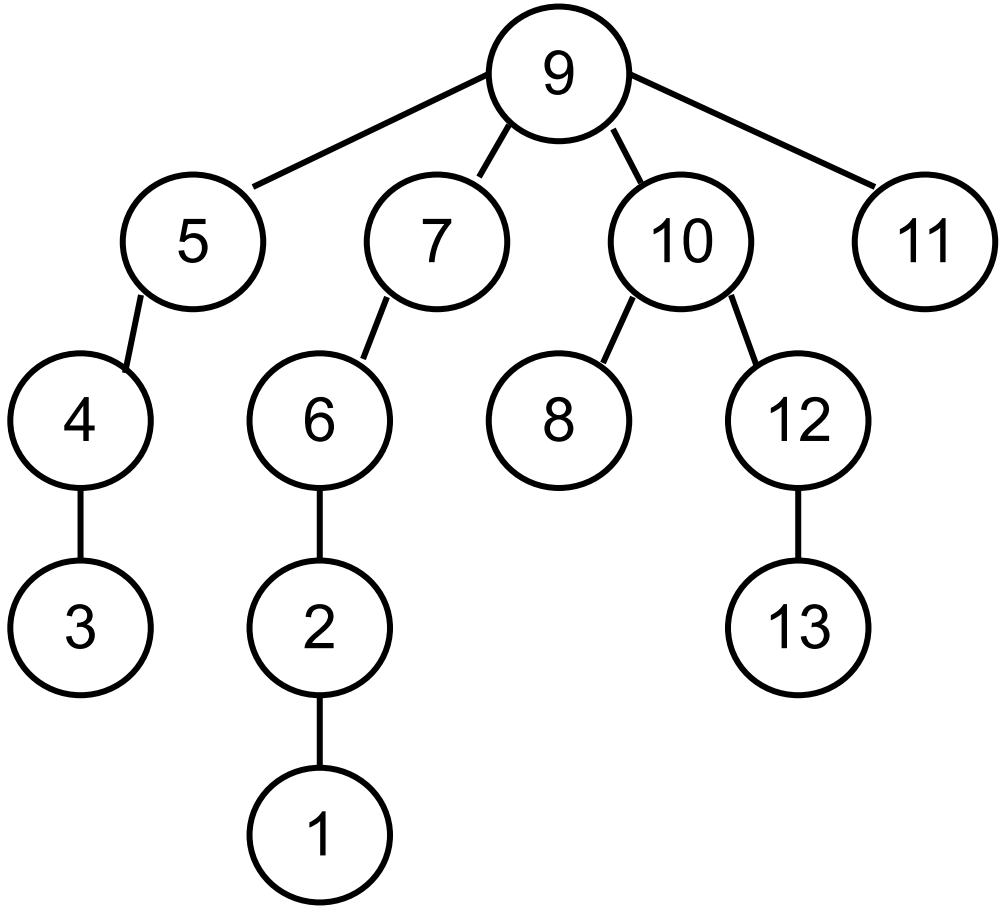
Árbol

Si el grafo tiene n nodos, tendrá n-1 aristas



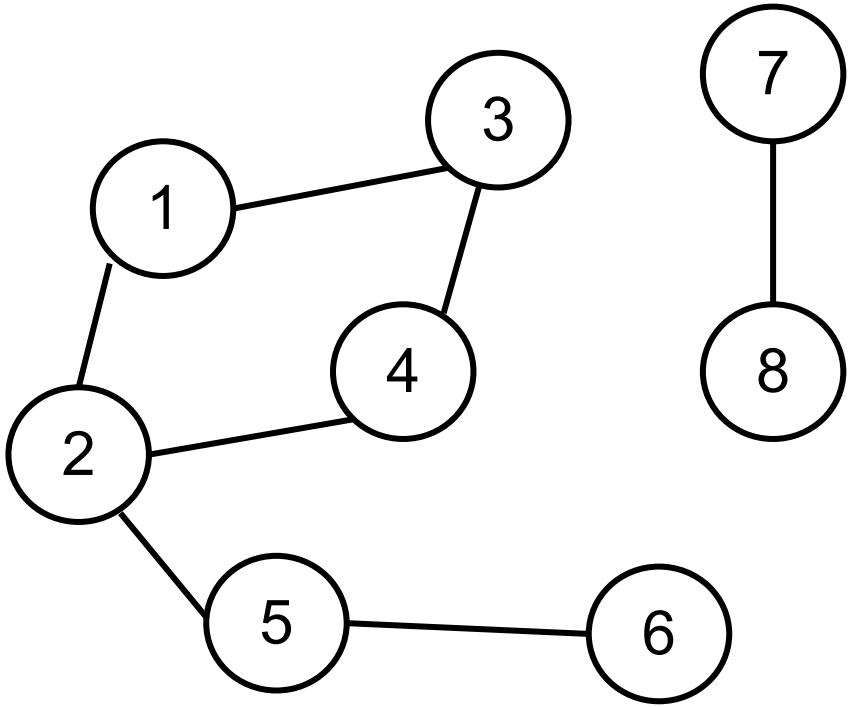
Árbol

- ➔ No tienen ciclos
- ➔ Si se le **quita** una arista deja de ser conexo
- ➔ Si se le **agrega** una arista se crean ciclos
- ➔ Existe solo un camino entre cada par de nodos
- ➔ Si el grafo tiene n nodos, tendrá n-1 aristas



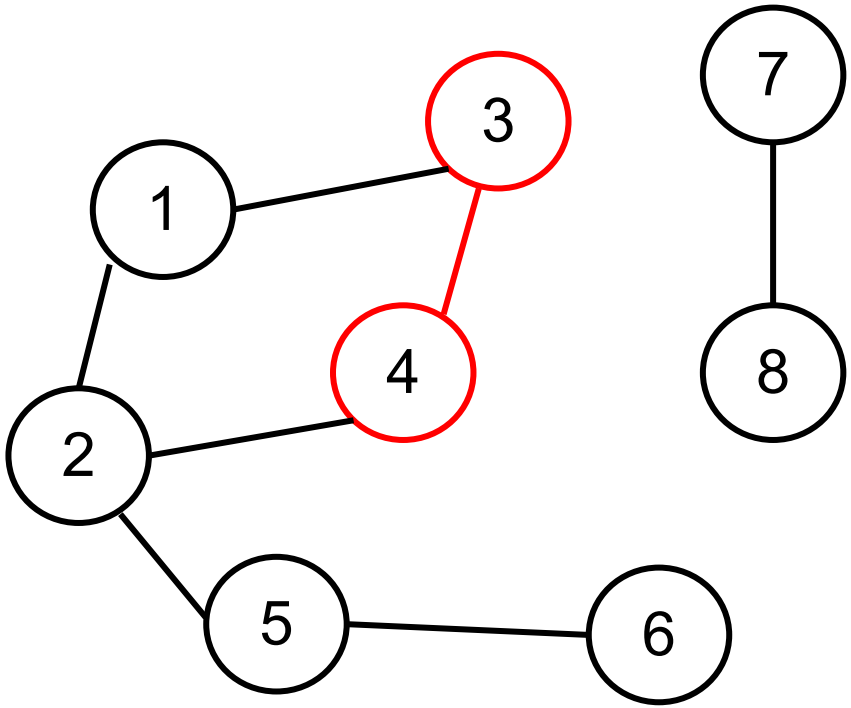
¿Cómo representamos un grafo en código?

Matriz de adyacencia



	1	**2**	**3**	**4**	**5**	**6**	**7**	**8**
1	0	1	1	0	0	0	0	0
2	1	0	0	1	1	0	0	0
3	1	0	0	1	0	0	0	0
4	0	1	1	0	0	0	0	0
5	0	1	0	0	0	1	0	0
6	0	0	0	0	1	0	0	0
7	0	0	0	0	0	0	0	1
8	0	0	0	0	0	0	1	0

Matriz de adyacencia



	1	2	3	4	5	6	7	8
1	0	1	1	0	0	0	0	0
2	1	0	0	1	1	0	0	0
3	1	0	0	**1**	0	0	0	0
4	0	1	**1**	0	0	0	0	0
5	0	1	0	0	0	1	0	0
6	0	0	0	0	1	0	0	0
7	0	0	0	0	0	0	0	1
8	0	0	0	0	0	0	1	0

Matriz de adyacencia

Pros:

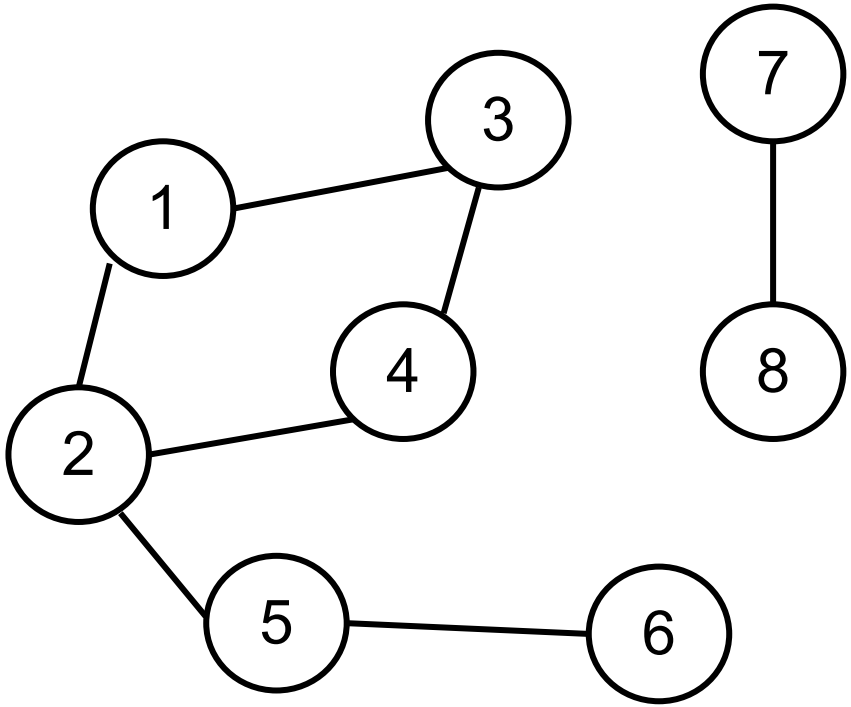
- Saber si dos nodos están conectados en $O(1)$
- Agregar y eliminar aristas en $O(1)$

Contras:

- Listar vecinos en $O(n)$
- Usa mucho espacio $O(n^2)$

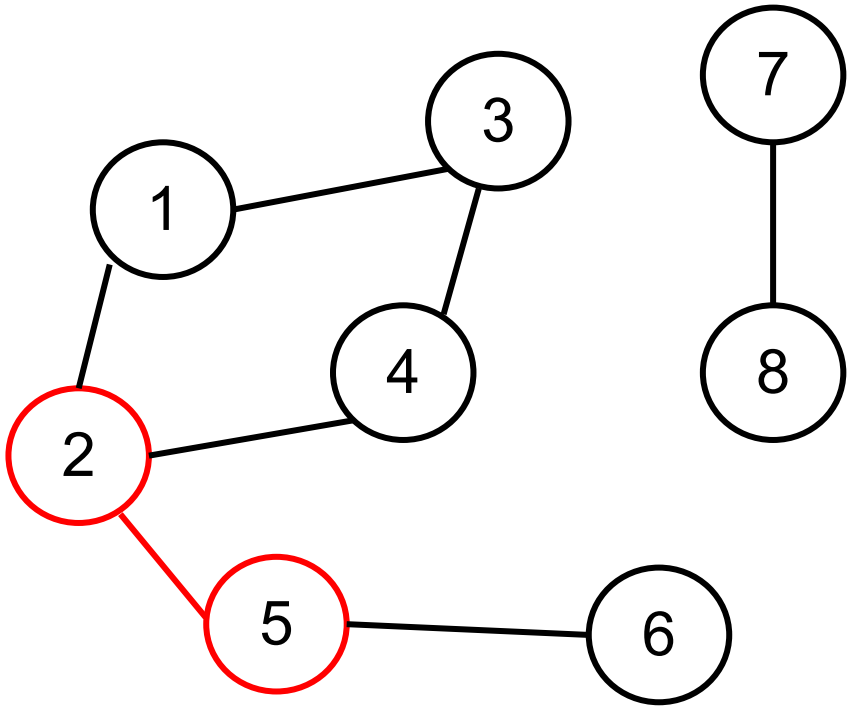
	1	**2**	**3**	**4**	**5**	**6**	**7**	**8**
1	0	1	1	0	0	0	0	0
2	1	0	0	1	1	0	0	0
3	1	0	0	1	0	0	0	0
4	0	1	1	0	0	0	0	0
5	0	1	0	0	0	1	0	0
6	0	0	0	0	1	0	0	0
7	0	0	0	0	0	0	0	1
8	0	0	0	0	0	0	1	0

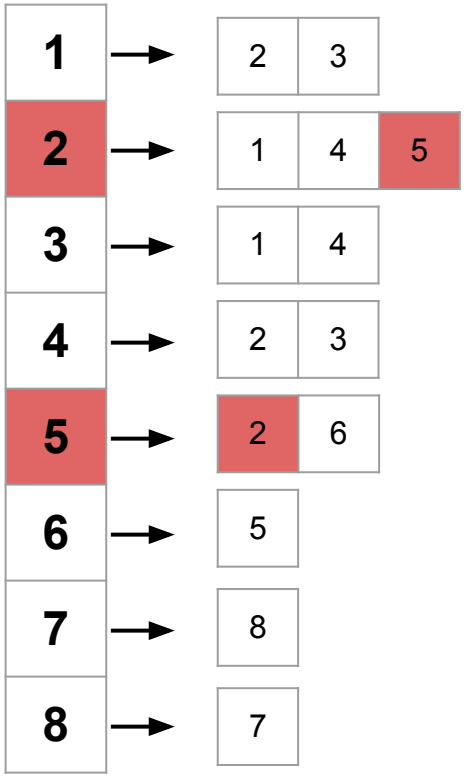
Lista de adyacencia



1	2, 3
2	1, 4, 5
3	1, 4
4	2, 3
5	2, 6
6	5
7	8
8	7

Lista de adyacencia





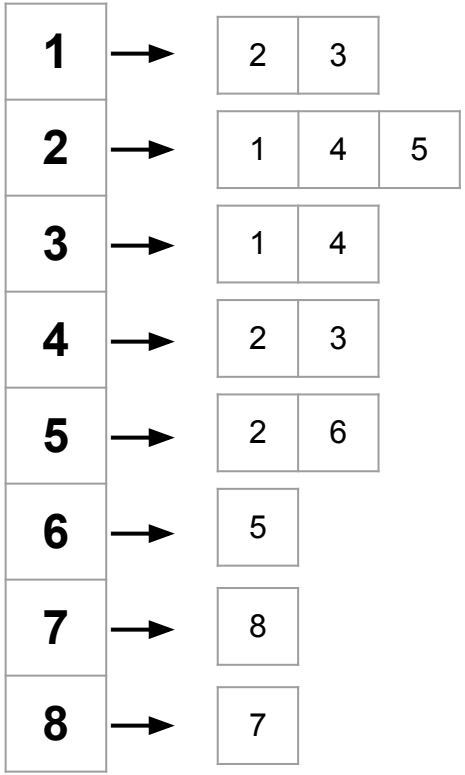
Lista de adyacencia

Pros:

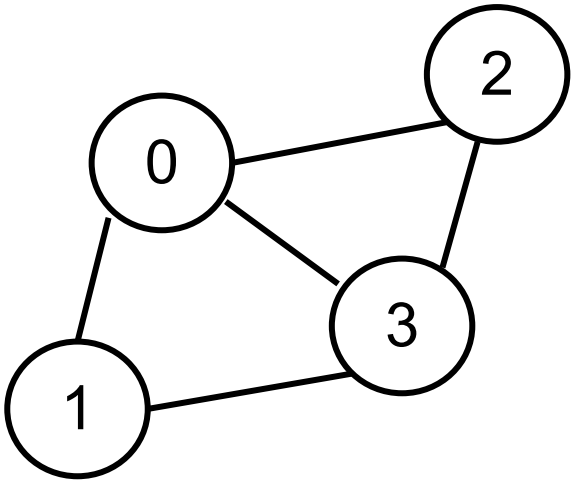
- Podemos listar a los vecinos de un nodo en O(vecinos)
- El espacio que utiliza en la memoria es O(aristas)

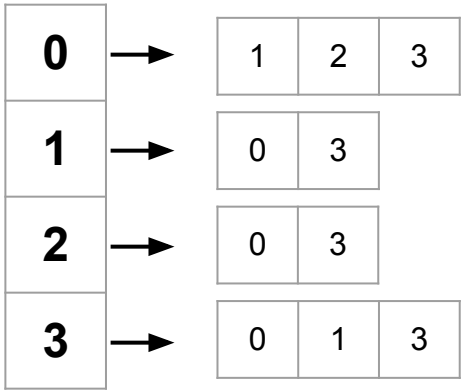
Contras:

- Saber si dos nodos están conectados es O(vecinos)
- Lo mismo pasa al agregar o eliminar aristas



