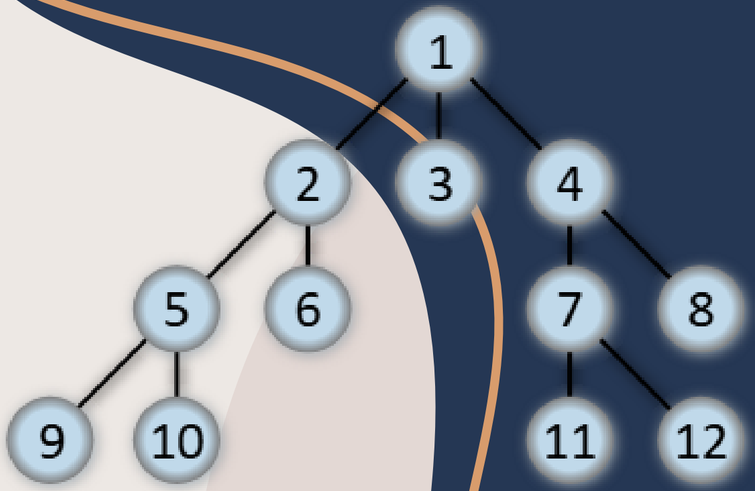
INTELIGENCIA ARTIFICIAL

BÚSQUEDA PRIMERO EN ANCHURA

Integrantes:
Scarlett Fernández Flores
Julio Cesar Ruiz Perez

INTRODUCCIÓN

La búsqueda primero en anchura es una técnica utilizada en inteligencia artificial para encontrar soluciones explorando primero los caminos más cortos. Funciona revisando todos los posibles movimientos a un mismo nivel antes de pasar al siguiente, lo que garantiza encontrar la solución más cercana al inicio.



¿QUÉ ES LA BÚSQUEDA PRIMERO EN ANCHURA?

La búsqueda primero en anchura (BFS) es una estrategia para explorar árboles o grafos en inteligencia artificial.

Explora nivel por nivel: primero el nodo raíz, luego todos sus sucesores, después los sucesores de estos, y así sucesivamente.

Es como revisar un edificio piso por piso, empezando desde arriba.

¿CÓMO FUNCIONA LA BÚSQUEDA PRIMERO EN ANCHURA?

Usa una cola FIFO (primero en entrar, primero en salir).
Pasos:

1.-Empieza con el nodo raíz.

2.-Expande el nodo raíz y agrega sus sucesores al final de la cola.

3.-Toma el primer nodo de la cola, expándelo y agrega sus sucesores al final.

4.-Repite hasta encontrar la solución o explorar todo el árbol.

Esto asegura que los nodos más superficiales se exploren primero.

EJEMPLO VISUAL: PROGRESO DE BFS EN UN ÁRBOL BINARIO

La Figura 3.10 muestra un árbol binario sencillo.
Progreso por etapas:
Etapa 1: Expande el nodo raíz A.
Etapa 2: Expande los sucesores de A: B y C.
Etapa 3: Expande los sucesores de B: D y E; y de C: F y G.
El próximo nodo a expandir se marca en cada etapa.
Esto demuestra cómo BFS explora nivel por nivel.

