
UNIVERSIDAD SANTO TOMÁS
FACIENTES VERITATEM
UNIVERSIDAD
SANTO TOMÁS

PRESENTACIÓN

Ingeniero Electrónico, Magister en Ingeniería con énfasis en electrónica y estudiante del doctorado en ingeniería con énfasis en eléctrica y electrónica de la UDFJC

Diego Alejandro Barragán Vargas

Docente de electrónica Universidad Santo Tomás de Aquino

Enlace de Interés:
https://scholar.google.com/citations?hl=es&user=Bp3QMQMAAAAJ

UNIVERSIDAD SANTO TOMÁS

Sesión 5-Algoritmos de Búsqueda No Informada

22 de Agosto, Bogotá D.C.

CONTENIDO

TEXTO COMPLEMENTARIO

Algoritmos de Búsqueda No Informada

BFS, DFS, UCS. Árboles de búsqueda.

Algoritmos de Búsqueda No Informada

La búsqueda no informada consiste en la selección e implementación de estrategias de búsqueda de un estado solución a partir de un estado inicial sin introducir al algoritmo de solución conocimiento sobre el impacto de las transiciones en la exploración del espacio de estados. [1]

CARACTERÍSTICAS

- **Falta de información** — No utilizan heurísticas ni información externa para guiar la búsqueda, por lo que no saben si un camino es mejor que otro para llegar al objetivo.
- **Exploración sistemática** — Siguen un proceso metódico para recorrer los posibles estados del problema, sin tener en cuenta el coste de alcanzar el objetivo.
- **Aplica a problemas simples** — Son valiosos para problemas sencillos o para sentar las bases de algoritmos más complejos, pero ineficientes en búsquedas grandes.

Conceptos Clave

- **Estado y acciones** — El estado es la representación del mundo en un momento y la acción es la operación que lleva de un estado a otro.
- **Sucesor y costo de paso** — La sucesión es el estado alcanzado aplicando una acción y el costo de paso es el costo de aplicar una acción.
- **Objetivo, Profundidad y Factor de ramificación.** — El objetivo verifica si un estado es solución, mientras la profundidad observa la longitud de la solución más corta y el F. de ramificación el número promedio de sucesores por estado.

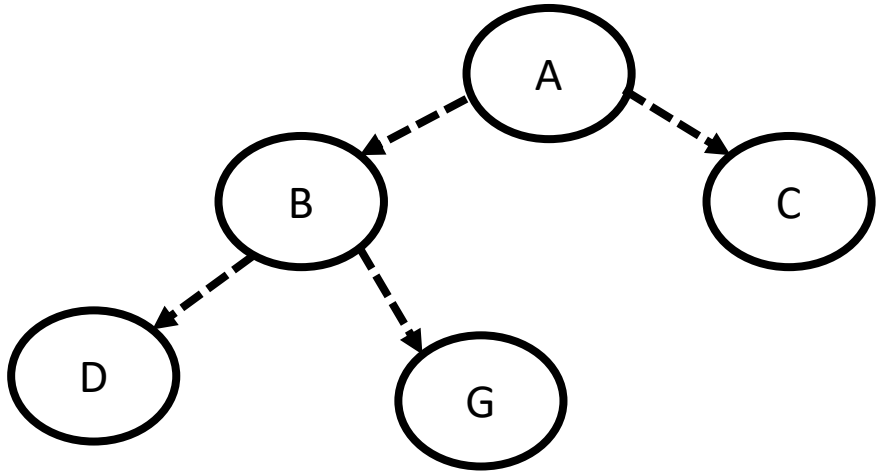
Búsqueda en Amplitud (BFS)

Se utiliza comúnmente para recorrer árboles o grafos. Explora los nodos vecinos en amplitud, visitando los nodos del mismo nivel antes de pasar al siguiente. Este enfoque utiliza una cola para facilitar su implementación [2].

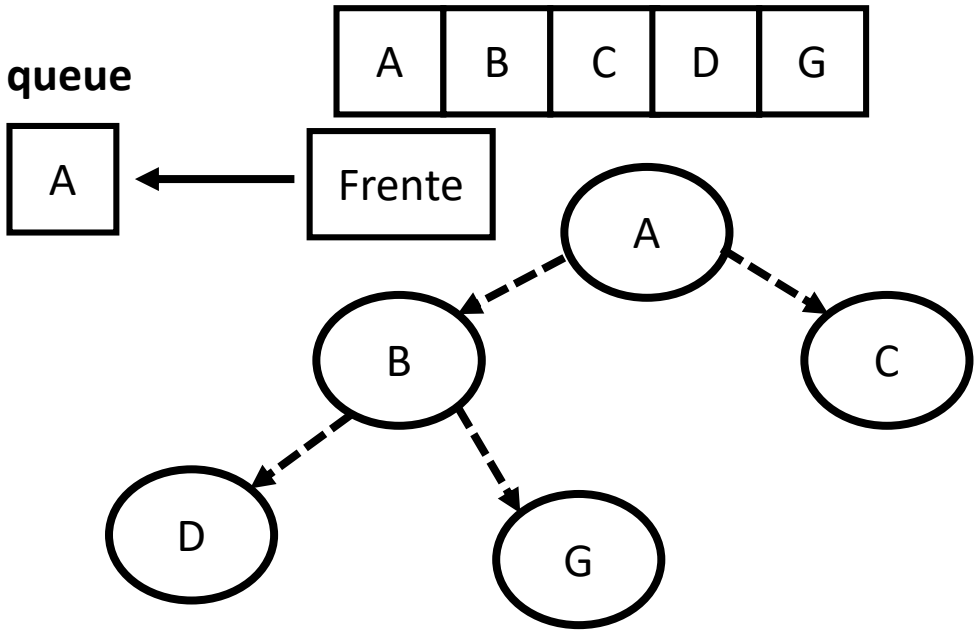
Ejemplo 1:

