CURSO DE PROGRAMACIÓN COMPETITIVA

ESTRUCTURAS DE DATOS II

CURSO DE PROGRAMACIÓN COMPETITIVA
URJC - 2026

Organizadores:

- Isaac Lozano (isaac.lozano@urjc.es)
- Sergio Salazar (sergio.salazar@urjc.es)
- Adaya Ruiz (am.ruiz.2020@alumnos.urjc.es)
- Olga Somalo (o.somalo.2021@alumnos.urjc.es)
- Lucas Martín (lucas.martin@urjc.es)
- Iván Penedo (ivan.penedo@urjc.es)
- Alicia Pina (alicia.pina@urjc.es)
- **Sara García** (sara.garciar@urjc.es)
- **Raúl Fauste** (raul.fauste@urjc.es)

INDICE

- Mapas
- Colas de Prioridad
- Grafos
- BFS / DFS

Mapas

MAPAS (DICTIONARY)

- Estructura de datos que almacena pares **clave-valor**.
- Basado en una tabla hash.
- **No** está **ordenado**.
- Aplicaciones:
  - **Búsqueda** rápida de valores **clave-valor**.
  - **Agrupación** de datos por clave

LISTAS

Índice	Valor
0	"mundo"
1	2
2	6.0

MAPAS

Clave	Valor
"hola"	"mundo"
1	2
(2, 4)	6.0

MAPAS (DICTIONARY)

Operaciones

Crear

```
mapa1 = {}
mapa2 = {"a": 10, "b": 20}
```

O(1) Insertar

```
mapa["w"] = 400
```

O(1) Acceder

```
mapa["a"]
```

O(1) Eliminar

```
del mapa["x"]
```

O(1) Buscar

```
"y" in mapa
```

Recorrer

```
for clave, valor in mapa.items():
```

Hardwood Species

- https://open.kattis.com/problems/hardwoodspecies
- ID: hardwoodspecies
- Ejemplo:

Entrada		Salida
Ash Ash Aspen Basswood Ash	→	Aspen 20 Ash 60 Basswood 20

Colas de prioridad

COLAS DE PRIORIDAD (PRIORITY QUEUES)

- Estructura donde los elementos se atienden según su **prioridad** y no solo su orden de llegada.

- **heapq :**
  - **Montículo**
  - Más eficiente que **list.**

```
import heapq
```

- Aplicaciones:
  - Algoritmo Dijkstra.
  - Gestión de tareas o recursos.
  - Simulaciones de eventos

COLAS DE PRIORIDAD (PRIORITY QUEUES)

Operaciones

```
import heapq
```

Complejidad	Operación	Código
	Crear	`colap = []`
O(log(n))	Insertar	`heapq.heappush(colap, 1)`
O(1)	Acceder primero	`colap[0]`
O(log(n))	Eliminar primero	`heapq.heappop(colap)`
O(1)	Comprobar vacía	`len(cola) == 0`

COLAS DE PRIORIDAD (PRIORITY QUEUES)

- Por defecto ordenación ascendente.

- Para ordenar descendentemente se introducen los elementos en negativo y se extraen también en negativo.

```python
import heapq

colap = []

heapq.heappush(colap, -1)
heapq.heappush(colap, -3)
heapq.heappush(colap, -2)

primero = -heapq.heappop(colap) #3
segundo = -heapq.heappop(colap) #2
tercero = -heapq.heappop(colap) #1
```

Annoyed coworkers

- https://open.kattis.com/problems/annoyedcoworkers
- ID: annoyedcoworkers
- Ejemplo:

```
4 4
1 2
2 3   →   7
3 4
4 5
```

Assigning Workstations

- https://open.kattis.com/problems/workstations
- ID: workstations
- Ejemplo:

```
3 5
1 5
6 3
14 6
```

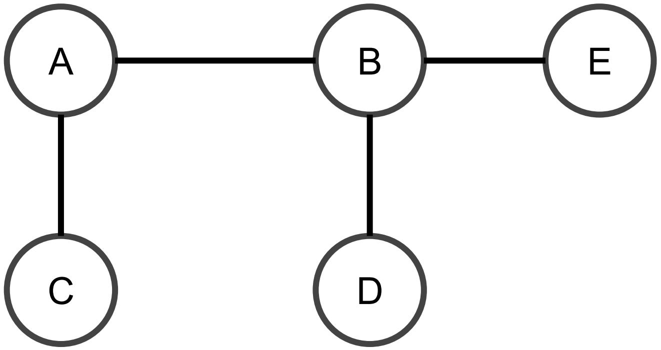
→ 2

GRAFOS

GRAFO - DEFINICIÓN

- Un grafo se denota como: G = (V,E)
  - Conjunto de vértices: V={A,B,C,D,E}
  - Conjunto de aristas:

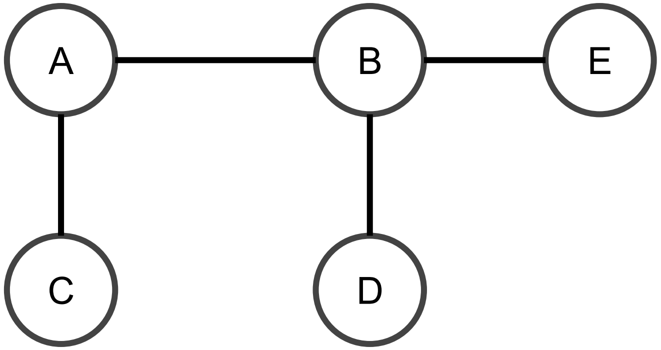
    E={(A,B),(A,C),(B,D),(B,E)}



GRAFO - TIPOS

- **Grafo NO dirigido** → Aristas **SIN dirección**

$(A,B) \Leftrightarrow (B,A)$



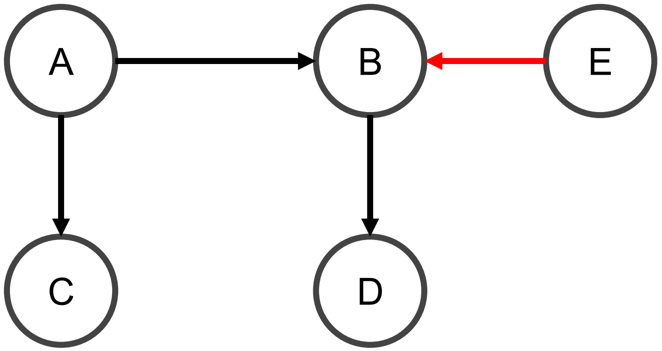
E={(A,B),(B,A),(A,C),(C,A),(B,D),(D,B),(B, E),(E,B)}

