



Estructura de Datos

Maria C. Torres

Maria C. Torres

Ing. Electrónica (UNAL)

M.E. Ing. Eléctrica (UPRM)

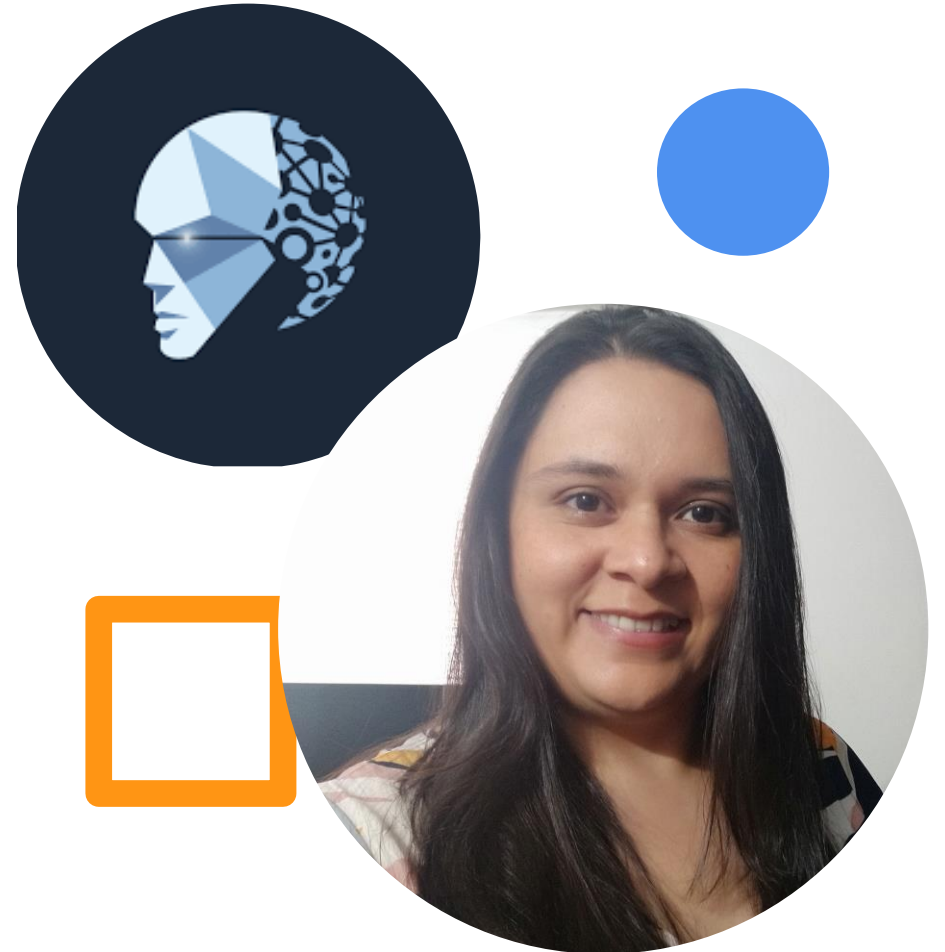
Ph.D. Ciencias e Ingeniería de la Computación y la Información (UPRM)


Profesora asociada

Dpto. Ciencias de la Computación y la Decisión


mctorresm@unal.edu.co

HORARIO DE ATENCIÓN: Martes 10:00 am a
12:00 m – Oficina 313 M8A






Contenido del Curso

- 
- ☐ Introducción: revisión fundamentos y POO
 - ☐ Análisis de complejidad
 - ☐ Arreglos
 - ☐ Listas enlazadas
 - ☐ Pilas y colas
 - ☐ Heap
 - ☐ Árboles binarios
 - ☐ Tablas hash
 - ☐ **Grafos**



Grafos

- 
- ❑ Listas y matrices de adyacencia
 - ❑ Recorrido
 - ❑ Árbol de expansión mínima
 - ❑ Algoritmos del camino más corto

Grafos

Historia

- ❑ Representaciones matemáticas introducidas por Euler en 1736 para representar vértices que pueden relacionarse libremente entre si mediante aristas.



Aplicaciones

- ❑ La representación de una red de carreteras
- ❑ Etapas de un proceso industrial
- ❑ Redes de telecomunicación

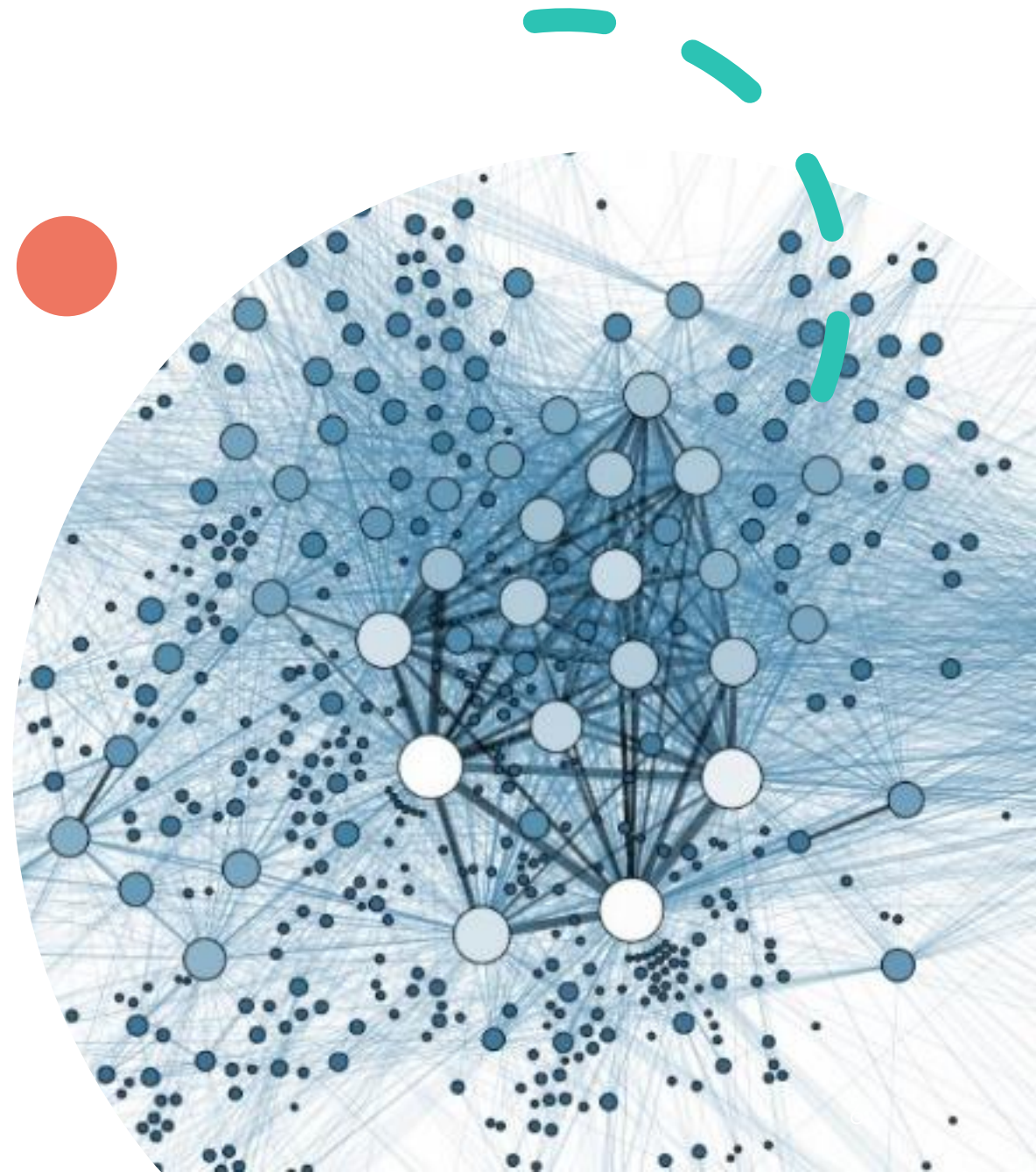


Grafos

DEFINICIÓN

Un grafo $G = (V, E)$ se define por:

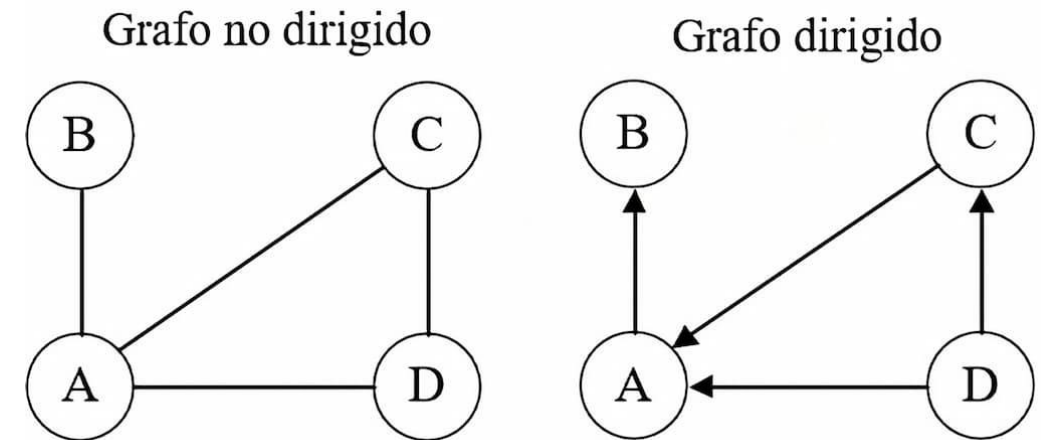
- Un conjunto de n vértices V , a los cuales se hace referencia por sus índices
- Un conjunto de m aristas E , que conectan vértices entre sí.
- Una arista es un par de vértices, indicados de la forma $\langle i, j \rangle$, que indica que el vértice i está conectado al vértice j .



Grafos

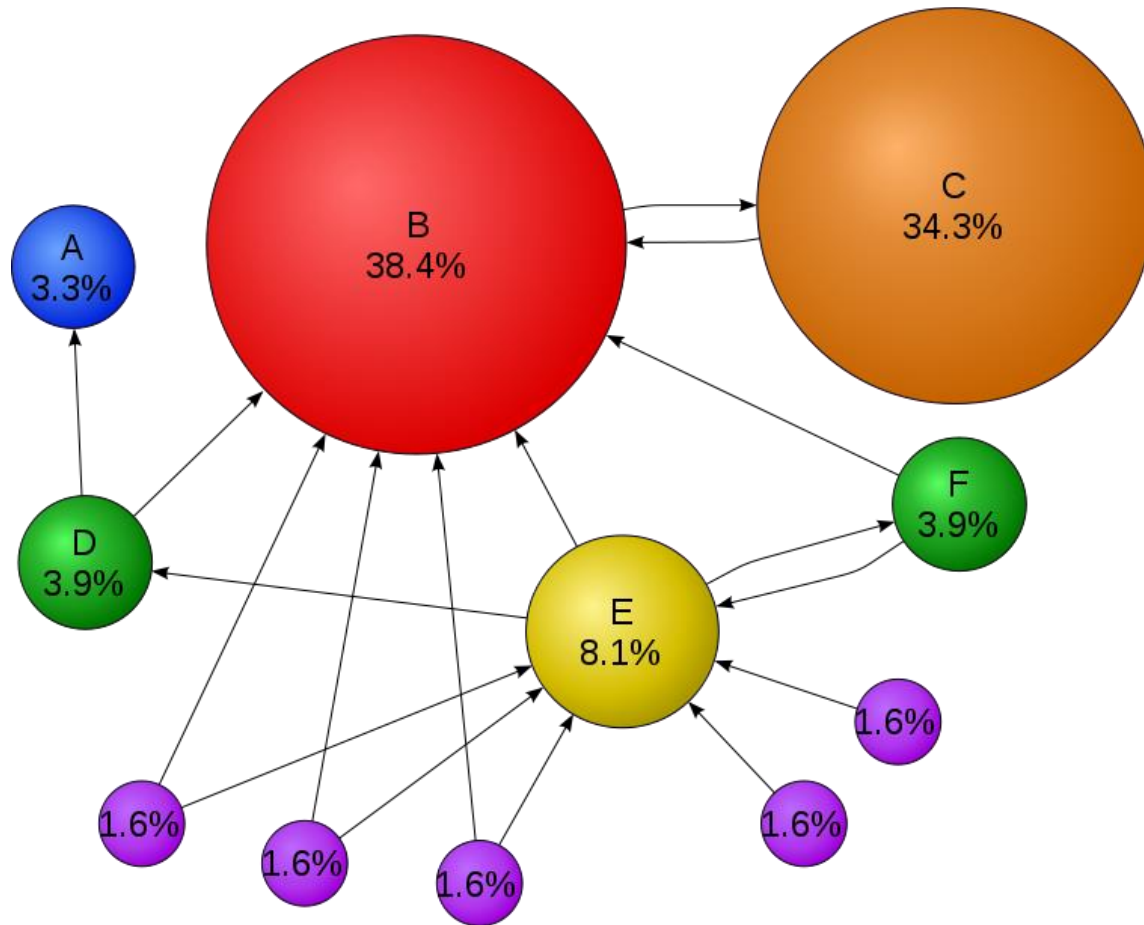
Dependiendo de si el orden de los vértices en las aristas importa o no tenemos dos clases de grafos:

- **Grafo dirigido:** El orden importa, es decir $\langle i, j \rangle \neq \langle j, i \rangle$. Si el vértice i está conectado al vértice j , no implica que el vértice j está conectado al vértice i .
- **Grafo no dirigido:** El orden no importa, $\langle i, j \rangle = \langle j, i \rangle$.



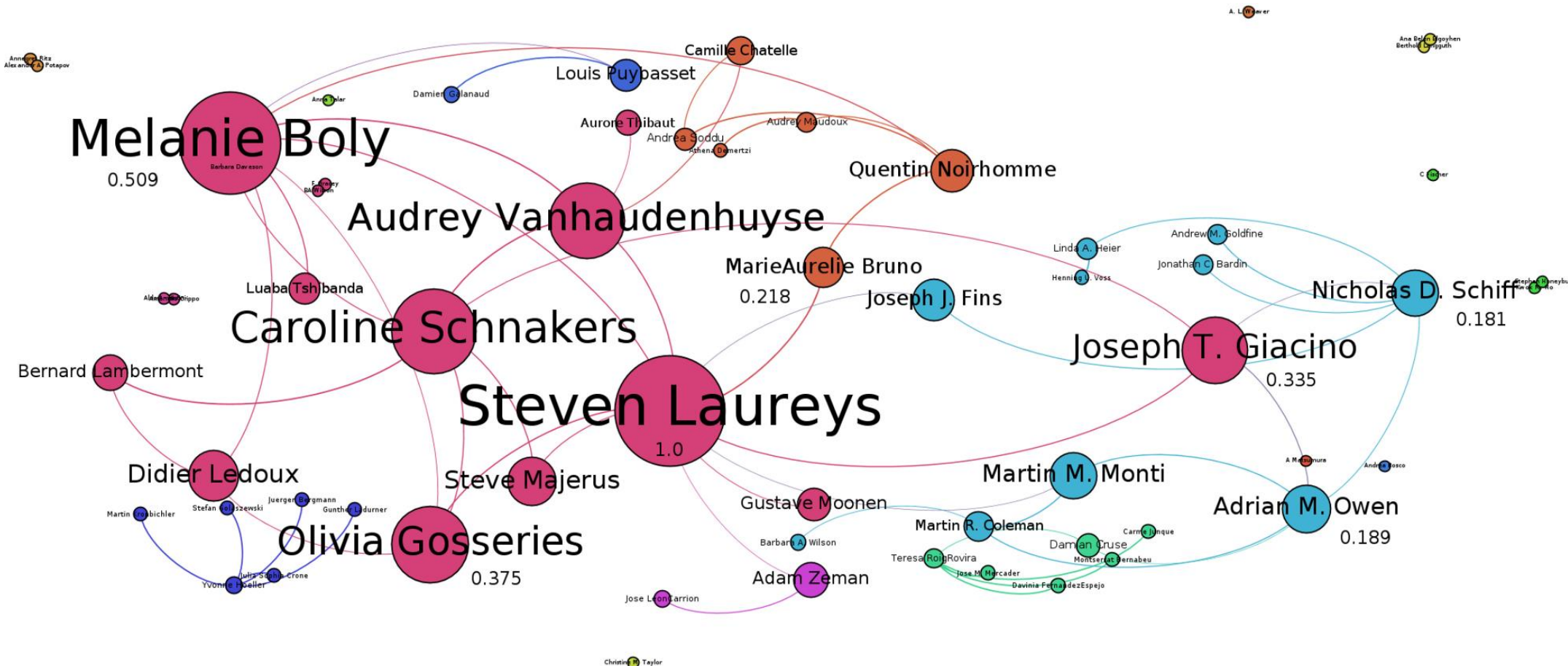
Aplicaciones

PageRank: algoritmo usado por Google en su buscador.



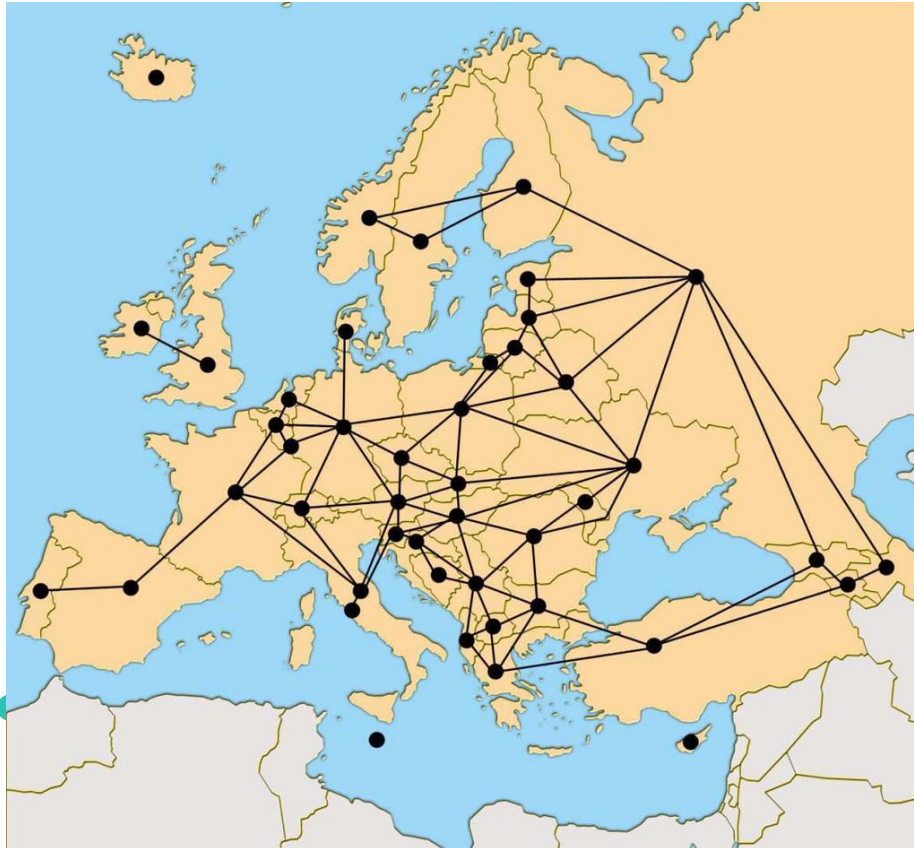
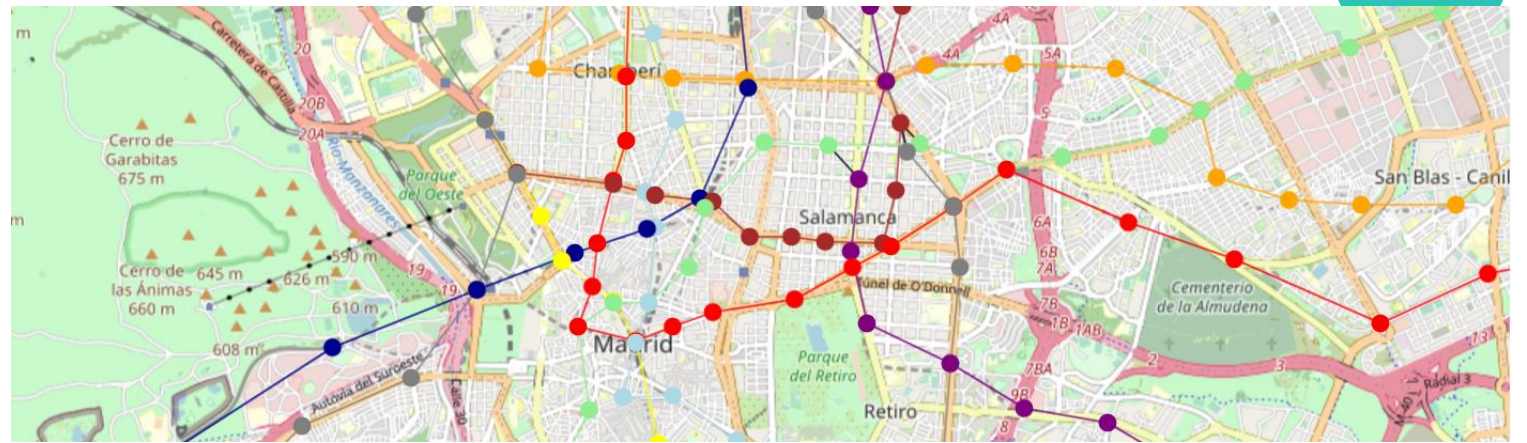
Aplicaciones