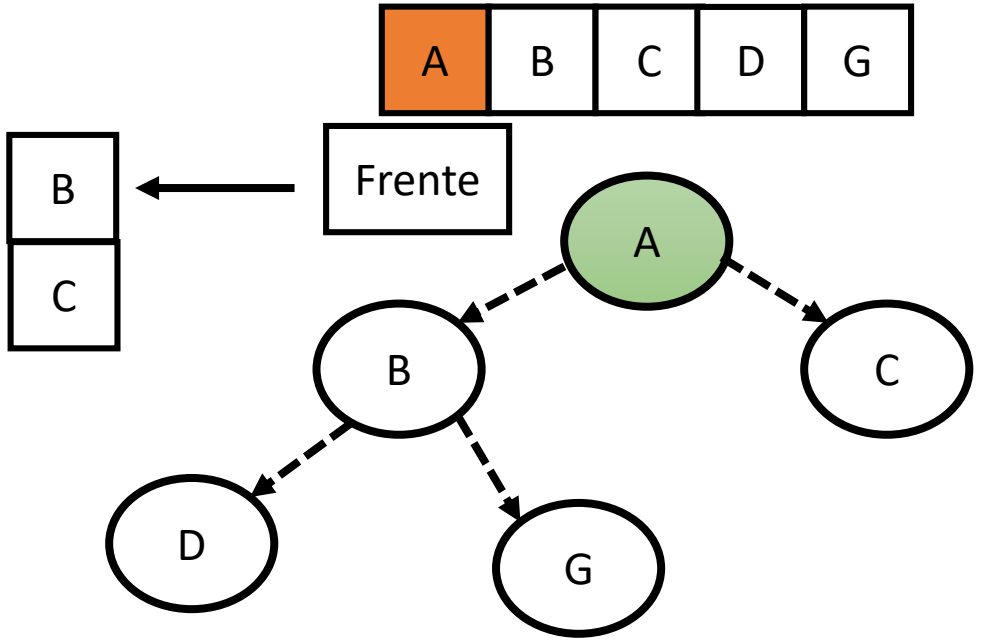
Visitado

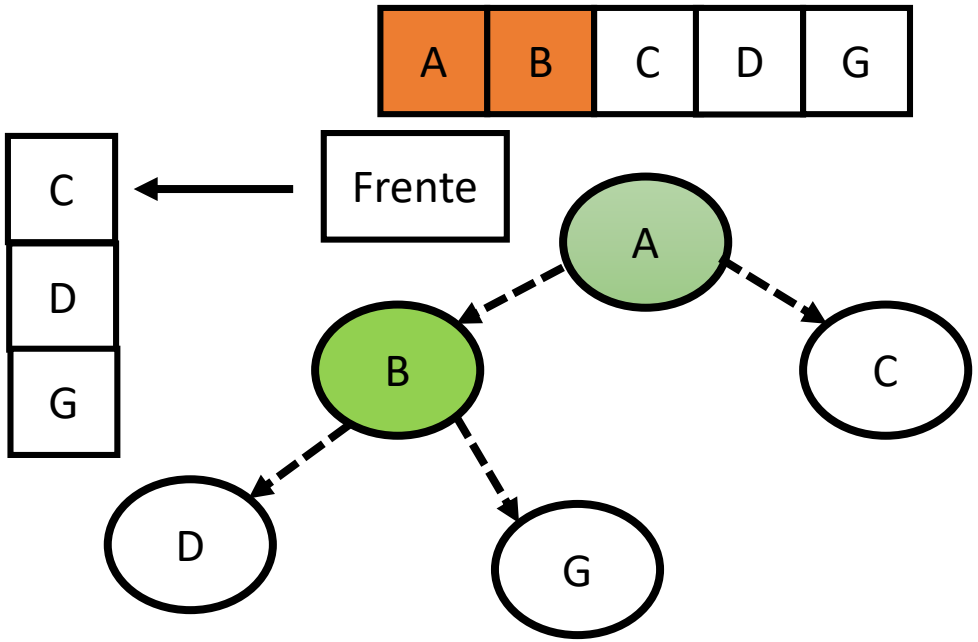
A	B	C	D	G

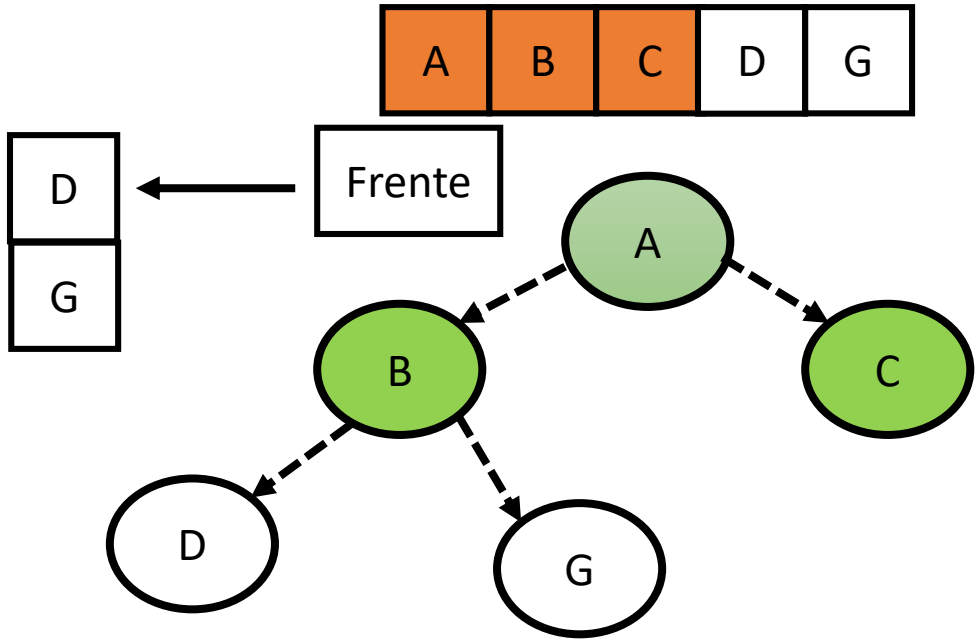


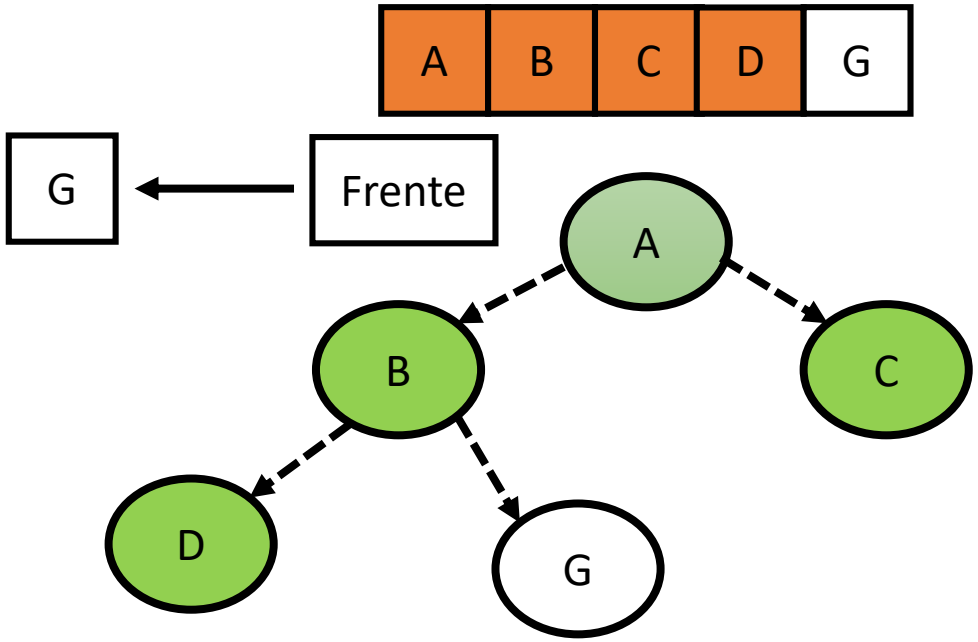
Partiendo del nodo fuente *A* , se explora las ramas de forma ordenada, es decir, de *A* a *B y* de ahí a *C* , donde se completa el nivel. Luego, se pasa al siguiente nivel y se explora *D* y *G.*