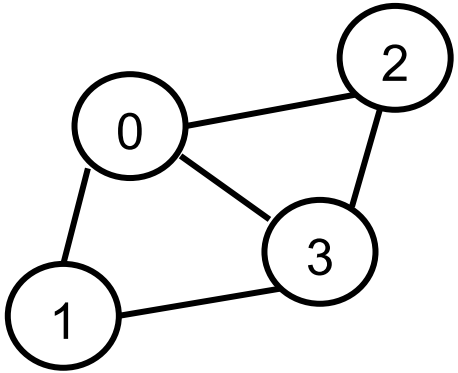
Lista de adyacencia - Código

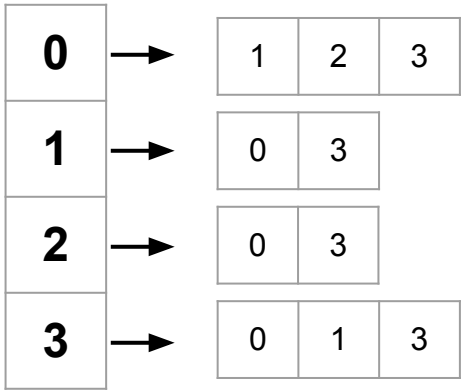




Lista de adyacencia - Código

```cpp
#include <bits/stdc++.h>
using namespace std;

int main(){

    int n = 4;
    vector <vector <int> > grafo(n);

    grafo[0].push_back(1);
    grafo[0].push_back(2);
    grafo[0].push_back(3);

    grafo[1].push_back(0);
    grafo[1].push_back(3);

    grafo[2].push_back(0);
    grafo[2].push_back(3);

    grafo[3].push_back(0);
    grafo[3].push_back(1);
    grafo[3].push_back(2);


    return 0;
}
```



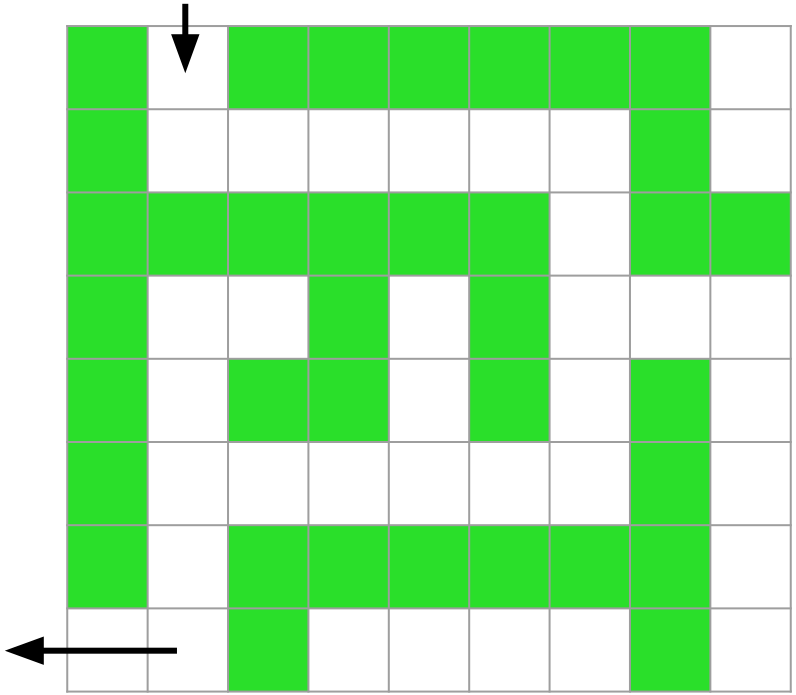


Grafo Implícito

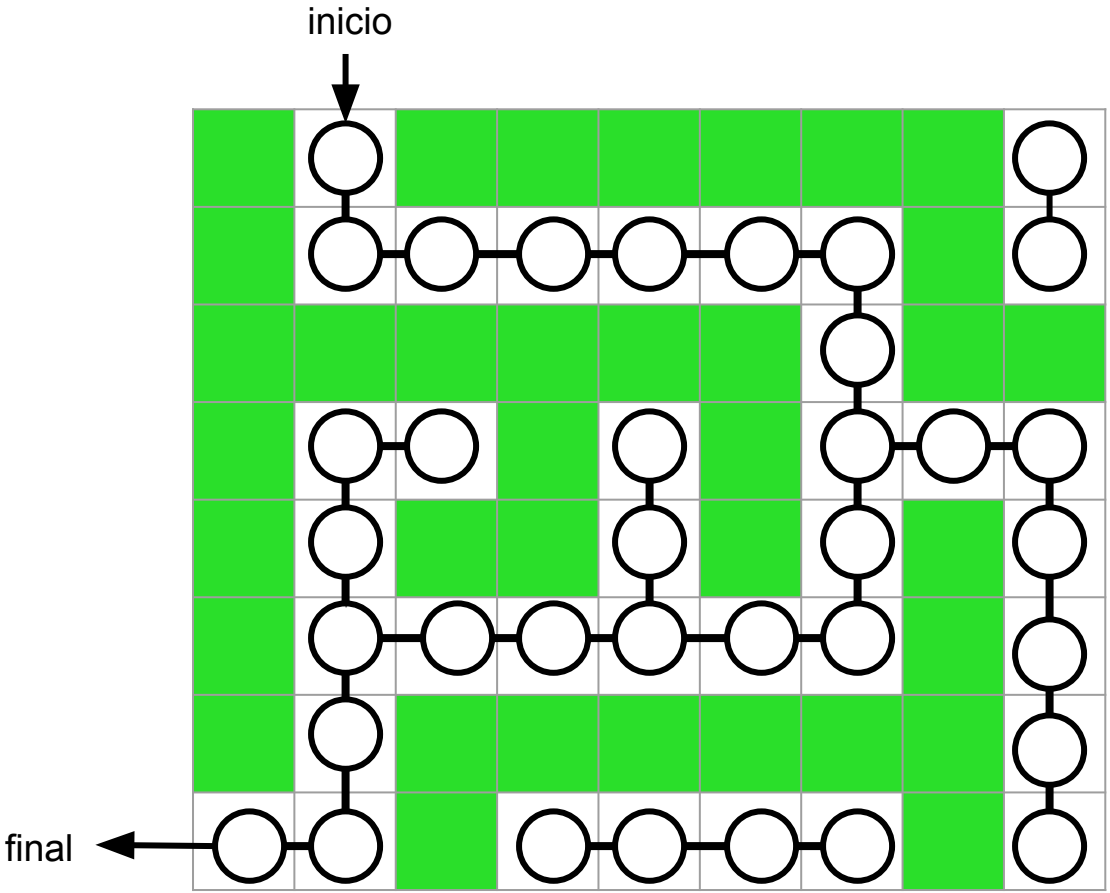
Podemos tener una situación donde no se nos entreguen los nodos y sus aristas como tal, y en cambio tenemos que **interpretar** el problema como un grafo

Grafo Implícito

Podemos tener una situación donde no se nos entreguen los nodos y sus aristas como tal, y en cambio tenemos que **interpretar** el problema como un grafo



Cada celda es un nodo con aristas hacia las celdas contiguas



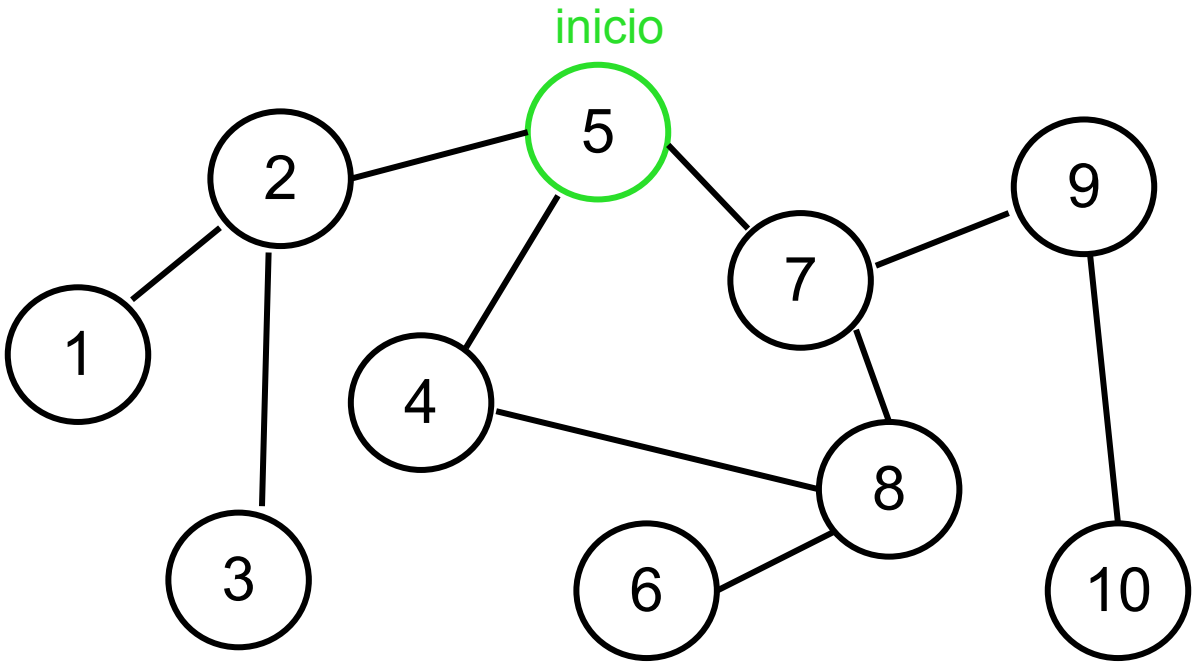
¿El laberinto tiene salida?

