

# ELEMENTARY GRAPH ALGORITHM

**Dani Hidayat**

**11220940000014**




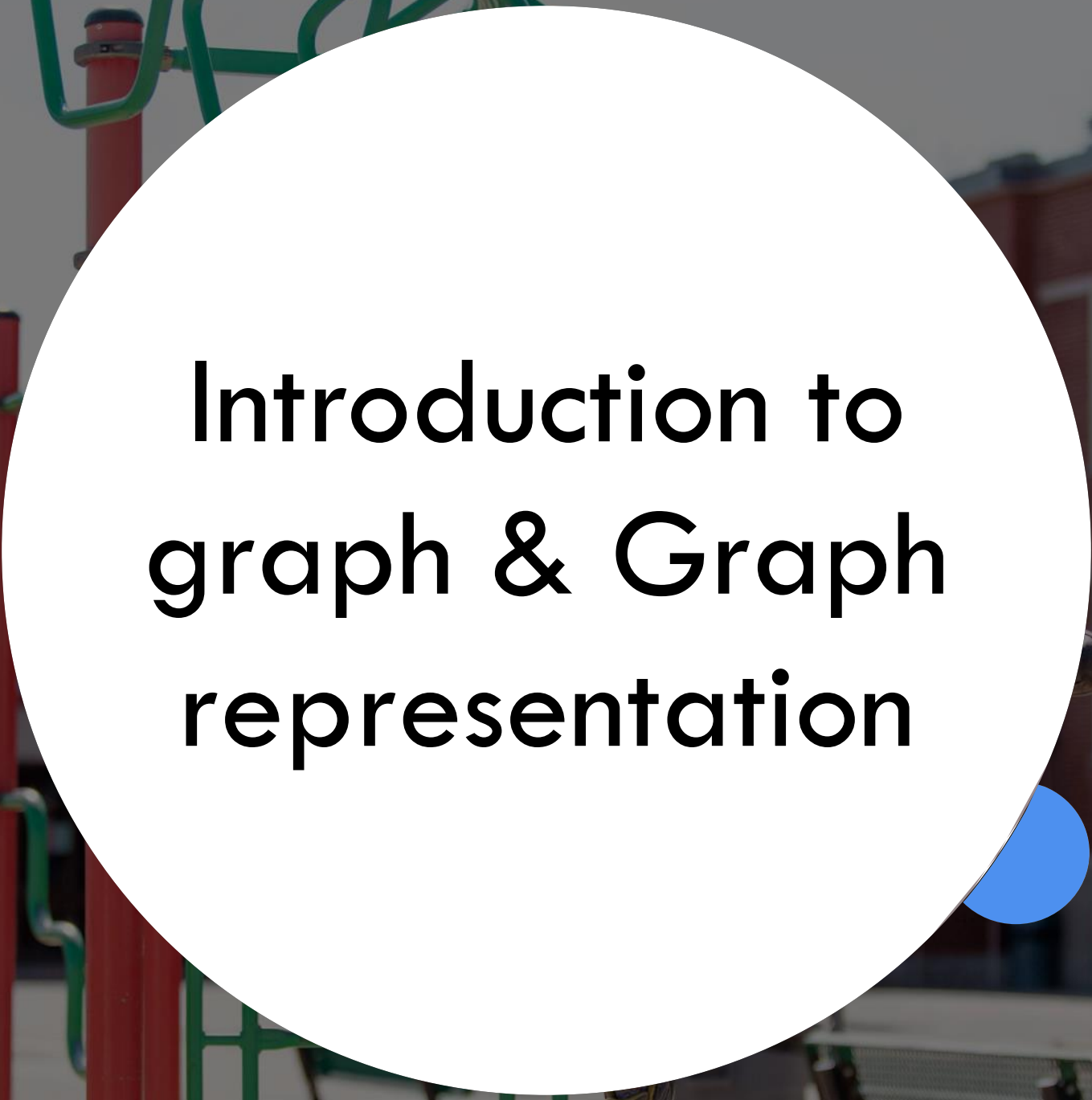
# Daftar Isi



Introduction to graph & Graph representation

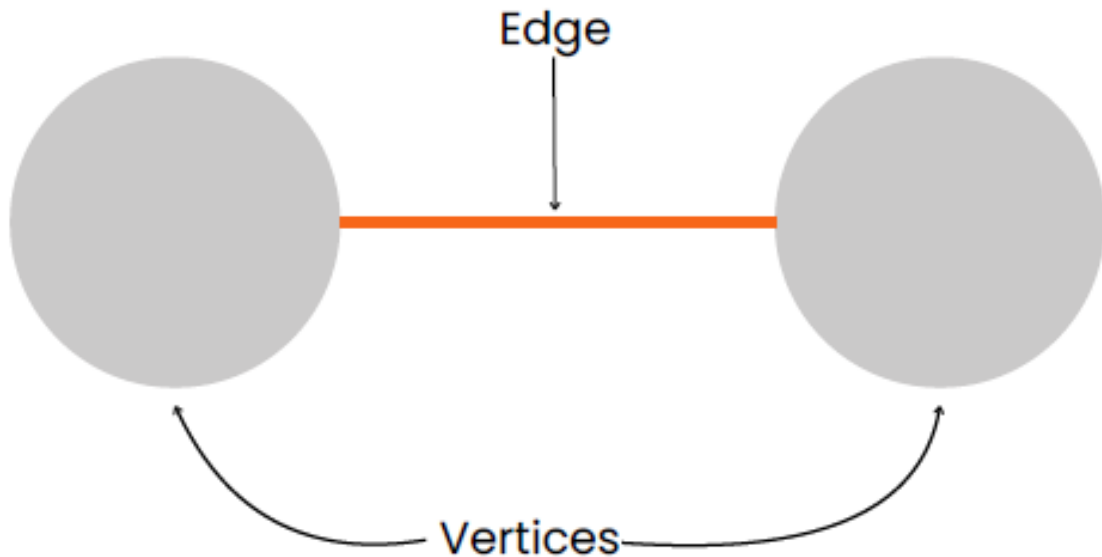
Breadth-first search

Depth-first search



# Introduction to graph & Graph representation

# Definisi Graph



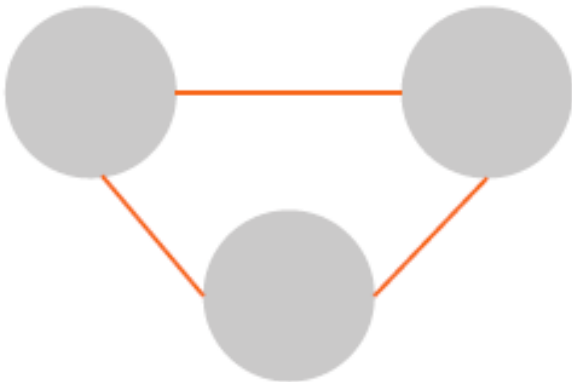
Graph adalah struktur data non-linier yang terdiri dari vertices dan edges yang mana dua vertices dihubungkan oleh sebuah edge.