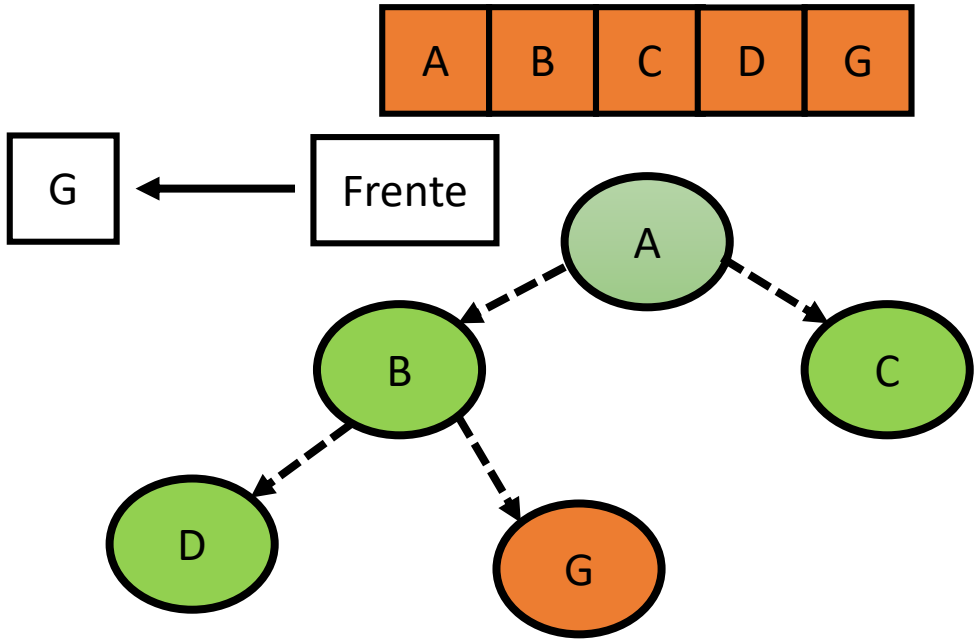

queue
A B C D G
A
Frente
A
B
C
D
G


A B C D G
B
C
Frente
A
B
C
D
G


A B C D G
C
D
G
Frente
A
B
C
D
G


A B C D G
D
G
Frente
A
B
C
D
G



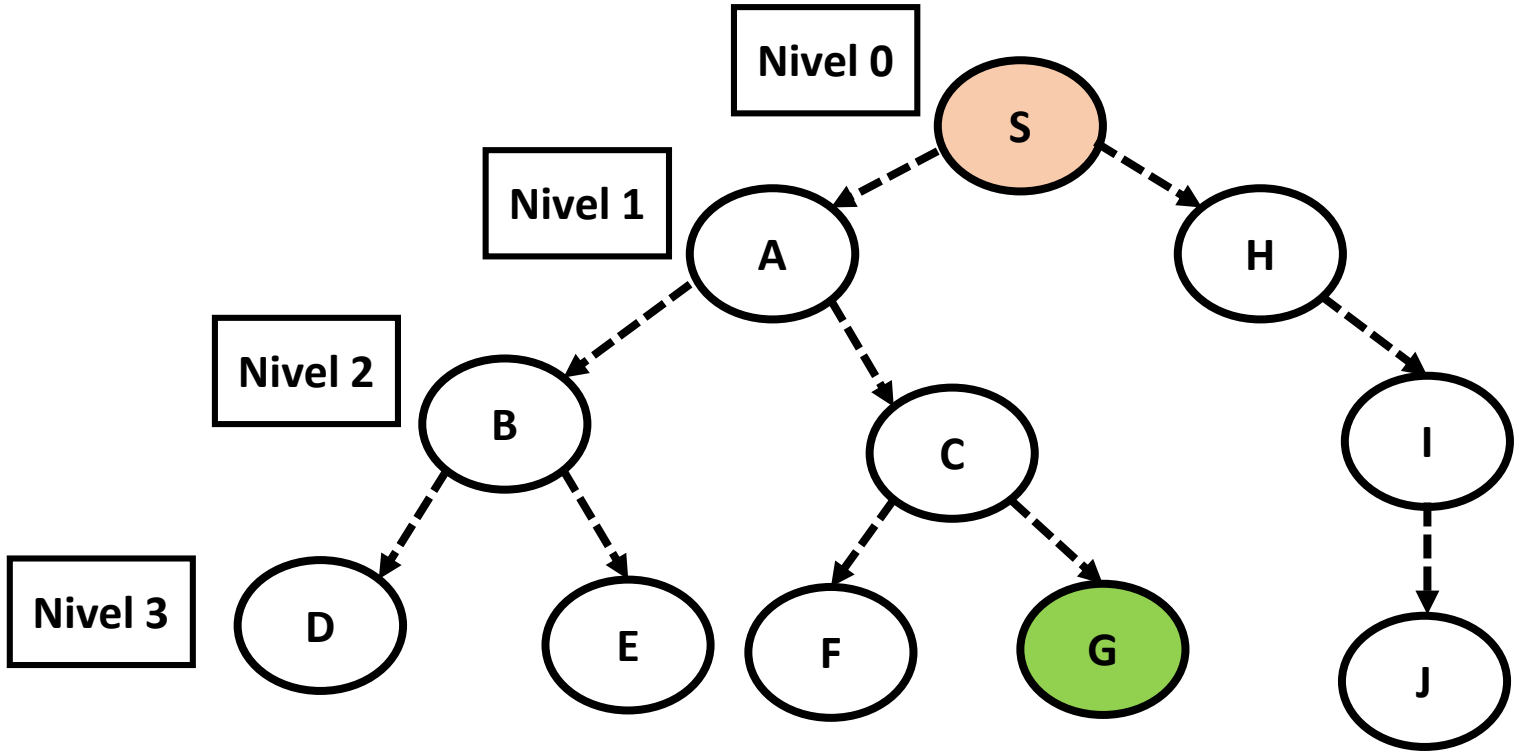


Complejidad Temporal

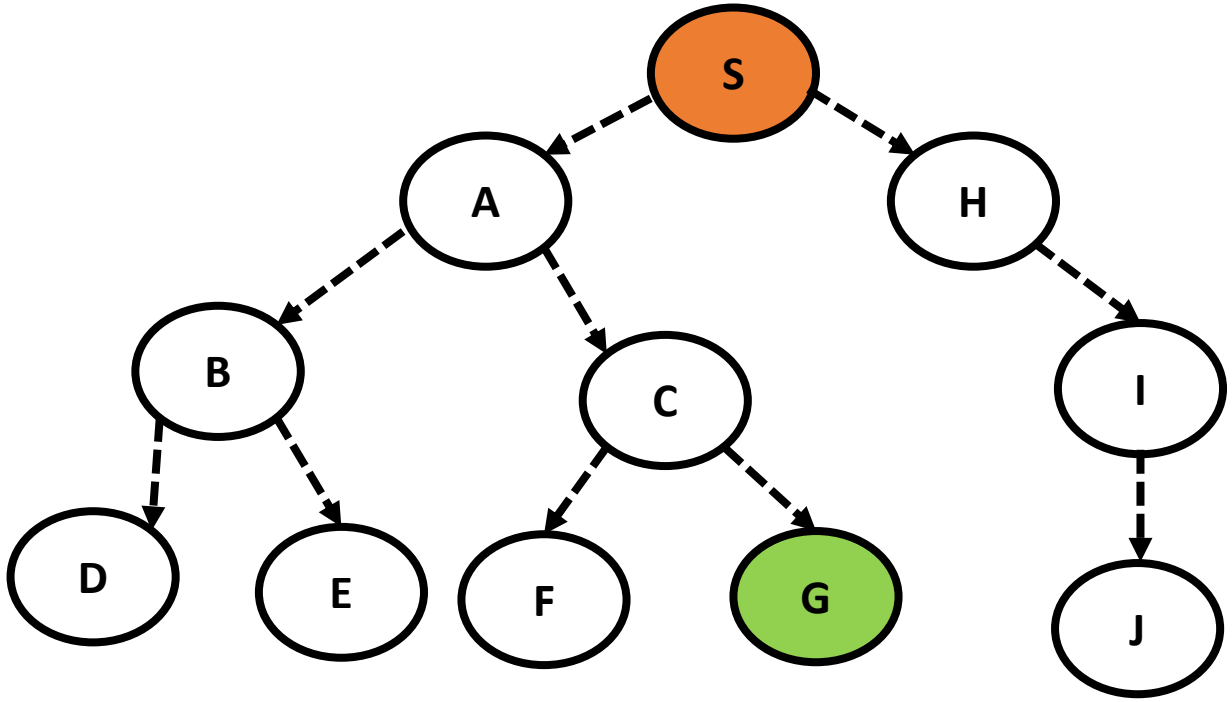
La complejidad temporal de BFS si se recorre todo el árbol es O(V) donde *V* es el número de nodos. En el caso de un grafo, la complejidad temporal es O(V+E) donde *V* es el número de vértices y *E* es el número de aristas.

Ejemplo 2:

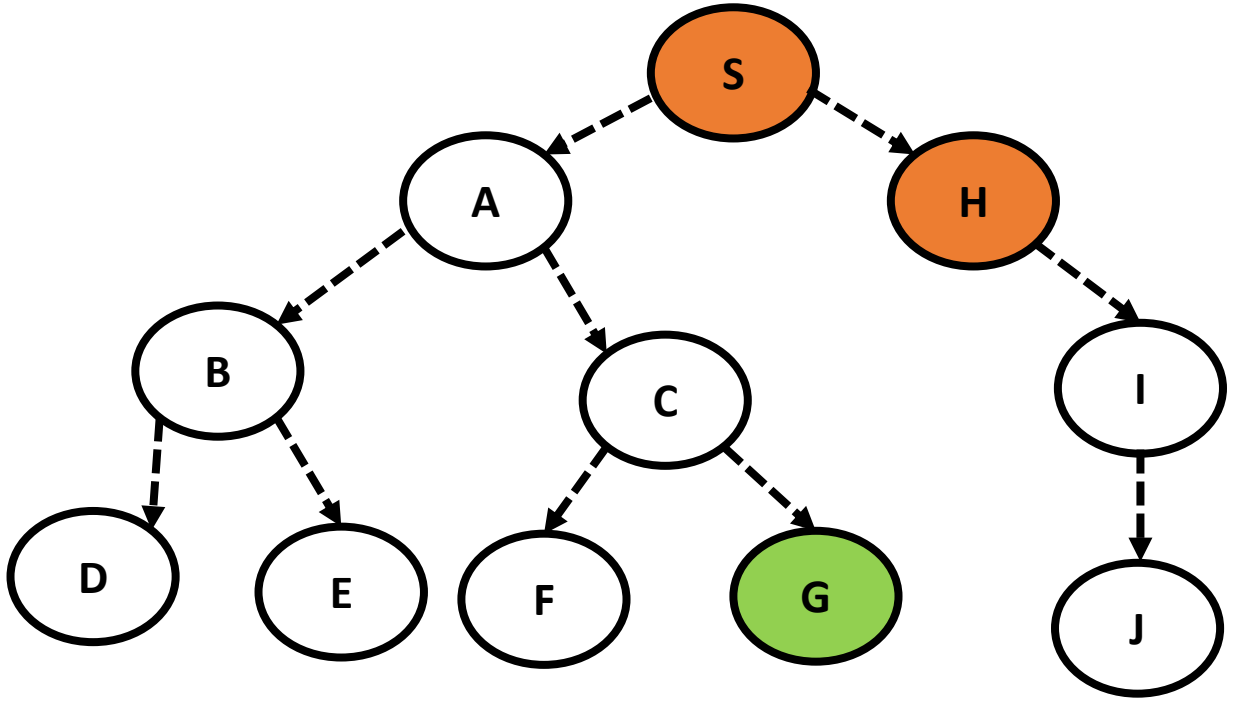
Se expone el recorrido de una búsqueda de amplitud, donde S representa el nodo inicial y G el nodo objetivo. La búsqueda finalizará al alcanzar el nodo objetivo G..