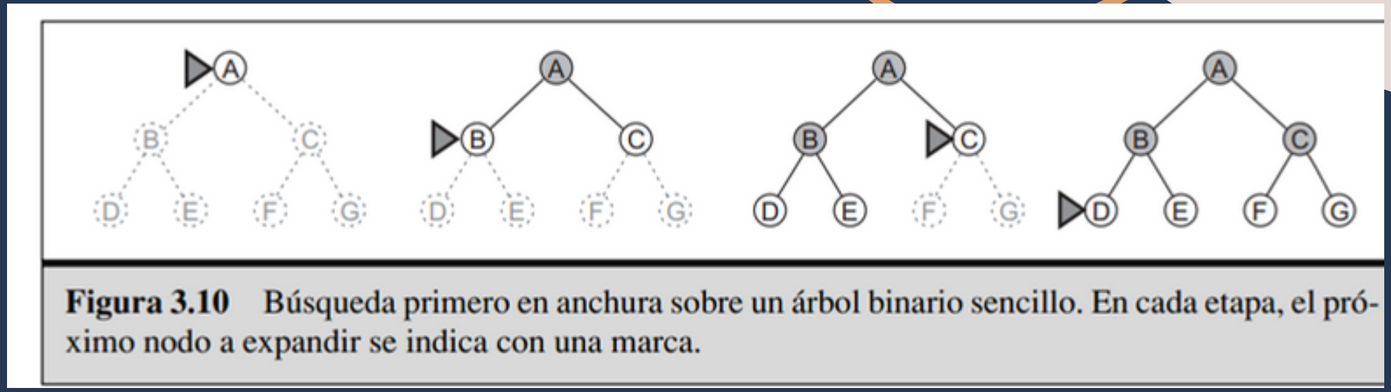


Figura 3.10 Búsqueda primero en anchura sobre un árbol binario sencillo. En cada etapa, el próximo nodo a expandir se indica con una marca.

Código de ejemplo en Python

BFS.py

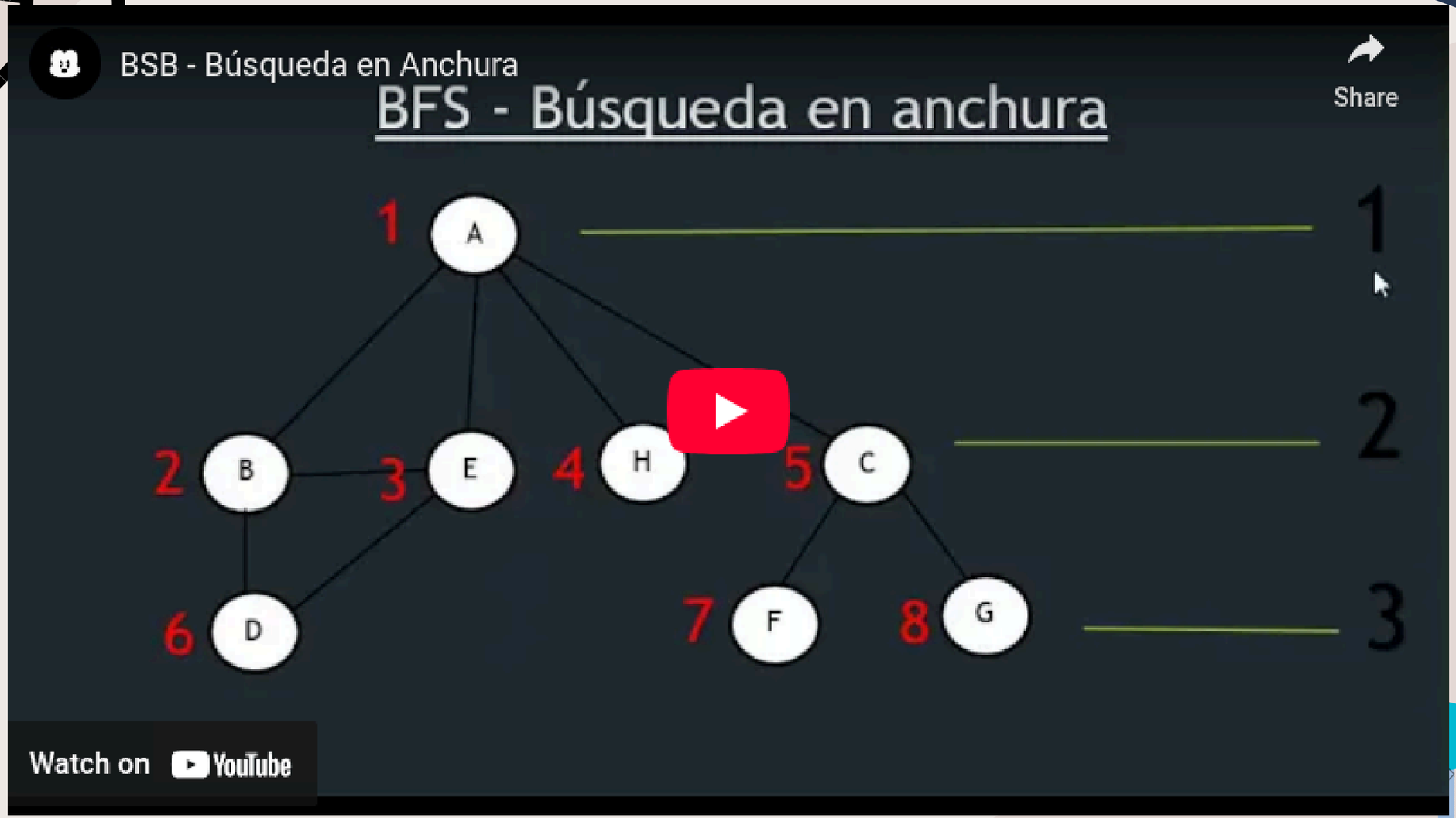
```python
# Importamos deque desde la biblioteca collections para usarlo como cola
from collections import deque
# Definimos la función de búsqueda en anchura (BFS)
def bfs(grafo, inicio):
    # Creamos un conjunto para registrar los nodos que ya fueron visitados
    visitados = set()
    # Creamos una cola (deque) e insertamos el nodo de inicio
    cola = deque([inicio])
    # Marcamos el nodo inicial como visitado
    visitados.add(inicio)
    # Mientras la cola no esté vacía, seguimos recorriendo el grafo
    while cola:
        # Sacamos el nodo al frente de la cola
        nodo = cola.popleft()
        # Mostramos el nodo actual (puedes cambiar esto por cualquier otra acción)
        print(nodo)
        # Recorremos todos los vecinos (nodos conectados) del nodo actual
        for vecino in grafo[nodo]:
            # Si el vecino no ha sido visitado aún
            if vecino not in visitados:
                # Lo marcamos como visitado
                visitados.add(vecino)
                # Lo añadimos a la cola para seguir explorando desde él
                cola.append(vecino)
```

```python
# ---------------- Ejemplo de uso ----------------
# Definimos un grafo como un diccionario, donde cada nodo tiene una lista de vecinos
grafo = {
    'A': ['B', 'C'],
    'B': ['D', 'E'],
    'C': ['F'],
    'D': [],
    'E': ['F'],
    'F': []
}
# Llamamos a la función BFS comenzando desde el nodo 'A'
bfs(grafo, 'A')
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
PS C:\xampp\htdocs\123> & C:/Python313/python.exe c:/xampp/htdocs/123/BFS.py
A
B
C
D
E
F
PS C:\xampp\htdocs\123>
```