Representasi visual dari graph adalah dengan menyatakan objek sebagai node atau titik (vertice), sedangkan hubungan antara objek dinyatakan dengan garis (edge).

# Type Graph

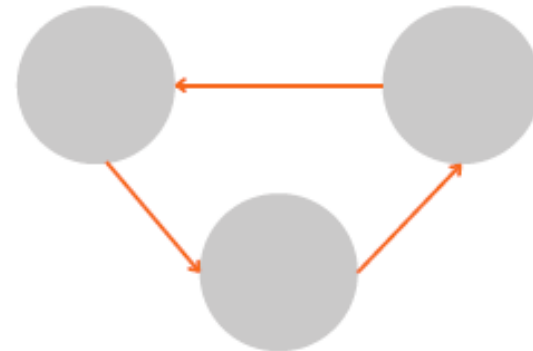
- **undirected graph**

$(u,v)$  dikatakan pasangan tak terurut (unordered), jika  $(u,v) = (v,u)$  dan self-loops tidak boleh ada.



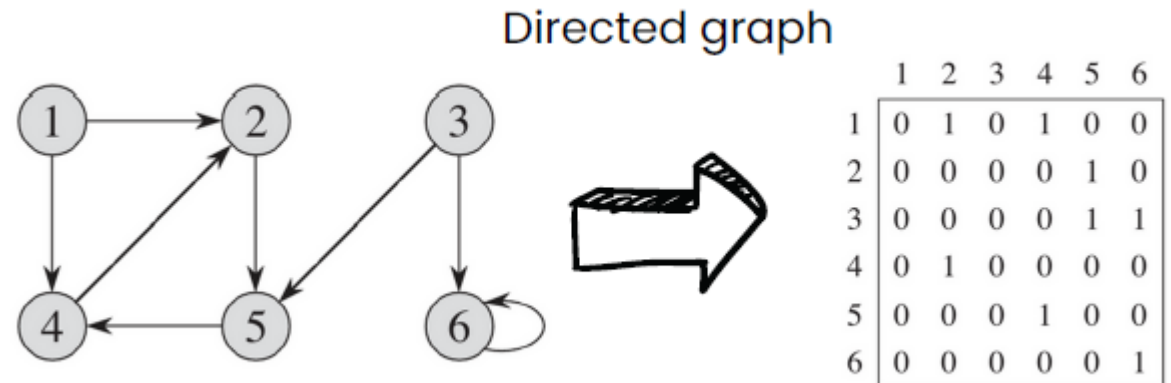
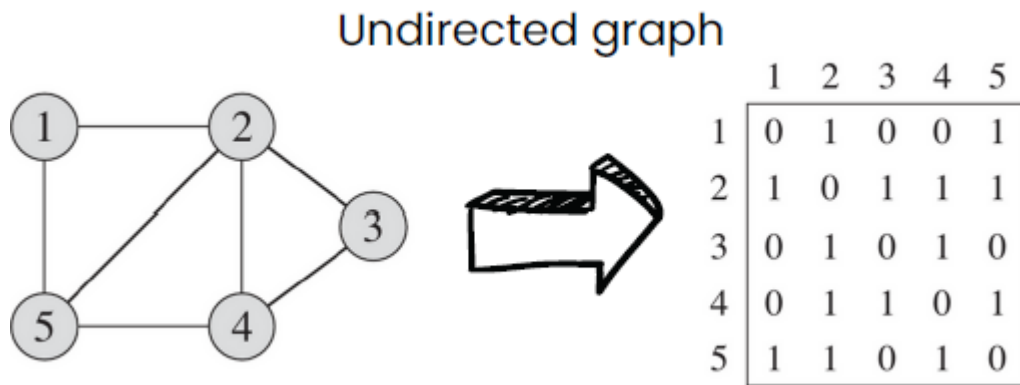
- **directed graph**

$(u,v)$  dikatakan pasangan terurut (ordered), jika  $(u,v) \neq (v,u)$  dan self-loops kemungkinannya ada.