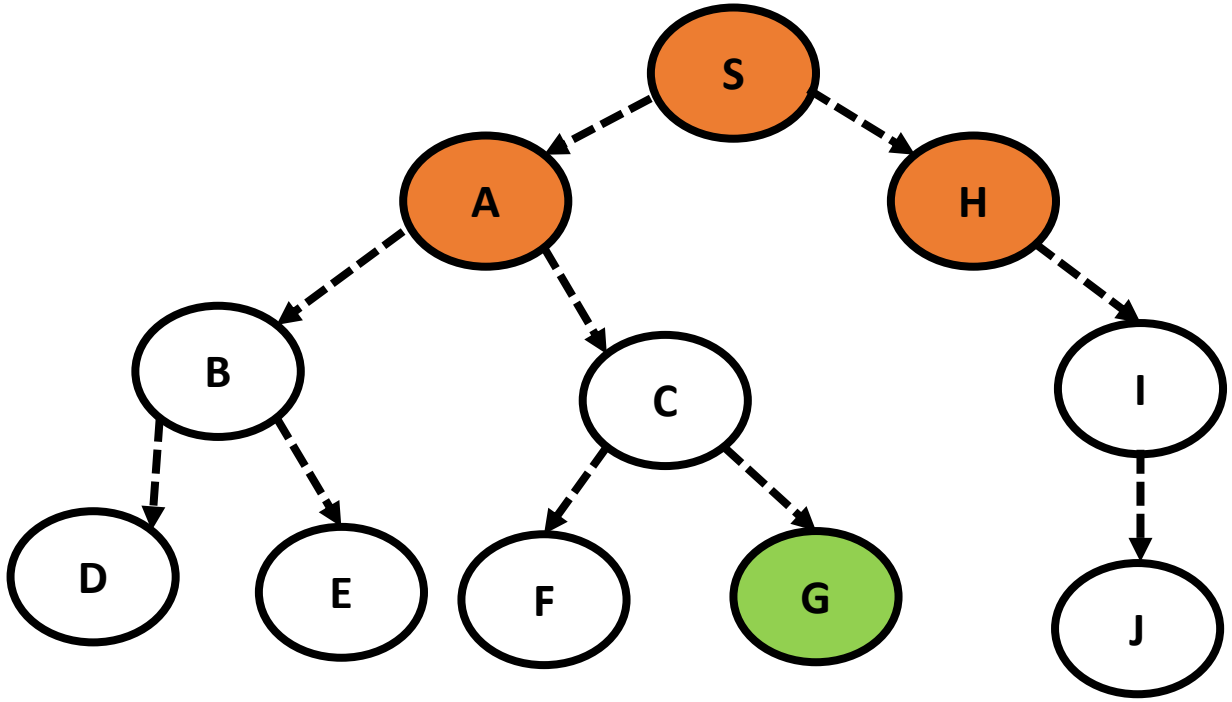
Recorrido 1



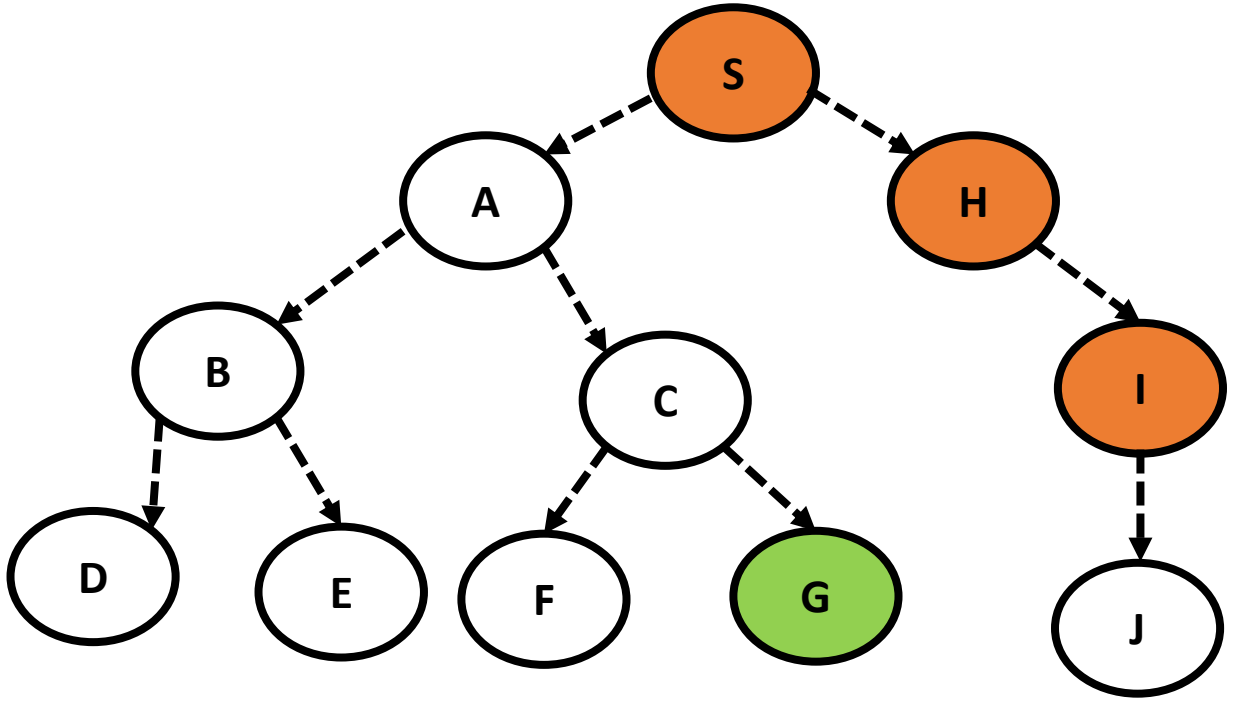
Recorrido 2



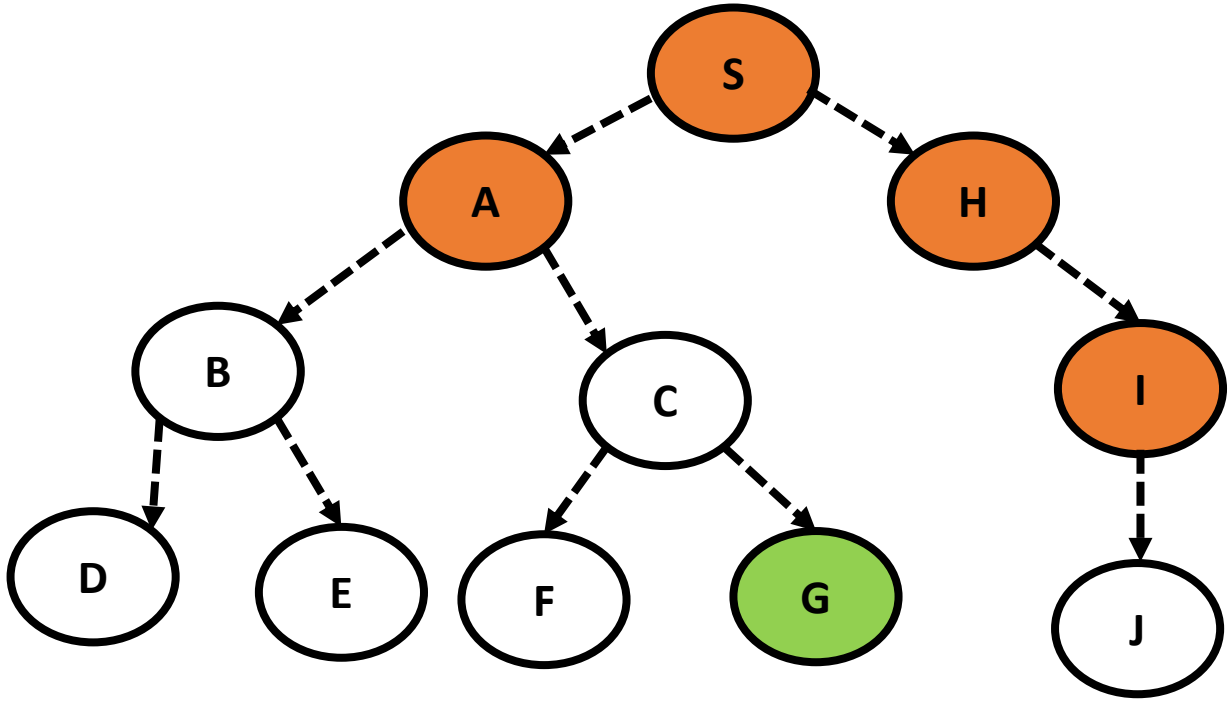
Recorrido 3



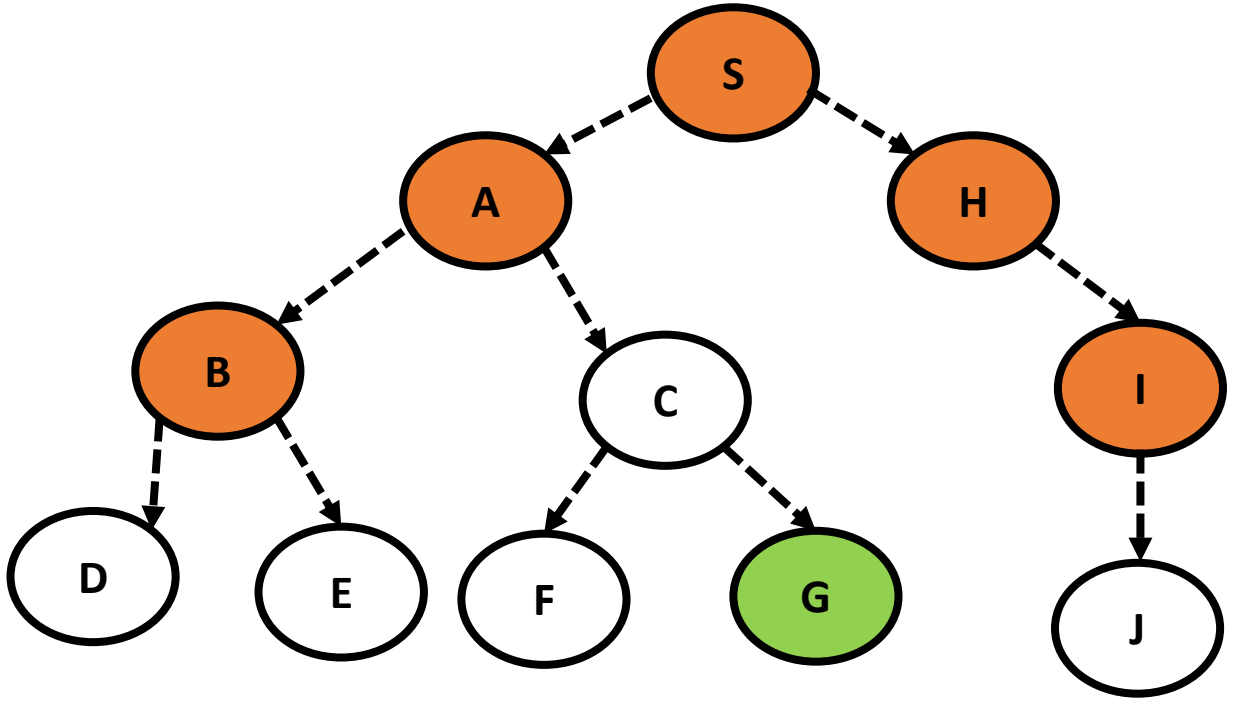
Recorrido 4



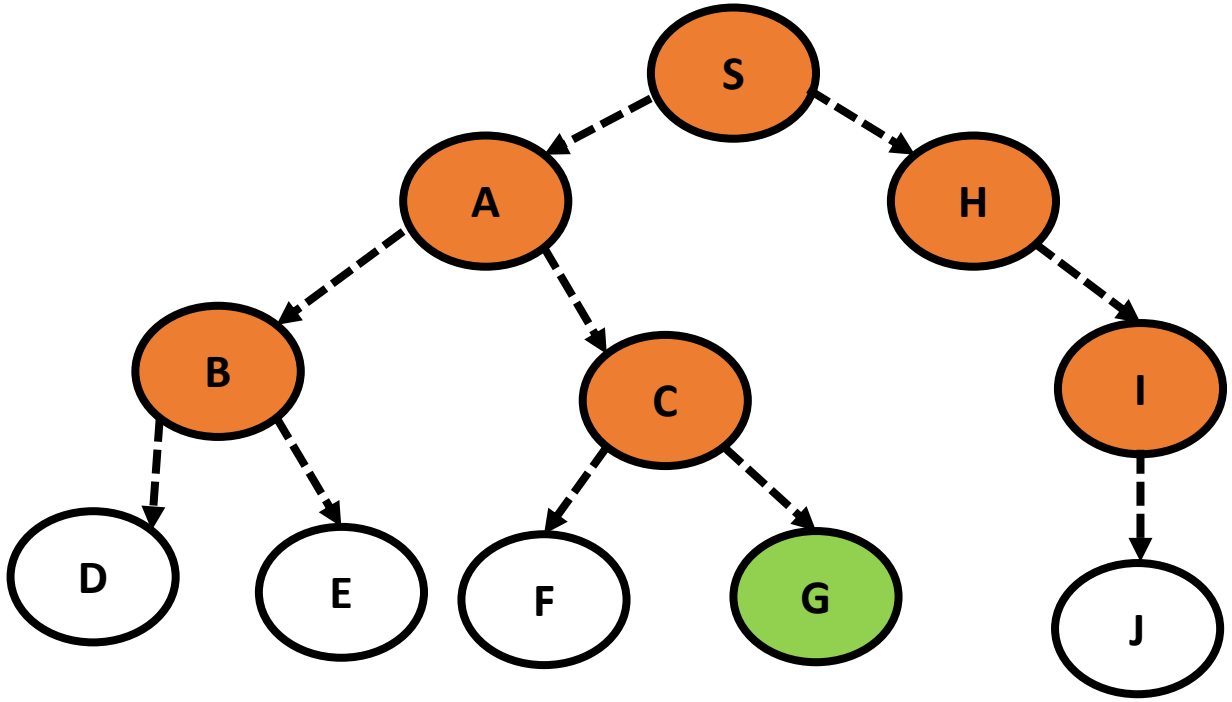
Recorrido 5



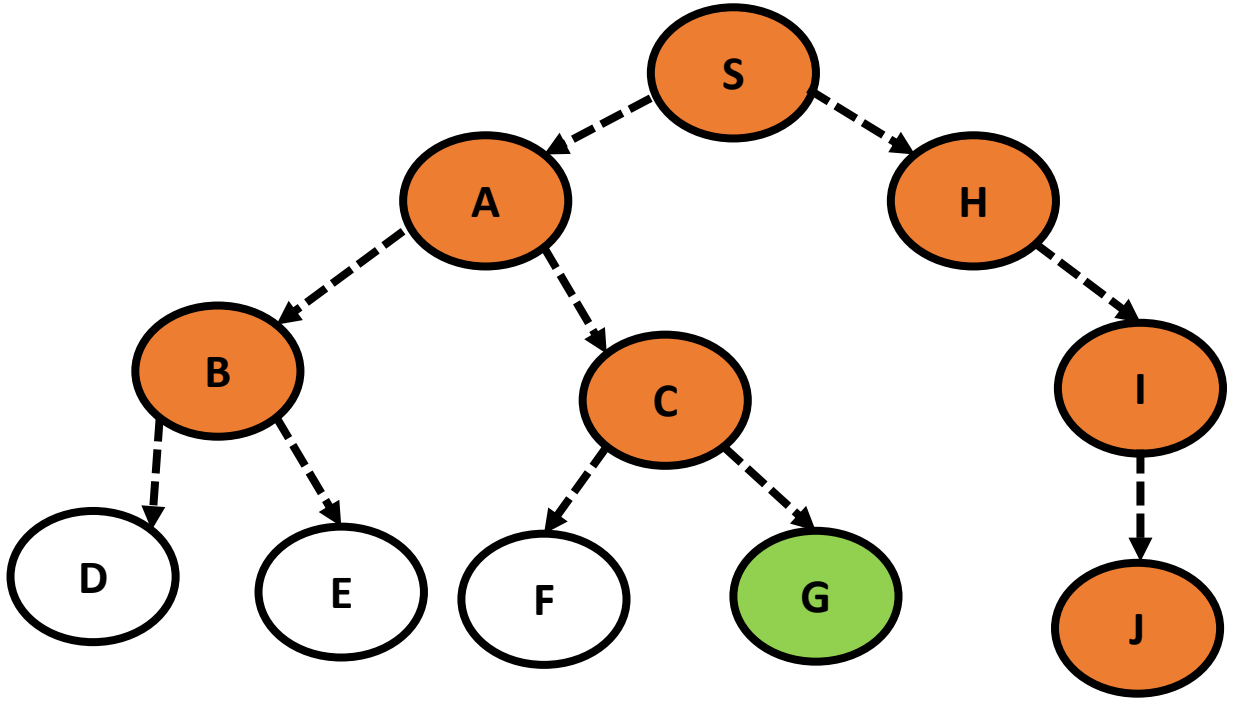
Recorrido 6



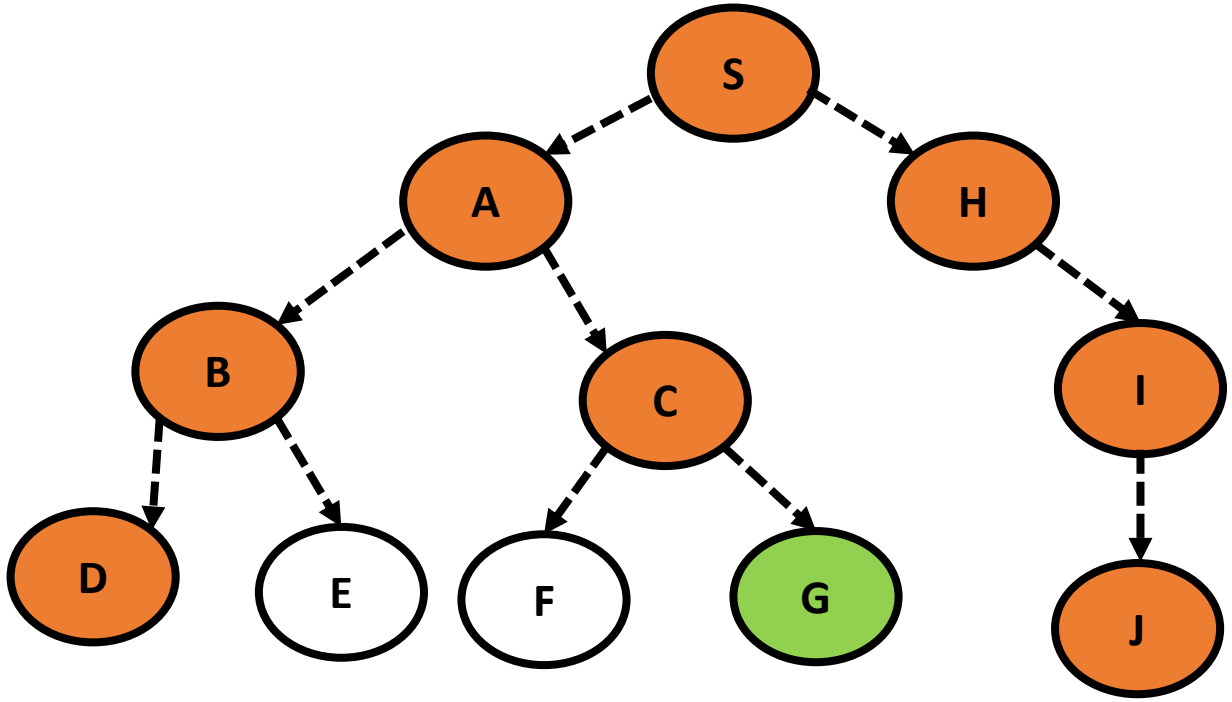
Recorrido 7



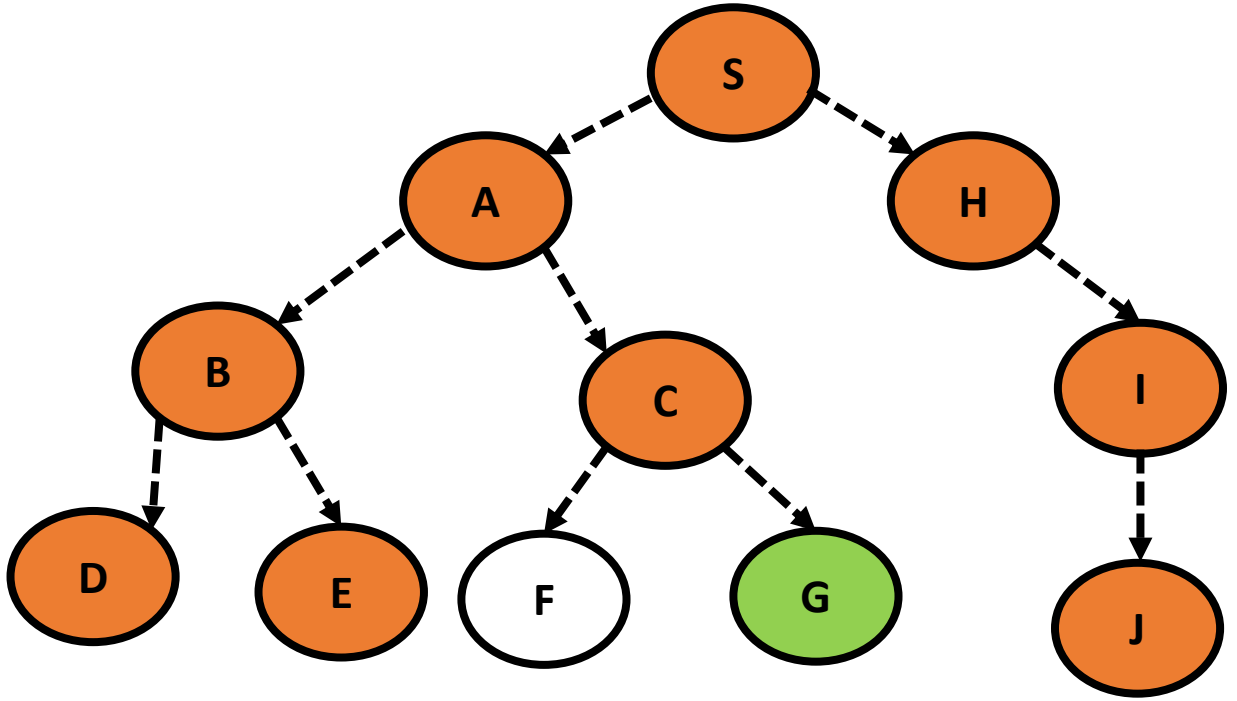
Recorrido 8



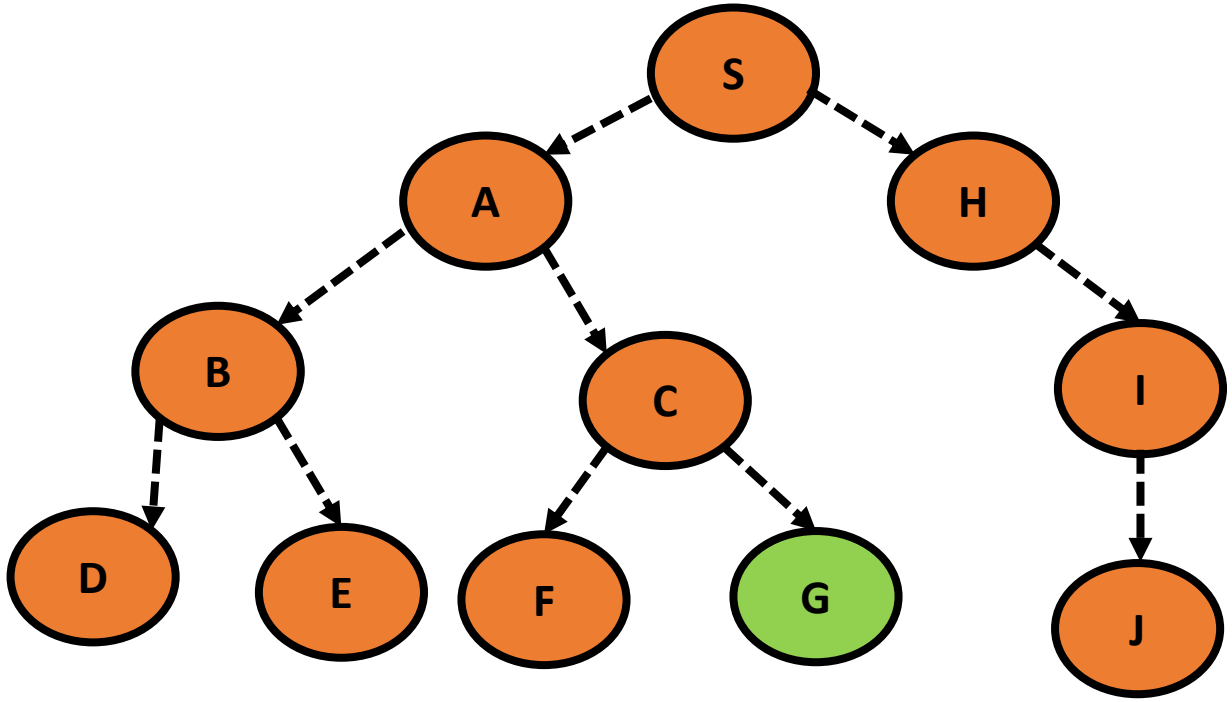