VIDEO ALGORITMO BFS



VENTAJAS

01

Completitud:

Siempre encuentra una solución si existe (mientras el factor de ramificación sea finito).

02

Óptima:

Lo es solo si todas las acciones tienen el mismo coste.

DESVENTAJAS

01

Altos requerimientos de memoria:

Necesita almacenar todos los nodos generados hasta alcanzar la solución.

02

Crecimiento exponencial:

A mayor profundidad, necesita mucho más tiempo y memoria.

REQUISITOS DE TIEMPO Y MEMORIA (FIGURA 3.11)

Suponiendo b = 10, 10.000 nodos/segundo, 1.000 bytes/nodo:

Profundidad	Nodos	Tiempo	Memoria
2	1.100	11 segundos	1 megabyte
4	111.100	11 segundos	106 megabytes
6	10^7	19 minutos	10 gigabytes
8	10^9	31 horas	1 terabytes
10	10^{11}	129 días	101 terabytes
12	10^{13}	35 años	10 petabytes
14	10^{15}	3.523 años	1 exabyte

Figura 3.11 Requisitos de tiempo y espacio para la búsqueda primero en anchura. Los números que se muestran suponen un factor de ramificación $b = 10$; 10.000 nodos/segundo; 1.000 bytes/nodo.

LECCIONES CLAVE

- Memoria es el mayor problema: A profundidad 8, se necesitan 1 TB de memoria, lo que es difícil para la mayoría de los computadores.
- Tiempo también es un factor: A profundidad 12, tomaría 35 años encontrar la solución.
- Para problemas profundos (complejidad exponencial), BFS no es práctica sin información adicional.

CONCLUSIÓN

Aunque la búsqueda primero en anchura es simple y asegura una solución si existe, puede volverse muy lenta y consumir mucha memoria en problemas grandes. Por eso, es ideal solo para problemas pequeños o cuando se espera que la solución esté cerca del inicio.