

# Complejidad Algorítmica

**Unidad 1: Comportamiento asintótico, métodos de búsquedas y grafos**

**Módulo 5: Grafos - Técnicas de recorrido y búsquedas**

# Complejidad Algorítmica

Semana 5 / Sesión 1

## MÓDULO 5: Grafos - Técnicas de recorrido y búsquedas



### Contenido

1. Caminos y recorridos en grafos
2. Búsquedas en grafos
3. Búsquedas por costo uniforme (UCS)
4. Búsquedas por profundidad limitada (DLS)
5. Búsquedas por profundidad iterativa (IDS)



### Preguntas

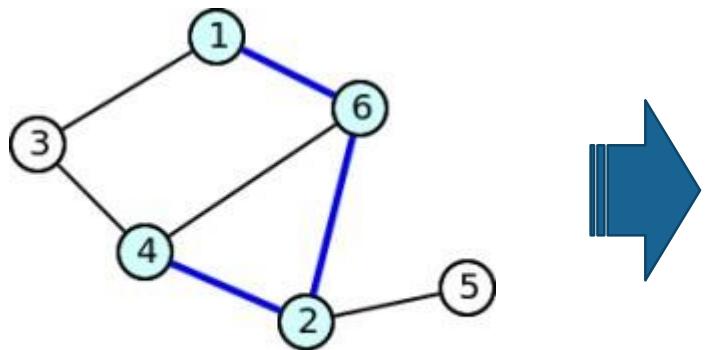
# 1. Caminos y recorridos en grafos

¿Qué es un CAMINO dentro de un Grafo?



- Un camino es una secuencia de vértices dentro de un grafo tal que existe una arista entre cada vértice y el siguiente.
- En un camino, se define también que:
  - ✓ Dos vértices están conectados si hay un camino entre ellos.
  - ✓ Dos vértices pueden estar conectados por varios caminos.
  - ✓ El número de aristas dentro de un camino es su **longitud**.
  - ✓ Un CICLO es un camino que empieza y termina en el mismo vértice.

# 1. Caminos y recorridos en grafos



- **CAMINO:** es una secuencia de vértices dentro de un grafo tal que existe una arista entre cada vértice y el siguiente.

Camino (Path) =  $1 \rightarrow 6 \rightarrow 2 \rightarrow 4$

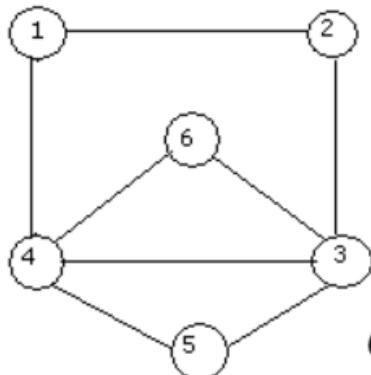
- **CICLO:** camino que comienza y termina en el mismo nodo (ej.: para el grafo,  $1 \rightarrow 6 \rightarrow 4 \rightarrow 3 \rightarrow 1$  es un ciclo)

# 1. Caminos y recorridos en grafos

## TIPOS DE CAMINOS

### Camino Euleriano

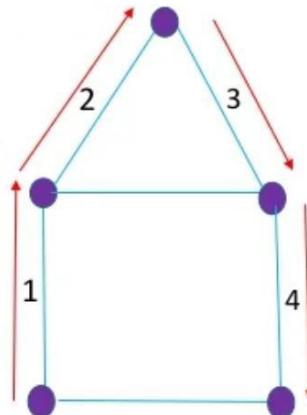
- Camino cerrado que empieza y termina en el mismo vértice.
- Pasa por cada vértice al menos una vez y sólo una vez por cada arista.
- No importa la repetición de vértices



$$C = \{1, 2, 3, 4, 6, 3, 5, 4, 1\}$$

### Camino Hamiltoniano

- Un ciclo hamiltoniano, es un camino donde el último vértice es adyacente al primero (no termina en el mismo vértice que inicia).
- Pasa por cada vértice exactamente una vez.



¿Cómo reconocemos un camino Hamiltoniano?