Recorrido 9



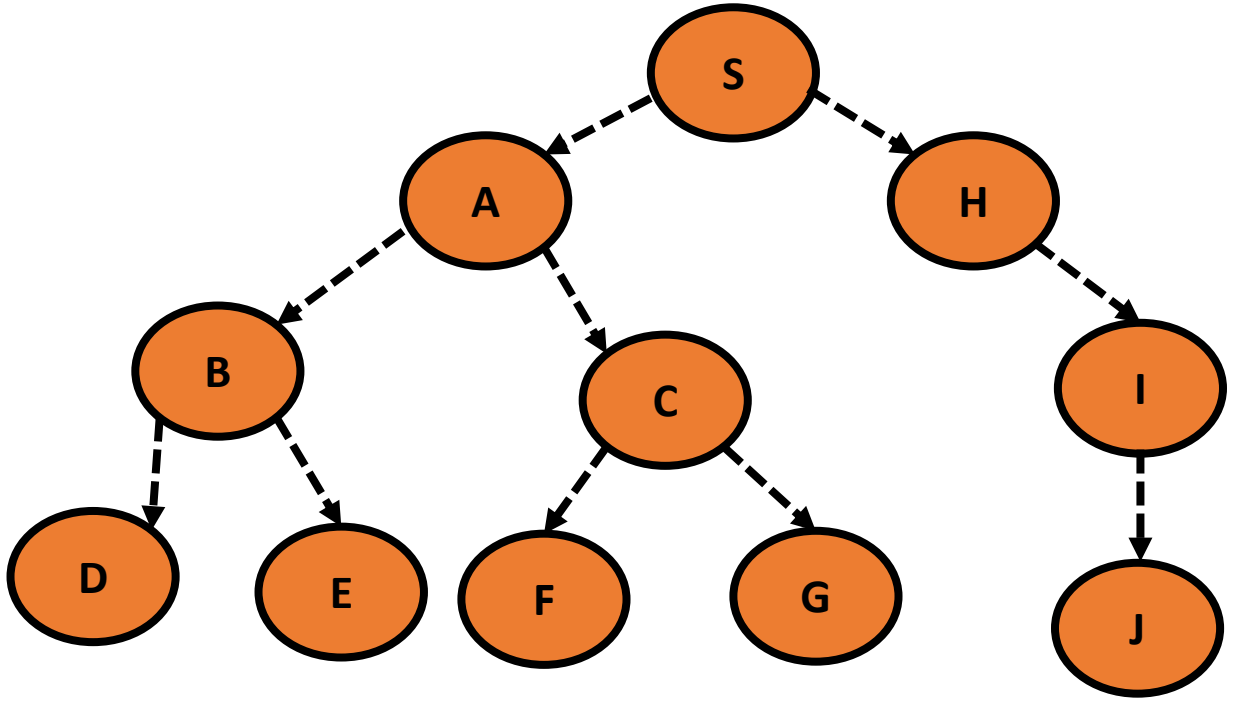
Recorrido 10



Recorrido 11



Recorrido 12

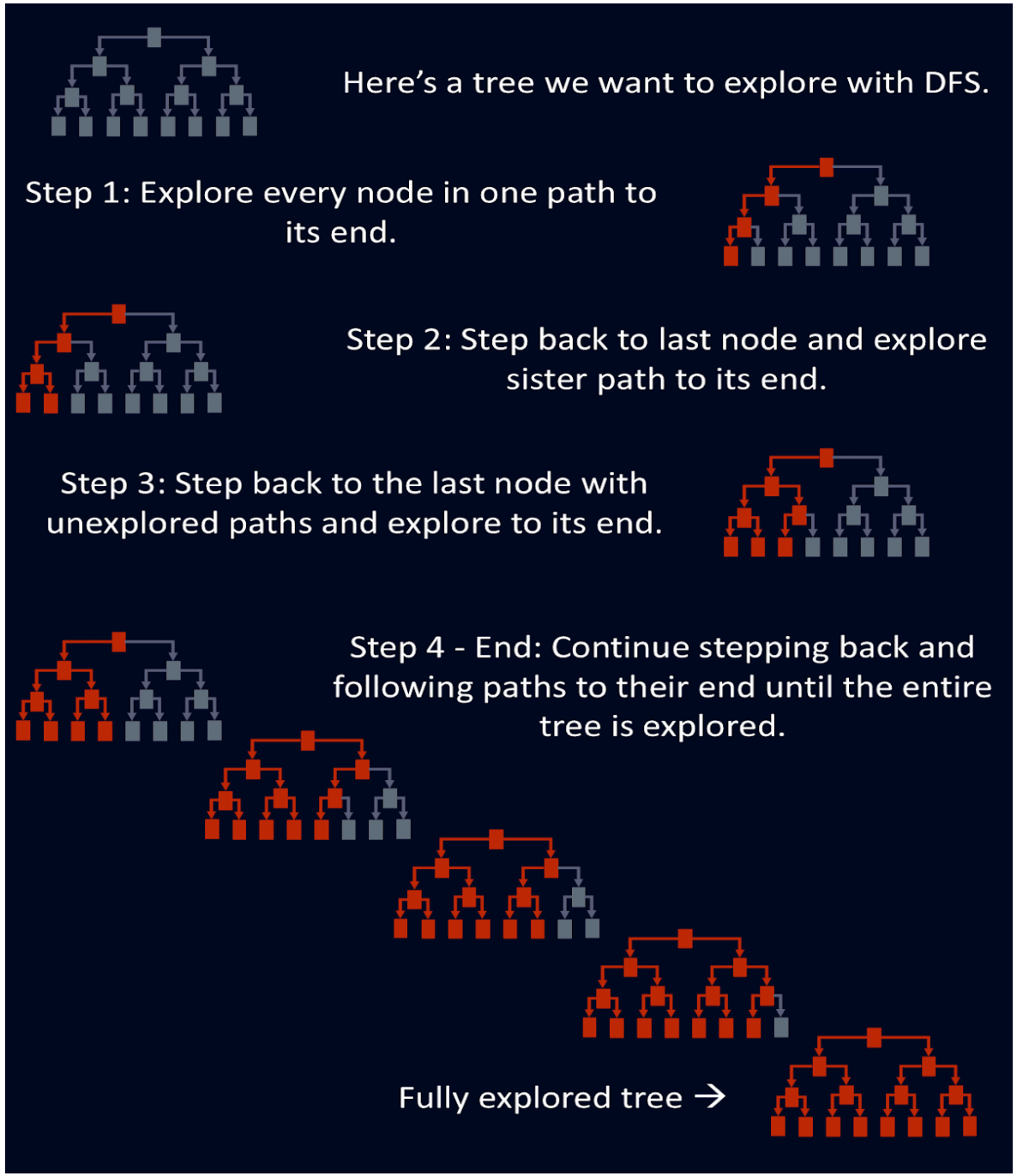


Código

```python
from collections import deque

def bfs(graph, start, goal):
    visited = set()   # Set to keep track of visited nodes
    queue = deque([[start]])  # Queue for BFS, starting with the initial node

    if start == goal:
        return "Start and goal nodes are the same"

    while queue:
        path = queue.popleft()  # Get the first path from the queue

        node = path[-1]  # Get the last node from the path

        if node not in visited:
            neighbors = graph[node]  # Get neighbors of the current node

            for neighbor in neighbors:
                new_path = list(path)  # Copy the current path
                new_path.append(neighbor)  # Add the neighbor to the path
                queue.append(new_path)  # Add the new path to the queue

                if neighbor == goal:
                    return new_path  # If neighbor is the goal, return the path

            visited.add(node)  # Mark the node as explored

    return "No path found between start and goal"

# Example graph represented as an adjacency list
graph = {
    'S' : ['A','H'],
    'A': ['B', 'C'],
    'B': ['D', 'E', 'E'],
    'C': ['F', 'G'],
    'H': ['I'],
    'I': ['J']
}

start_node = 'S'
End_node = 'G'
print("BFS Path:", bfs(graph, start_node, End_node))
```