GRAFO - TIPOS

- **Grafo dirigido** → Aristas **CON dirección**

E={(A,B),(A,C),(B,D),(E, B)}

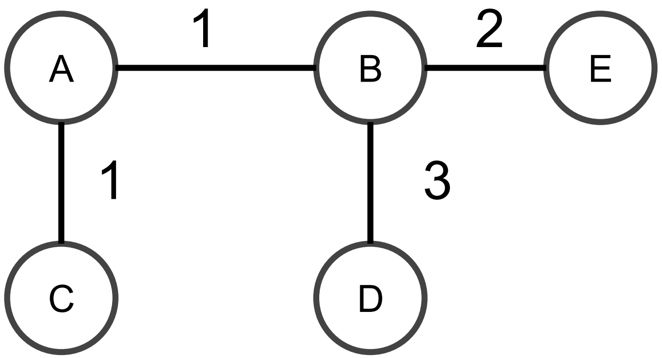
(E,B) ≠ (B,E)



GRAFO - TIPOS

- **Grafo ponderado →** Aristas con **peso**.

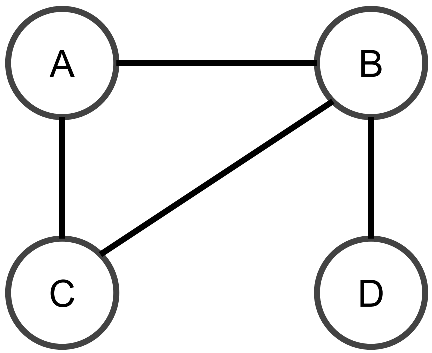
E={(A,B,1),(A,C,1),(B,D,3),(B, E,2)}



GRAFO - REPRESENTACIONES

- Representaciones
  - Matriz de adyacencia
  - Lista de adyacencia
  - Lista de aristas

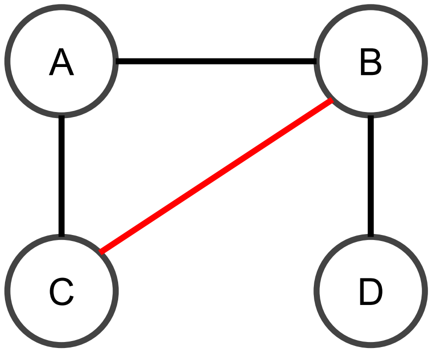
GRAFOS – MATRIZ ADYACENCIA



	A	B	C	D
A	0	1	1	0
B	1	0	1	1
C	1	1	0	0
D	0	1	0	0

GRAFOS – MATRIZ ADYACENCIA

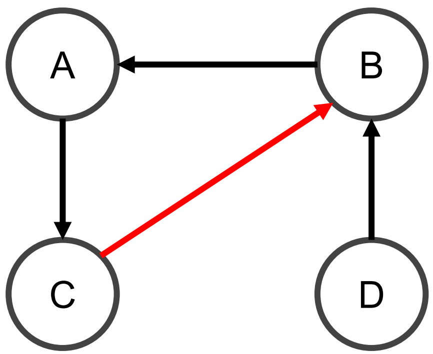
- Matriz **simétrica en grafos no dirigidos**



	A	B	C	D
A	0	1	1	0
B	1	0	1	1
C	1	1	0	0
D	0	1	0	0

GRAFOS – MATRIZ ADYACENCIA

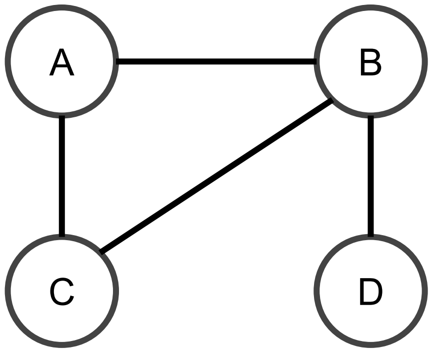
- Con **grafo dirigido** la matriz **no** es **simétrica**



	A	B	C	D
A	0	0	1	0
B	1	0	0	0
C	0	1	0	0
D	0	1	0	0

GRAFOS – LISTA ADYACENCIA

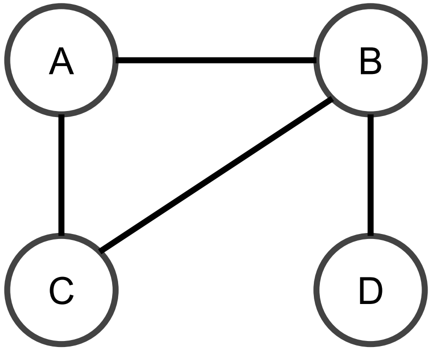
- De cada vértice se almacena con que otros vértices se conecta.