# Graph Representation

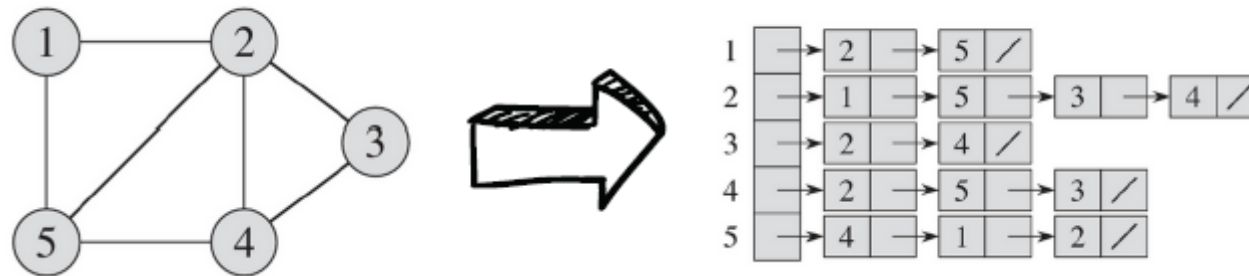
**Adjacency Matrix:** Representasi graph dengan matriks dua dimensi dimana entri  $(i, j)$  adalah 1 jika ada edge dari vertice  $i$  ke vertice  $j$ , dan 0 jika sebaliknya.