¿Hay un camino entre el nodo inicial y final?

¿Cómo recorremos un grafo?

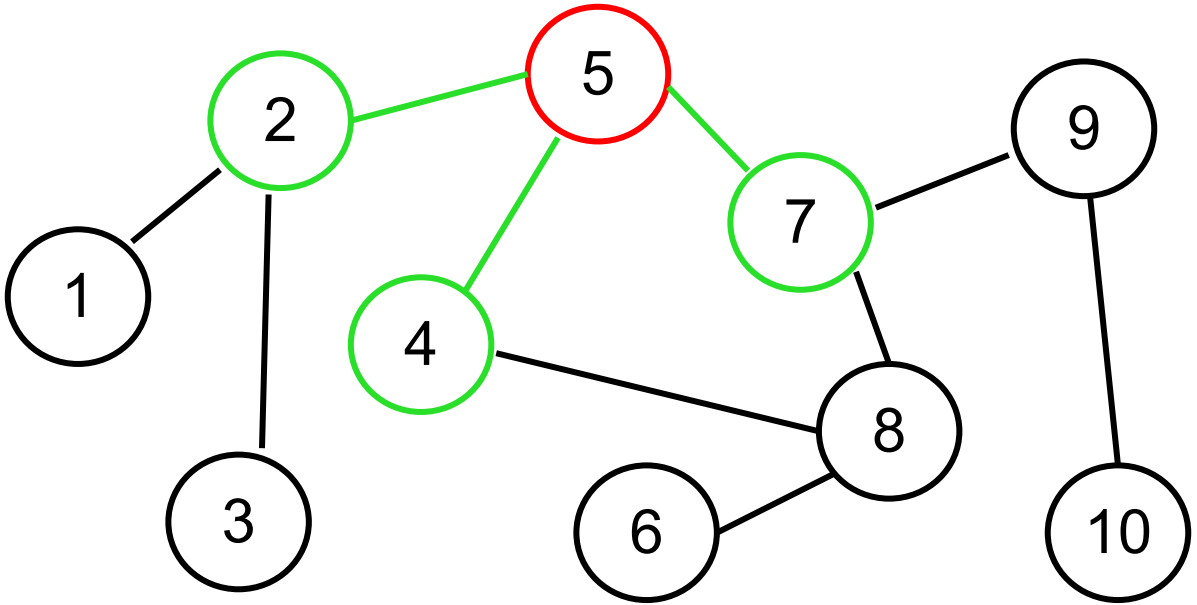
Breadth first search (BFS)