A	⇒	{B,C}
B	⇒	{A,C,D}
C	⇒	{A,B}
D	⇒	{B}

GRAFOS – LISTA DE ARISTAS

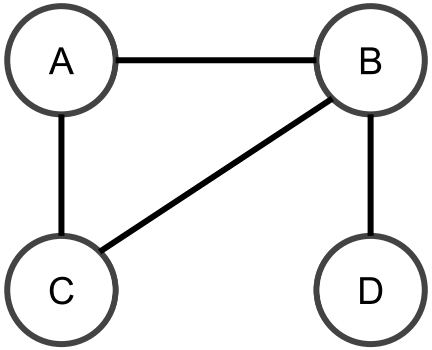
- El grafo se representa mediante sus aristas.



(A,B)
(A,C)
(B,C)
(B,D)

GRAFOS – LISTA DE ARISTAS

- El grafo se representa mediante sus aristas.



(A,B)
(A,C)
(B,C)
(B,D)

Útil con KRUSKAL

GRAFOS - RECORRIDOS

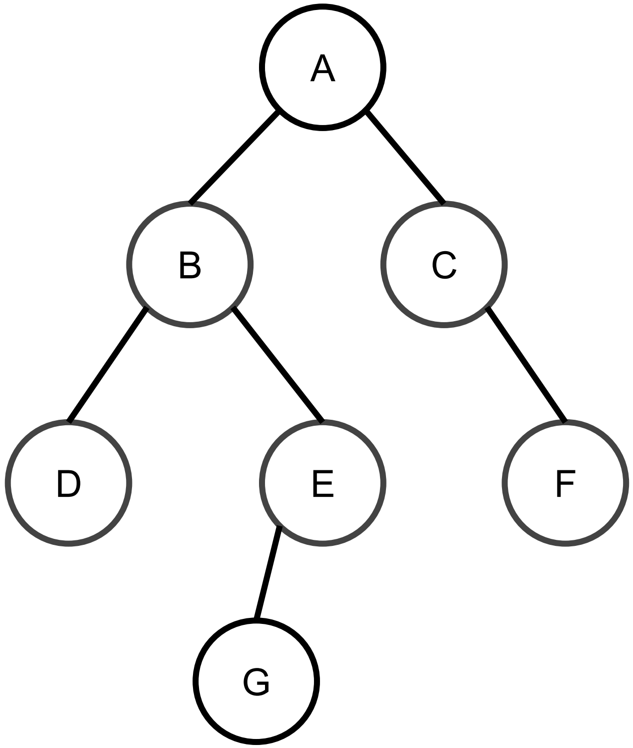
- Recorrido en anchura:

  BFS - Breadth First Search

- Recorrido en profundidad

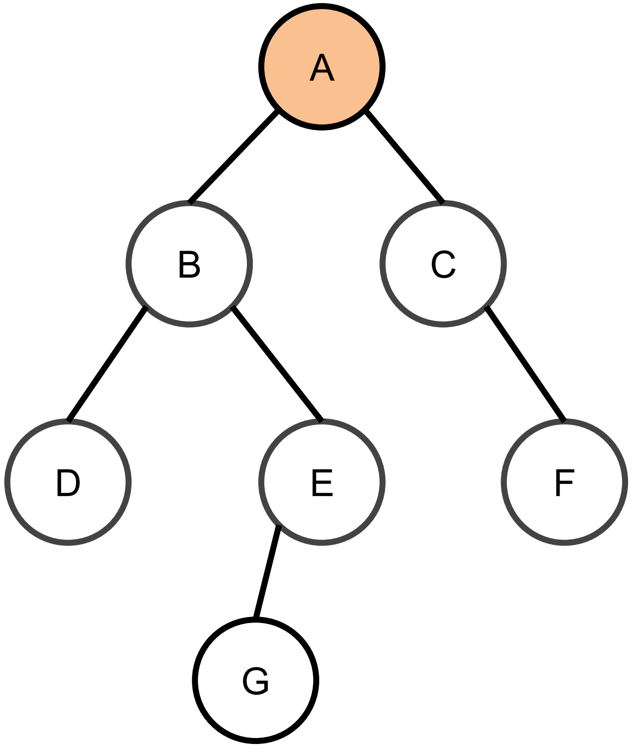
  DFS - Depth First Search

GRAFOS - BFS



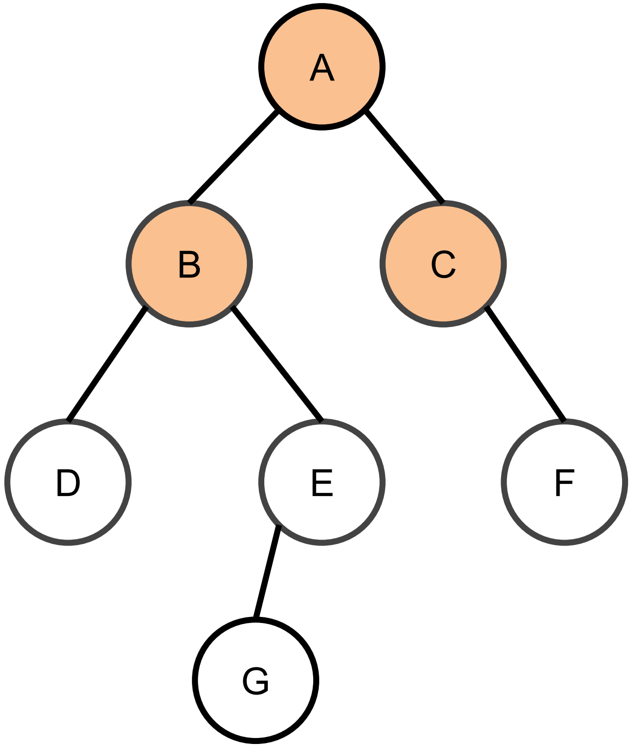
GRAFOS - BFS

A



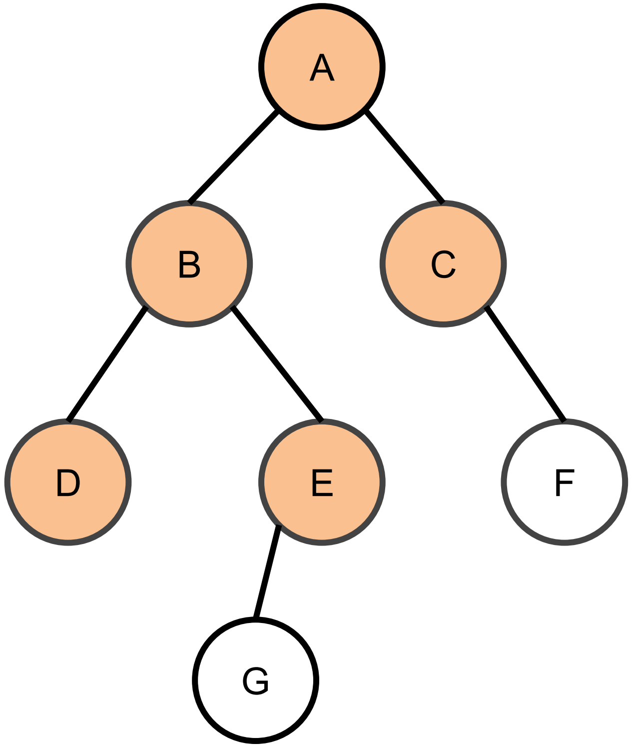
GRAFOS - BFS

A, B ,C



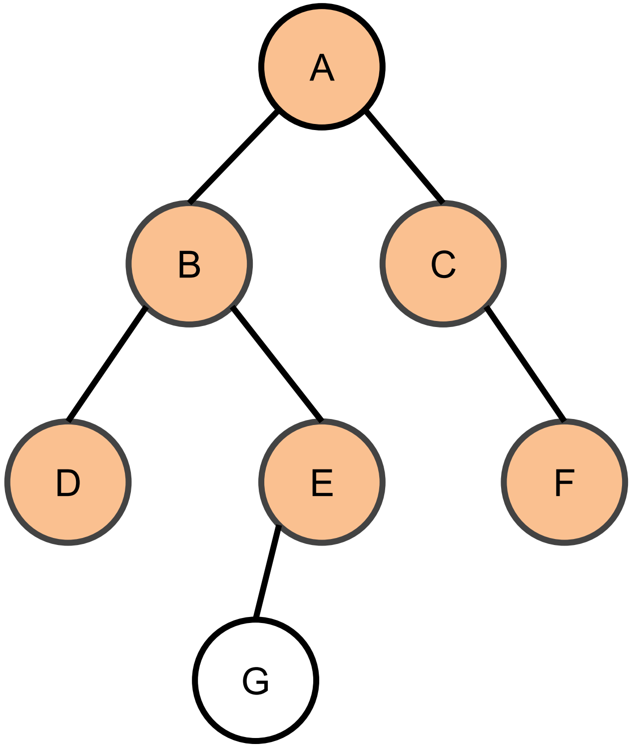