Si uno escoge una pareja de vértices, y sumando sus grados, el resultado tiene que ser mayor o igual a  $n-1$  (donde  $n$  es el grado del grafo)

# 1. Caminos y recorridos en grafos

## TIPOS DE RECORRIDOS EN GRAFOS

Existen dos formas de recorrer un grafo:

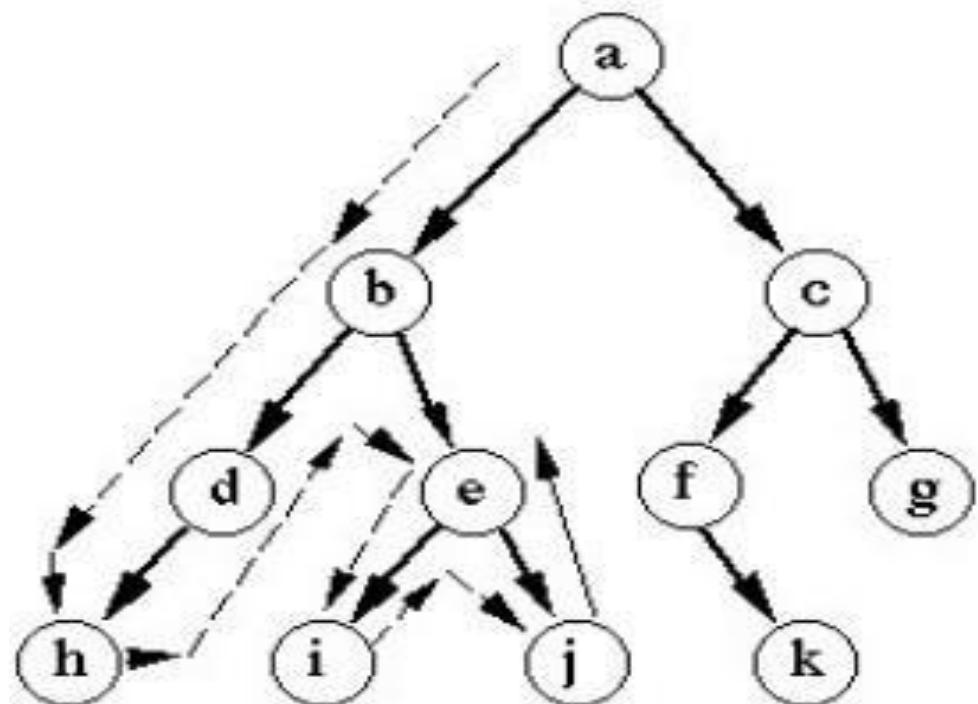
1. Recorrido en profundidad (Deep First Search – DFS)
  2. Recorrido en amplitud o anchura (Breadth First Search – BFS)
- En esencia, **Deep First Search (DFS)** y **Breadth First Search (BFS)** hacen lo mismo: recorrer todos los nodos de un grafo.
  - ¿Cuándo usar uno u otro? depende del problema y el orden en que nos interesa pasar por los nodos.

Veamos cómo implementar ambos...