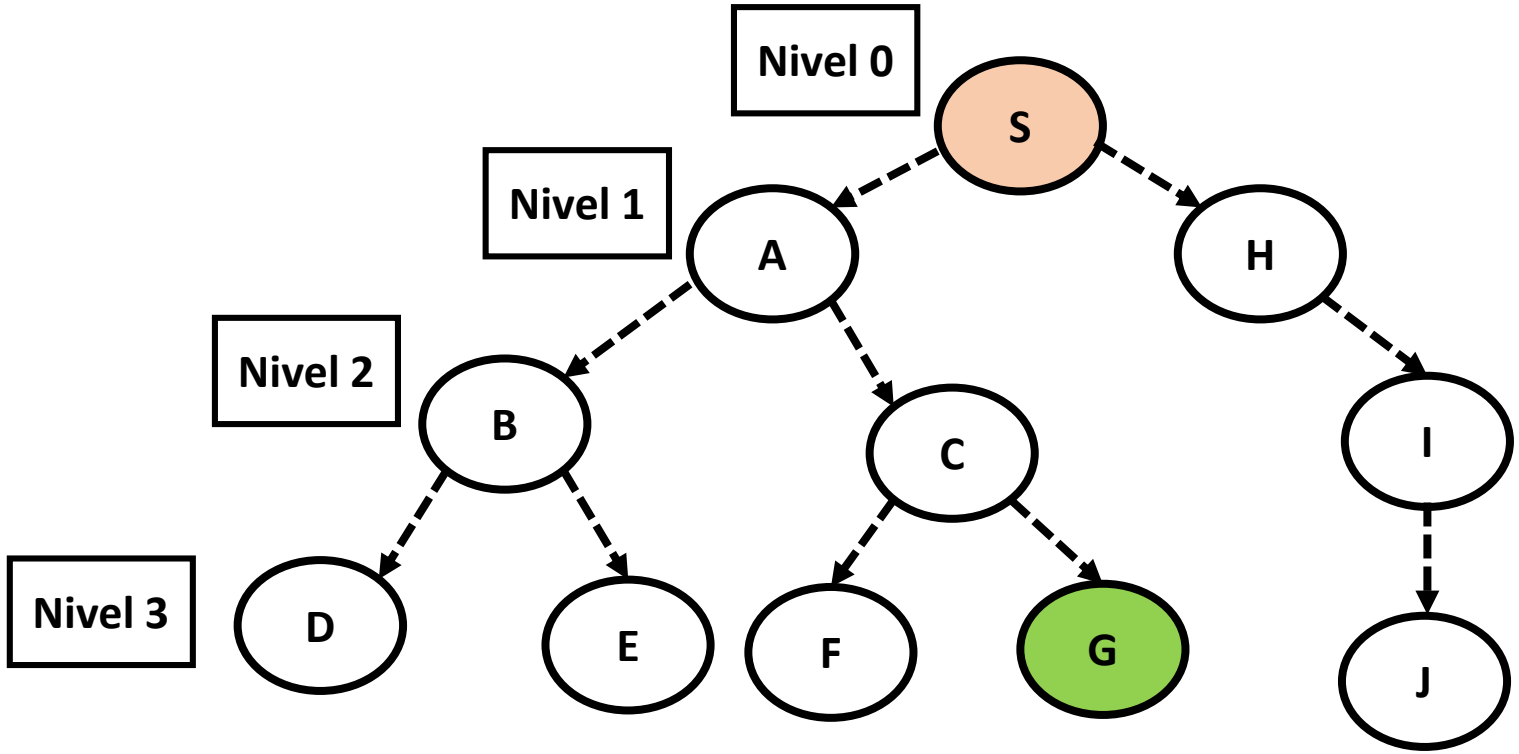
Búsqueda en Profundidad (DFS)

Es un algoritmo de búsqueda no informado que permite recorrer un árbol o grafo mediante el concepto de retroceso. Este algoritmo recorre todos los nodos avanzando por una ruta específica y utiliza el retroceso al llegar al final de dicha ruta, lo que permite explorar rutas alternativas. El enfoque DFS utiliza una pila para facilitar su implementación.[3].



Ejemplo:

Se expone el recorrido de una búsqueda de profundidad, donde S representa el nodo inicial y G el nodo objetivo. La búsqueda finalizará al alcanzar el nodo objetivo G..



Código

```python
def dfs(graph, start, goal, path=None):
    if path is None:
        path = [start]  # Initialize the path

    if start == goal:
        return path  # If start is goal, return the path

    if start not in graph:
        return None  # If start is not in graph, return None

    for node in graph[start]:
        if node not in path:  # Avoid cycles
            new_path = path + [node]  # Append node to the path
            result = dfs(graph, node, goal, new_path)  # Recursive call

            if result is not None:
                return result  # If a path is found, return it

    return None  # If no path is found, return None

# Example graph represented as an adjacency list
graph = {
    'S' : ['A','H'],
    'A': ['B', 'C'],
    'B': ['D', 'E', 'E'],
    'C': ['F', 'G'],
    'H': ['I'],
    'I': ['J']
}

start_node = 'S'
End_node = 'G'
print("DFS Path:", dfs(graph, start_node, End_node))
```

Búsqueda de Costos Uniformes(UCS)

Es un algoritmo de búsqueda no informado que explora los nodos de un grafo según el costo acumulado más bajo, buscando la ruta más rentable desde el origen hasta el destino. Comienza desde el nodo raíz y expande los nodos según su costo acumulado mínimo. El enfoque UCS utiliza una cola de prioridad para facilitar su implementación [4].

```python
import heapq

def ucs(grafo, inicio, meta):
    cola = [(0, inicio)]  # (costo, nodo)
    visitados = set()
    costo_acumulado = {inicio: 0}
    padre = {inicio: None}

    while cola:
        costo, nodo = heapq.heappop(cola)
        if nodo in visitados:
            continue
        visitados.add(nodo)
        if nodo == meta:
            break
        for vecino, costo_arista in grafo[nodo]:
            nuevo_costo = costo + costo_arista
            if vecino not in costo_acumulado or nuevo_costo < costo_acumulado[vecino]:
                costo_acumulado[vecino] = nuevo_costo
                heapq.heappush(cola, (nuevo_costo, vecino))
                padre[vecino] = nodo

    return reconstruir_camino(padre, meta), costo_acumulado.get(meta, float('inf'))

# Grafo con costos
grafo_costo = {
    'A': [('B', 1), ('C', 4)],
    'B': [('D', 3), ('E', 2)],
    'C': [('F', 5)],
    'D': [],
    'E': [('F', 1)],
    'F': []
}

camino, costo = ucs(grafo_costo, 'A', 'F')
print("UCS - Camino:", camino, "Costo:", costo)
```

Referencias

[1] H. Franco, “Búsqueda no informada”, *Edu.co*. [En línea]. Disponible en: https://hpclab.ucentral.edu.co/~hfranco/int_art/Session_6_UninformedSearch.pdf. [Consultado: 22-ago-2025].

[2] “What is Breadth First Search?”, *HowDev*. [En línea]. Disponible en: https://how.dev/answers/what-is-breadth-first-search. [Consultado: 22-ago-2025].

[3] *Datacamp.com*. [En línea]. Disponible en: https://www.datacamp.com/es/tutorial/depth-first-search-in-python. [Consultado: 22-ago-2025].

[4]

UNIVERSIDAD SANTO TOMÁS

GRACIAS