Tendencia de publicaciones

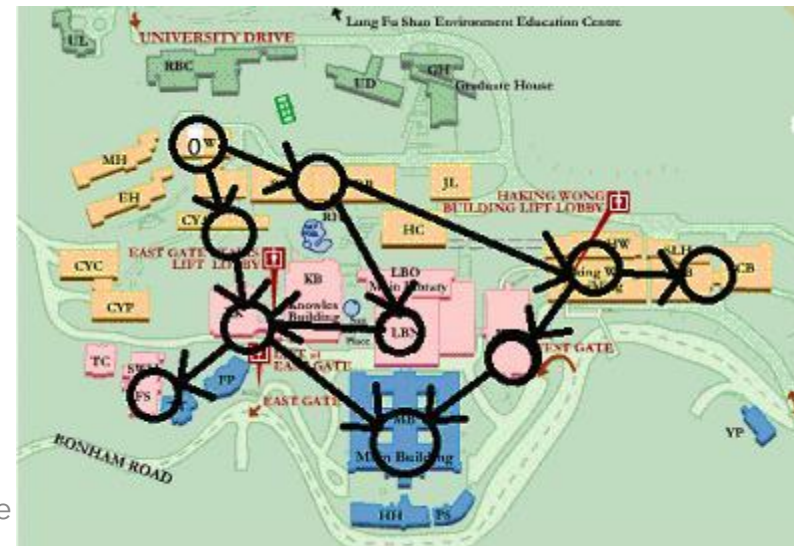


Aplicaciones

Camino más corto

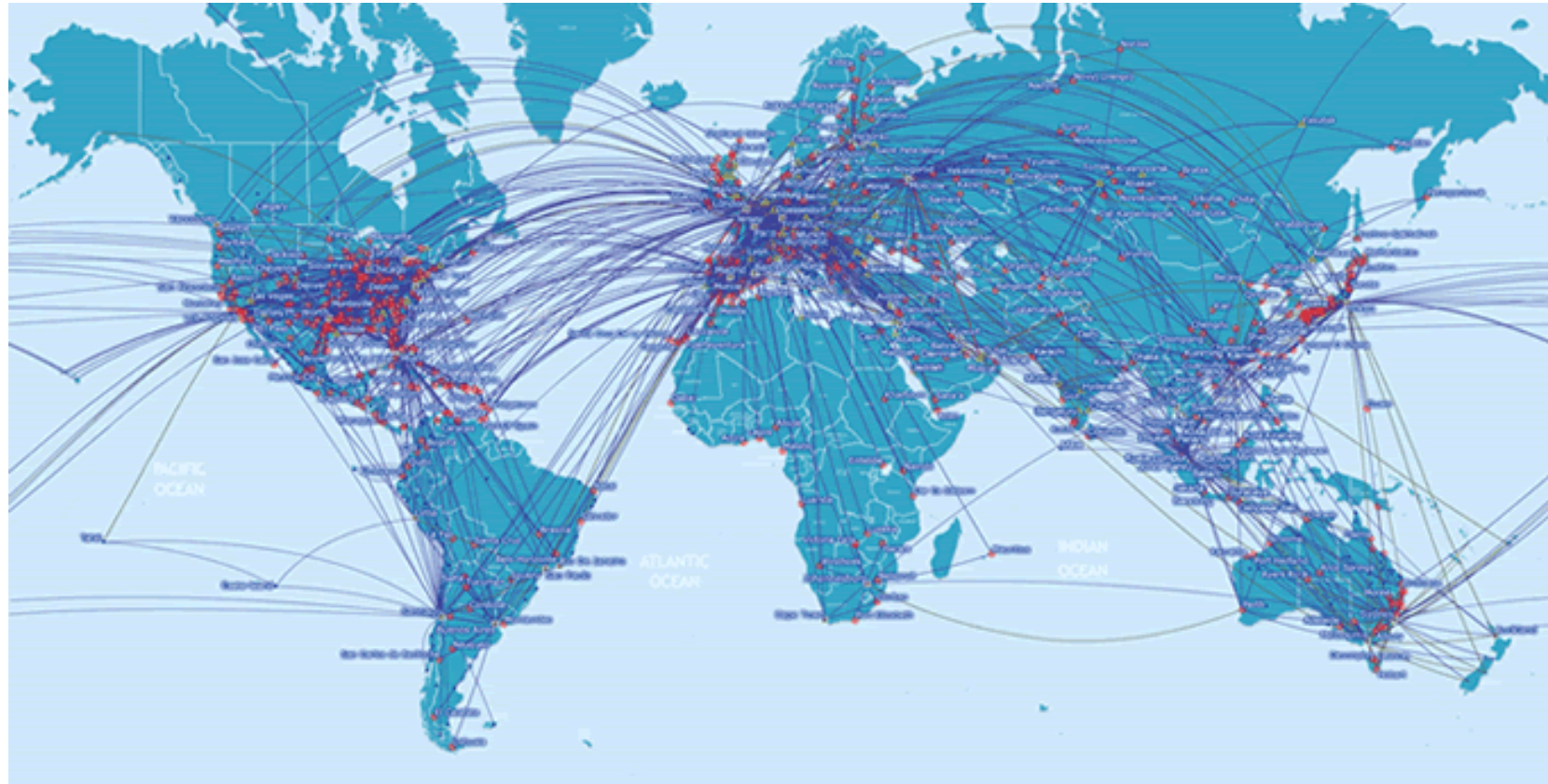


Google Maps



Aplicaciones

Trafico aéreo





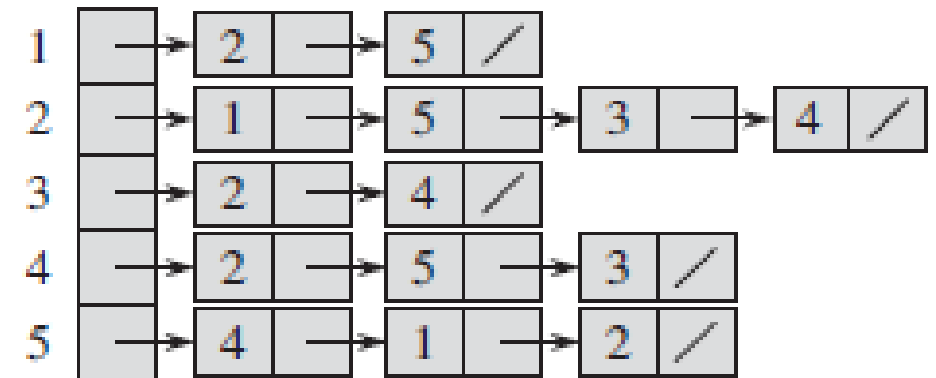
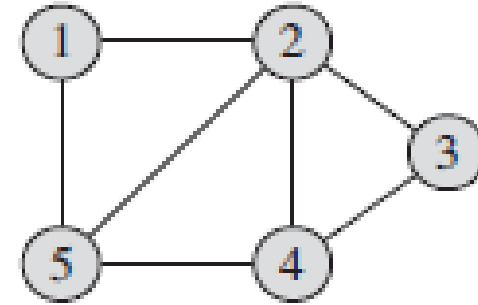
Grafos

- ❑ **Listas y matrices de adyacencia**
- ❑ Recorrido
- ❑ Árbol de expansión mínima
- ❑ Algoritmos del camino más corto

Grafos

Representación - Listas de adyacencia:

- ❑ proporciona un camino compacto para representar grafos esparcidos, es decir grafos con pocas aristas.
- ❑ Esta representación consiste en un arreglo Adj con $|V|$ listas, una para cada vértice.
- ❑ Para cada $u \in V$, la lista de adyacencia $Adj[u]$ contiene todos los vértices v tal que exista la arista $\langle u, v \rangle$.
- ❑ Si G es un **grafo dirigido**, la suma de las longitudes de todas las listas de adyacencia es $|E|$.
- ❑ Si G es un **grafo no dirigido**, la suma de las longitudes de todas las listas de adyacencia es $2|E|$.

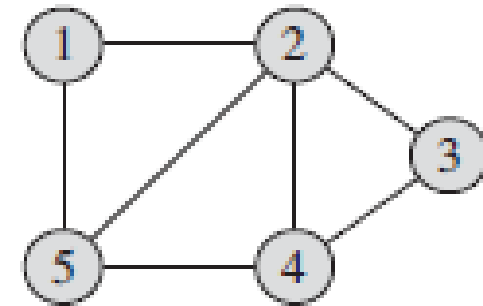


Grafos

Representación - Matriz de adyacencia:

- ❑ Cuando el grafo es denso, es decir tiene muchas aristas, o deseamos saber rápidamente si dos nodos están conectados, la representación más apropiada es la **matriz de adyacencia**.
- ❑ Para esta representación de un grafo $G=(V,E)$ asumimos que los vértices están enumerados de $1...|V|$
- ❑ La matriz de adyacencia es una matriz $|V|*|V|$ tal que:

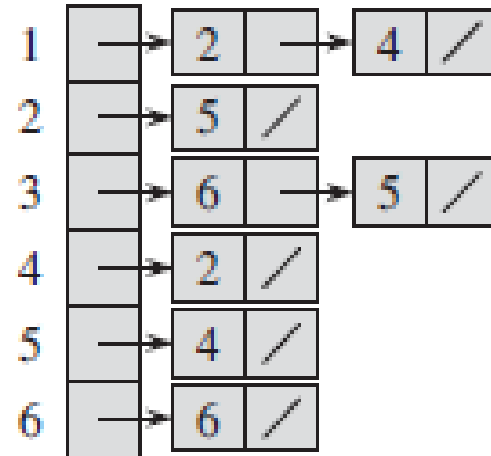
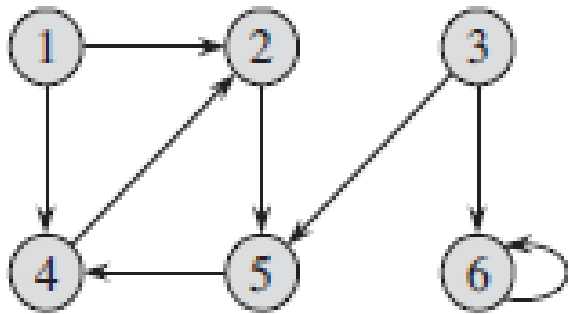
$$a_{ij} = \begin{cases} 1 & \text{si } \langle i, j \rangle \in E \\ 0 & \text{en otro caso} \end{cases}$$



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

Grafos

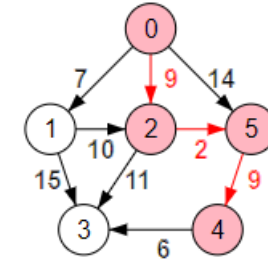
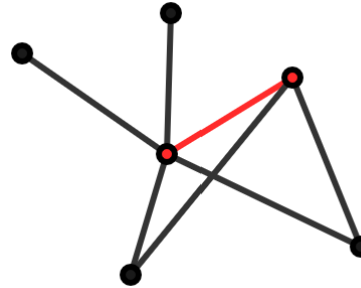
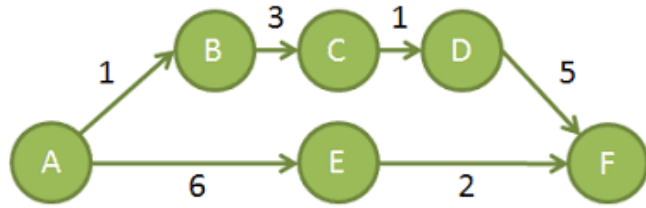
Representación - Grafos dirigidos



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

Grafos

Conceptos básicos



Grafos ponderados:

La arista tiene asociada un valor numérico relacionado con un costo. Este costo se almacena también en la lista o matriz de adyacencia.

Vértices adyacentes:

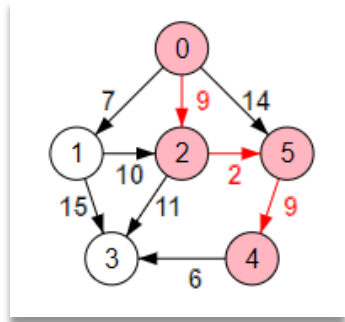
Dos vértices u y v son adyacentes si existe una arista que los conecte, es decir, $\langle u, v \rangle \in E$

Camino:

Entre dos vertices u y v , un camino es una secuencia de vértices tal que cada par de vértices contiguos forman una arista

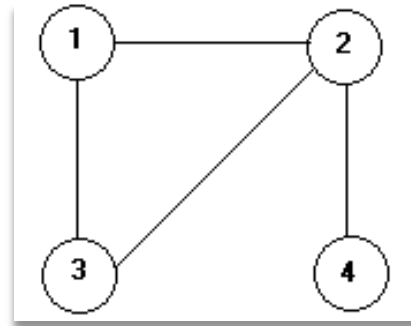
Grafos

Conceptos básicos



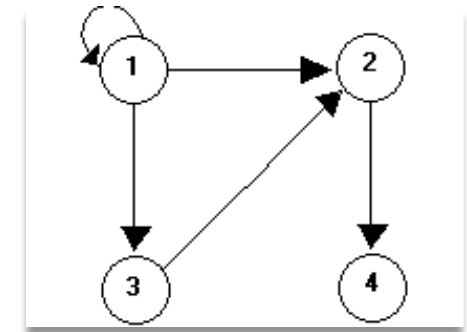
Camino simple:

Camino donde no hay vértices repetidos



Grado de un vértice:

En un grafo no dirigido es igual al número de vértices adyacentes al vértice.

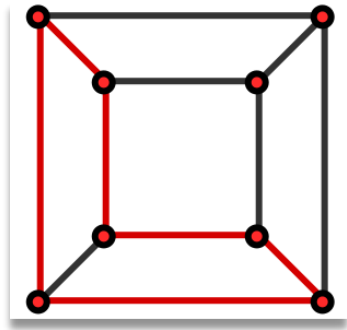


Grado de un vertice:

En un grafo dirigido se distingue entre el grado interior de un vértice (número de aristas que llegan a él) y grado exterior (número de aristas que salen de él).

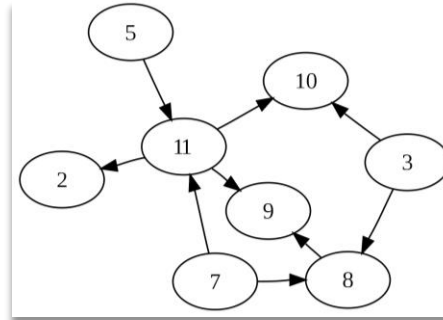
Grafos

Conceptos básicos



Ciclo:

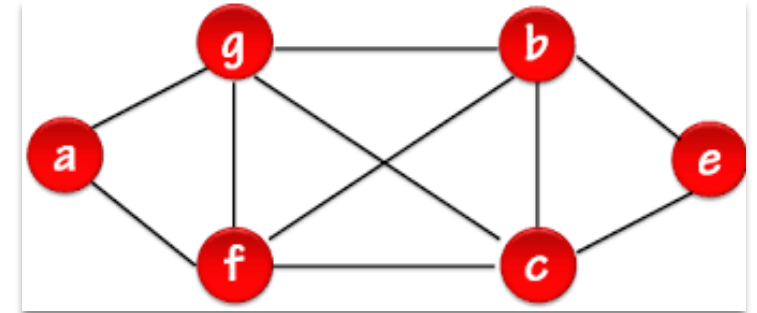
Un camino simple donde el vértice inicial y el final son el mismo.



Grafo acíclico:

Todos sus posibles caminos son simples, no existen ciclos.

Un árbol es un grafo no dirigido y acíclico

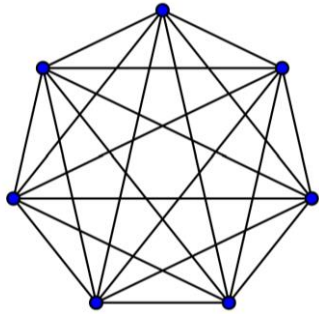


Grado conectado:

Existe como mínimo un camino entre cualquier par de vértices distintos

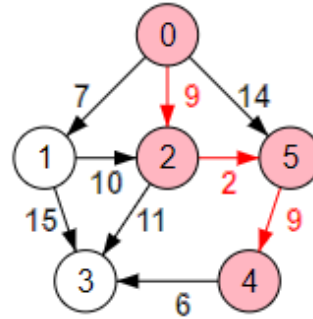
Grafos

Conceptos básicos



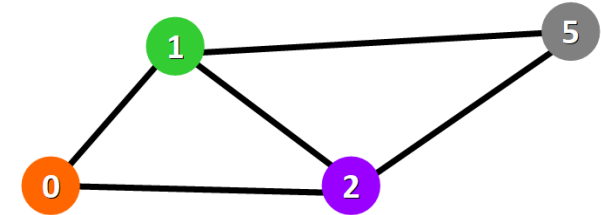
Grafo completamente conectado:

Para cada vértice existe aristas que lo conecten con los $n-1$ vértices restantes.



Costo de un camino:

Suma de los costos o pesos de las aristas que recorre el camino, si el grafo no es ponderado cada arista tiene un peso igual a 1.



Ruta optima:

Camino de costo mínimo.

Grafos

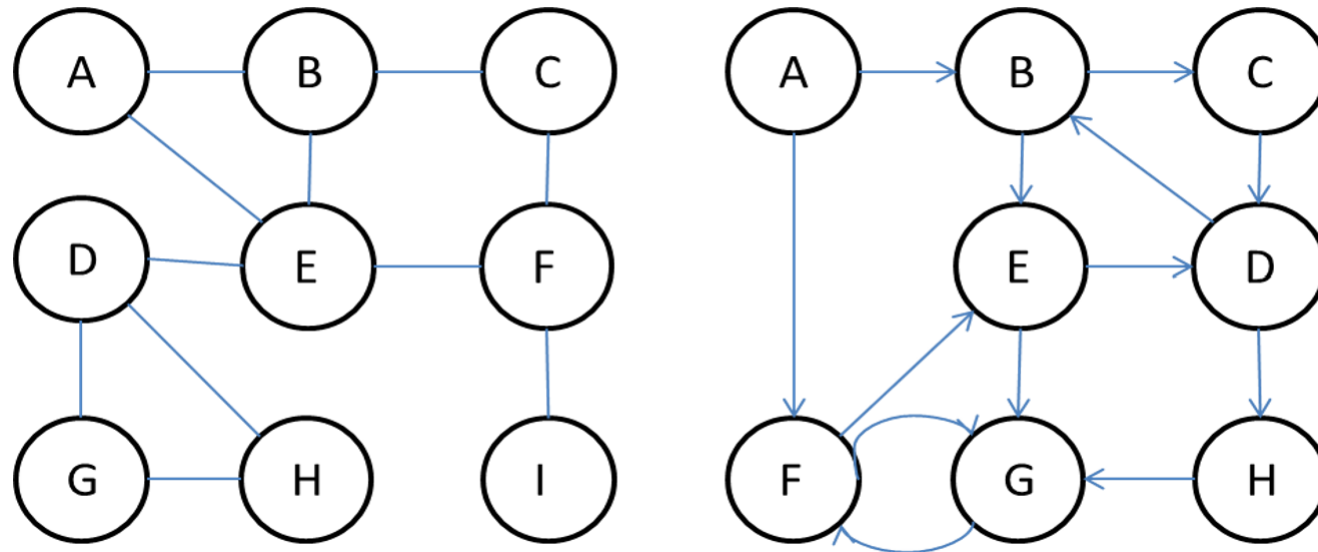
Costo computacional:

Operación	Matriz de adyacencia	Lista de adyacencia
Uso de memoria	$\theta(n^2)$	$\theta(n + m)$
Existencia de arista	$O(1)$	$O(\text{grado}(i))$
Recorrido vértices adyacentes	$\theta(n)$	$\theta(\text{grado}(i))$
Recorrido de todas las aristas	$\theta(n^2)$	$\theta(m)$
Insertar/eliminar arista	$O(1)$	$O(\text{grado}(i))$
Insertar/eliminar vértice	$O(n^2)$	$O(n)$

Grafos

Ejemplo:

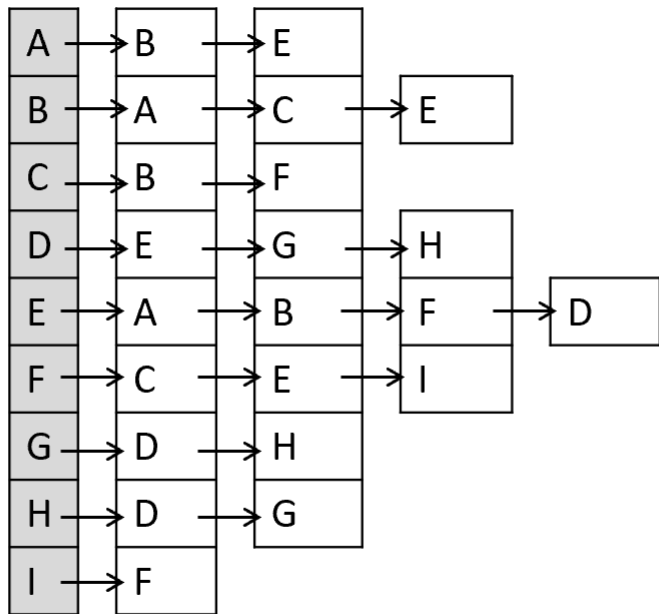
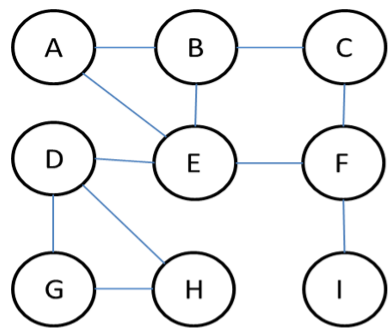
Vamos a construir la representación para cada uno de estos grafos.