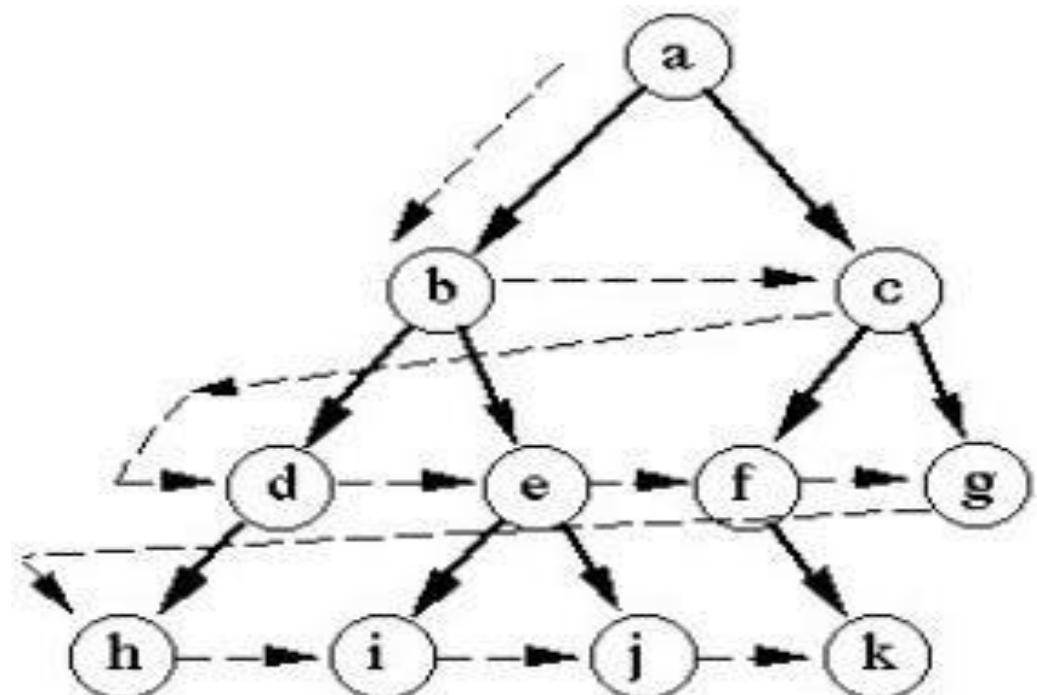


# 1. Caminos y recorridos en grafos

## TIPOS DE RECORRIDOS EN GRAFOS



Depth-first search



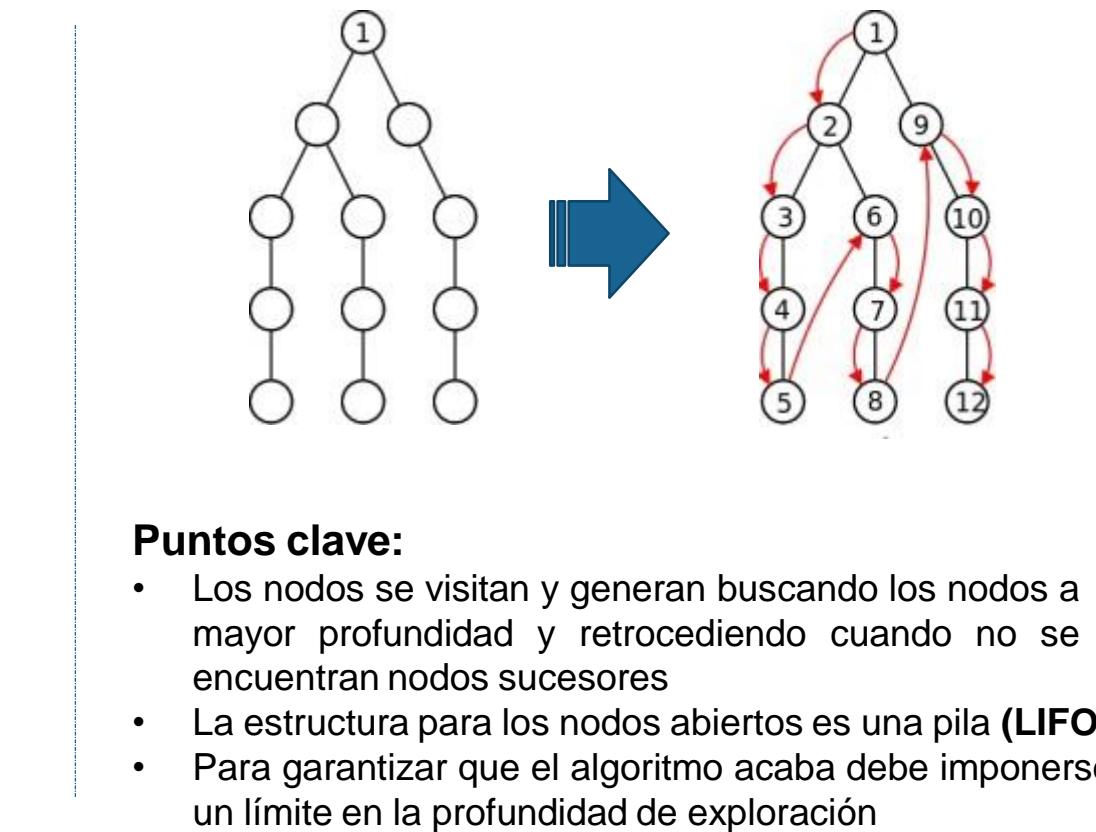
Breadth-first search

# 1. Caminos y recorridos en grafos

## 1. Recorrido en profundidad (Deep First Search – DFS)

### ¿Cómo se busca?

- Se busca en todo el gráfico conectado a un nodo adyacente antes de ingresar al siguiente nodo adyacente
- En esta búsqueda siempre se expande uno de los nodos que se encuentre en lo más profundo del árbol.
- Sólo si la búsqueda conduce a un callejón sin salida (un nodo que no es meta y que no tiene expansión), se revierte la búsqueda y se expanden los nodos de niveles menos profundos.