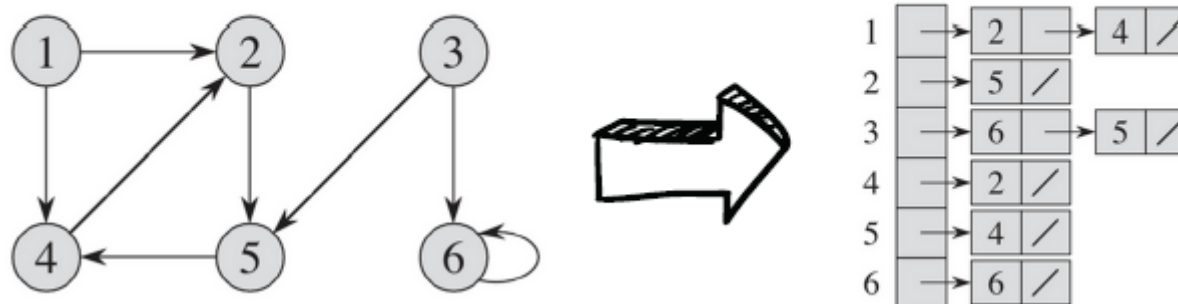
# Graph Representation

**Adjacency List:** Menyimpan vertices dalam graph dan vertices yang berdekatan dengannya.

Undirected graph



Directed graph








# BREADTH-FIRST SEARCH



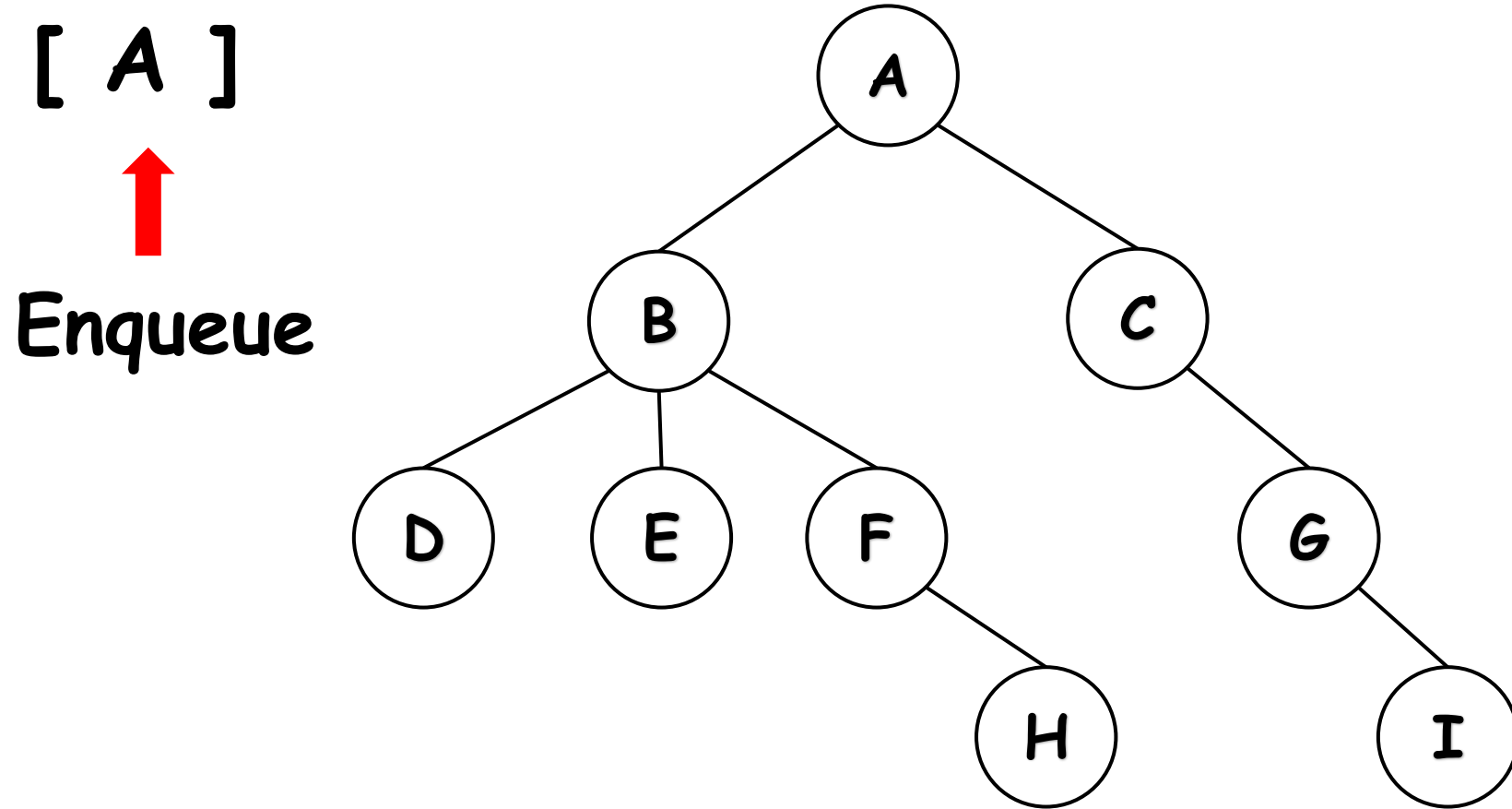


# BREADTH-FIRST SEARCH

Breadth-First Search (BFS) adalah algoritma traversal graf yang mengeksplorasi semua vertices dalam graf pada kedalaman saat ini sebelum berpindah ke vertices pada tingkat kedalaman berikutnya. Ia dimulai dari titik tertentu dan mengunjungi semua tetangganya sebelum berpindah ke tetangga tingkat berikutnya.