Grafos

Ejemplo:

Vamos a construir la representación para cada uno de estos grafos.



Lista de adyacencia

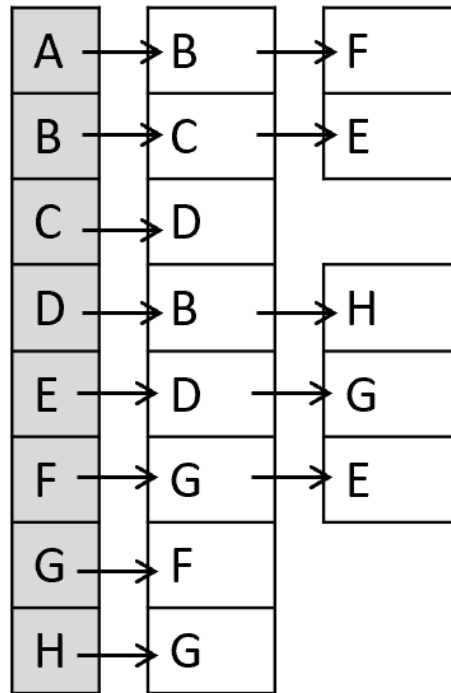
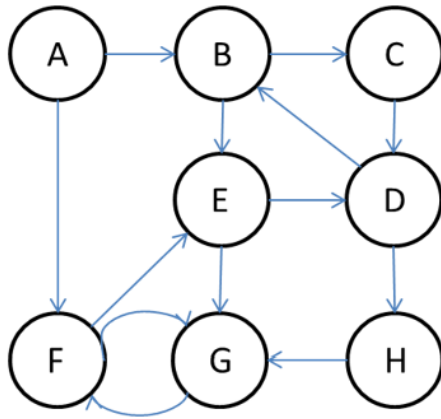
	A	B	C	D	E	F	G	H	I
A	0	1	0	0	1	0	0	0	0
B	1	0	1	0	1	0	0	0	0
C	0	1	0	0	0	1	0	0	0
D	0	0	0	0	1	0	1	1	0
E	1	1	0	1	0	1	0	0	0
F	0	0	1	0	1	0	0	0	1
G	0	0	0	1	0	0	0	1	0
H	0	0	0	1	0	0	1	0	0
I	0	0	0	0	0	1	0	0	0

Matriz de adyacencia

Grafos

Ejemplo:


Vamos a construir la representación para cada uno de estos grafos.



	A	B	C	D	E	F	G	H
A	0	1	0	0	0	1	0	0
B	0	0	1	0	1	0	0	0
C	0	0	0	1	0	0	0	0
D	0	1	0	0	0	0	0	1
E	0	0	0	1	0	0	1	0
F	0	0	0	0	1	0	1	0
G	0	0	0	0	0	1	0	0
H	0	0	0	0	0	0	1	0



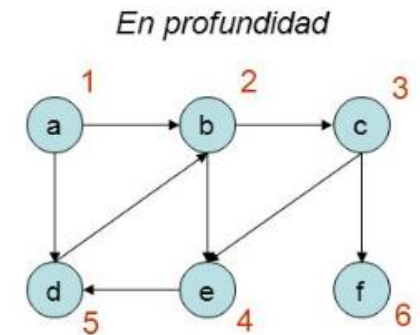
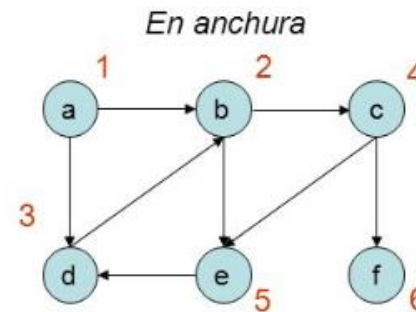
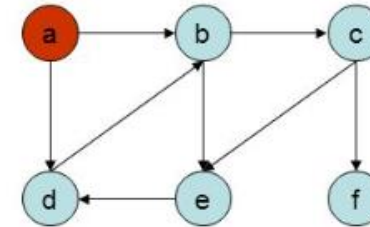
Grafos

- 
- ❑ Listas y matrices de adyacencia
 - ❑ **Recorrido**
 - ❑ Árbol de expansión mínima
 - ❑ Algoritmos del camino más corto

Grafos

Recorrido

- ❑ Se emplean cuando se requiere recorrer (visitar) de forma sistemática todos los vértices del grafo
- ❑ Dos casos:
 - ❑ No importa el orden de visita
 - ❑ El orden de visita depende de las aristas existentes
 - ❑ Recorrido en profundidad
 - ❑ Recorrido en anchura
 - ❑ Recorrido por orden topológico
 - ❑ Estas estrategias utilizan otras estructuras de datos (e.g. colas) para almacenar los vértices visitados



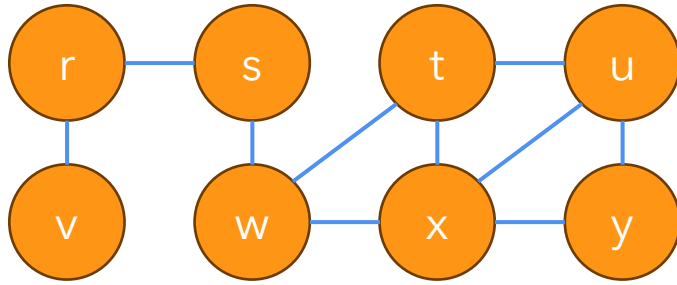
Grafos

Recorrido en profundidad

- ❑ Se emplea en grafos dirigidos, para grafos no dirigidos se considera cada arista como un par de aristas dirigidas.
 - ❑ Se empieza visitando un nodo seleccionado de forma aleatoria.
 - ❑ Se recorre en profundidad los componentes conexos, es decir, se examina los caminos hasta que se llega a nodos ya visitados o sin sucesores.
 - ❑ Si después de visitar todos los sucesores del primer nodo todavía quedan nodos por visitar, se repite el proceso a partir de cualquiera de los nodos no visitados.
- ❑ No hay un único recorrido en profundidad, sino un conjunto de ellos.
- ❑ El recorrido depende del vértice inicial y del orden de visita.
- ❑ El orden de visita puede interpretarse como un árbol: árbol de expansión en profundidad asociada al grafo

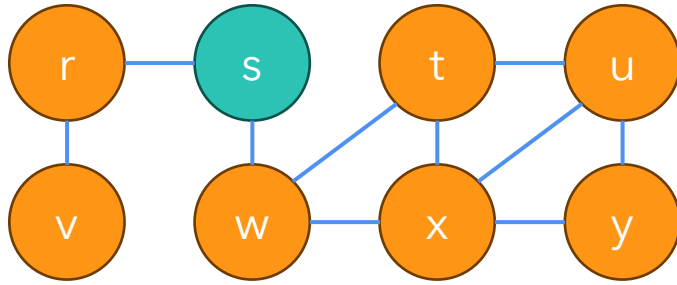
Grafos

Recorrido en profundidad

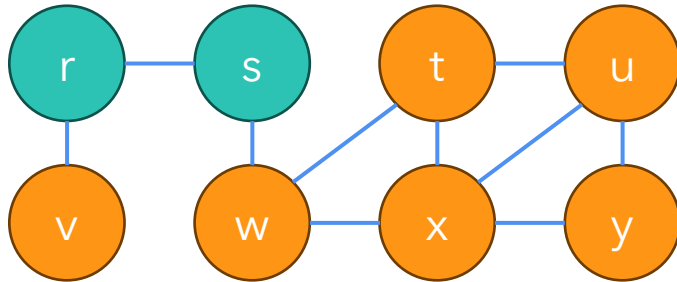


Grafos

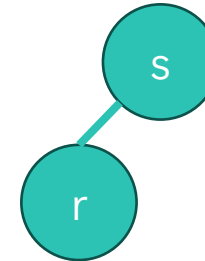
Recorrido en profundidad



Pila S = { s }

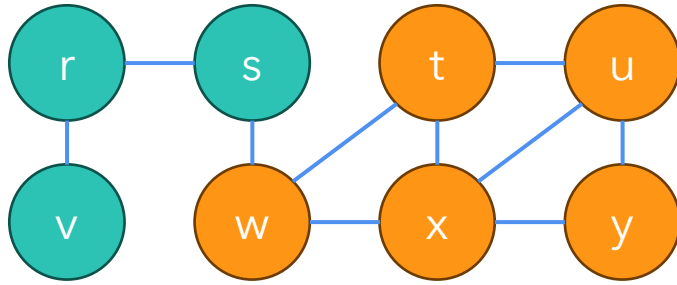


Pila S = { r, s }

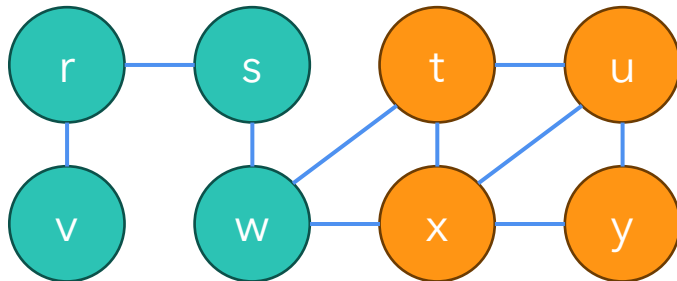


Grafos

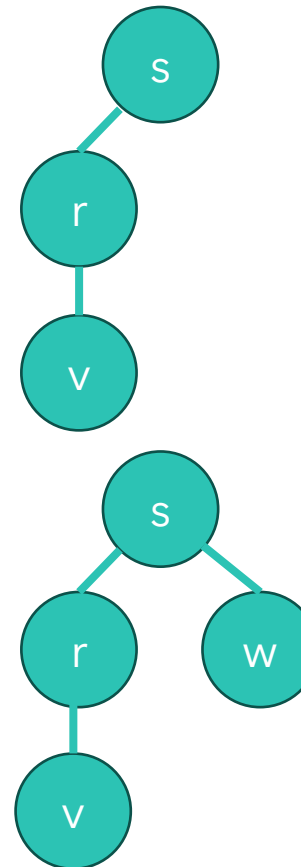
Recorrido en profundidad



Pila $S = \{v, r, s\}$

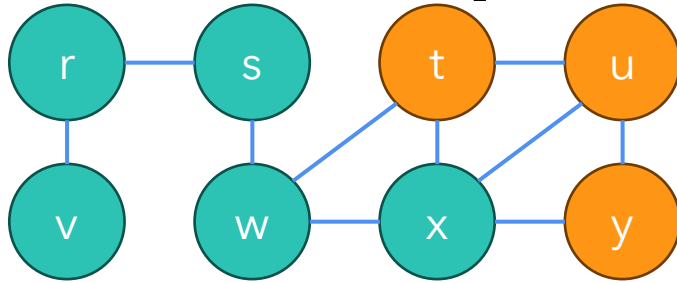


Pila $S = \{w, v, r, s\}$

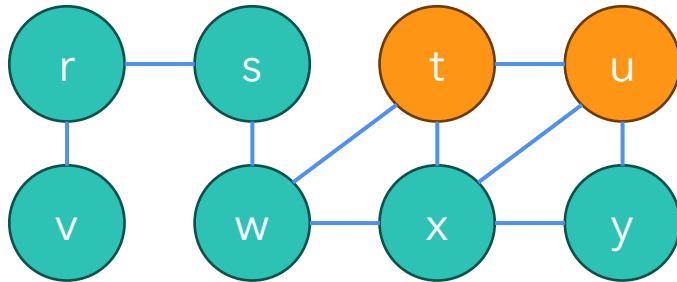
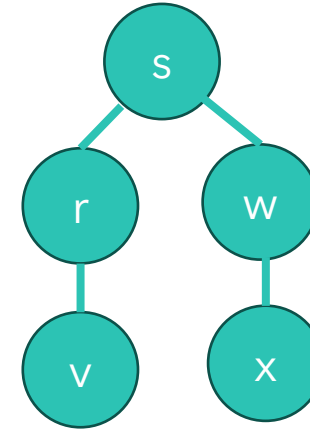


Grafos

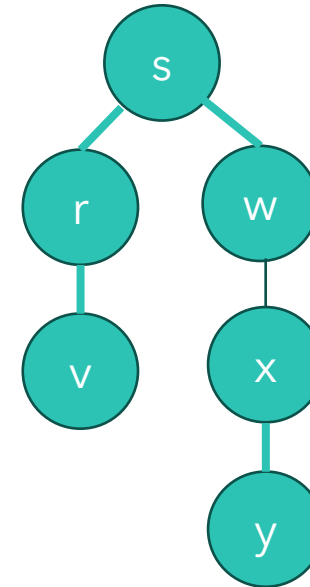
Recorrido en profundidad



Pila $S = \{x, w, v, r, s\}$

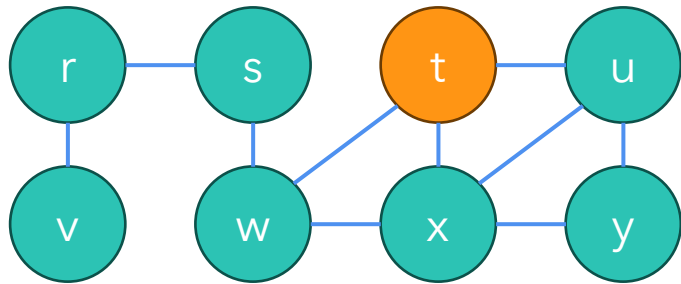


Pila $S = \{y, x, w, v, r, s\}$

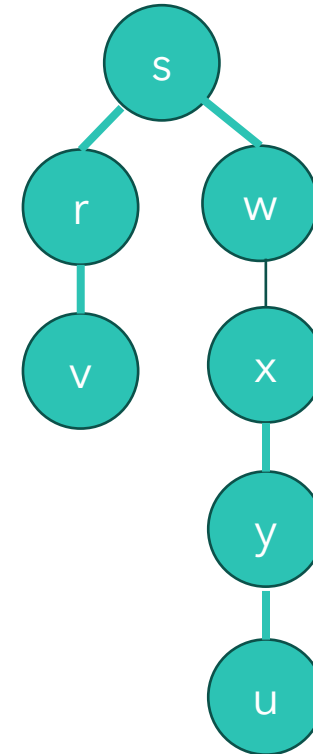


Grafos

Recorrido en profundidad

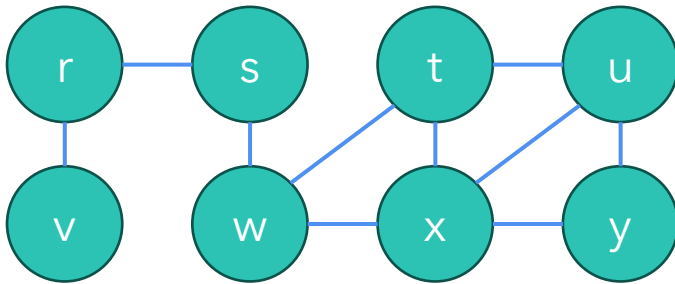


Pila S = { u, y, x, w, v, r, s }

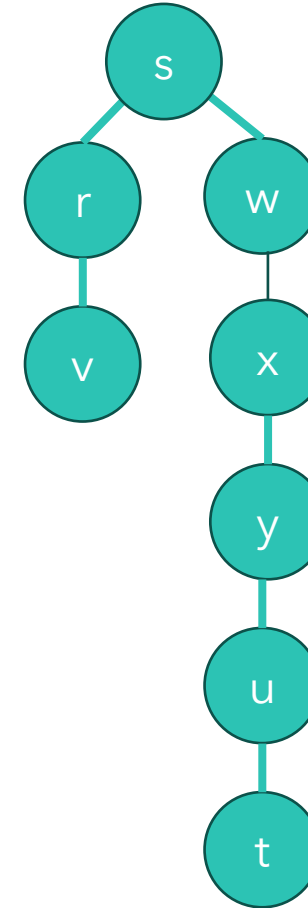


Grafos

Recorrido en profundidad



Pila S = { t, u, y, x, w, v, r, s }



Grafos

Recorrido en profundidad

DFS(G)

1. FOR $i = 0$ TO $i < V.length$
2. Visitado[i] = false
3. FOR $i = 0$ TO $i < V.length$
4. IF (!Visitado[i])
5. DFS-VISIT(G,i)

DFS-VISIT(G,i)

1. Visitado[i] = true
2. FOR each node j adjacent to i
3. IF (!Visitado[j])
4. DFS-VISIT(G,j)

Grafos

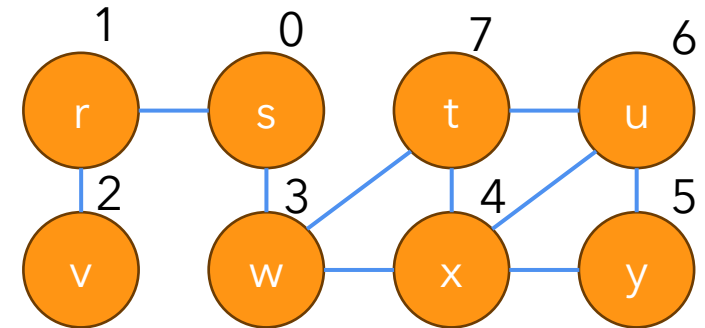
Recorrido en profundidad

DFS(G)

1. FOR $i = 0$ TO $i < V.length$
2. Visitado[i] = false
3. FOR $i = 0$ TO $i < V.length$
4. IF (!Visitado[i])
5. DFS-VISIT(G,i)

DFS-VISIT(G,i)

1. Visitado[i] = true
2. FOR each node j adjacent to i
3. IF (!Visitado[j])
4. DFS-VISIT(G,j)



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

Grafos

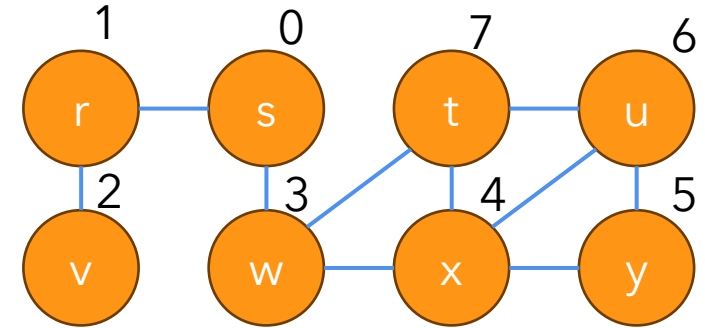
Recorrido en profundidad

DFS(G)

1. FOR $i = 0$ TO $i < V.length$
2. Visitado[i] = false
- ➔ 3. FOR $i = 0$ TO $i < V.length$
4. IF (!Visitado[i])
5. DFS-VISIT(G,i)

DFS-VISIT(G,i)

1. Visitado[i] = true
2. FOR each node j adjacent to i
3. IF (!Visitado[j])
4. DFS-VISIT(G,j)



DFS-VISIT(G,0)

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

Grafos

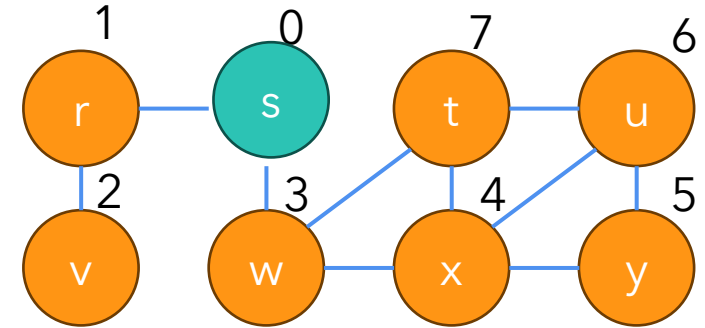
Recorrido en profundidad

DFS(G)

1. FOR $i = 0$ TO $i < V.length$
2. Visitado[i] = false
- ➔ 3. FOR $i = 0$ TO $i < V.length$
4. IF (!Visitado[i])
5. DFS-VISIT(G,i)

DFS-VISIT(G,i)

1. Visitado[i] = true
- ➔ 2. FOR each node j adjacent to i
3. IF (!Visitado[j])
4. DFS-VISIT(G,j)



DFS-VISIT(G,0)

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

Grafos

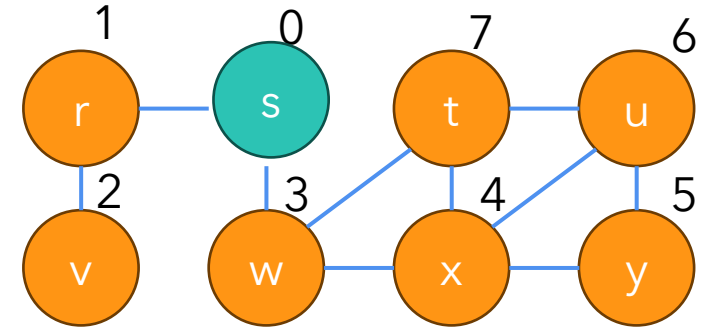
Recorrido en profundidad

DFS(G)