# 1. Caminos y recorridos en grafos

## 1. Recorrido en profundidad (Deep First Search – DFS)

**Pseudocódigo:**

DFS (versión recursiva)

**dfs**(nodo v):

    marcar **v** como visitado

**Para** todos los nodos w adyacentes a v **hacer**

**Si** w aún no ha sido visitado **entonces**

            DFS (w)

**Complejidades**

**Temporal:**

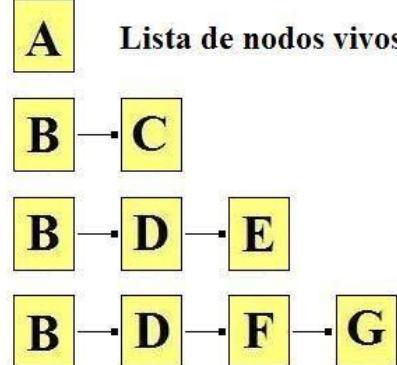
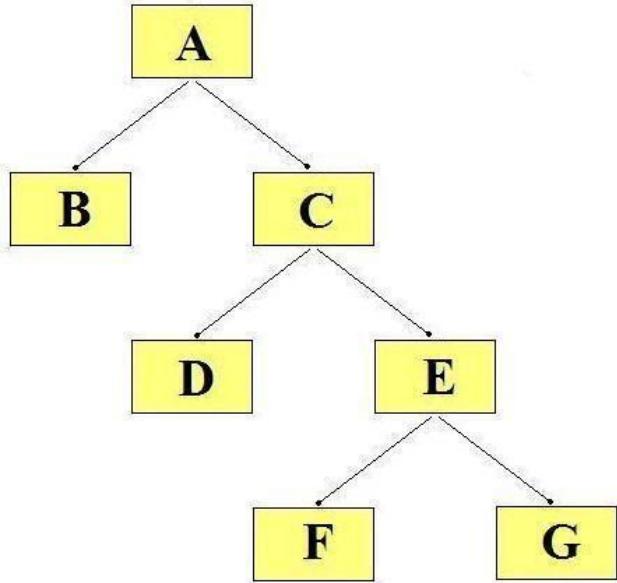
- Lista de adyacencia:  $O(|V| + |E|)$
- Matriz de Adyacencia:  $O(|V|^2)$

**Espacial:**  $O(|V|)$

# 1. Caminos y recorridos en grafos

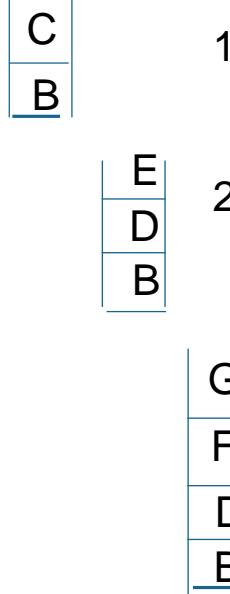
## 1. Recorrido en profundidad (Deep First Search – DFS)

Recorrido de un árbol en profundidad (LIFO)



### Recorrido LIFO

1. Generar hijos del nodo A y colocar estos nodos activos en una pila (B, C).
2. Quitar el elemento el ultimo ingresado a la pila y generar sus hijos, coloque esos nodos en la pila. C se quita de la pila. Los hijos de C son D, E.
3. Nuevamente, elimine un elemento de la pila, es decir, se elimina el nodo E y los nodos generados por E son F, G que se introducen a la pila.