GRAFOS - BFS

A, B ,C, D, E



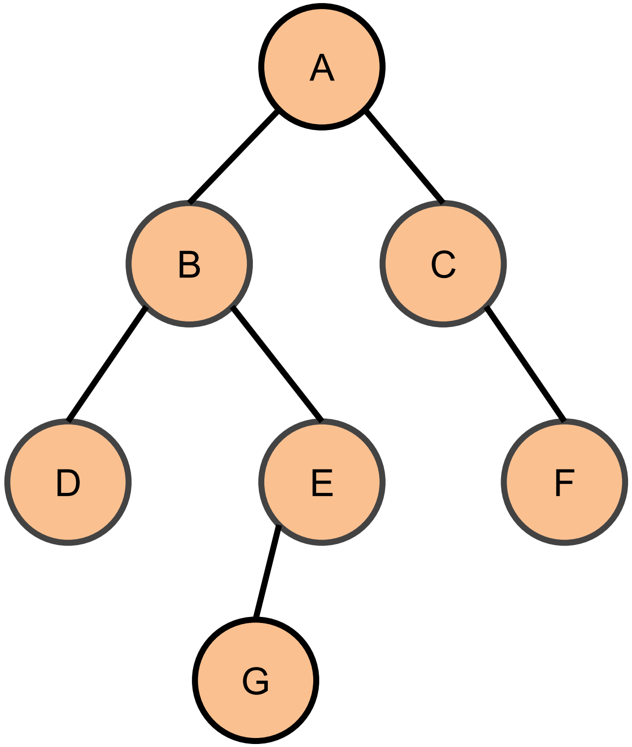
GRAFOS - BFS

A, B ,C, D, E, F



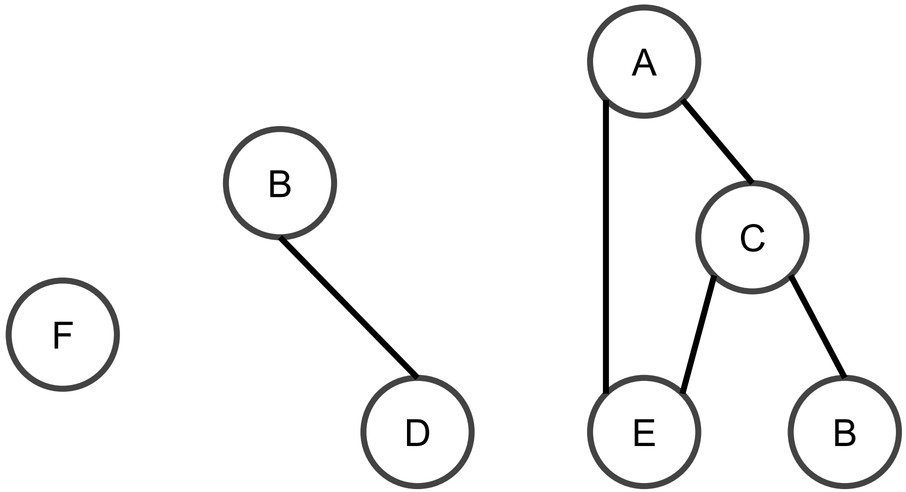
GRAFOS - BFS

A, B ,C, D, E, F, G



GRAFOS - BFS

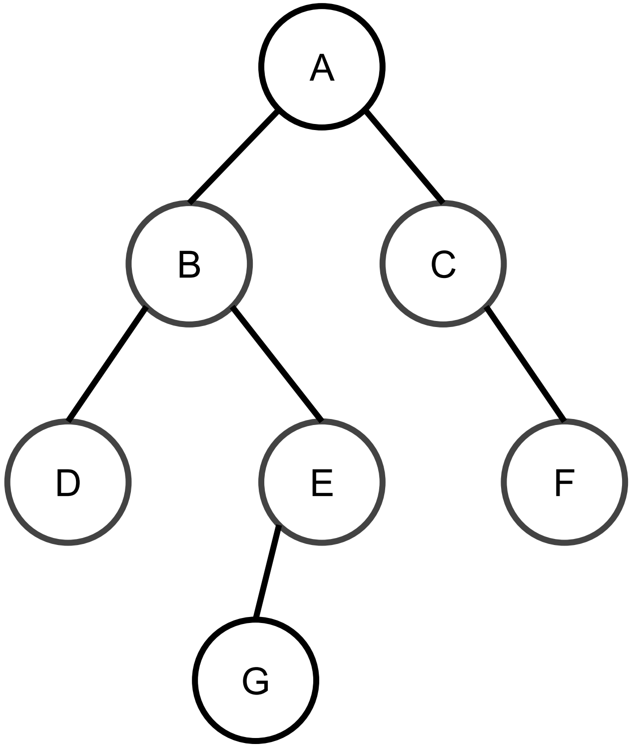
- Cosas importantes:
  - Llevar cuenta de los nodos visitados.
  - Cuidado con:



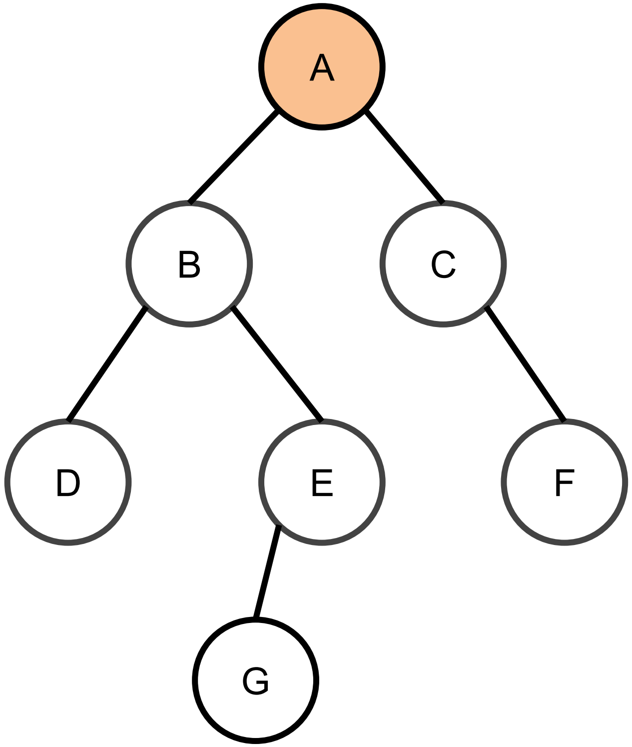
A - PARTY

- https://codeforces.com/contest/115/problem/A

GRAFOS - DFS



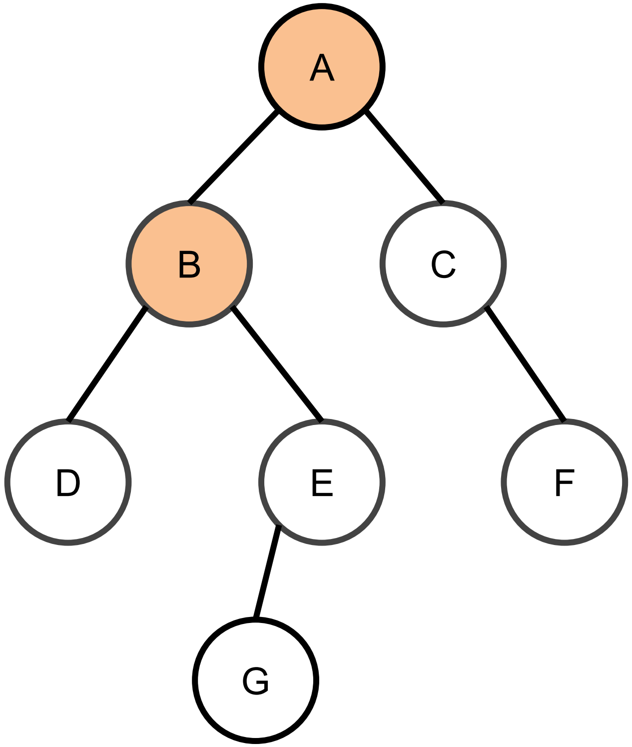
GRAFOS - DFS



A

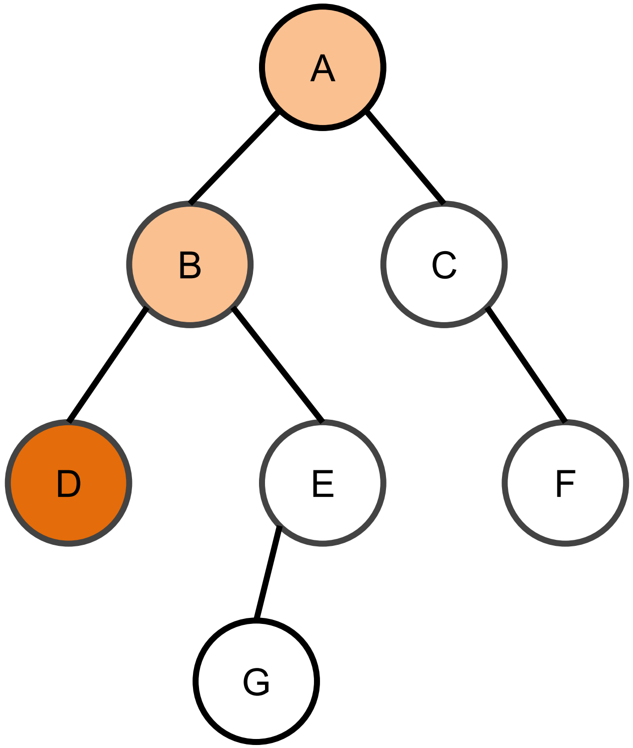
GRAFOS - DFS

A, B



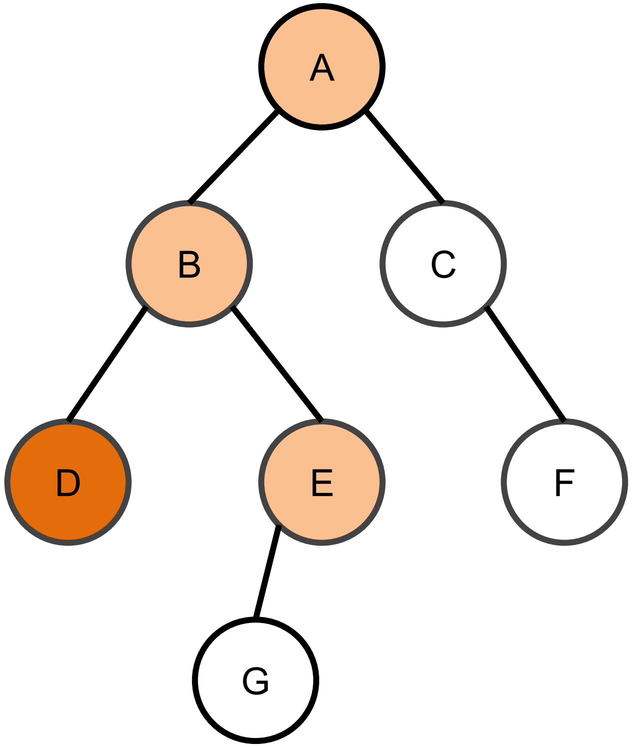
GRAFOS - DFS

A, B, D



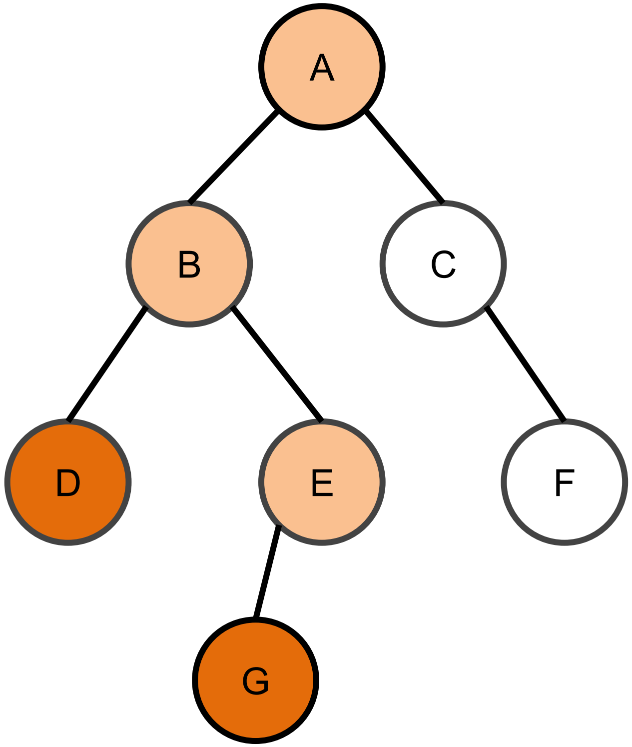
GRAFOS - DFS

A, B, D, E