1. FOR $i = 0$ TO $i < V.length$
2. Visitado[i] = false
- ➔ 3. FOR $i = 0$ TO $i < V.length$
4. IF (\neg Visitado[i])
5. DFS-VISIT(G,i)

DFS-VISIT(G,i)

1. Visitado[i] = true
2. FOR each node j adjacent to i
3. IF (\neg Visitado[j])
- ➔ 4. DFS-VISIT(G,j)



DFS-VISIT(G,1)

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

Grafos

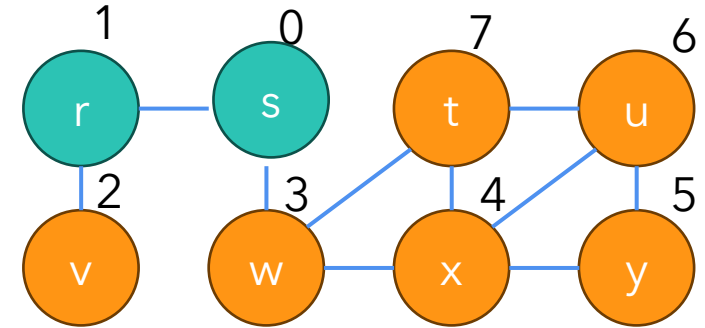
Recorrido en profundidad

DFS(G)

1. FOR $i = 0$ TO $i < V.length$
2. Visitado[i] = false
- ➔ 3. FOR $i = 0$ TO $i < V.length$
4. IF (!Visitado[i])
5. DFS-VISIT(G,i)

DFS-VISIT(G,i)

1. Visitado[i] = true
- ➔ 2. FOR each node j adjacent to i
3. IF (!Visitado[j])
4. DFS-VISIT(G,j)



DFS-VISIT(G,1)

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

Grafos

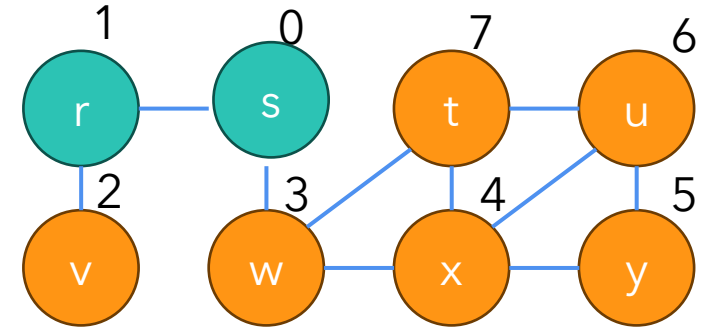
Recorrido en profundidad

DFS(G)

1. FOR $i = 0$ TO $i < V.length$
2. Visitado[i] = false
- ➔ 3. FOR $i = 0$ TO $i < V.length$
4. IF (!Visitado[i])
5. DFS-VISIT(G,i)

DFS-VISIT(G,i)

1. Visitado[i] = true
2. FOR each node j adjacent to i
3. IF (!Visitado[j])
- ➔ 4. DFS-VISIT(G,j)



DFS-VISIT(G,2)

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

Grafos

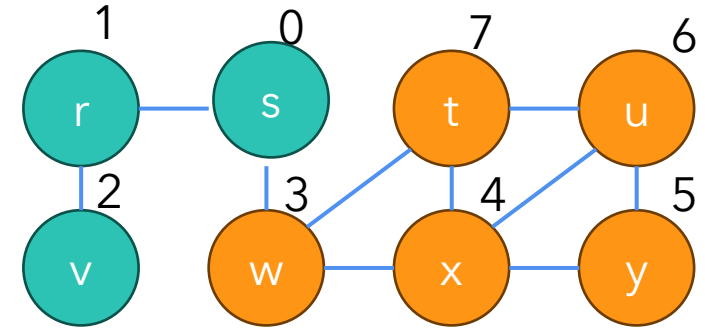
Recorrido en profundidad

DFS(G)

1. FOR $i = 0$ TO $i < V.length$
2. Visitado[i] = false
- ➔ 3. FOR $i = 0$ TO $i < V.length$
4. IF (!Visitado[i])
5. DFS-VISIT(G,i)

DFS-VISIT(G,i)

- ➔ 1. Visitado[i] = true
2. FOR each node j adjacent to i
3. IF (!Visitado[j])
4. DFS-VISIT(G,j)



DFS-VISIT(G,2)

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

Grafos

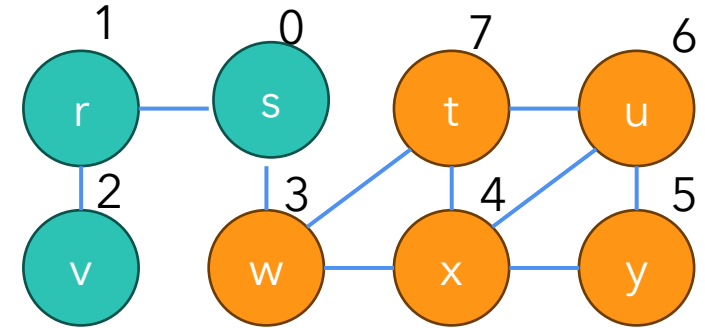
Recorrido en profundidad

DFS(G)

1. FOR $i = 0$ TO $i < V.length$
2. Visitado[i] = false
- ➔ 3. FOR $i = 0$ TO $i < V.length$
4. IF (!Visitado[i])
5. DFS-VISIT(G,i)

DFS-VISIT(G,i)

1. Visitado[i] = true
- ➔ 2. FOR each node j adjacent to i
3. IF (!Visitado[j])
4. DFS-VISIT(G,j)



DFS-VISIT(G,2)

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

Grafos

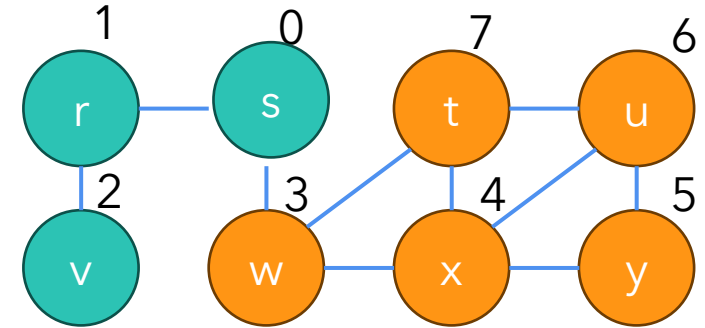
Recorrido en profundidad

DFS(G)

1. FOR $i = 0$ TO $i < V.length$
2. Visitado[i] = false
- 3. FOR $i = 0$ TO $i < V.length$
4. IF (!Visitado[i])
5. DFS-VISIT(G,i)

DFS-VISIT(G,i)

1. Visitado[i] = true
- 2. FOR each node j adjacent to i
3. IF (!Visitado[j])
4. DFS-VISIT(G,j)



DFS-VISIT(G,0)

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

Grafos

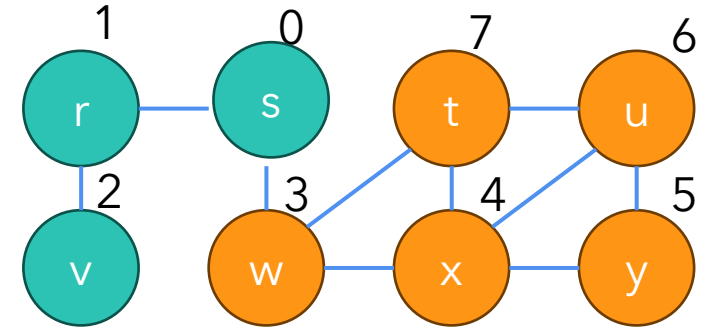
Recorrido en profundidad

DFS(G)

1. FOR $i = 0$ TO $i < V.length$
2. Visitado[i] = false
- ➔ 3. FOR $i = 0$ TO $i < V.length$
4. IF (!Visitado[i])
5. DFS-VISIT(G,i)

DFS-VISIT(G,i)

1. Visitado[i] = true
- ➔ 2. FOR each node j adjacent to i
3. IF (!Visitado[j])
4. DFS-VISIT(G,j)



DFS-VISIT(G,3)

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

Grafos

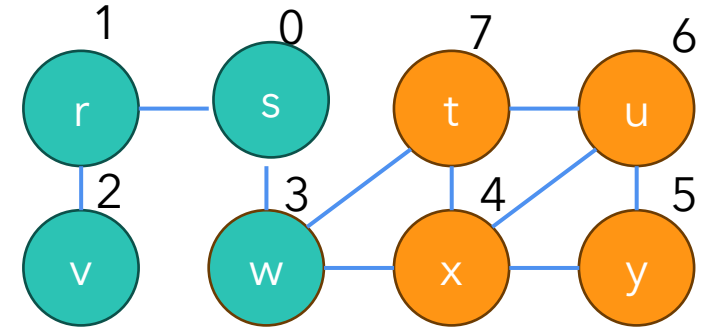
Recorrido en profundidad

DFS(G)

1. FOR $i = 0$ TO $i < V.length$
2. Visitado[i] = false
- ➔ 3. FOR $i = 0$ TO $i < V.length$
4. IF (!Visitado[i])
5. DFS-VISIT(G,i)

DFS-VISIT(G,i)

1. Visitado[i] = true
- ➔ 2. FOR each node j adjacent to i
3. IF (!Visitado[j])
4. DFS-VISIT(G,j)



DFS-VISIT(G,3)

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

Grafos

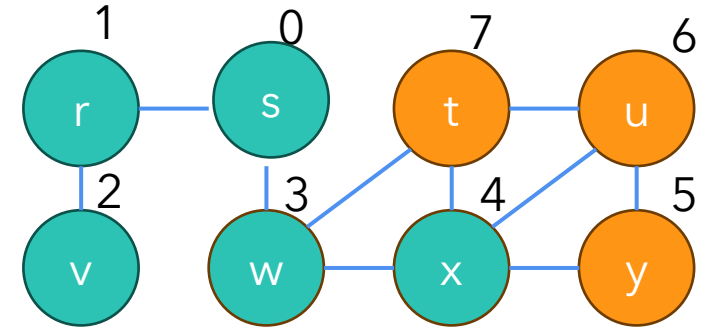
Recorrido en profundidad

DFS(G)

1. FOR $i = 0$ TO $i < V.length$
2. Visitado[i] = false
- ➔ 3. FOR $i = 0$ TO $i < V.length$
4. IF (!Visitado[i])
5. DFS-VISIT(G,i)

DFS-VISIT(G,i)

1. Visitado[i] = true
- ➔ 2. FOR each node j adjacent to i
3. IF (!Visitado[j])
4. DFS-VISIT(G,j)



DFS-VISIT(G,4)

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

Grafos

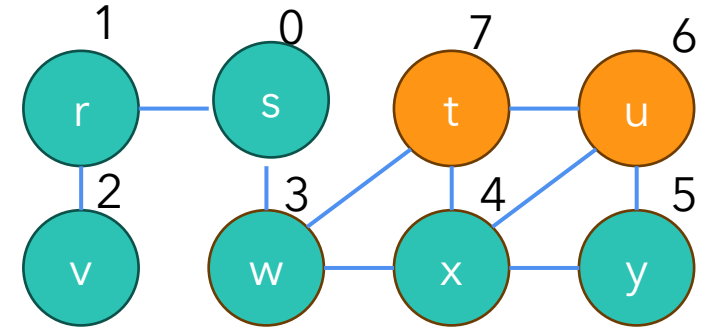
Recorrido en profundidad

DFS(G)

1. FOR $i = 0$ TO $i < V.length$
2. Visitado[i] = false
- ➔ 3. FOR $i = 0$ TO $i < V.length$
4. IF (!Visitado[i])
5. DFS-VISIT(G,i)

DFS-VISIT(G,i)

1. Visitado[i] = true
- ➔ 2. FOR each node j adjacent to i
3. IF (!Visitado[j])
4. DFS-VISIT(G,j)



DFS-VISIT(G,5)

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

Grafos

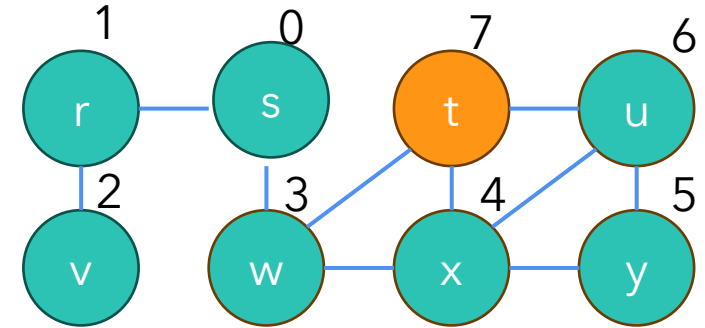
Recorrido en profundidad

DFS(G)

1. FOR $i = 0$ TO $i < V.length$
2. Visitado[i] = false
- ➔ 3. FOR $i = 0$ TO $i < V.length$
4. IF (!Visitado[i])
5. DFS-VISIT(G,i)

DFS-VISIT(G,i)

1. Visitado[i] = true
- ➔ 2. FOR each node j adjacent to i
3. IF (!Visitado[j])
4. DFS-VISIT(G,j)



DFS-VISIT(G,6)

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

Grafos

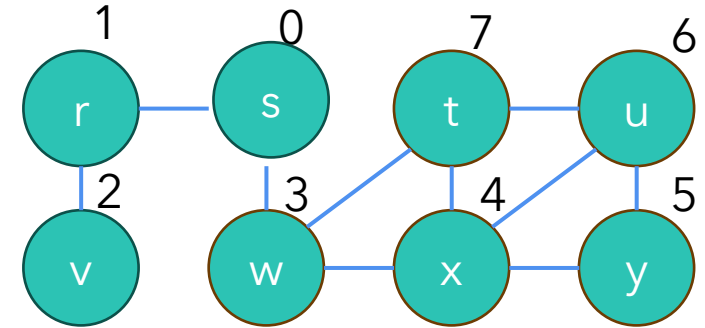
Recorrido en profundidad

DFS(G)

1. FOR $i = 0$ TO $i < V.length$
2. Visitado[i] = false
- ➔ 3. FOR $i = 0$ TO $i < V.length$
4. IF (!Visitado[i])
5. DFS-VISIT(G,i)

DFS-VISIT(G,i)

1. Visitado[i] = true
- ➔ 2. FOR each node j adjacent to i
3. IF (!Visitado[j])
4. DFS-VISIT(G,j)



DFS-VISIT(G,7)

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

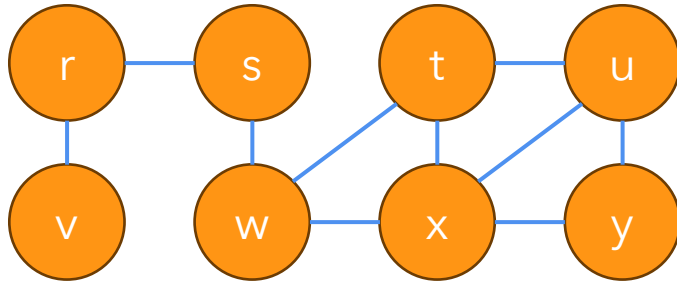
Grafos

Recorrido en anchura

- ❑ Se empieza visitando un nodo seleccionado de forma aleatoria.
- ❑ Luego se visitan todos sus vértices adyacentes
- ❑ A continuación, los adyacentes a estos y así sucesivamente
- ❑ El algoritmo utiliza una cola para almacenar los vértices

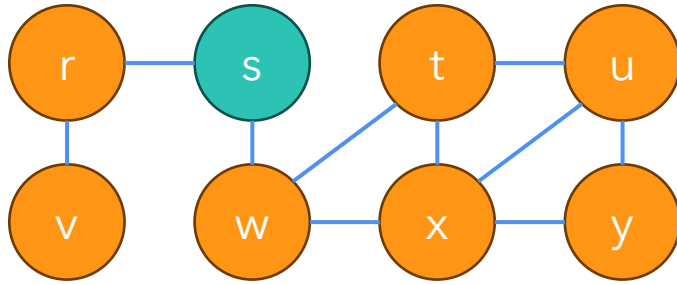
Grafos

Recorrido en anchura

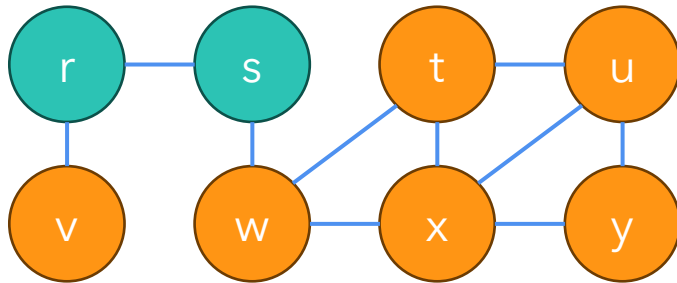


Grafos

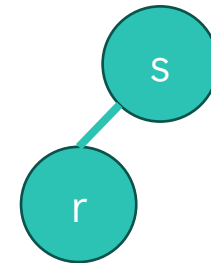
Recorrido en anchura



Cola $Q = \{ s \}$

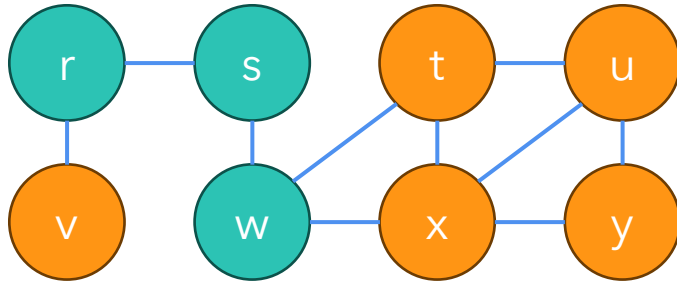


Cola $Q = \{ r, w \}$

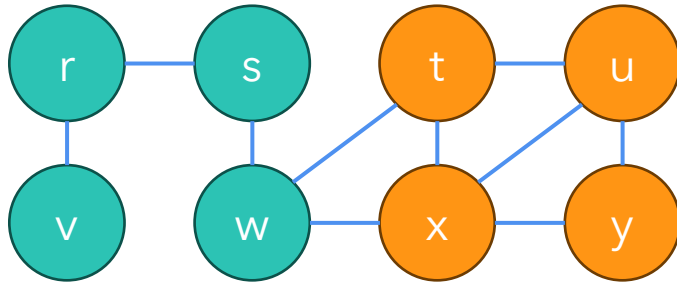
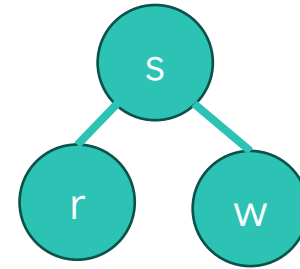