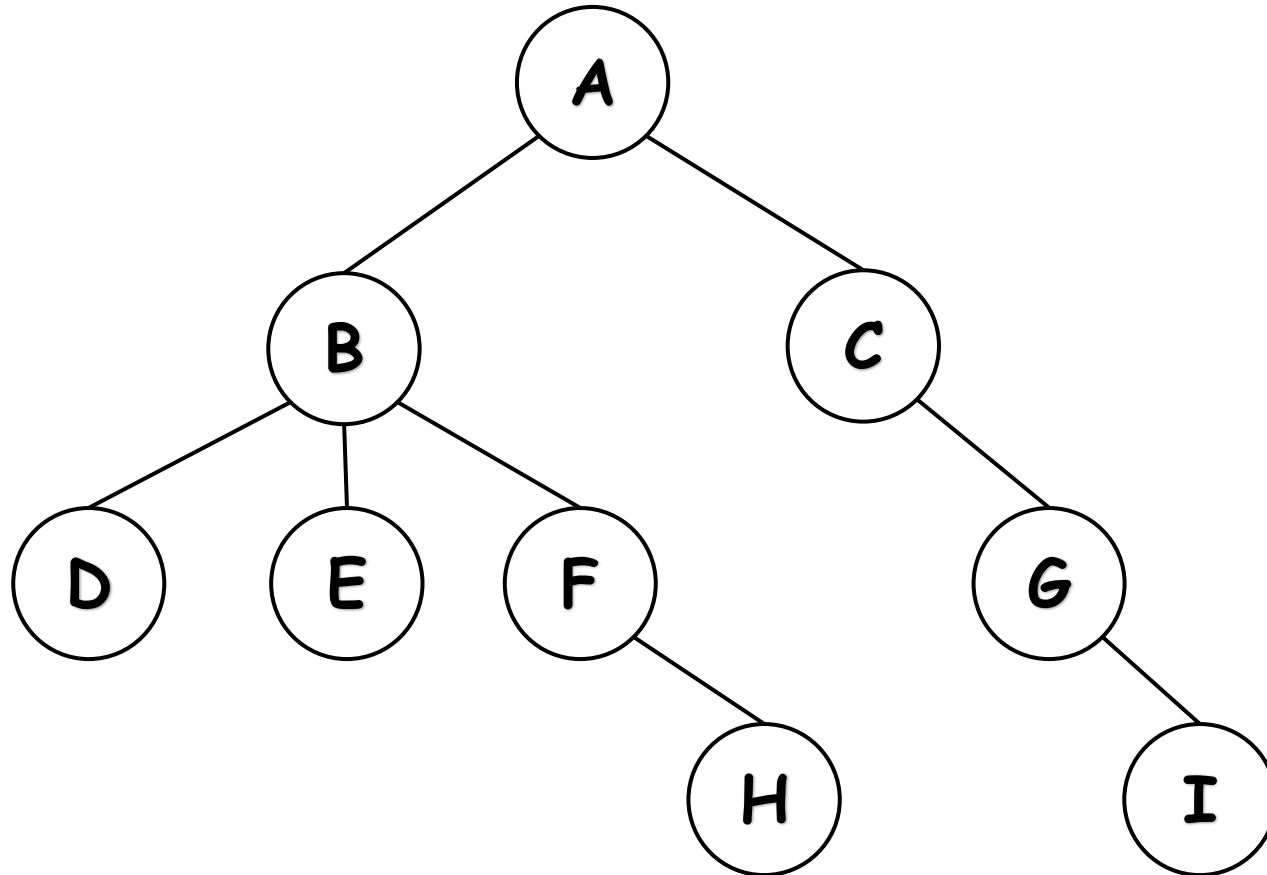


# Ilustrasi Breadth-First Search

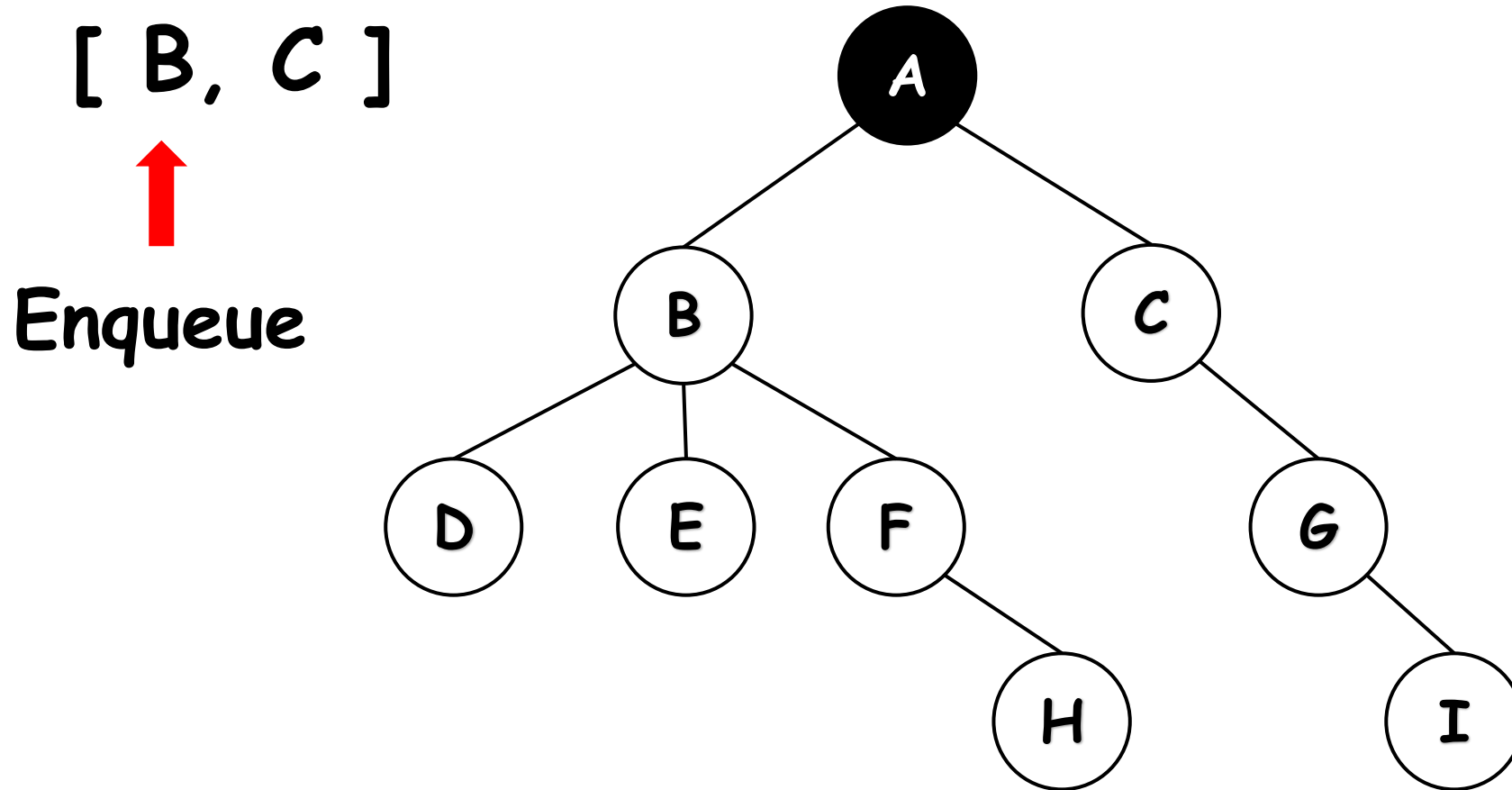


# Ilustrasi Breadth-First Search

[ ]