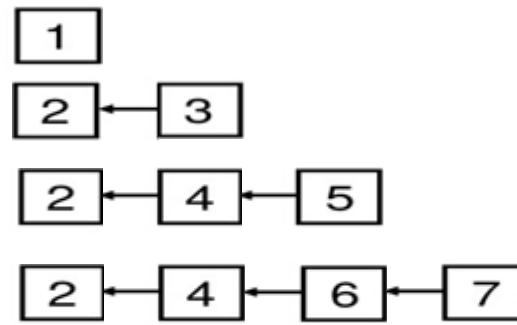
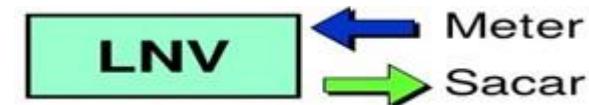
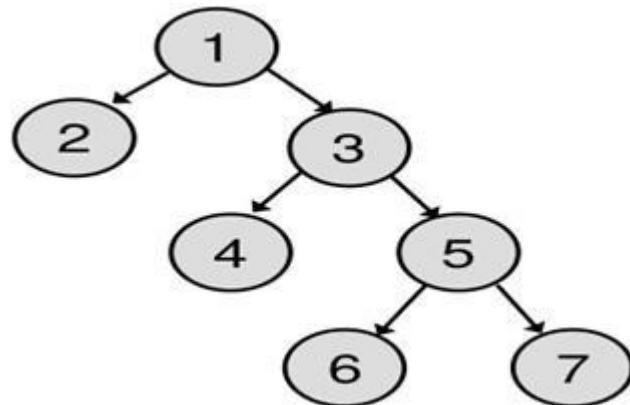


# 1. Caminos y recorridos en grafos

## 1. Recorrido en profundidad (Deep First Search – DFS)

Ejemplo: Recorrido de un árbol en profundidad (LIFO)

- La estructura de la lista de nodos vivos (LNV) se trata como una pila (LIFO)
- LIFO = LAST IN FIRST OUT = EL ULTIMO EN ENTRAR ES EL PRIMERO EN SALIR.

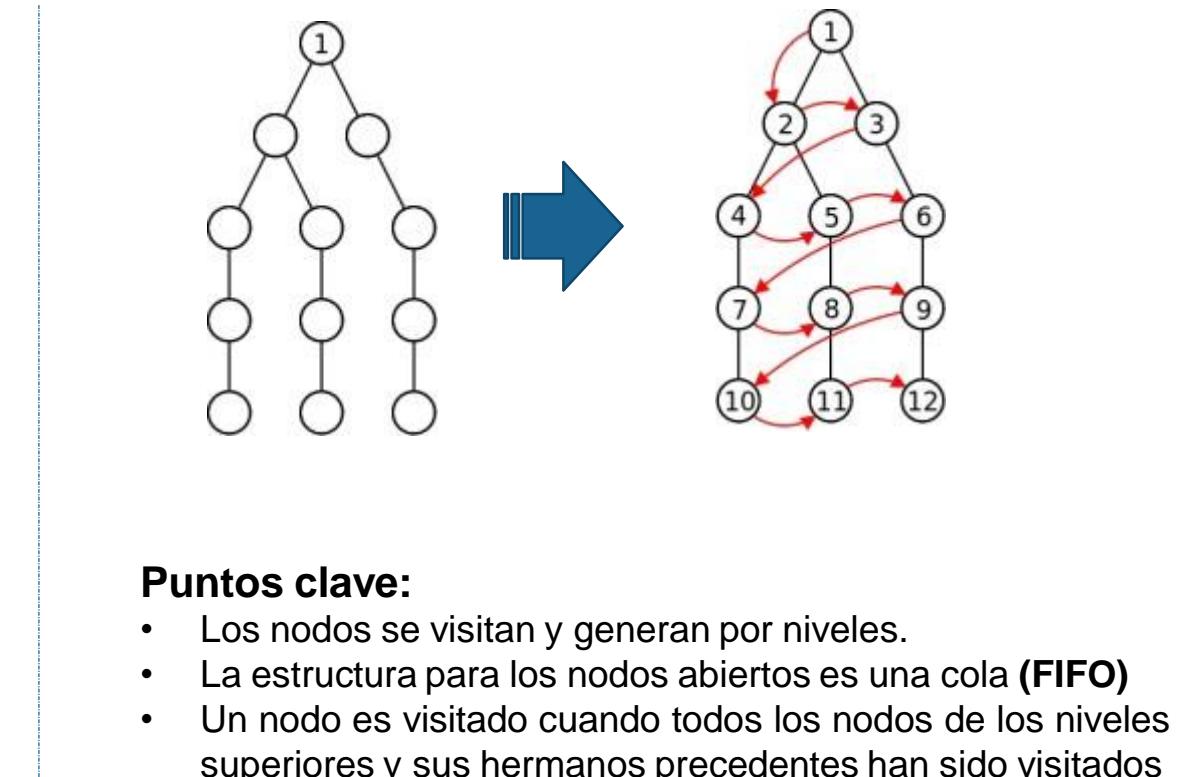


# 1. Caminos y recorridos en grafos

## 2. Recorrido en amplitud o anchura (Breadth First Search – BFS)

### ¿Cómo se busca?

- Primero se expande el nodo inicial (raíz) y luego todos los nodos generados por éste, luego sus sucesores y así sucesivamente.
- Todos los nodos que están a profundidad  $d$  se expanden primero, antes que los nodos con profundidad  $d + 1$ .