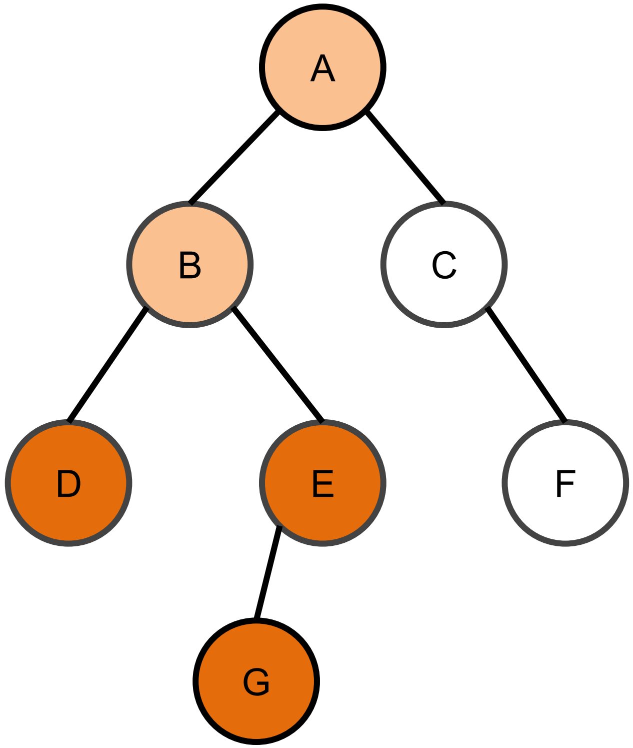
GRAFOS - DFS

A, B, D, E, G



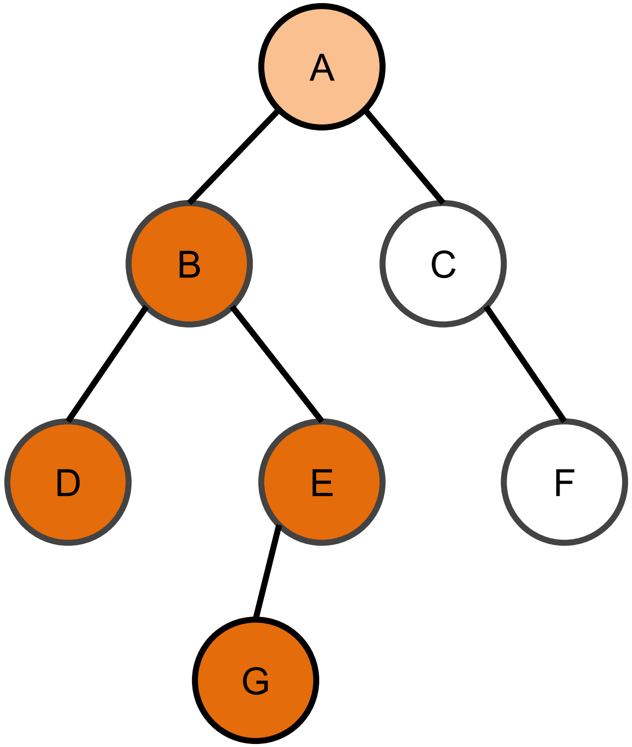
GRAFOS - DFS

A, B, D, E, G



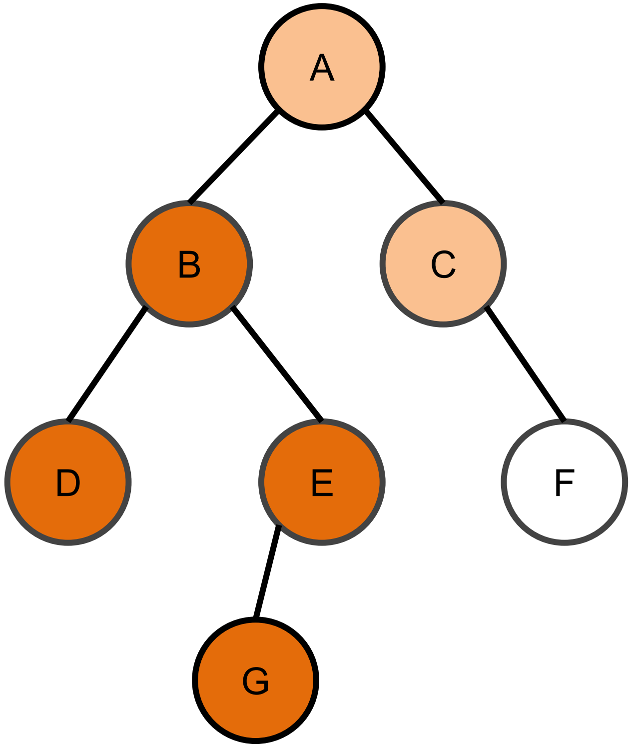
GRAFOS - DFS

A, B, D, E, G



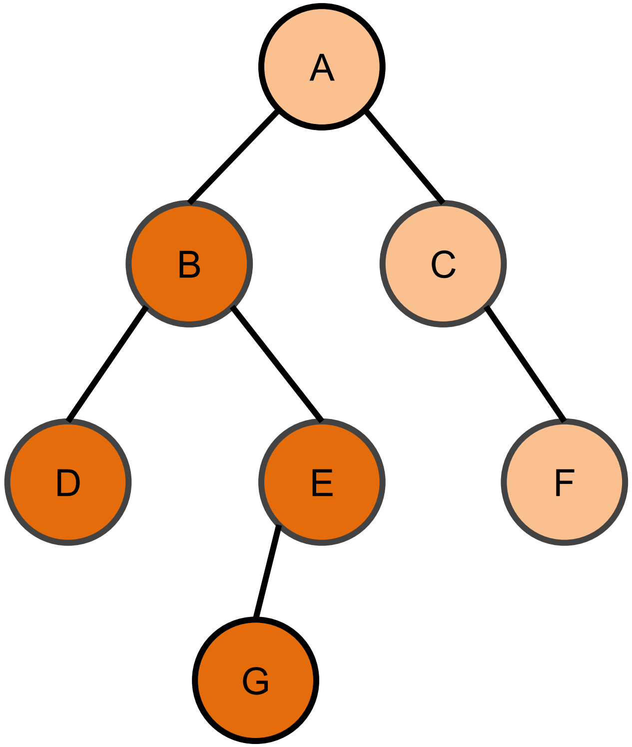
GRAFOS - DFS

A, B, D, E, G, C



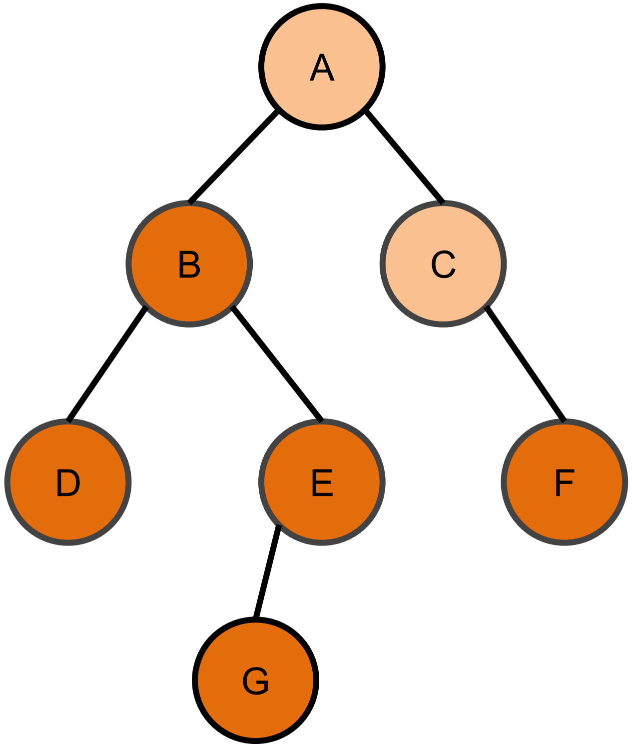
GRAFOS - DFS

A, B, D, E, G, C, F