BFS



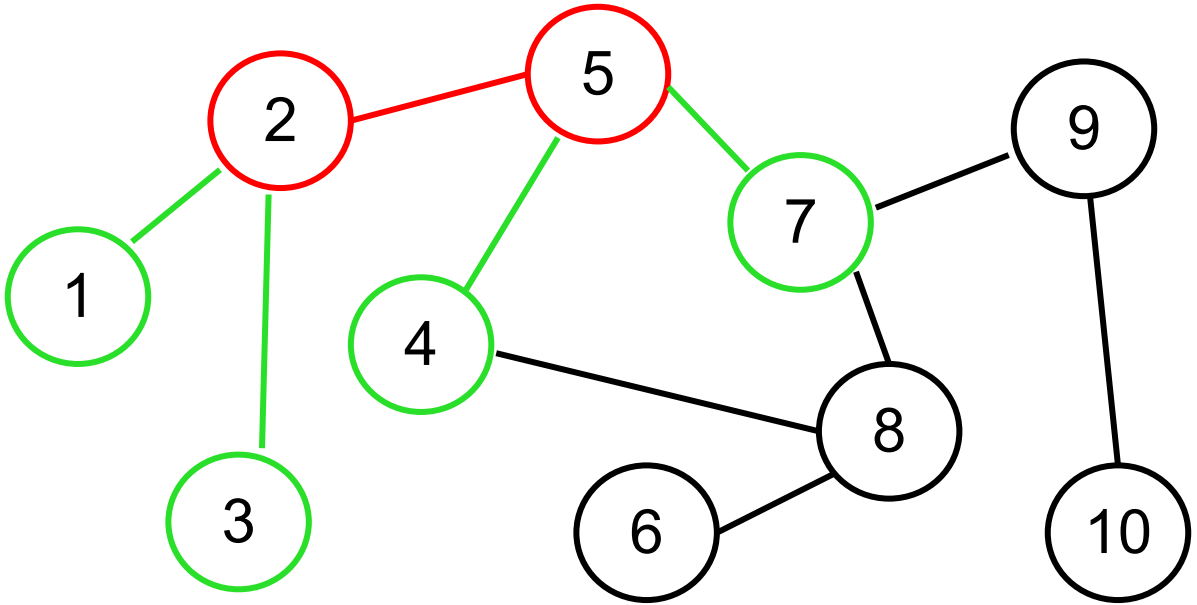
5

BFS



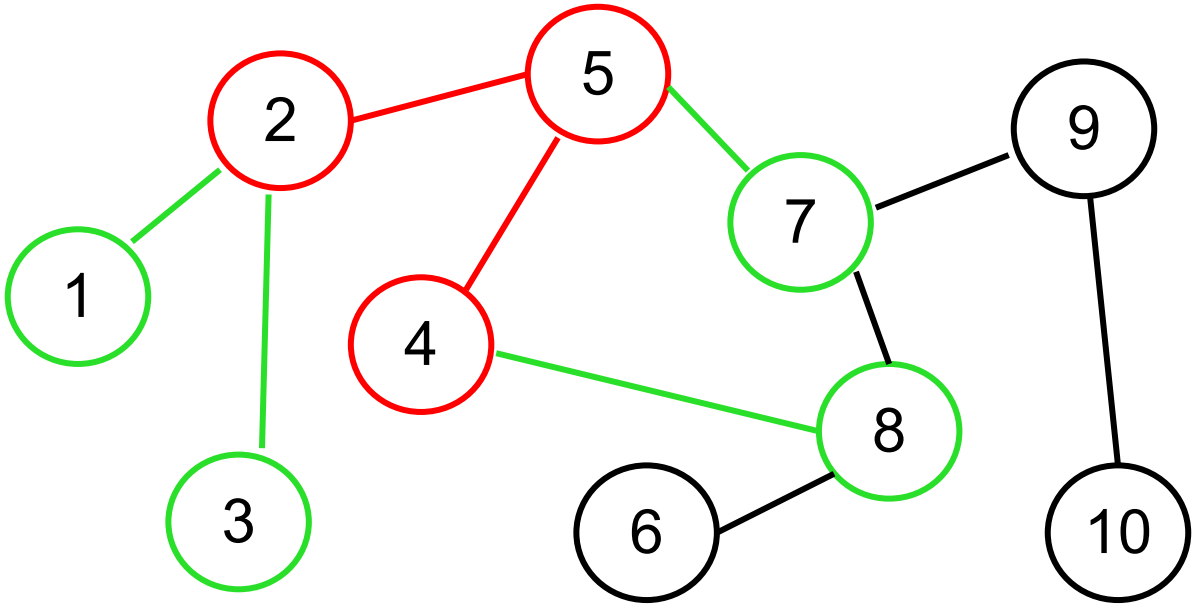
2	4	7

BFS



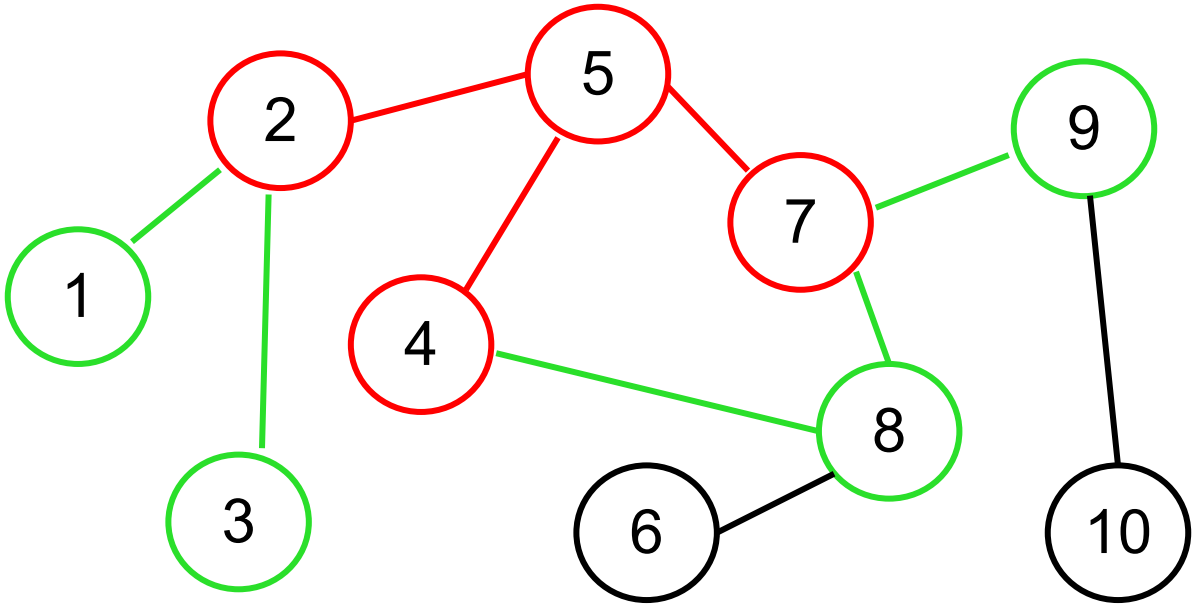
4	7	1	3

BFS



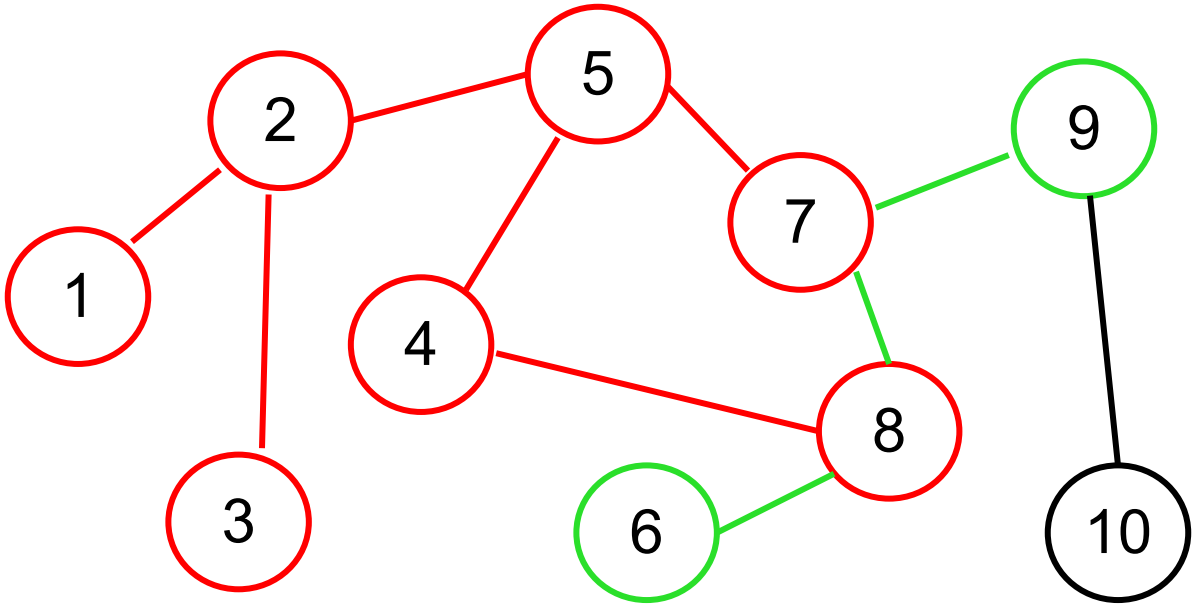
7	1	3	8

BFS



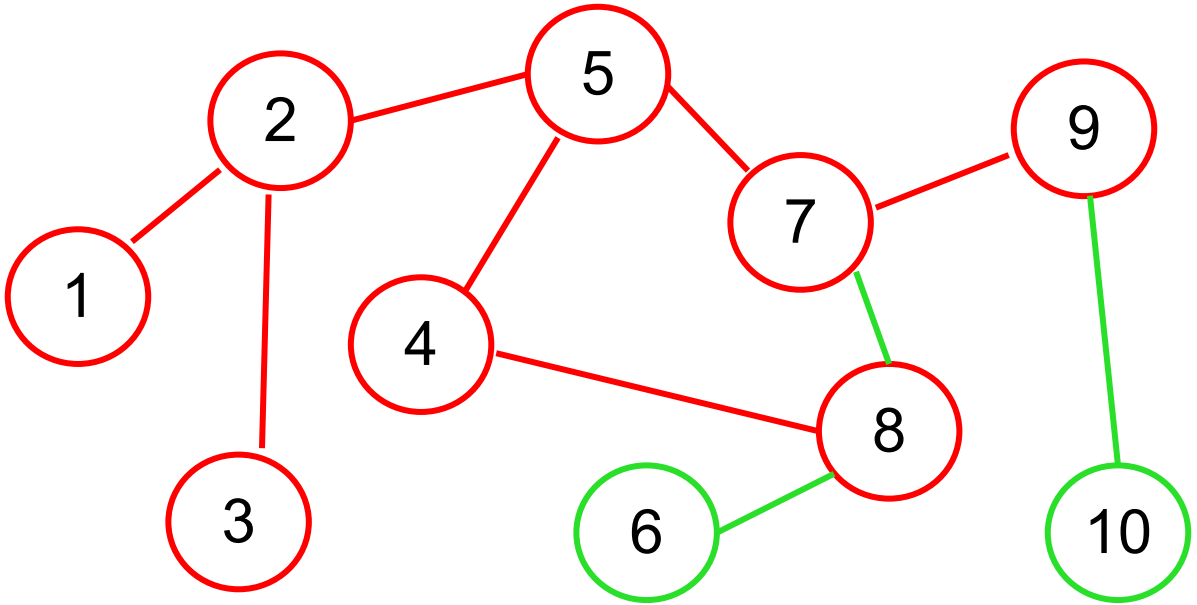
1	3	8	9	8

BFS



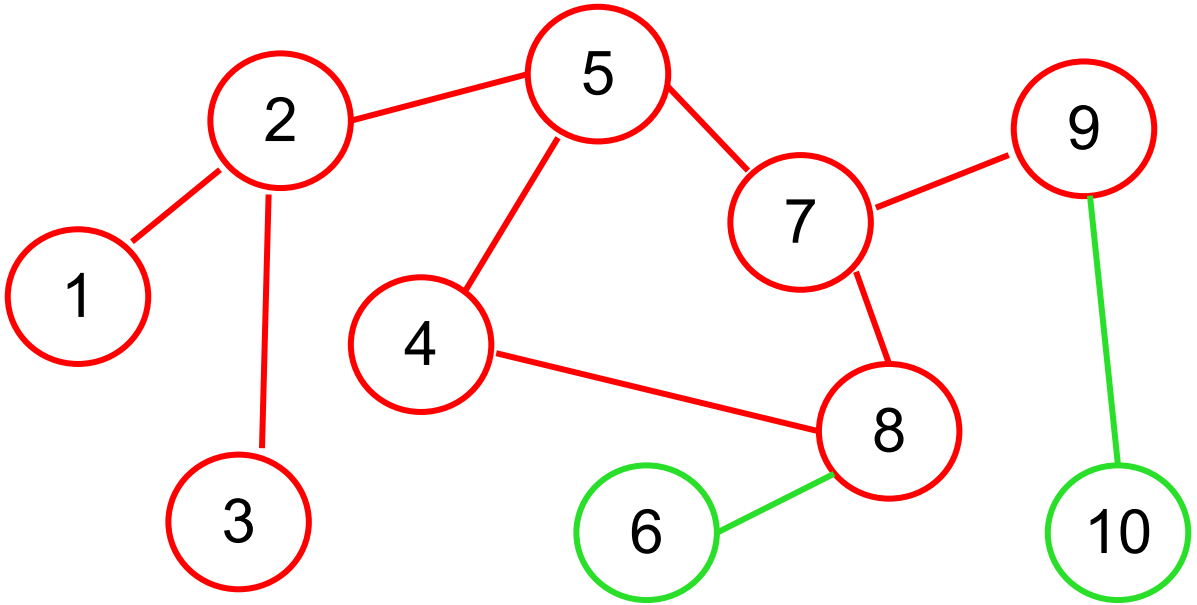
9	8	6

BFS



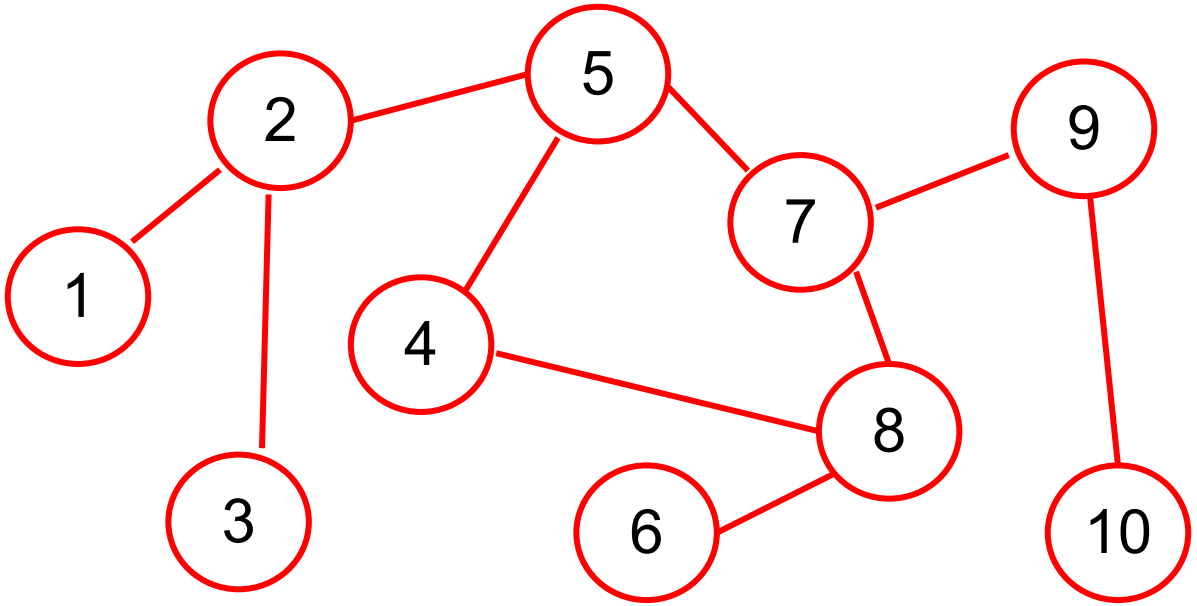
8	6	10

BFS

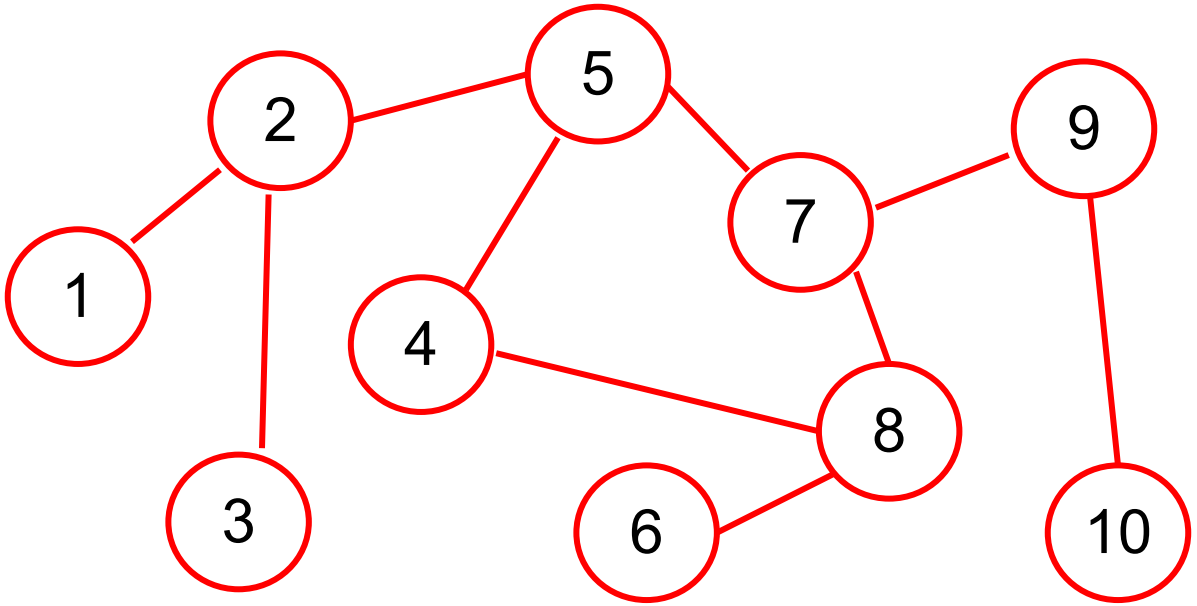


6	10

BFS



BFS



Con bfs podemos encontrar la distancia entre un nodo y todos los demás

BFS

```cpp
#include <bits/stdc++.h>
using namespace std;

vector <vector <int> > grafo; //la lista de adyacencia
vector <int> dist; //para guardar la distancia
vector <int> parent; //para guardar el padre de cada nodo

void bfs(int s, int n){ //nodo inicial, cantidad de nodos
    dist.assign(n, -1);
    parent.assign(n, -1);

    queue <int> Cola;
    dist[s] = 0;
    Cola.push(s);
    while(!Cola.empty()){
        int u = Cola.front(); Cola.pop();

        for( int v : grafo[u]){          //por cada vecino de u
            if(dist[v] == -1){           //veo si ya lo visite