# 1. Caminos y recorridos en grafos

## 2. Recorrido en amplitud o anchura (Breadth First Search – BFS)

- Una búsqueda de amplitud (BFS) es muy similar a un DFS.
- Solamente cambia el orden en el que se visitan los nodos
- En lugar de usar la recursividad, mantenemos explícitamente una cola de nodos no visitados (q)

### Pseudocódigo:

**BFS** (nodo v):

q  $\leftarrow \emptyset$  /\* Cola de nodos no visitados \*/

q.inqueue (v)

marcar v como visitado

**Mientras** q  $\neq \emptyset$  /\* Mientras haya nodos a procesar \*/

  u  $\leftarrow$  q.dequeue() /\* Elimina el primer elemento de q \*/

**Para** todos los nodos w adyacentes a u **hacer**

**Si** w aún no ha sido visitado **entonces** /\* ¡Nuevo nodo! \*/

      q.inqueue (w)

      marcar w como visitado

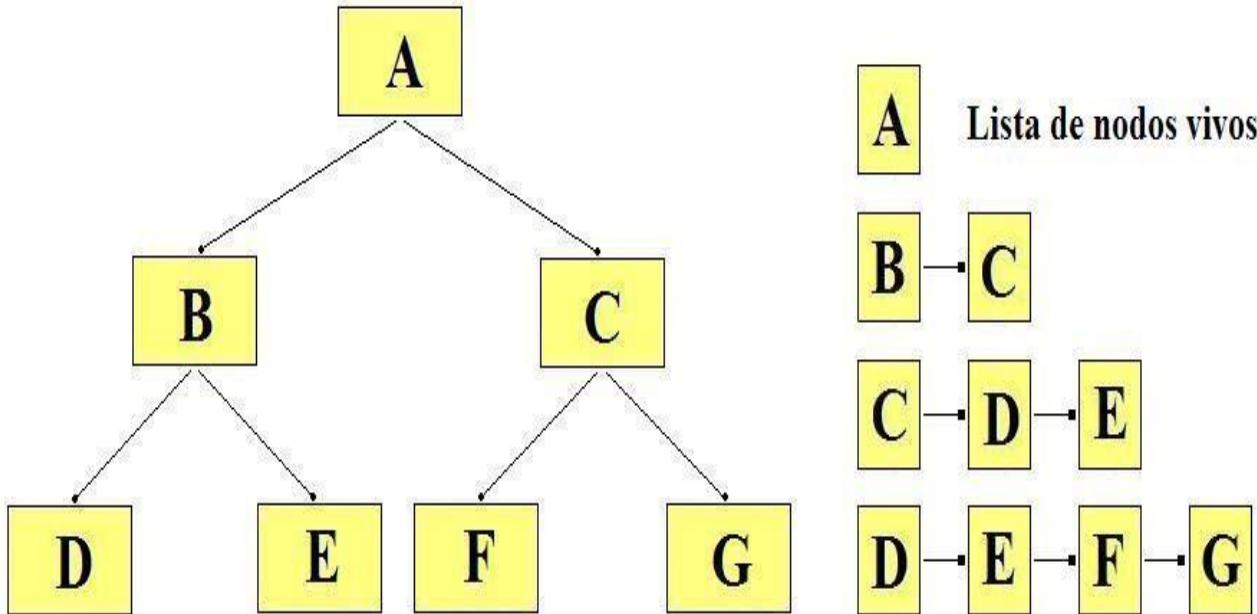
### Complejidad:

- Temporal:  $O(V+E)$

# 1. Caminos y recorridos en grafos

## 2. Recorrido en amplitud o anchura (Breadth First Search – BFS)

Recorrido de un árbol en anchura (FIFO)