Grafos

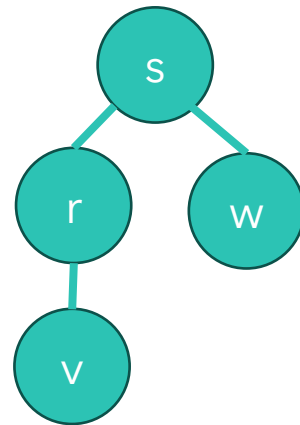
Recorrido en anchura



Cola $Q = \{ w, v \}$

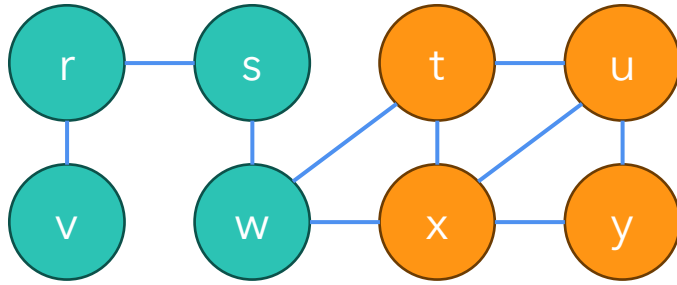


Cola $Q = \{ v, s, t, x \}$

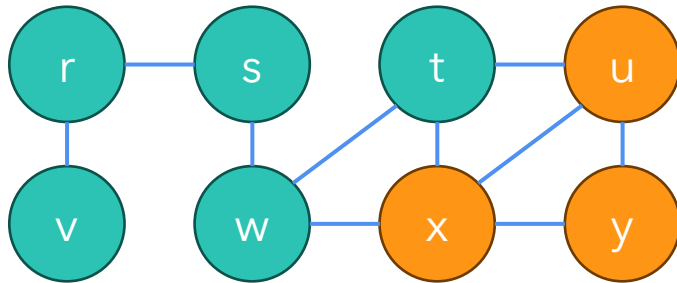


Grafos

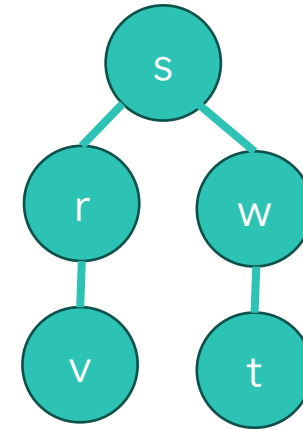
Recorrido en anchura



Cola $Q = \{s, t, x\}$

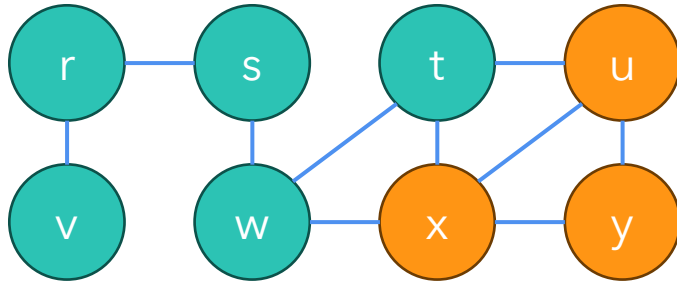


Cola $Q = \{t, x\}$

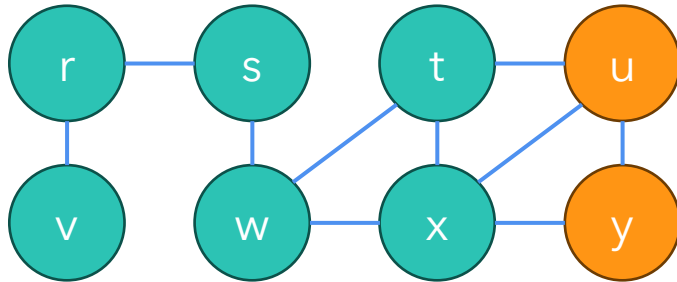


Grafos

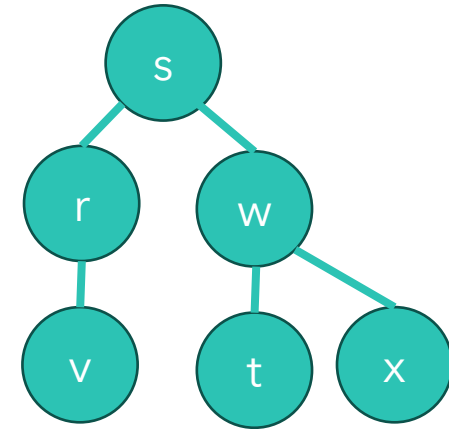
Recorrido en anchura



Cola $Q = \{x, w, x, u\}$

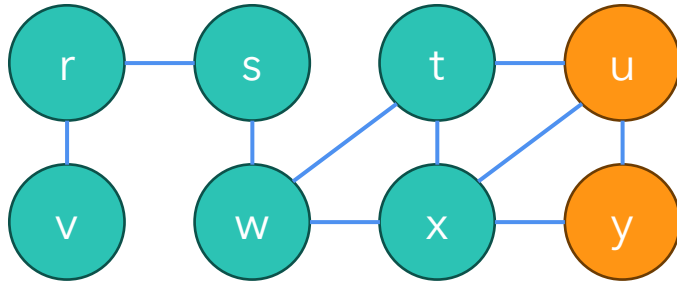


Cola $Q = \{w, x, u, w, t, u, y\}$

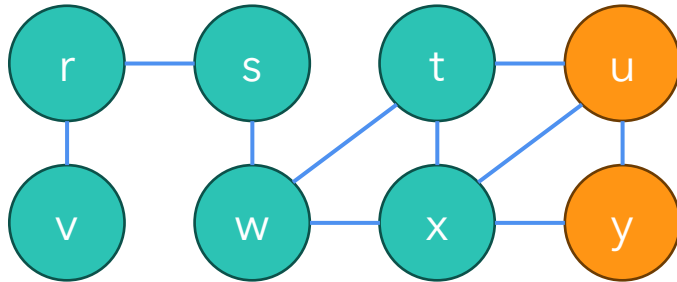


Grafos

Recorrido en anchura



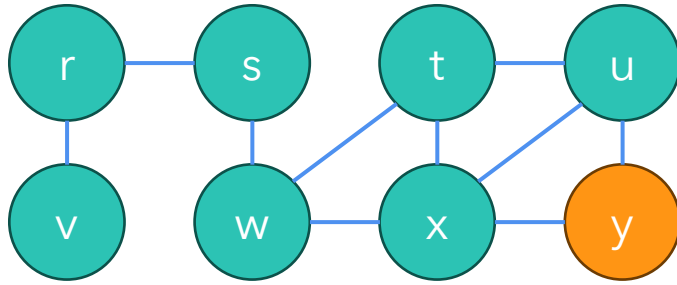
Cola $Q = \{x, u, w, t, u, y\}$



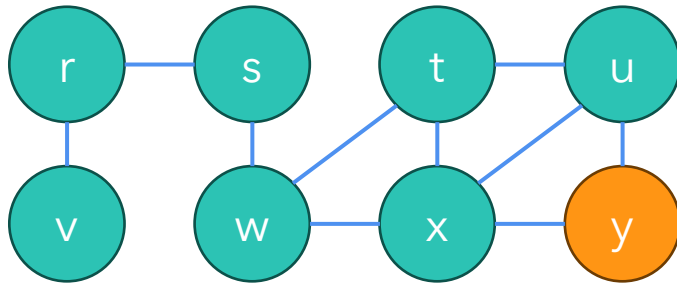
Cola $Q = \{u, w, t, u, y\}$

Grafos

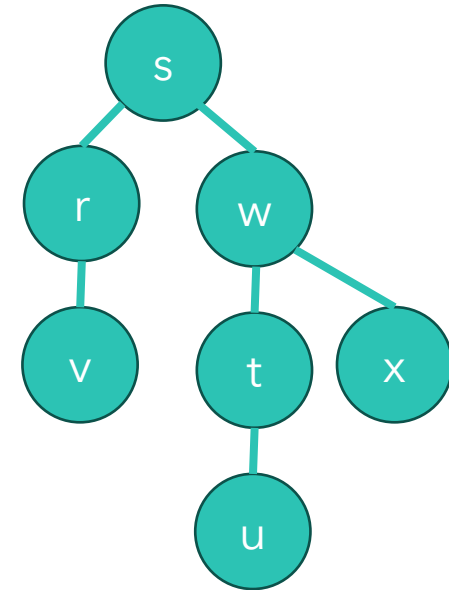
Recorrido en anchura



Cola $Q = \{w, t, u, y\}$

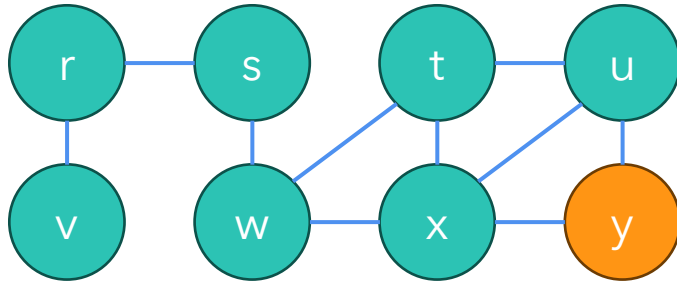


Cola $Q = \{w, t, u, y, t, x, y\}$

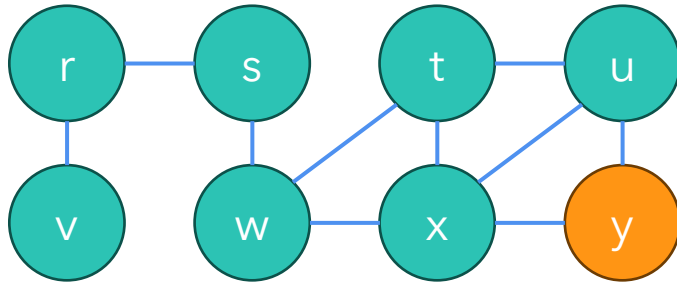


Grafos

Recorrido en anchura



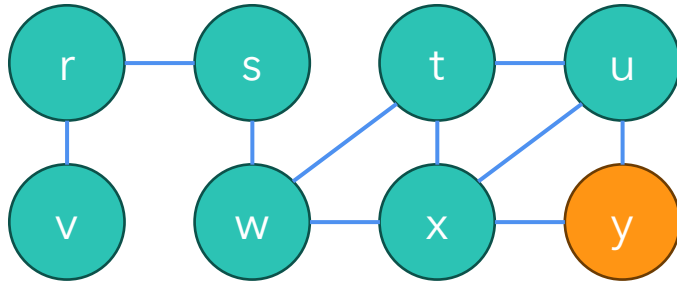
Cola Q = {t, u, y, t, x, y}



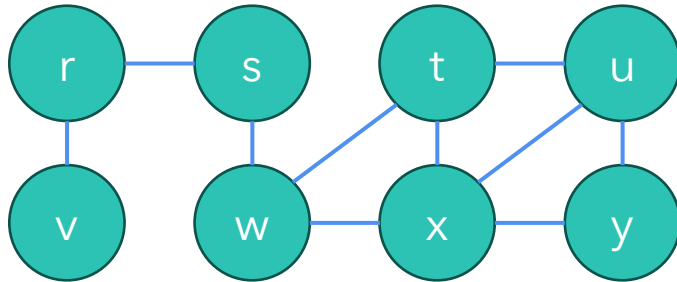
Cola Q = { u, y, t, x, y}

Grafos

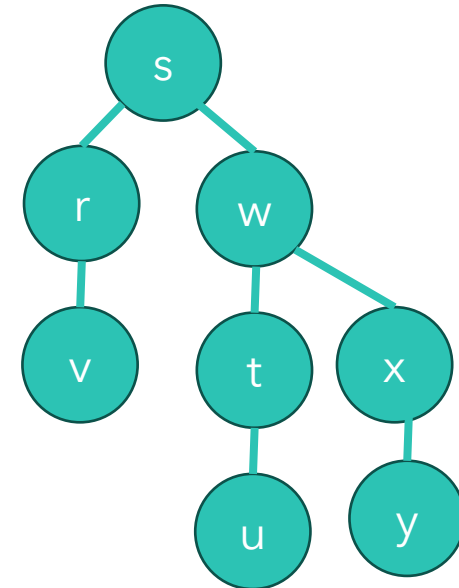
Recorrido en anchura



Cola $Q = \{y, t, x, y\}$

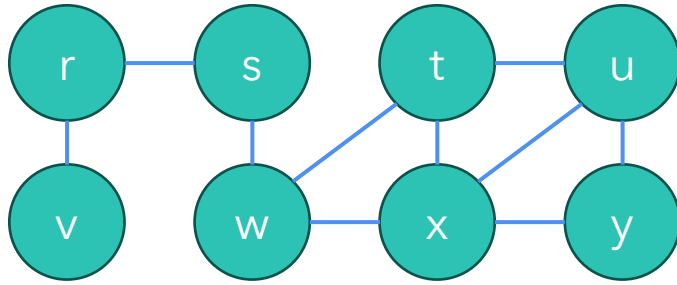


Cola $Q = \{t, x, y, u, x\}$

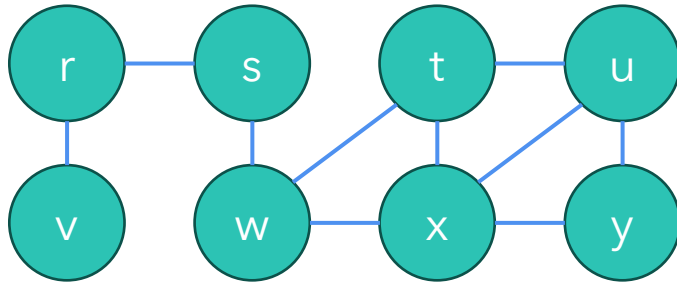


Grafos

Recorrido en anchura



Cola $Q = \{t, x, y, u, x\}$



Cola $Q = \{ \}$

Grafos

Recorrido en anchura

BFS(G)

1. FOR $i = 0$ TO $i < V.length$
2. $Visitado[i] = false$
3. FOR $i = 0$ TO $i < V.length$
4. IF ($!Visitado[i]$)
5. BFS-VISIT(G,i)

BFS-VISIT(G,i)

1. Queue Q = new Queue()
2. $Visitado[i] = true$
3. Q.enqueue(vi)
4. WHILE ($!Q.isEmpty()$)
5. X = Q.dequeue();
6. FOR each node j adjacent to X
7. IF ($!Visitado[j]$)
8. $Visitado[j] = true$
9. Q.enqueue(vj)

Grafos

Tiempos de ejecución

- ❑ Usando la representación por listas de adyacencia

Recorrido en profundidad (DFS)	$O(V + E)$
Recorrido en anchura (BFS)	$O(V + E)$

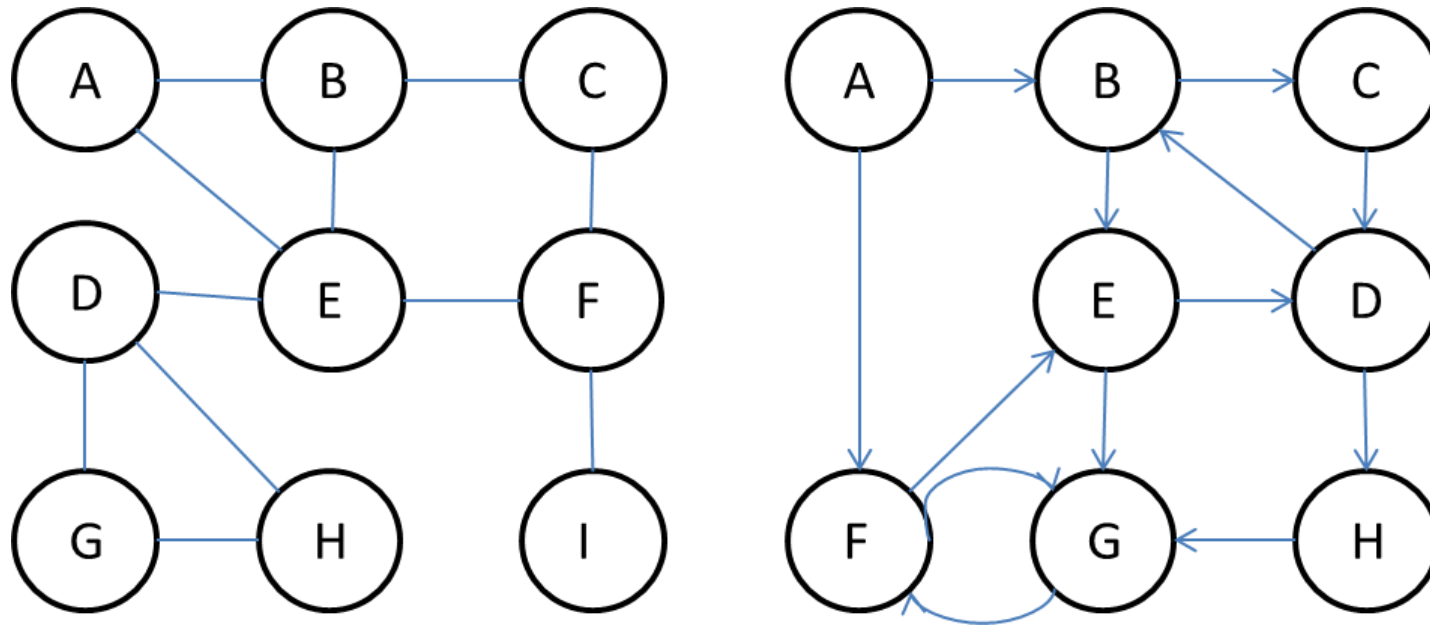
Nota

- ❑ El DFS permite encontrar los nodos conectados al grafo, resolver rompecabezas, encontrar nodos fuertemente conectados
- ❑ El BFS permite encontrar el camino más corto entre 2 nodos medido por el número de aristas, probar si un grafo es bipartito, en sistemas de navegación GPS permite encontrar localizaciones vecinas

Grafos


Ejemplo

❑ Vamos a realizar los recorridos DFS y BFS para los siguientes grafos





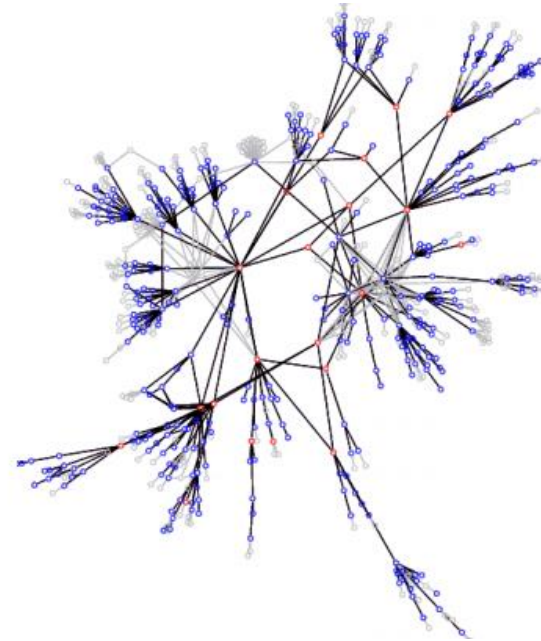
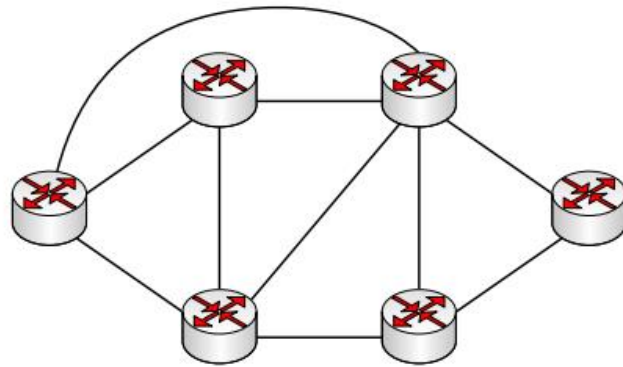
Grafos

- 
- ☐ Listas y matrices de adyacencia
 - ☐ Recorrido
 - ☐ **Árbol de expansión mínima**
 - ☐ Algoritmos del camino más corto

Grafos

Árbol de expansión mínima - Minimum Spanning Tree:

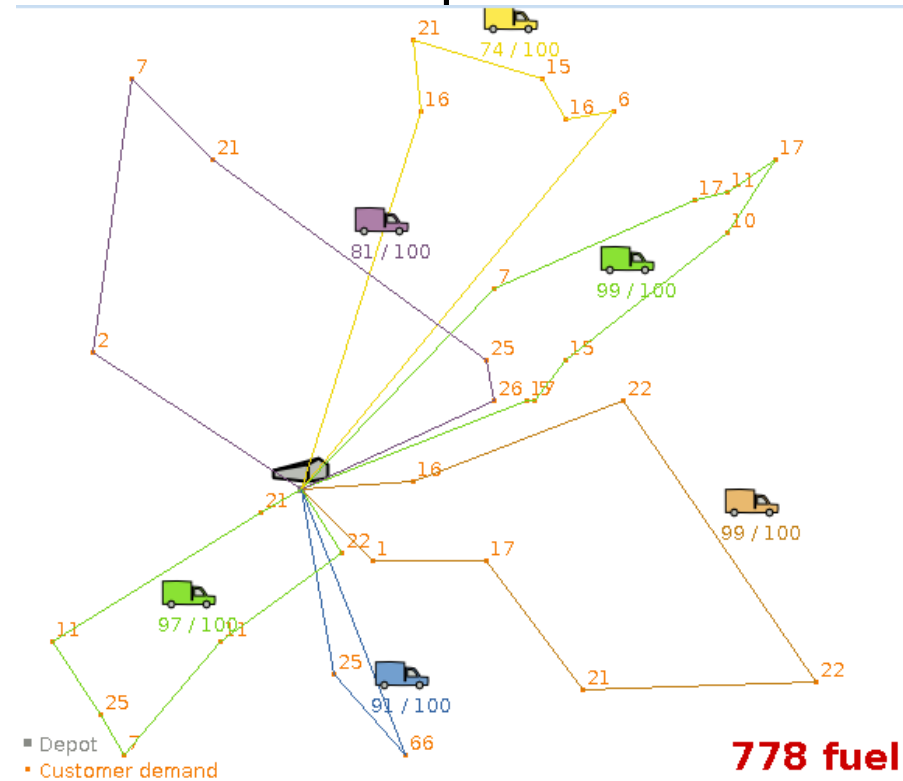
- ❑ Ejemplo de aplicación: Enviar un mensaje a todos los nodos de una red. Los nodos corresponden a los routers de la red y las aristas corresponden a los enlaces físicos entre ellos.



Grafos

Árbol de expansión mínima - Minimum Spanning Tree:

- Ejemplo de aplicación: Entrega de correspondencia exprés. Los nodos corresponden a los puntos de entrega.