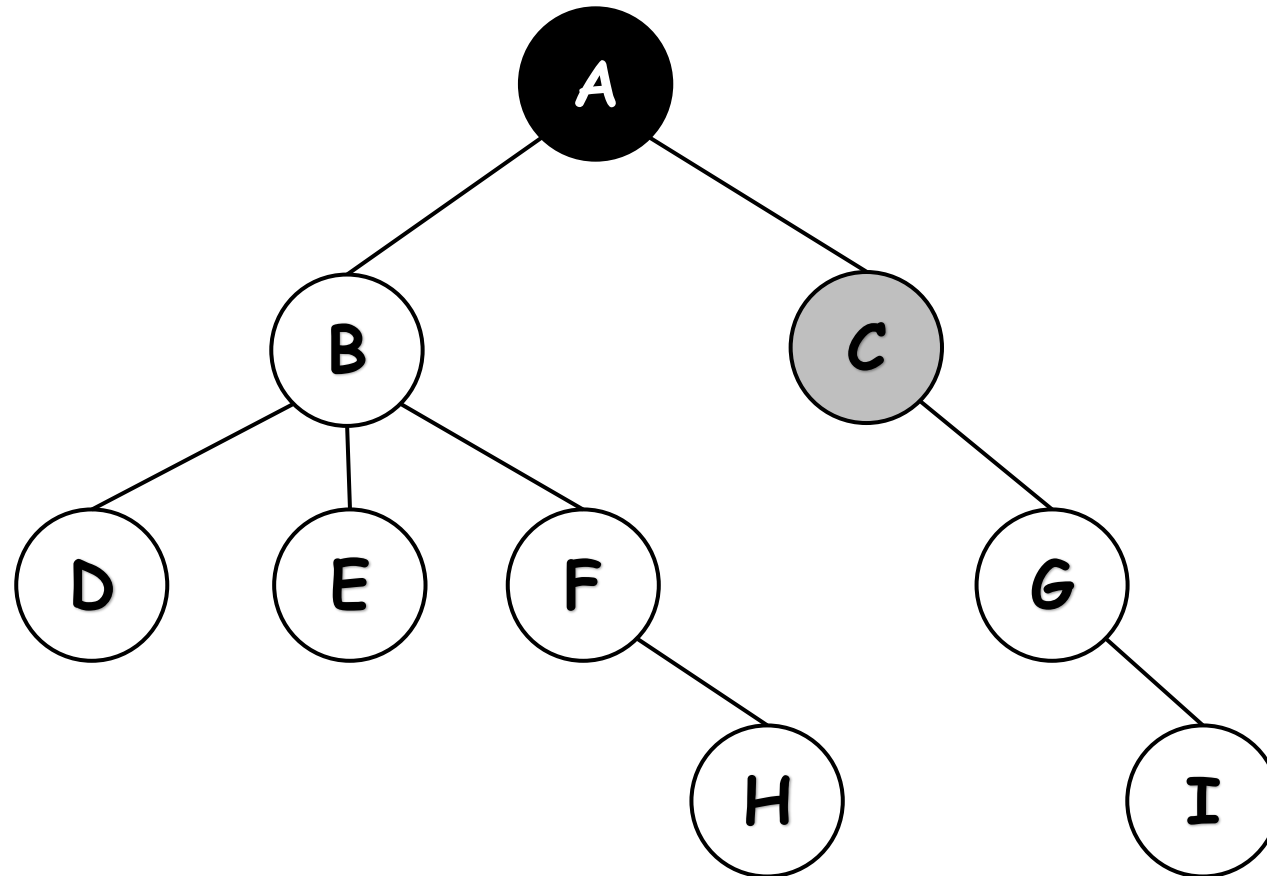


# Ilustrasi Breadth-First Search



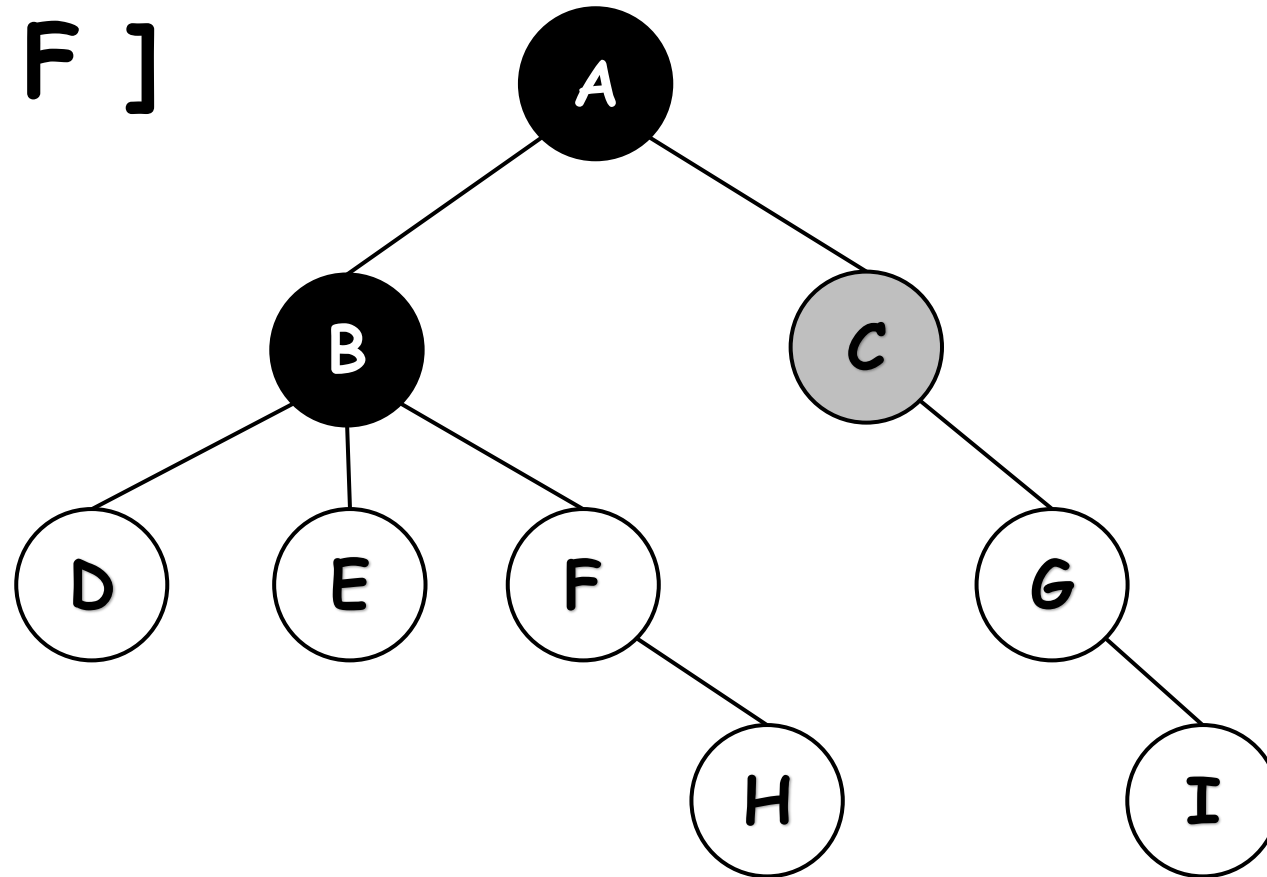
# Ilustrasi Breadth-First Search

[ C ]