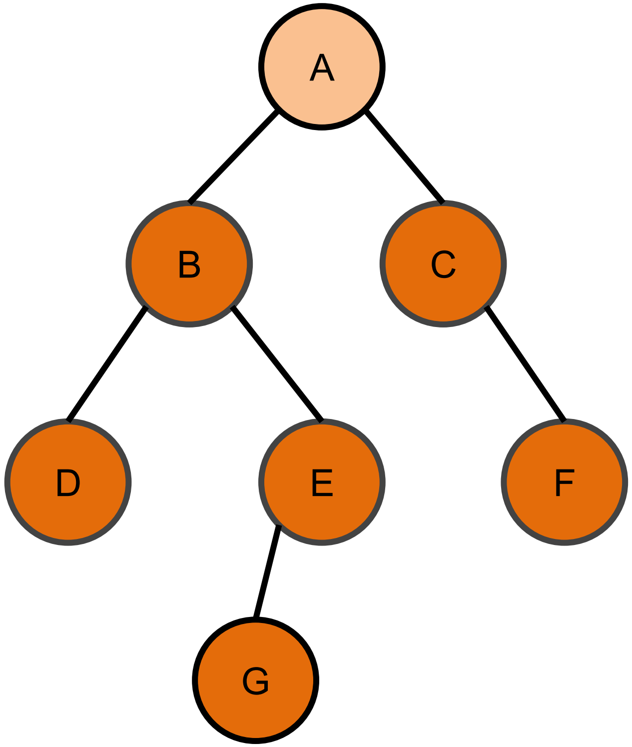
GRAFOS - DFS

A, B, D, E, G, C, F



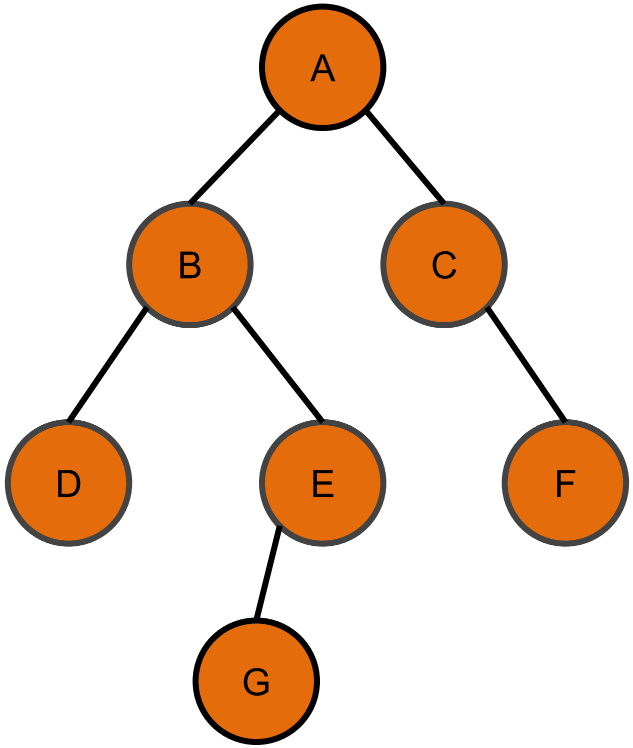
GRAFOS - DFS

A, B, D, E, G, C, F



GRAFOS - DFS

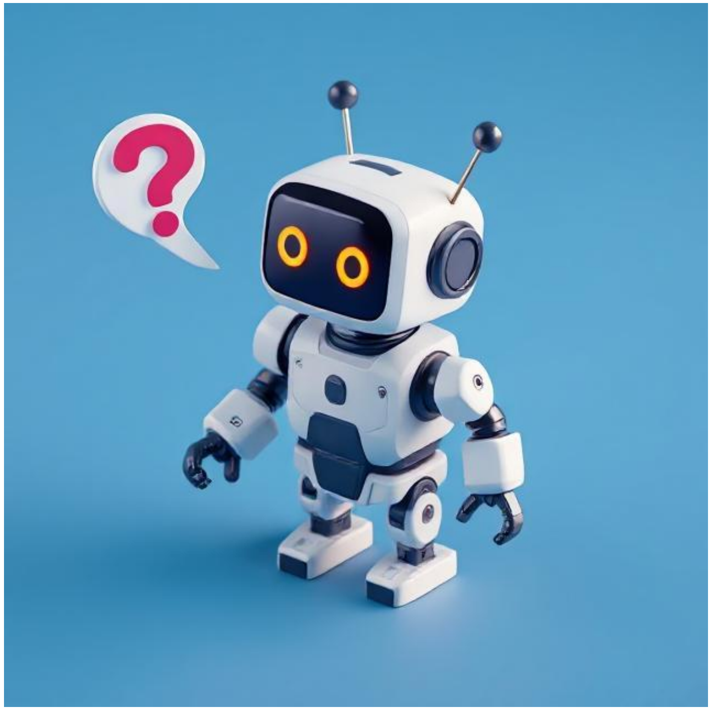
A, B, D, E, G, C, F



A - PARTY

- https://codeforces.com/contest/115/problem/A

¿PREGUNTAS?



HASTA LA SEMANA QUE VIENE!

@URJC_CP

@Dijkstraidos