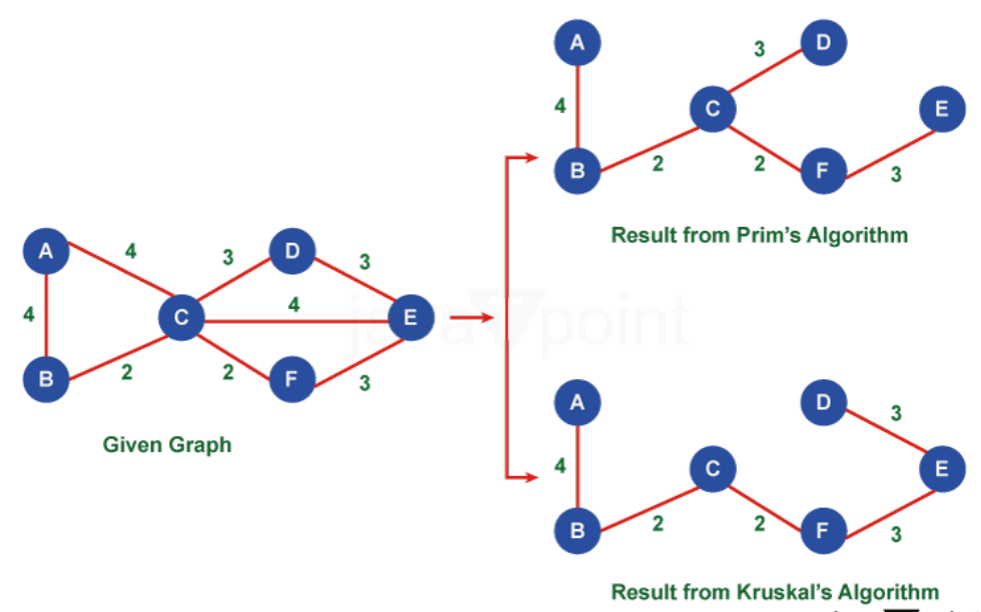
Grafos

Árbol de expansión mínima - Minimum Spanning Tree:

Dado un grafo pesado $G=(V,E)$ sin dirección y sin componentes desconectados:

Generic_MST(G,w)

1. $A = \text{null}$
2. WHILE A no sea un árbol de expansión
3. Encuentre una arista (u,v) segura para
4. $A = AU(u,v)$
5. return A

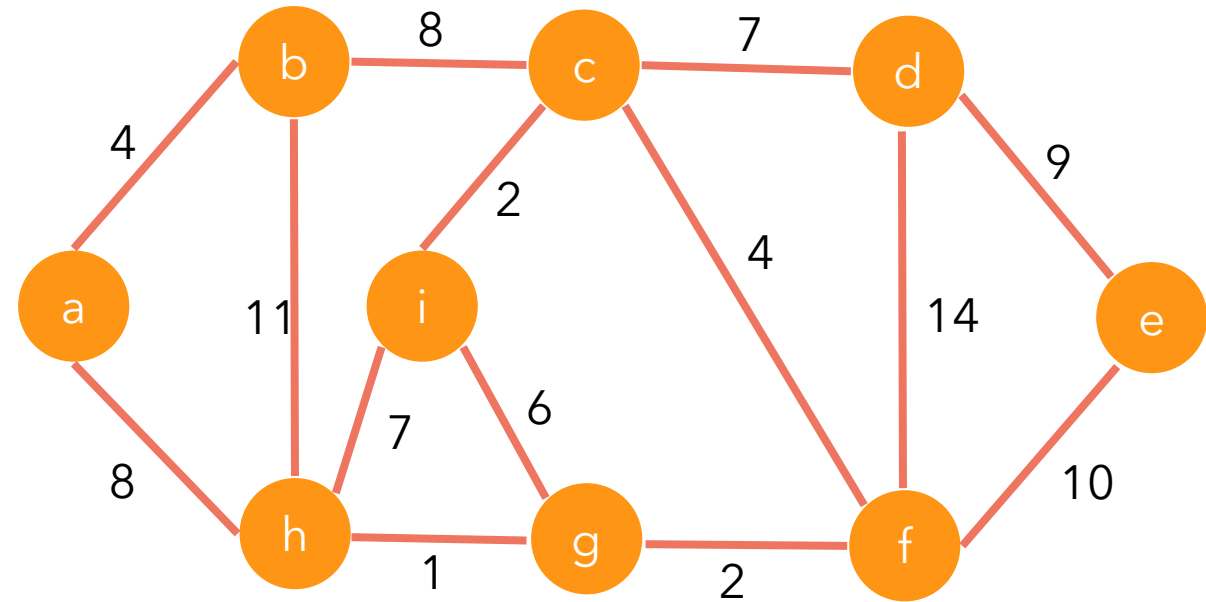


- ❑ Varios algoritmos implementan esta idea genérica:
 - Kruskal
 - Prim
- ❑ Difieren en la forma en que seleccionan la arista segura

Grafos

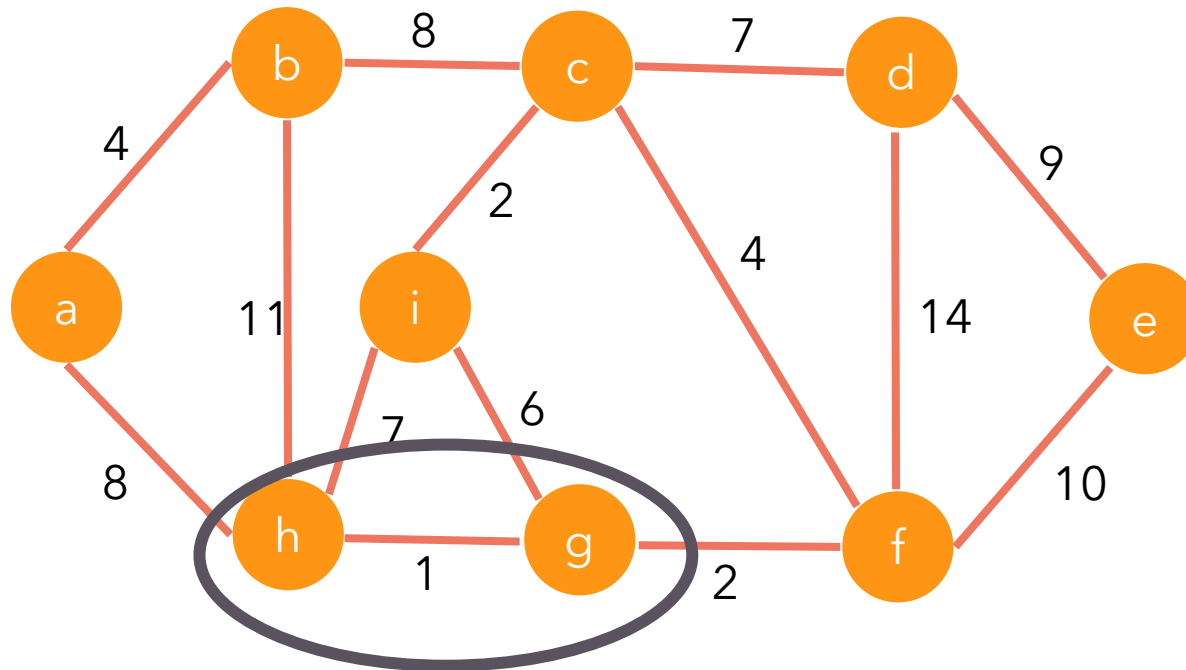
Minimum Spanning Tree - Kruskal

- ❑ Se marca la arista con menor peso. Si hay más de una, se elige cualquiera de ellas.
- ❑ De las aristas restantes, se marca la que tenga menor valor, si hay más de una, se elige cualquiera de ellas.
- ❑ Repetir el paso 2 siempre que la arista elegida no forme un ciclo con las ya seleccionadas.
- ❑ El proceso termina cuando todos los nodos están en alguna de las aristas seleccionadas.