# Ilustrasi Breadth-First Search

[ C, D, E, F ]  
↑  
Enqueue

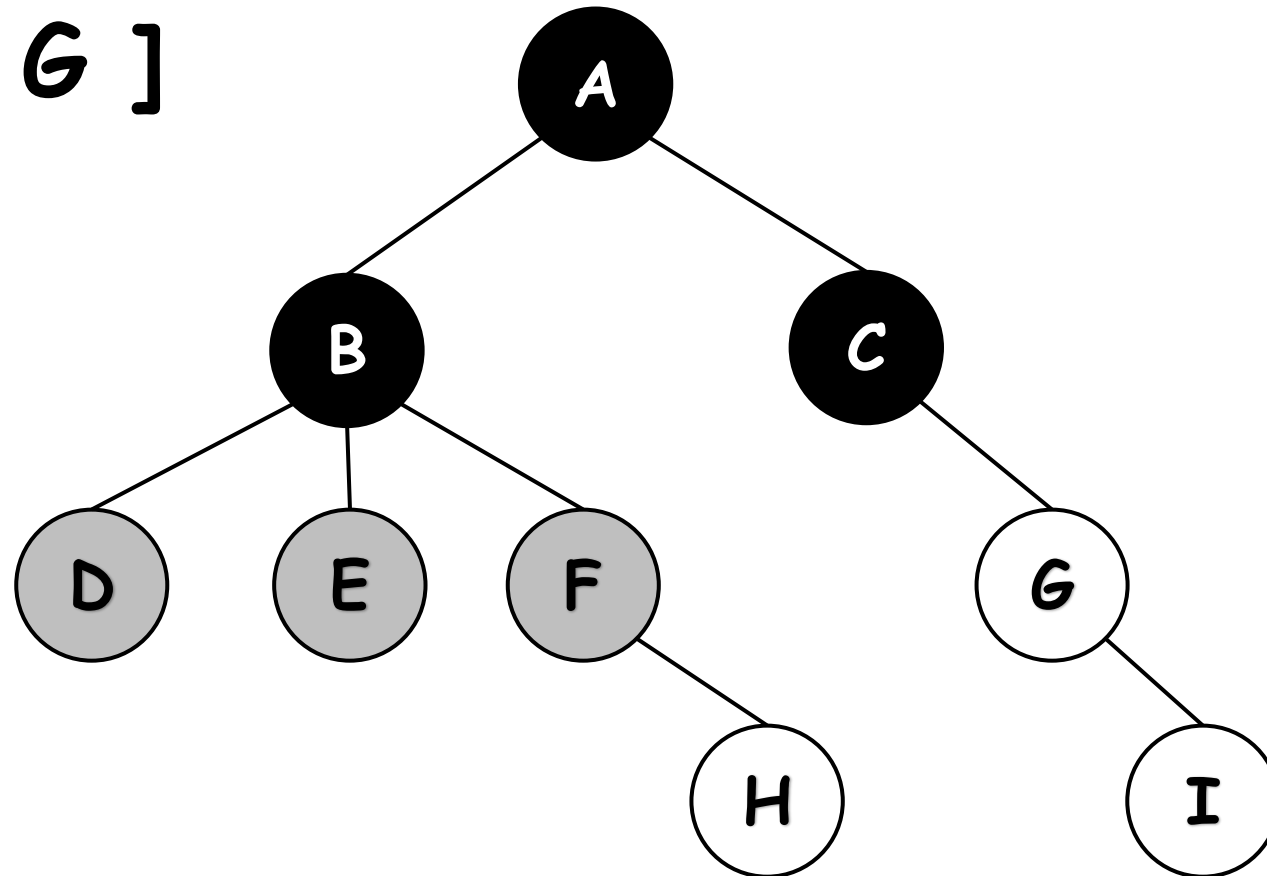


# Ilustrasi Breadth-First Search

[ D, E, F, G ]