                dist[v] = dist[u] + 1;   //guardo la distancia hasta v
                parent[v] = u;           //guardo el padre de v
                Cola.push(v);            //lo agrego a la cola
            }
        }
    }

}
```

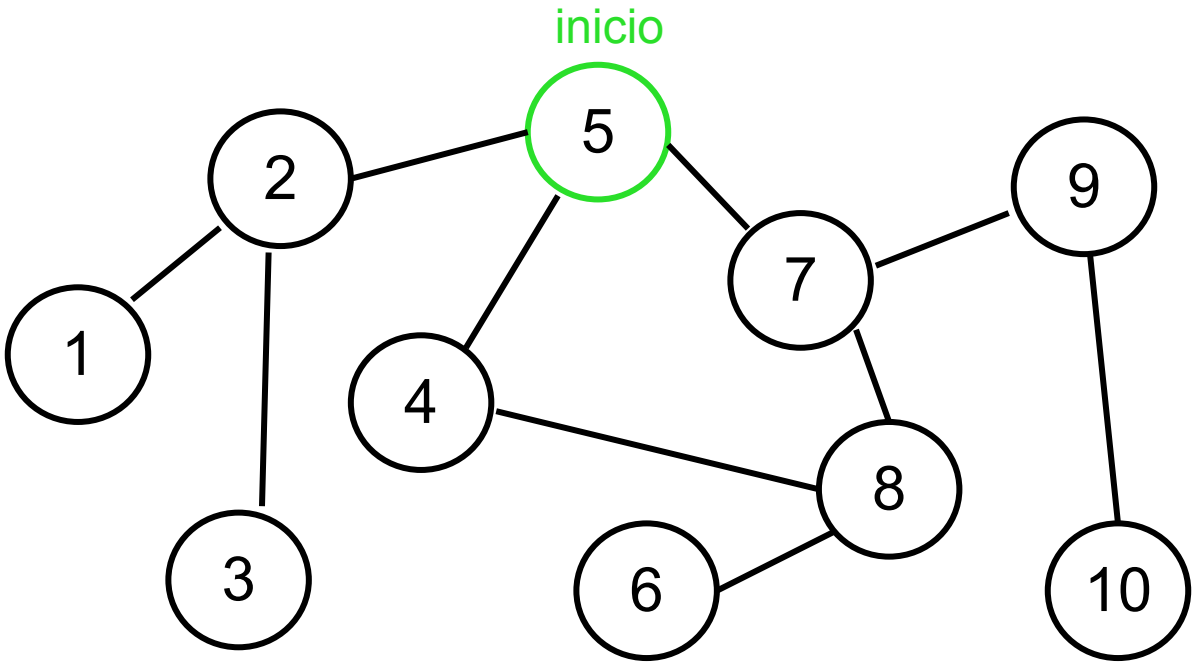
BFS

Ojo que no pasa por más de un componente conexo (si es que hay más de uno)

```cpp
#include <bits/stdc++.h>
using namespace std;

vector <vector <int> > grafo; //la lista de adyacencia
vector <int> dist; //para guardar la distancia
vector <int> parent; //para guardar el padre de cada nodo

void bfs(int s, int n){ //nodo inicial, cantidad de nodos
    dist.assign(n, -1);
    parent.assign(n, -1);

    queue <int> Cola;
    dist[s] = 0;
    Cola.push(s);
    while(!Cola.empty()){
        int u = Cola.front(); Cola.pop();

        for( int v : grafo[u]){          //por cada vecino de u
            if(dist[v] == -1){           //veo si ya lo visite
                dist[v] = dist[u] + 1;   //guardo la distancia hasta v
                parent[v] = u;           //guardo el padre de v
                Cola.push(v);            //lo agrego a la cola
            }
        }
    }

}
```