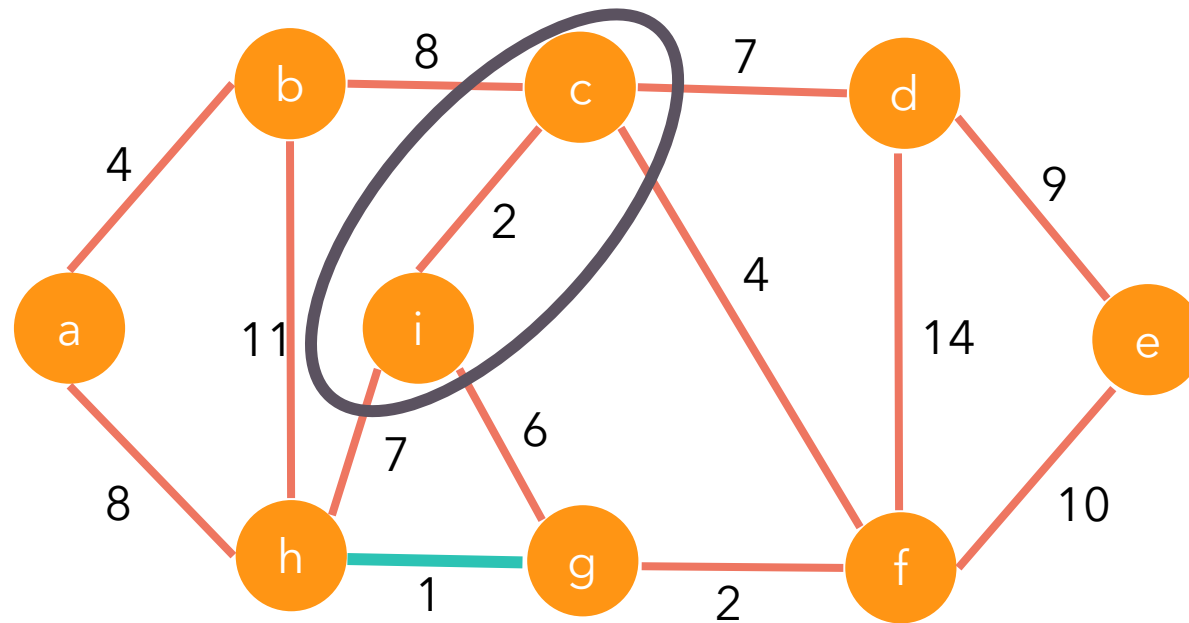
Grafos

Minimum Spanning Tree - Kruskal



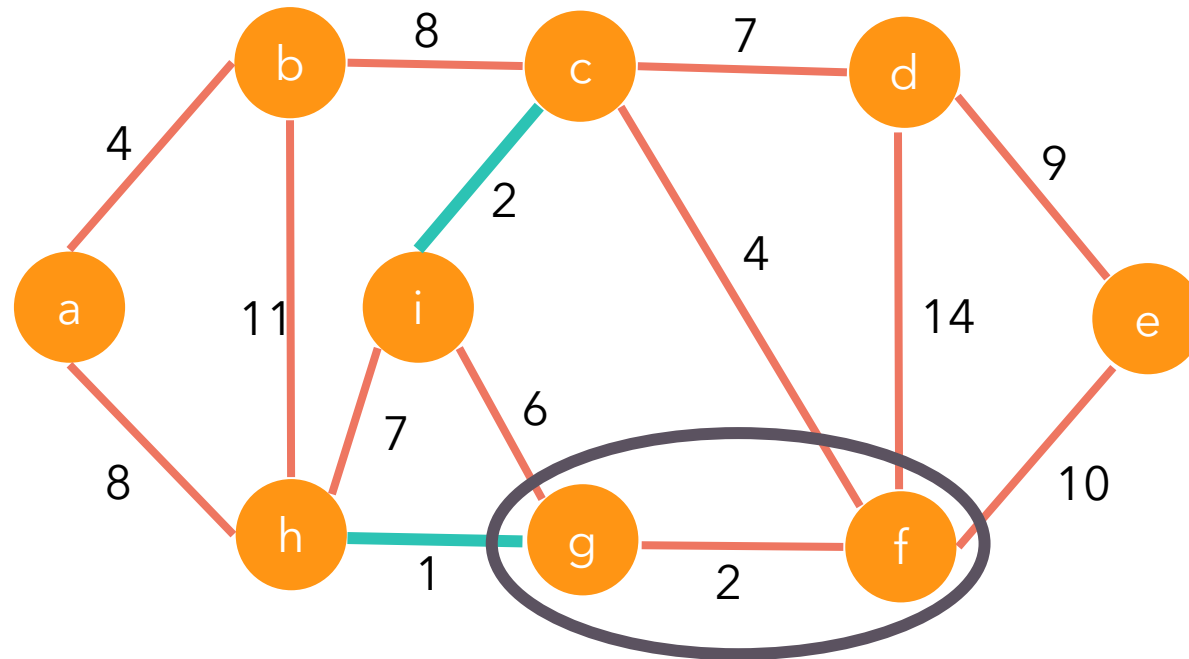
Grafos

Minimum Spanning Tree - Kruskal



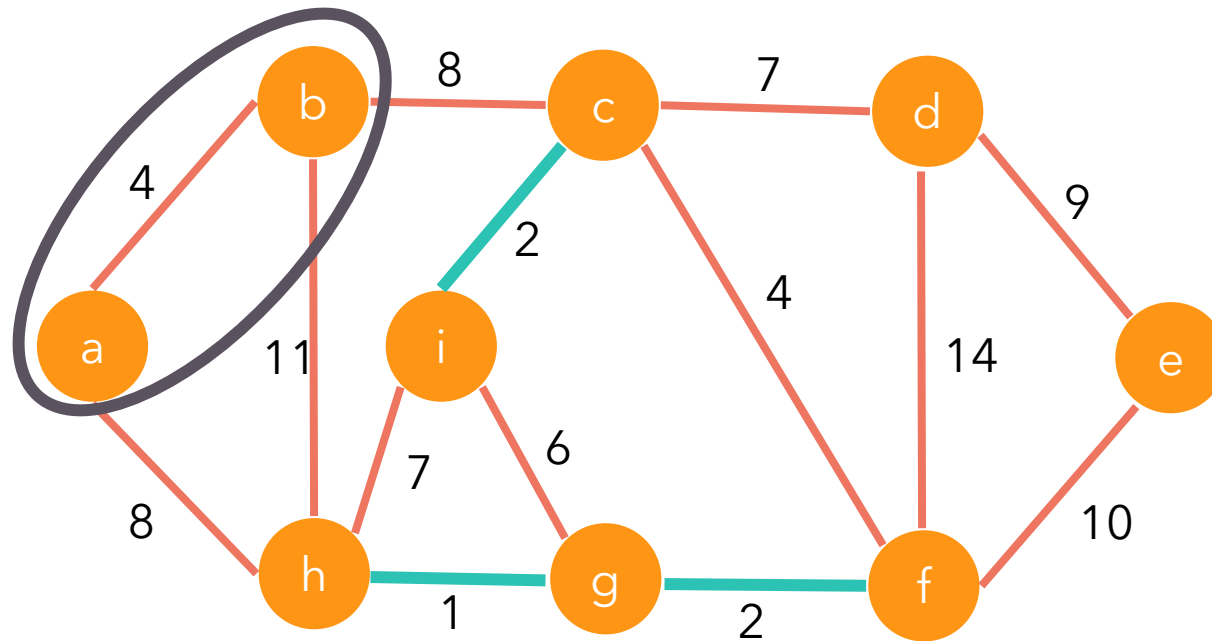
Grafos

Minimum Spanning Tree - Kruskal



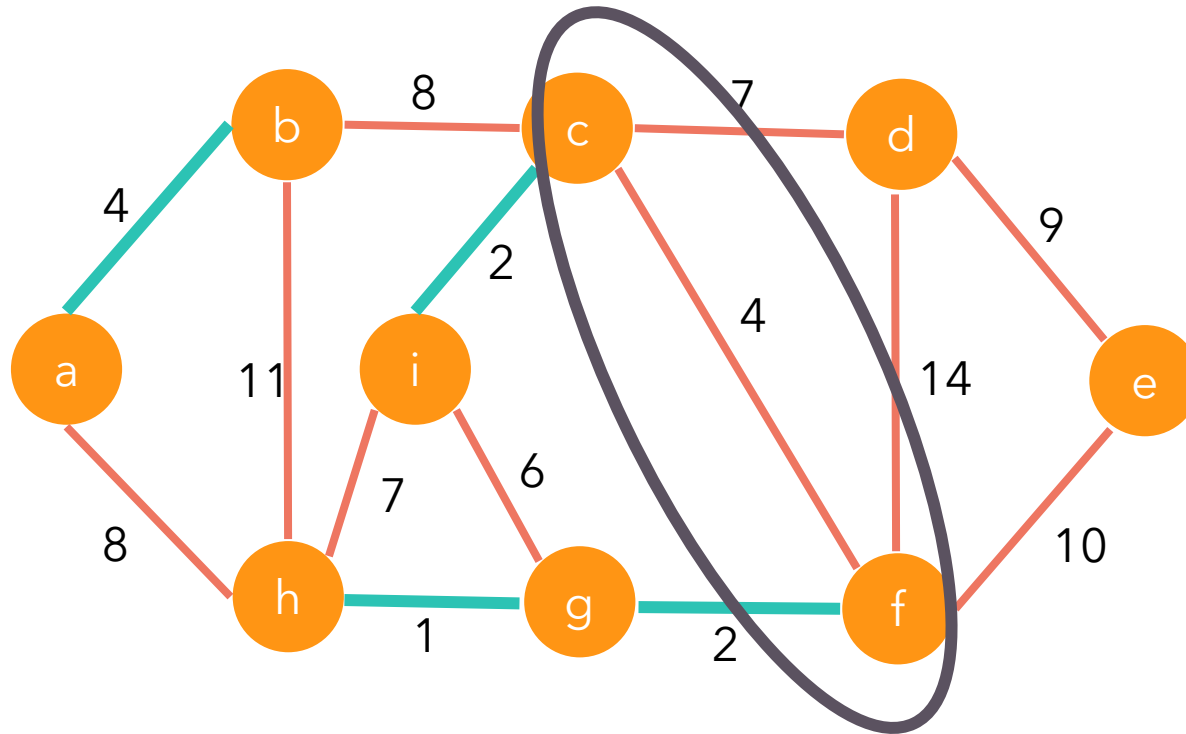
Grafos

Minimum Spanning Tree - Kruskal



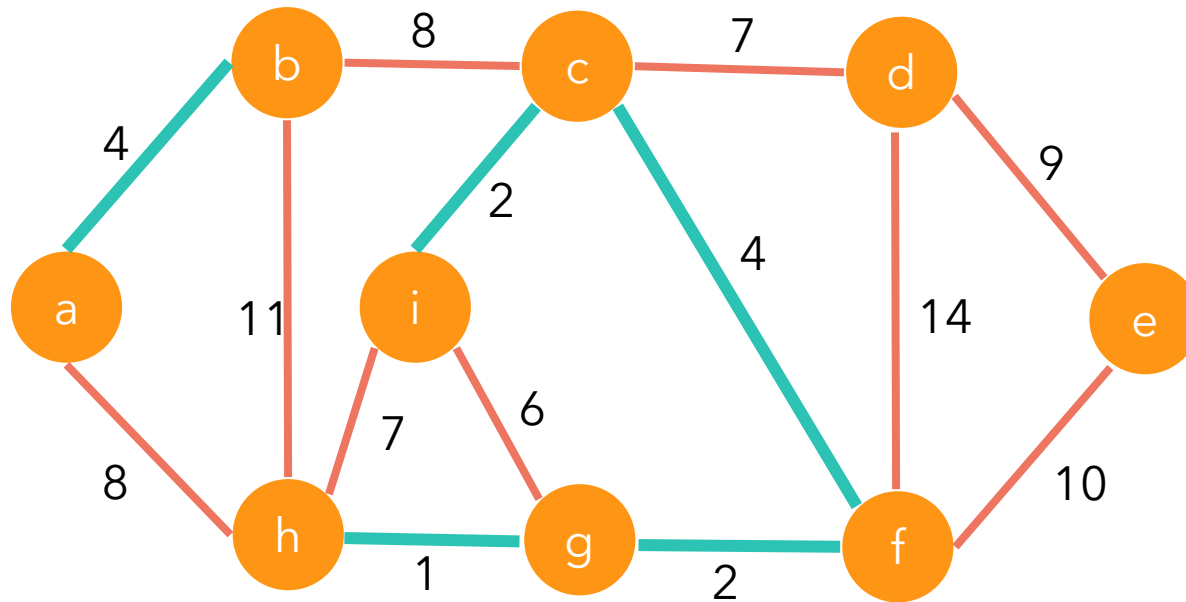
Grafos

Minimum Spanning Tree - Kruskal



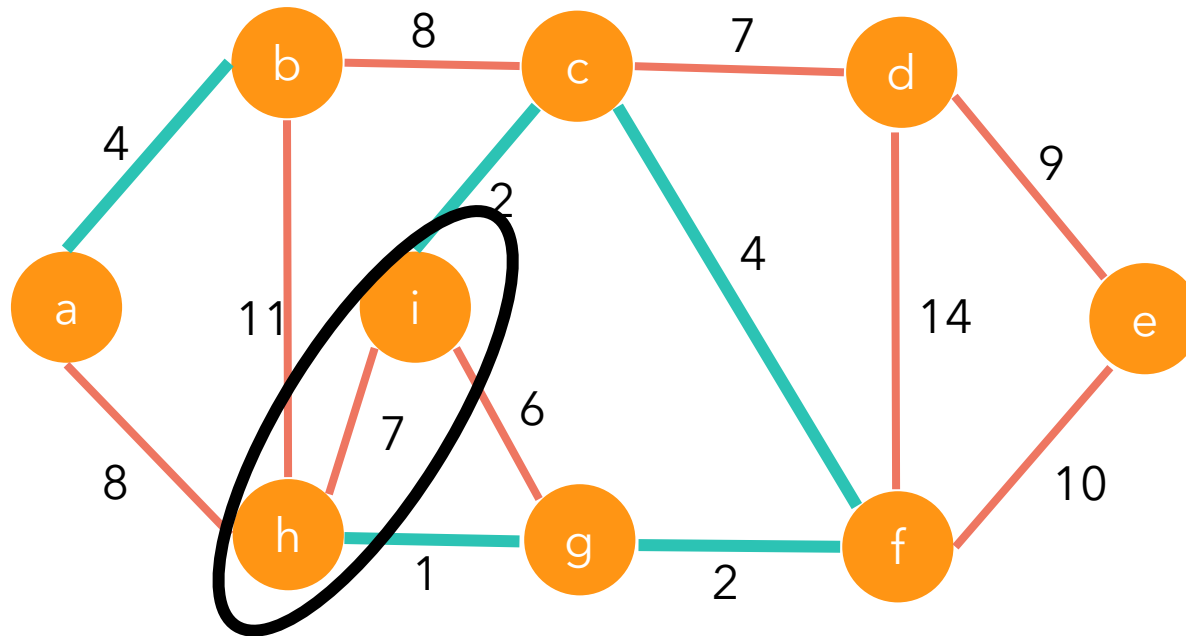
Grafos

Minimum Spanning Tree - Kruskal



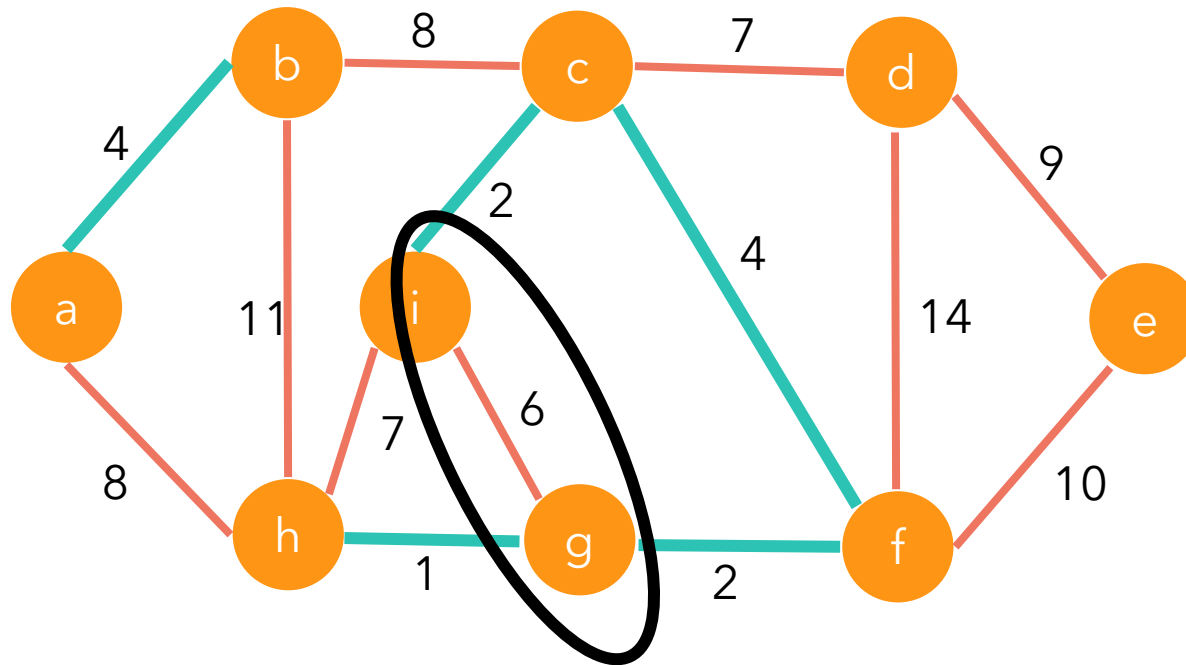
Grafos

Minimum Spanning Tree - Kruskal



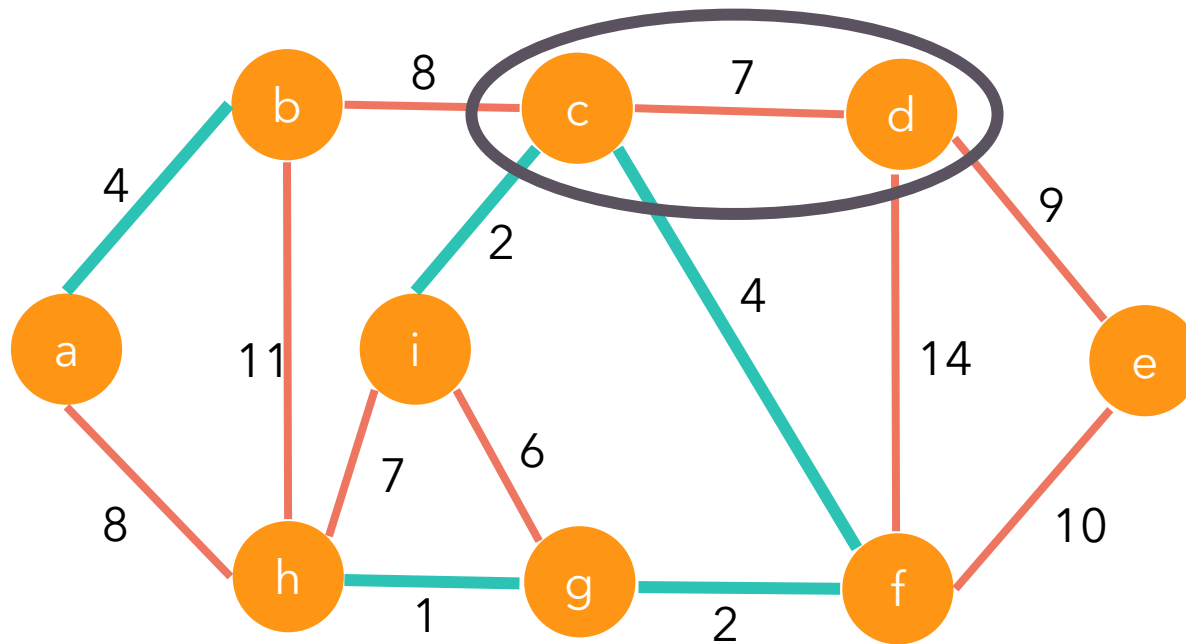
Grafos

Minimum Spanning Tree - Kruskal



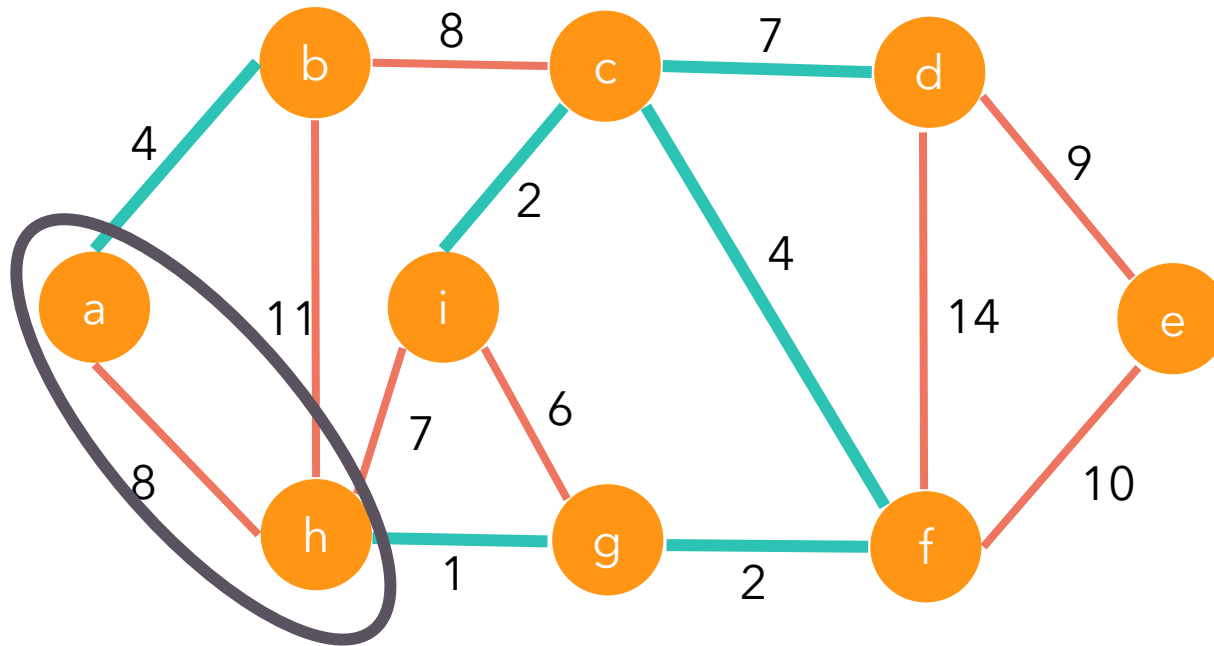
Grafos

Minimum Spanning Tree - Kruskal



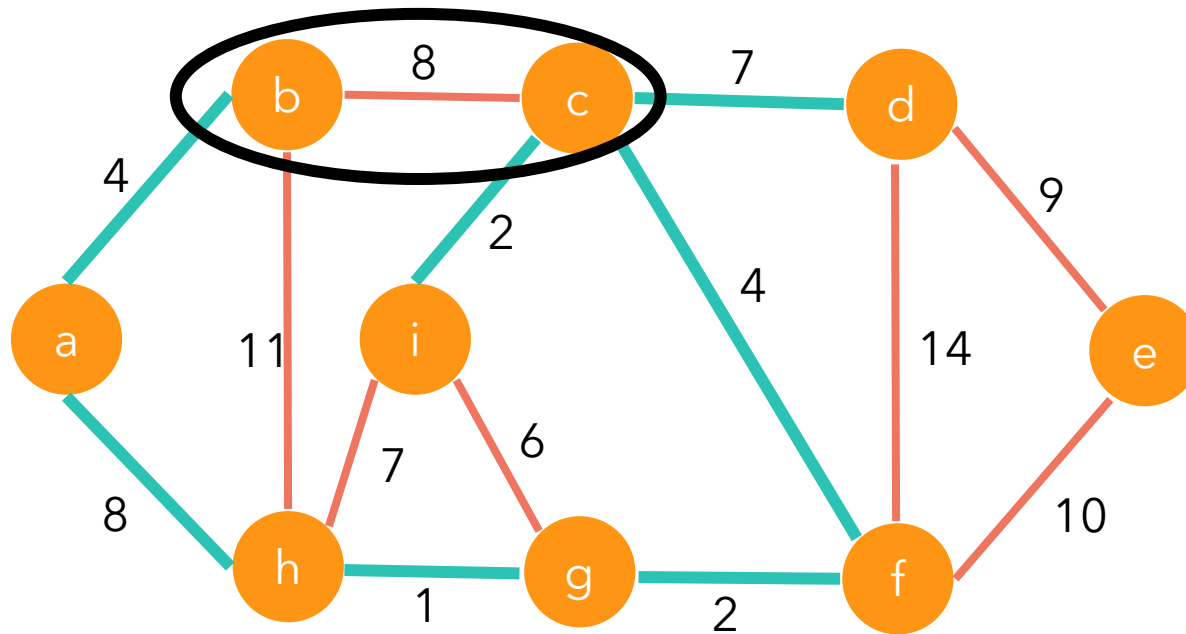
Grafos

Minimum Spanning Tree - Kruskal



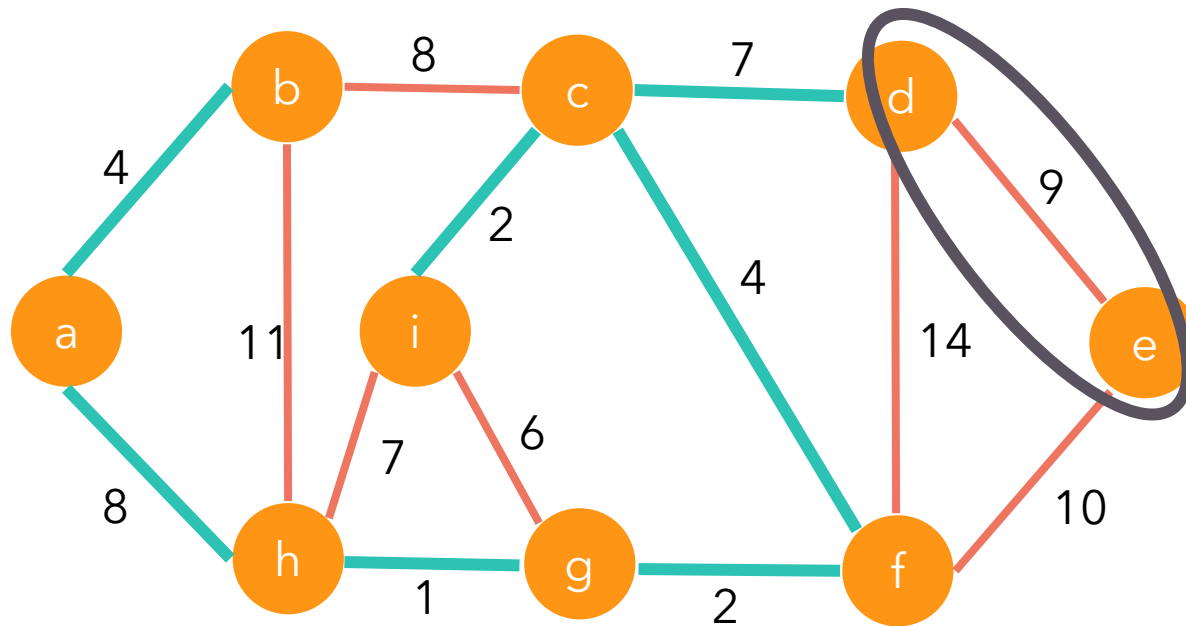
Grafos

Minimum Spanning Tree - Kruskal



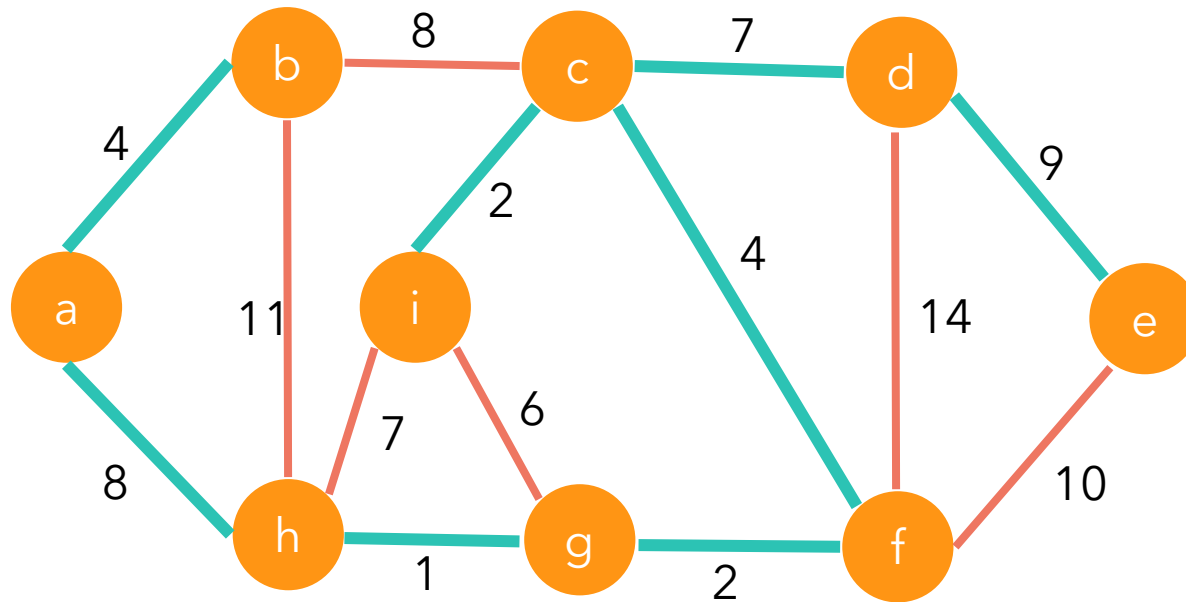
Grafos

Minimum Spanning Tree - Kruskal



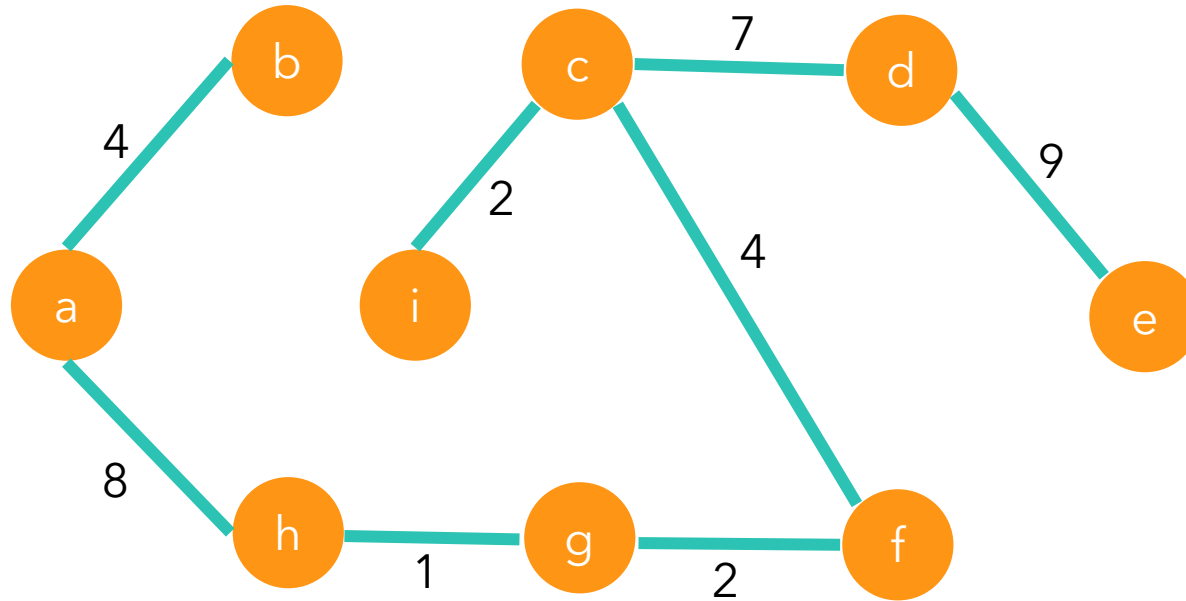
Grafos

Minimum Spanning Tree - Kruskal



Grafos

Minimum Spanning Tree - Kruskal



Grafos

Minimum Spanning Tree - Kruskal

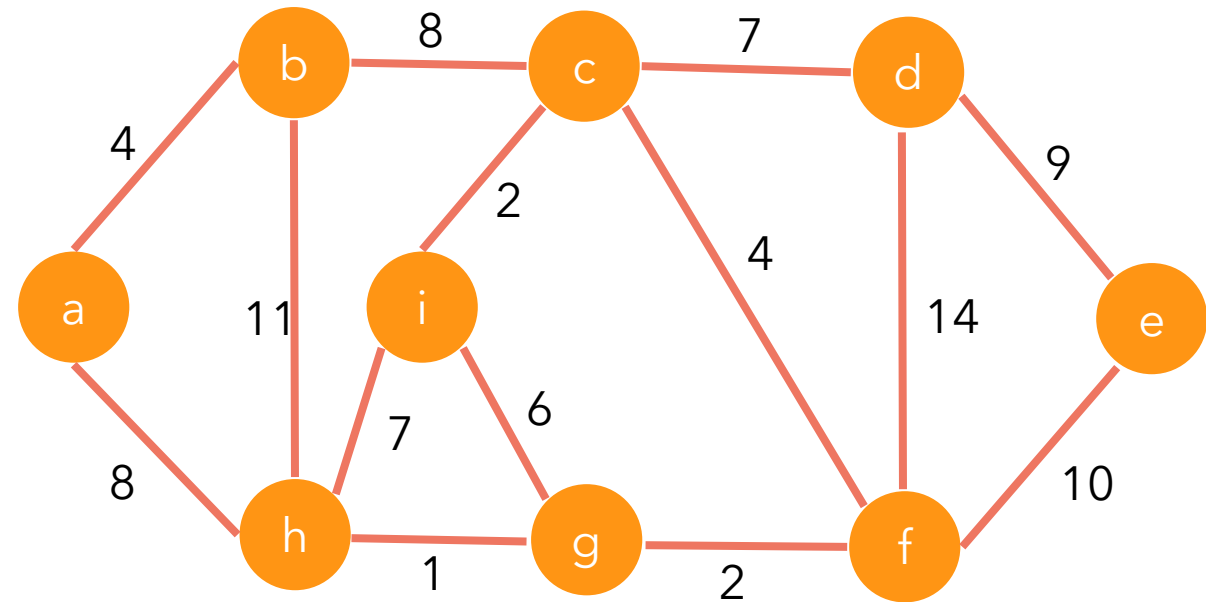
MST-KRUSKAL(G, w)

```
1   $A = \emptyset$ 
2  for each vertex  $v \in G.V$ 
3      MAKE-SET( $v$ )
4  sort the edges of  $G.E$  into nondecreasing order by weight  $w$ 
5  for each edge  $(u, v) \in G.E$ , taken in nondecreasing order by weight
6      if FIND-SET( $u$ )  $\neq$  FIND-SET( $v$ )
7           $A = A \cup \{(u, v)\}$ 
8          UNION( $u, v$ )
9  return  $A$ 
```

Grafos

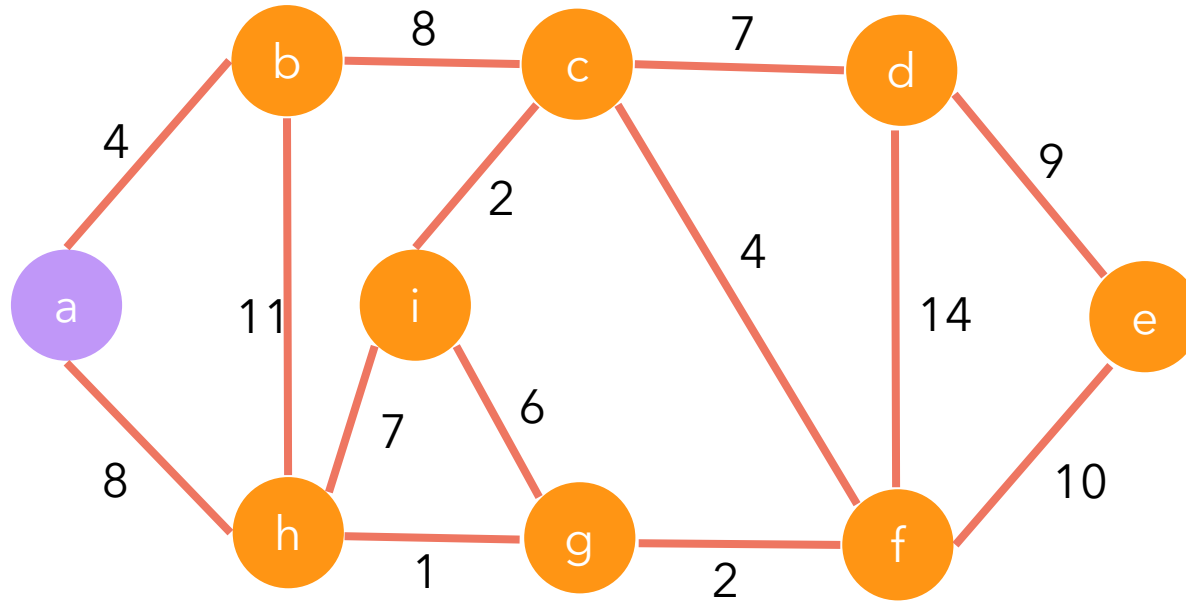
Minimum Spanning Tree - Prim

- ❑ Incrementa continuamente el tamaño de un árbol, comenzando con un vértice inicia al que se le agregan sucesivamente vértices cuya distancia a los anteriores es mínima.
- ❑ En cada paso, las aristas a considerar son aquellas que inciden en vértices que ya pertenecen al árbol.
- ❑ Algoritmos de Prim
- ❑ Se elige un vértice u de G y se considera el árbol $S = \{u\}$
- ❑ Se considera la arista e de mínimo peso que une un vértice en S y un vértice que no pertenece a S
- ❑ Si el número de aristas de T es $n-1$ el algoritmo termina, sino repite el paso 2.