### Recorrido FIFO

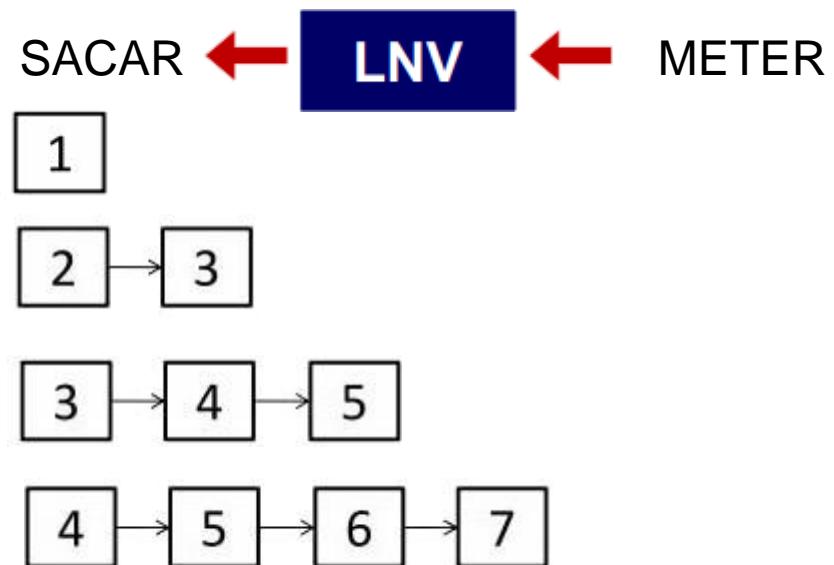
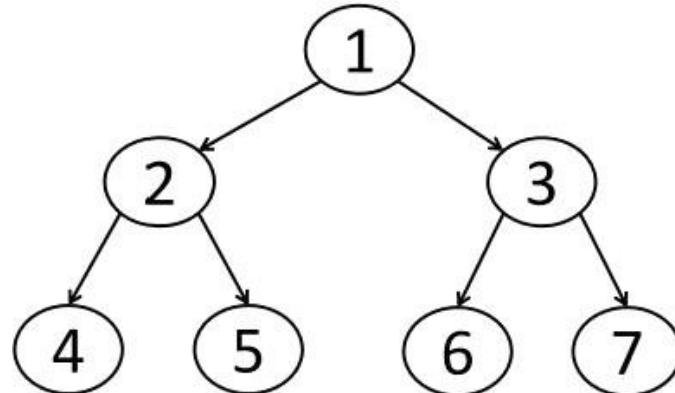
1. Inicia introduciendo en la LNV el nodo A.
2. Sacamos el nodo A de la cola y se expande generando los nodos B y C que son introducidos en la LNV.
3. Seguidamente se saca el primer nodo que es el B y se vuelve a expandir generando los nodos D y E que se introducen en la LNV.
4. Este proceso se repite mientras que quede algún elemento en la cola.

# 1. Caminos y recorridos en grafos

## 2. Recorrido en amplitud o anchura (Breadth First Search – BFS)

Ejemplo: Recorrido de un árbol en anchura (FIFO)

- La estructura de la lista de nodos vivos (LNV) se trata como una cola (FIFO), dando lugar a un recorrido en anchura del árbol.
- **FIFO = FIRST IN FIRST OUT = EL PRIMERO EN ENTRAR ES EL PRIMERO EN SALIR.**



# 1. Caminos y recorridos en grafos

## Ventajas y desventajas entre DFS y BFS

### Deep First Search – DFS

- Si la cantidad de soluciones en un problema es grande, se recomienda esta búsqueda sobre la búsqueda por amplitud (BFS).
- La desventaja de esta búsqueda es que se puede quedar estancada al avanzar por una ruta equivocada, ya que muchos árboles de búsqueda pueden ser muy profundos o infinitos.
- Por lo tanto, la DFS no es ni completa ni óptima.

### Breadth First Search – BFS

- Si hay solución, es seguro que se encontrará mediante la búsqueda preferente por amplitud (BFS).
- Si son varias soluciones, siempre encontrará primero el estado de meta más próximo (menos profundidad, más a la izquierda).
- La búsqueda preferente por amplitud es completa y óptima siempre y cuando el costo de ruta sea una función que no disminuya al aumentar la profundidad del nodo.

**DFS es más adecuado para encontrar la ruta más larga o el nodo más lejano al nodo inicial.**

**BFS es más adecuado para encontrar la ruta más corta o el nodo más cercano al nodo inicial.**

# PREGUNTAS

Dudas y opiniones