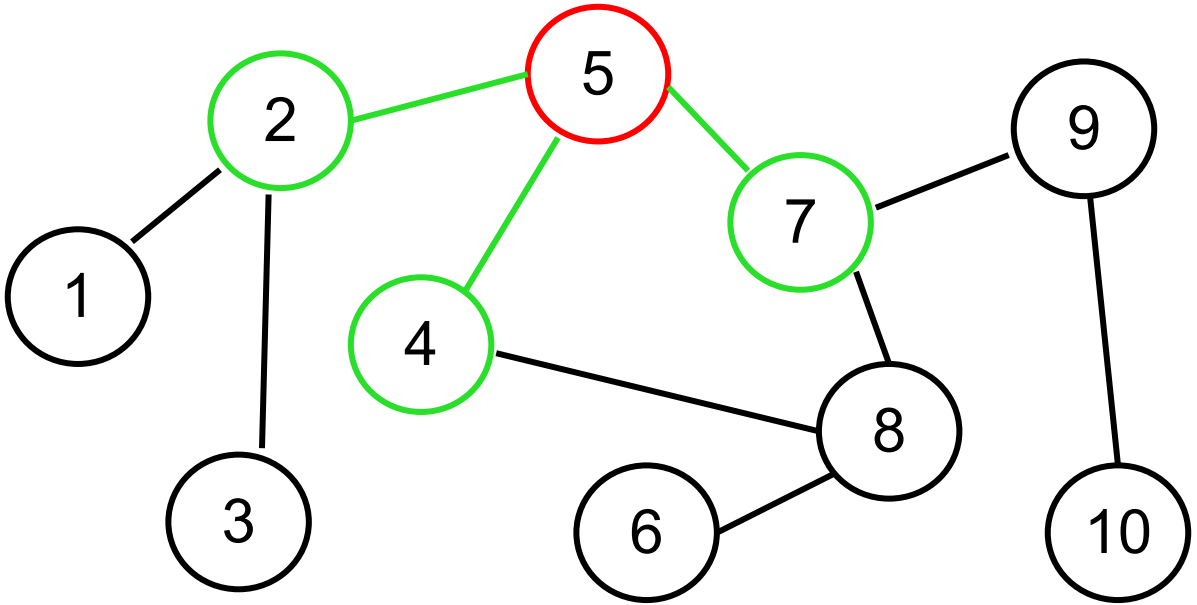
Depth first search (DFS)