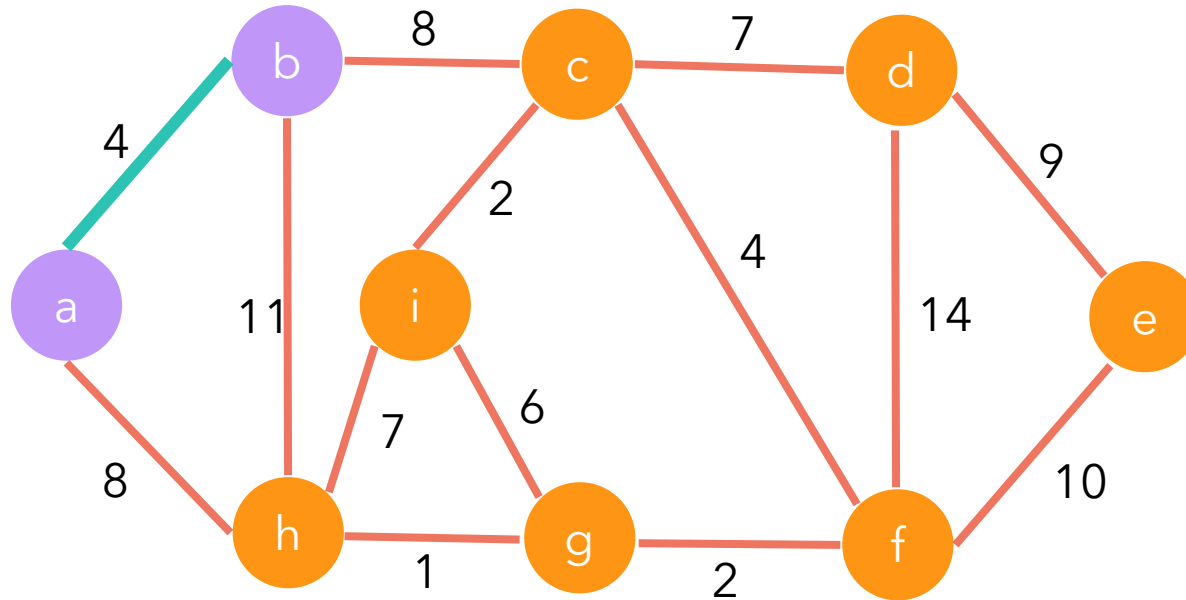
Grafos

Minimum Spanning Tree - Prim



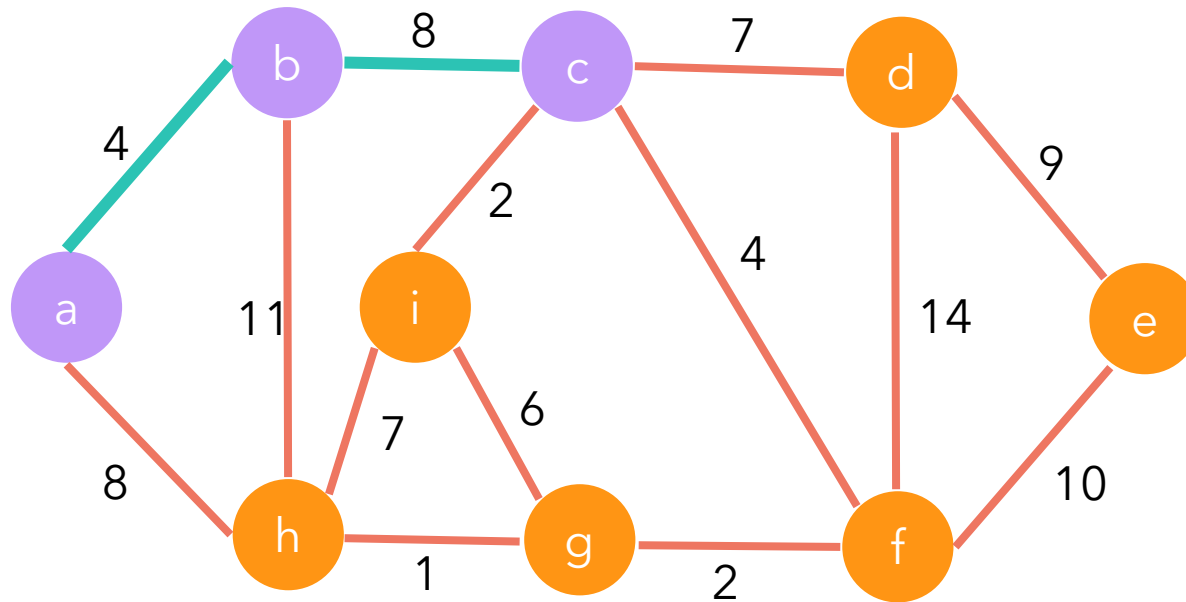
Grafos

Minimum Spanning Tree - Prim



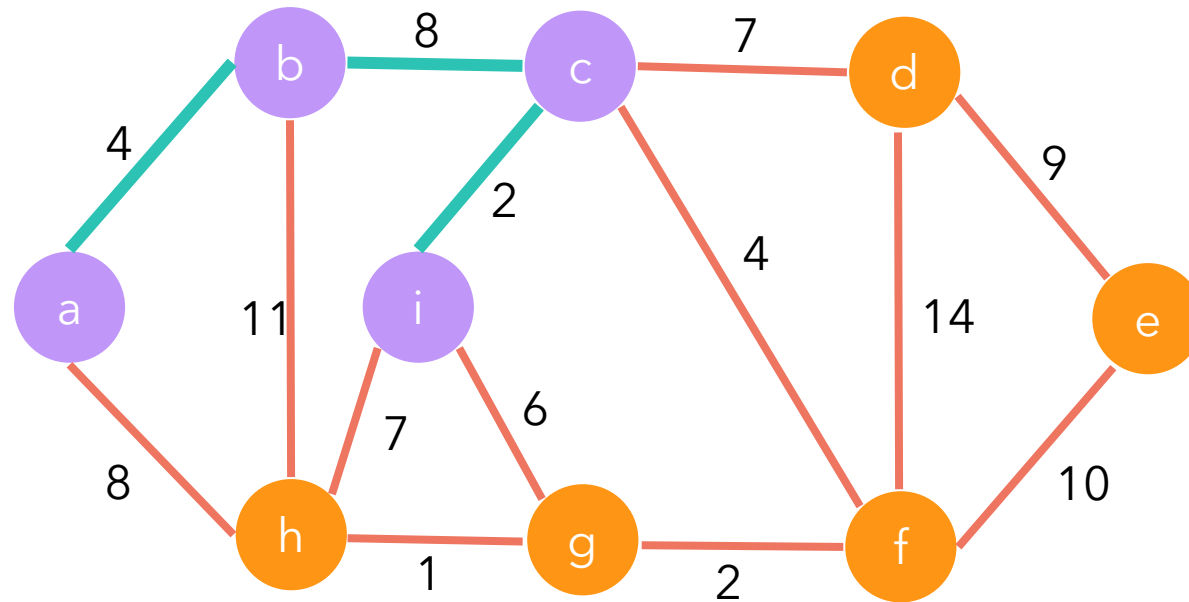
Grafos

Minimum Spanning Tree - Prim



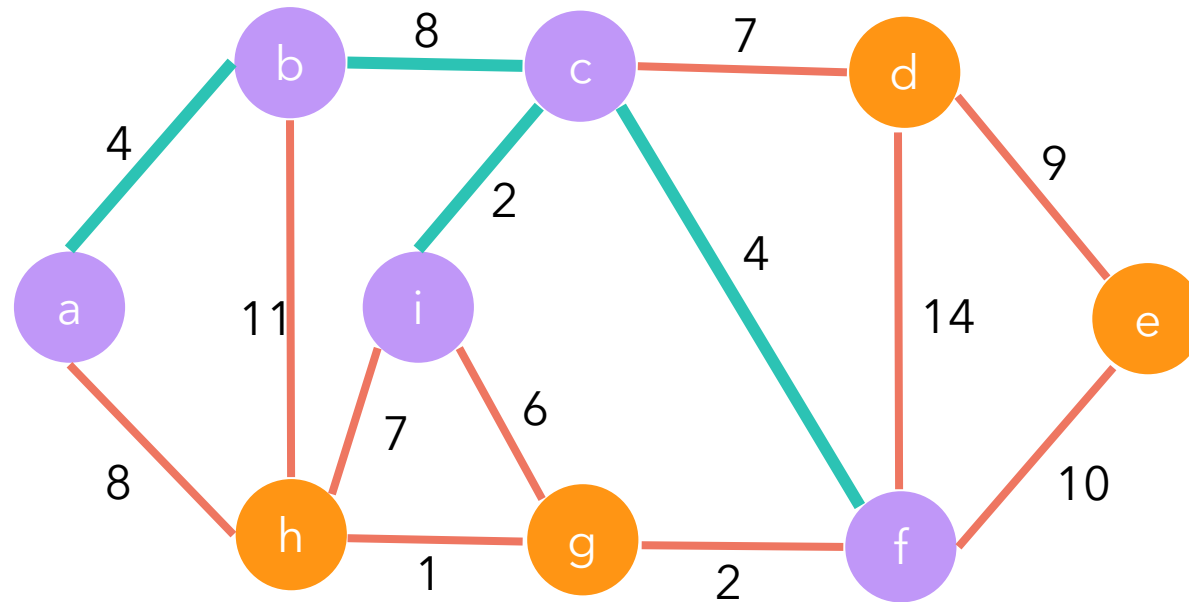
Grafos

Minimum Spanning Tree - Prim



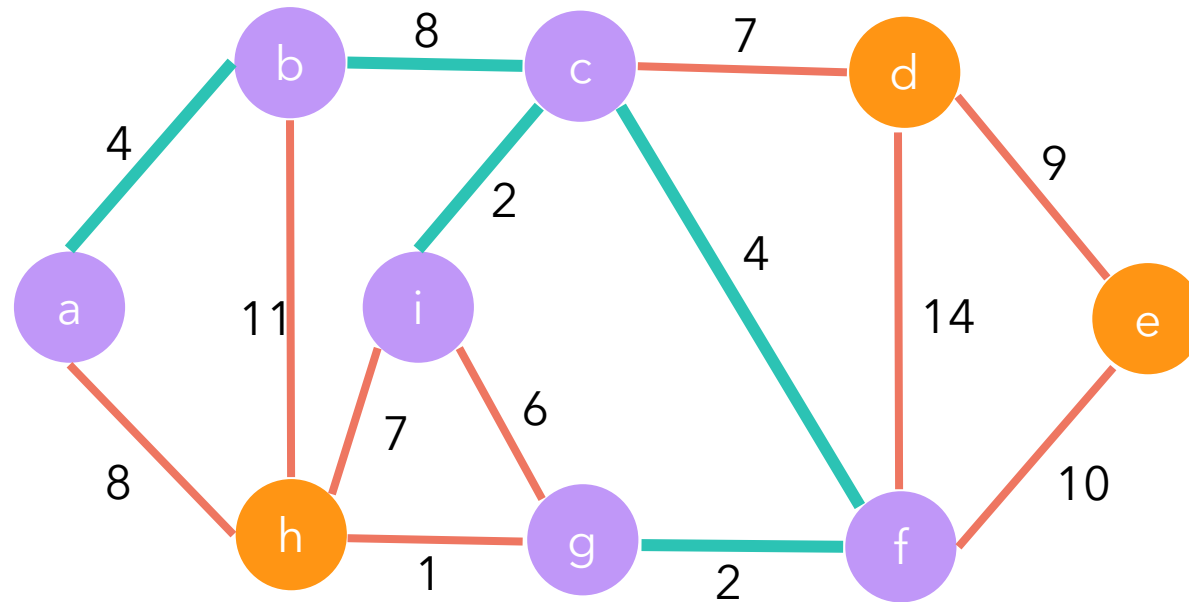
Grafos

Minimum Spanning Tree - Prim



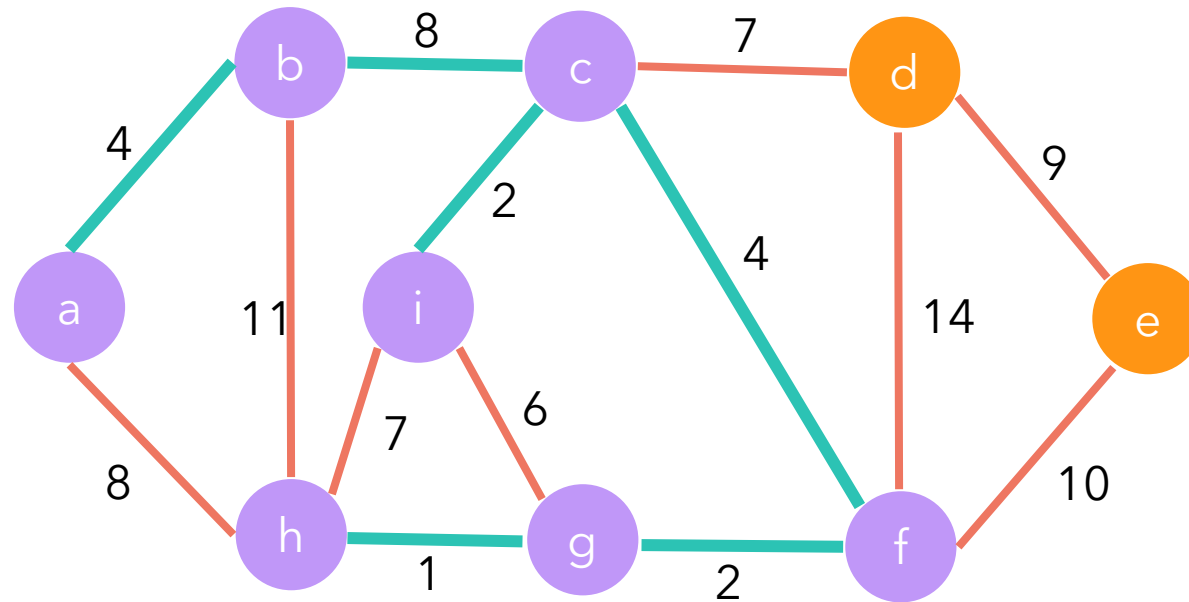
Grafos

Minimum Spanning Tree - Prim



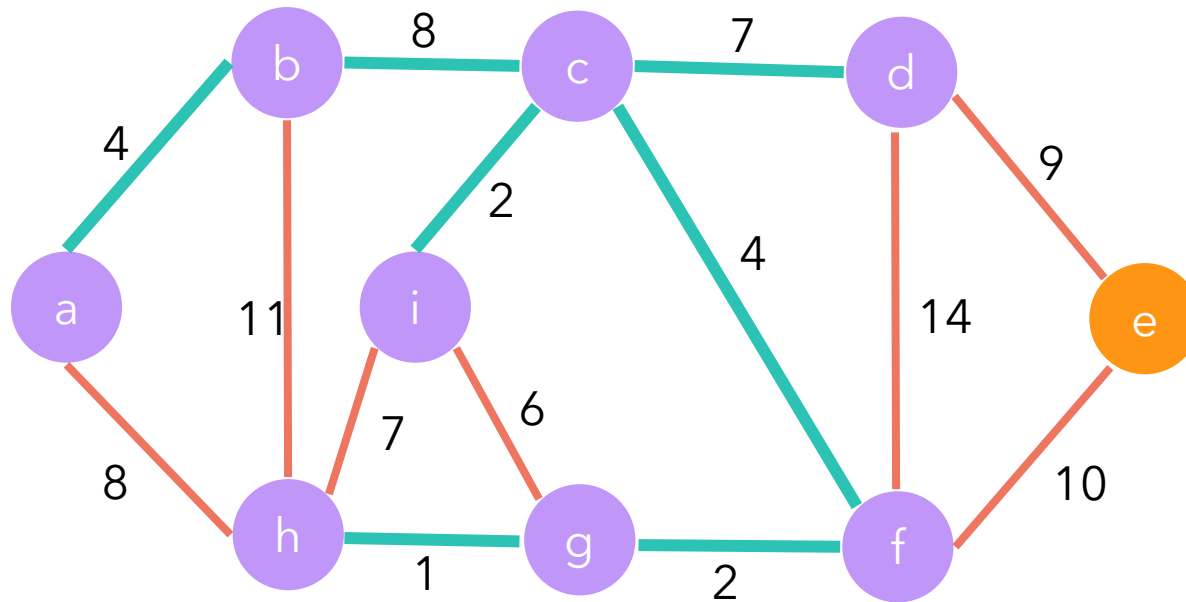
Grafos

Minimum Spanning Tree - Prim



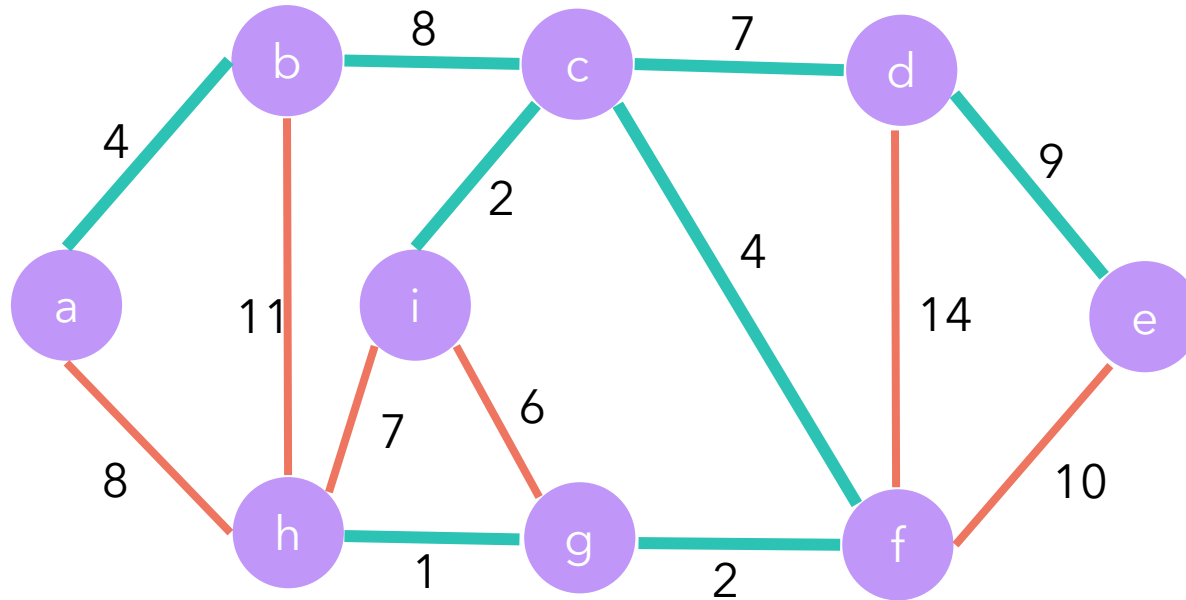
Grafos

Minimum Spanning Tree - Prim



Grafos

Minimum Spanning Tree - Prim



Grafos

Minimum Spanning Tree - Prim