DFS



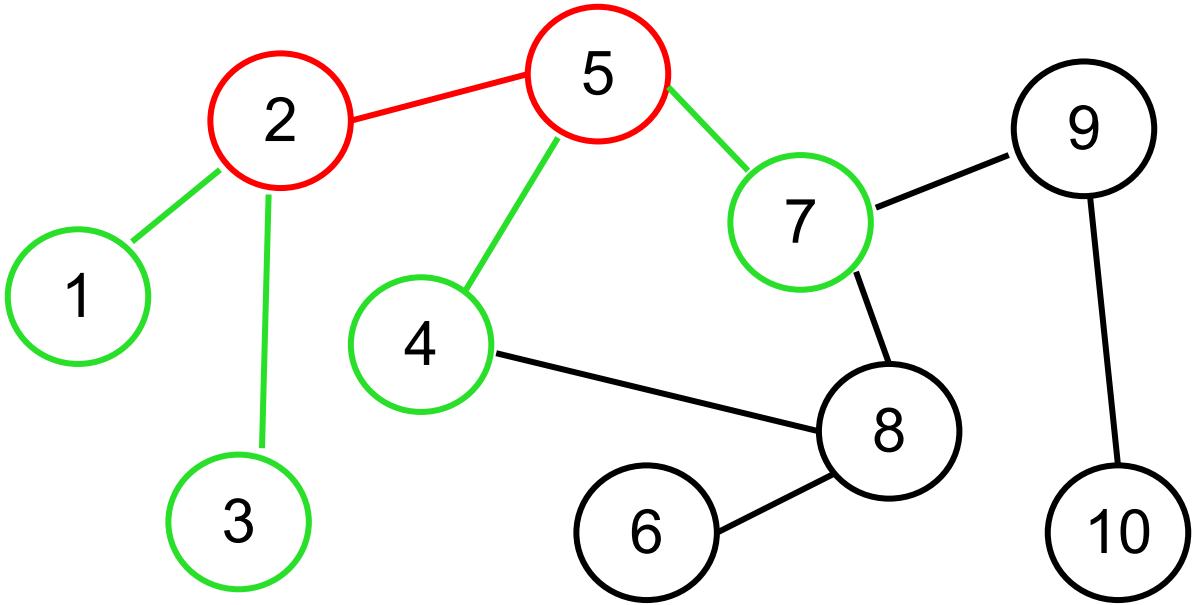
5

DFS



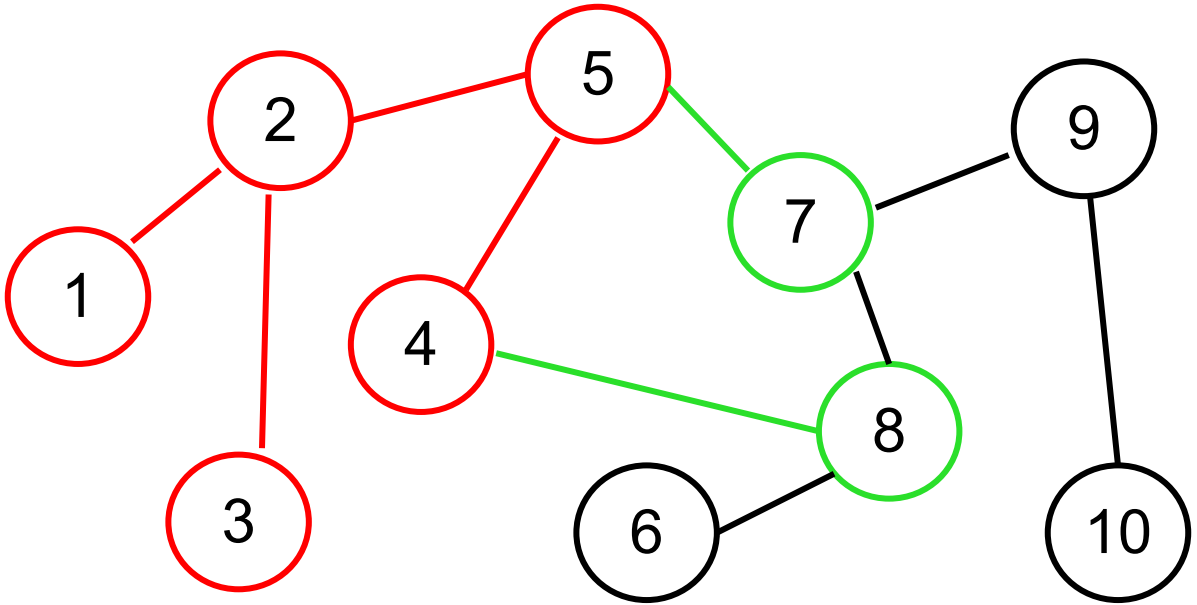
2	4	7

DFS



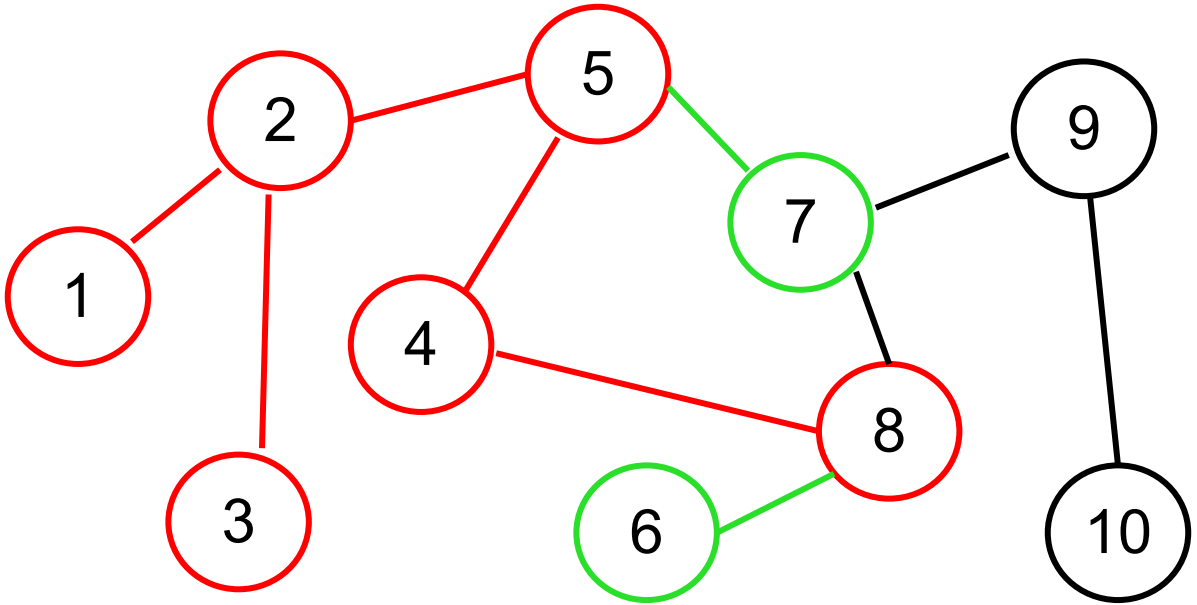
1	3	4	7

DFS



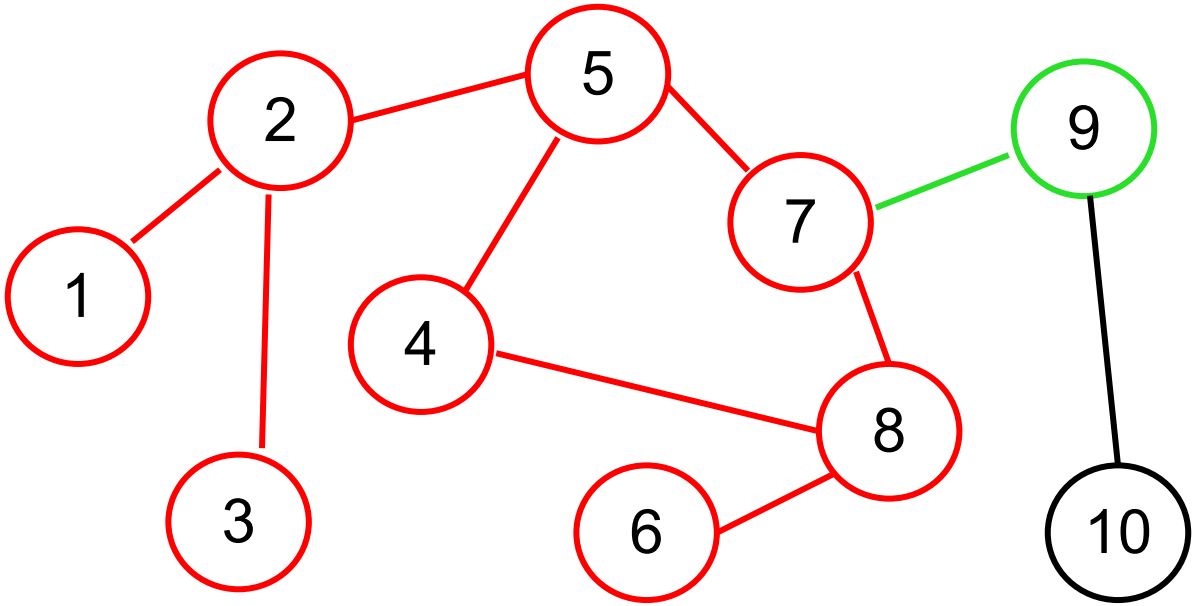
8	7

DFS



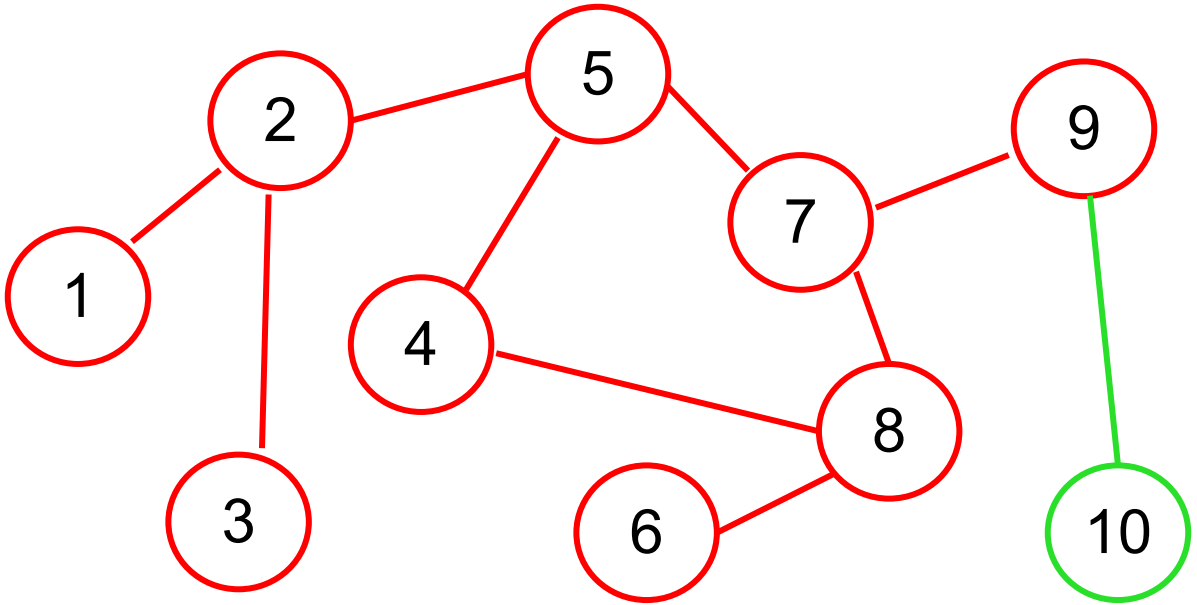
6	7

DFS



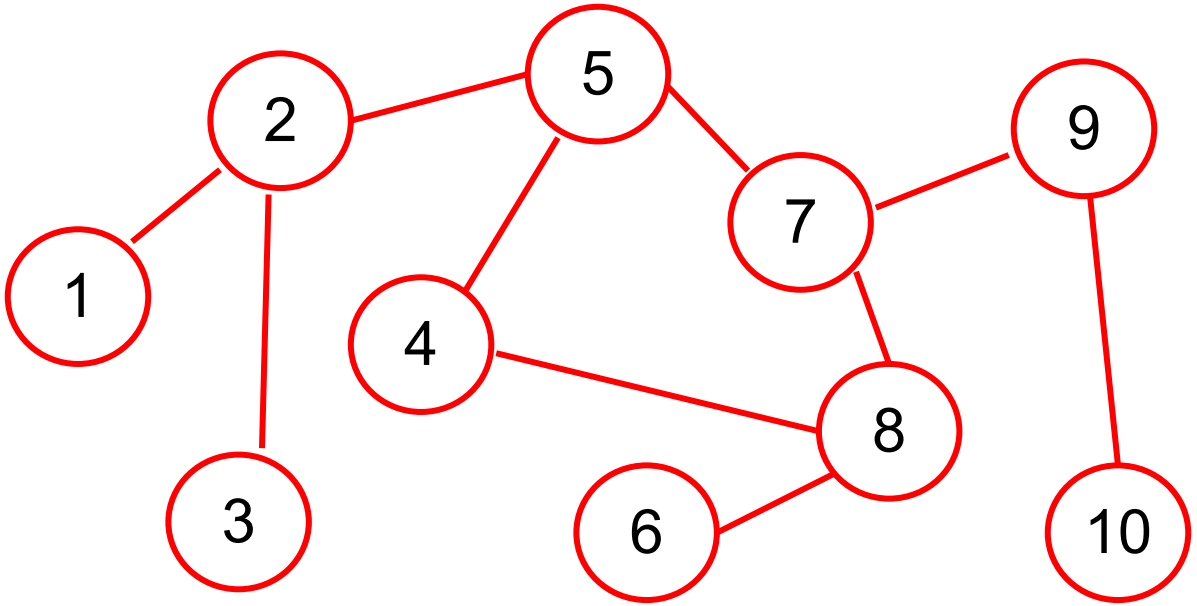
9

DFS



10

DFS



DFS

```cpp
#include <bits/stdc++.h>
using namespace std;

vector <vector <int> > grafo;
vector <int> visited;

void dfs_recursivo(int s, vector <vector <int> > &grafo, vector <int> &visited){
    visited[s] = 1;                                 //marcamos que pasamos por s
    for(int v : grafo[s]){                          //por cada vecino de s
        if(visited[v] == 0){                        //si es que no lo hemos visitado
            dfs_recursivo(v, grafo, visited);       //lo visitamos
        }
    }
}


void dfs_iterativo(int s, vector < vector <int> > &grafo, vector <int> &visited){
    stack <int> Pila;
    Pila.push(s);                           //metemos a la pila el nodo inicial

    while(!Pila.empty()){
        int u = Pila.top(); Pila.pop(); //tomamos un nodo
        visited[u] = 1;                 //lo marcamos como visitado

        for(int v: grafo[u]){           //para cada vecino del nodo
            if(visited[v] == 0){        //vemos si lo visitamos
                visited[v] = 1;         //si es que no, lo marcamos como visitado y
                Pila.push(v);           //lo metemos a la pila
            }
        }
    }
}
```

Te entregan un grafo no dirigido **G** compuesto por **V** nodos y **E** aristas. Tu tarea es encontrar el número de componentes conexos que existen en **G**.

Input

La primera línea contiene dos valores *V* y *E* el número de nodos y aristas respectivamente. A continuación siguen *E* líneas con dos valores *a* y *b* indicando una arista entre los nodos *a* y *b*

Ejemplo

6 4

1 2

3 6

4 5

1 6

2