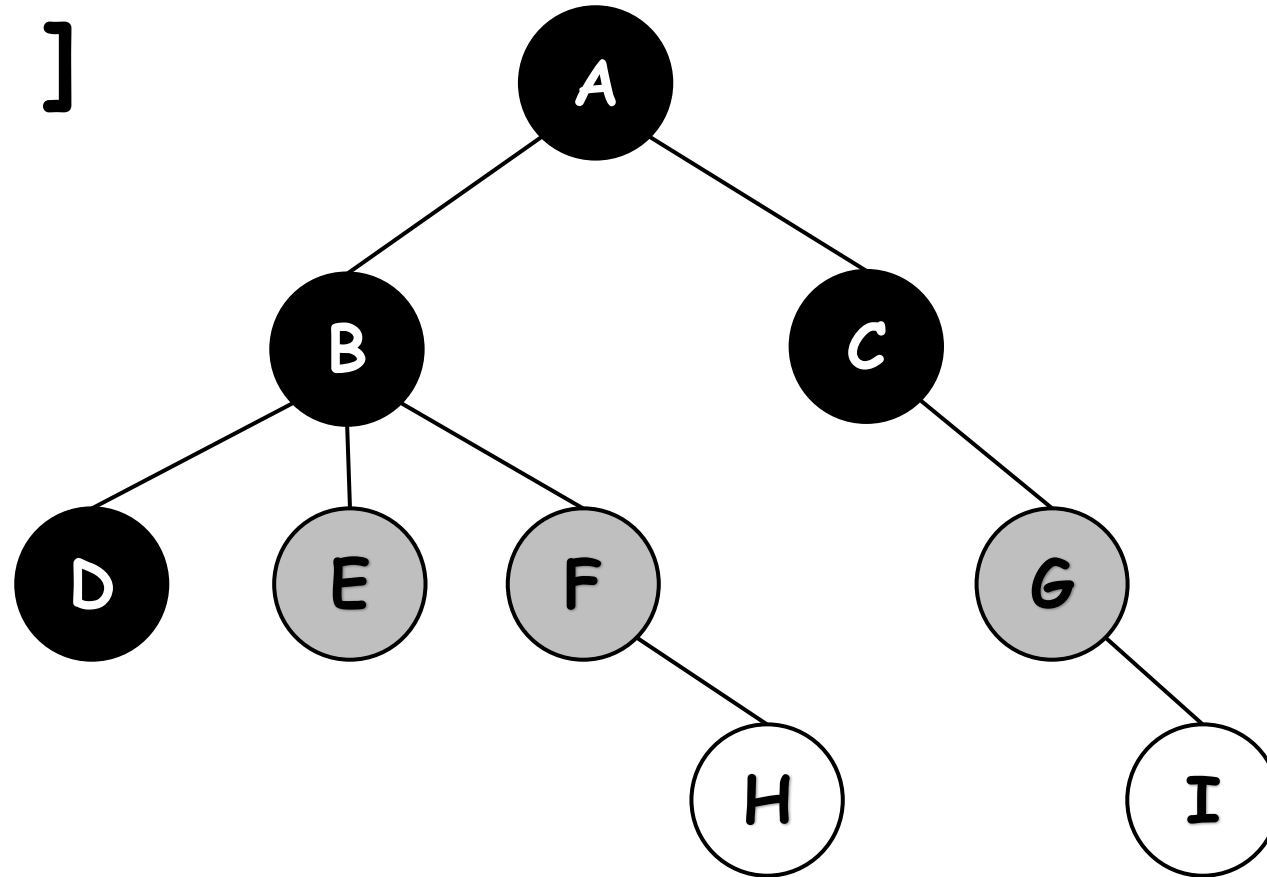
↑

Enqueue



# Ilustrasi Breadth-First Search

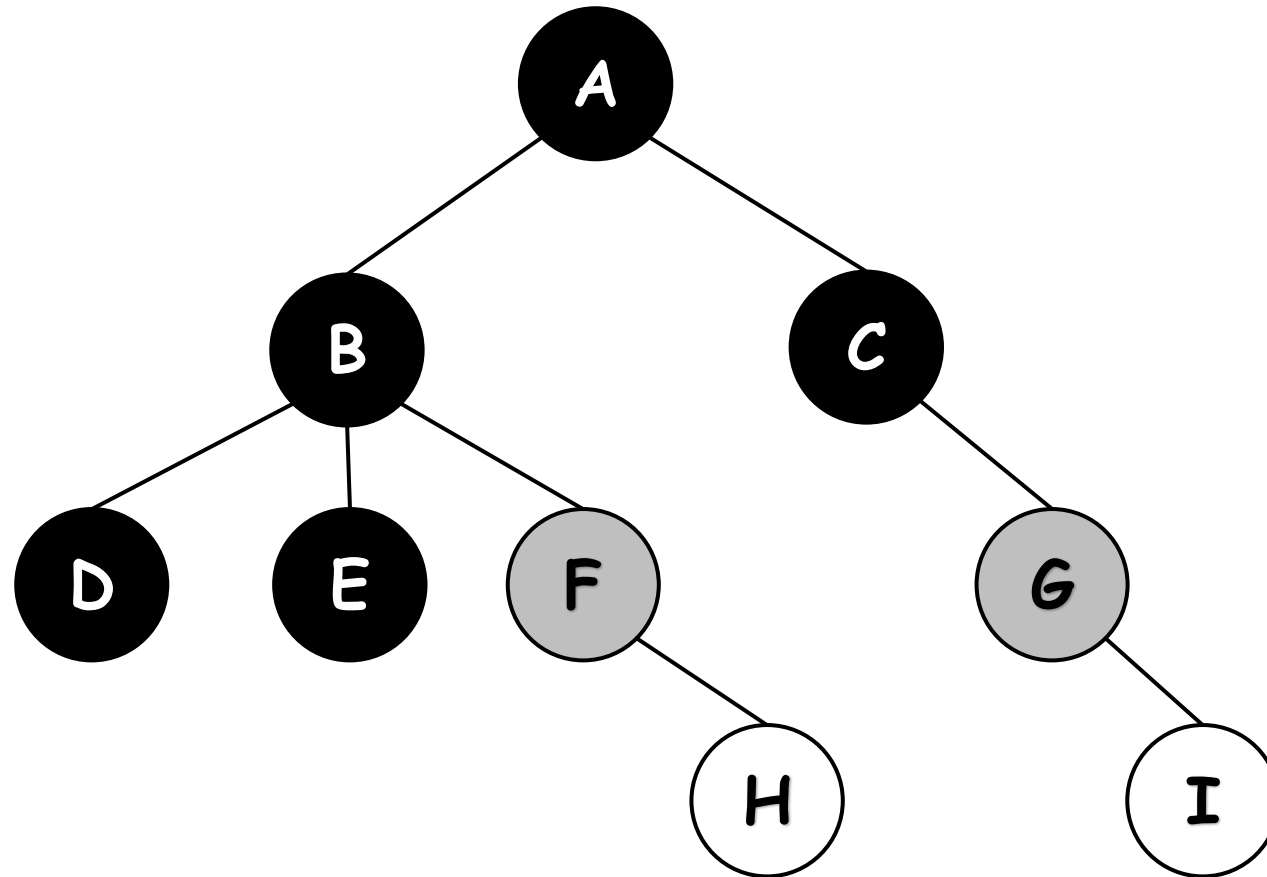
[ E, F, G ]  
↑  
Enqueue





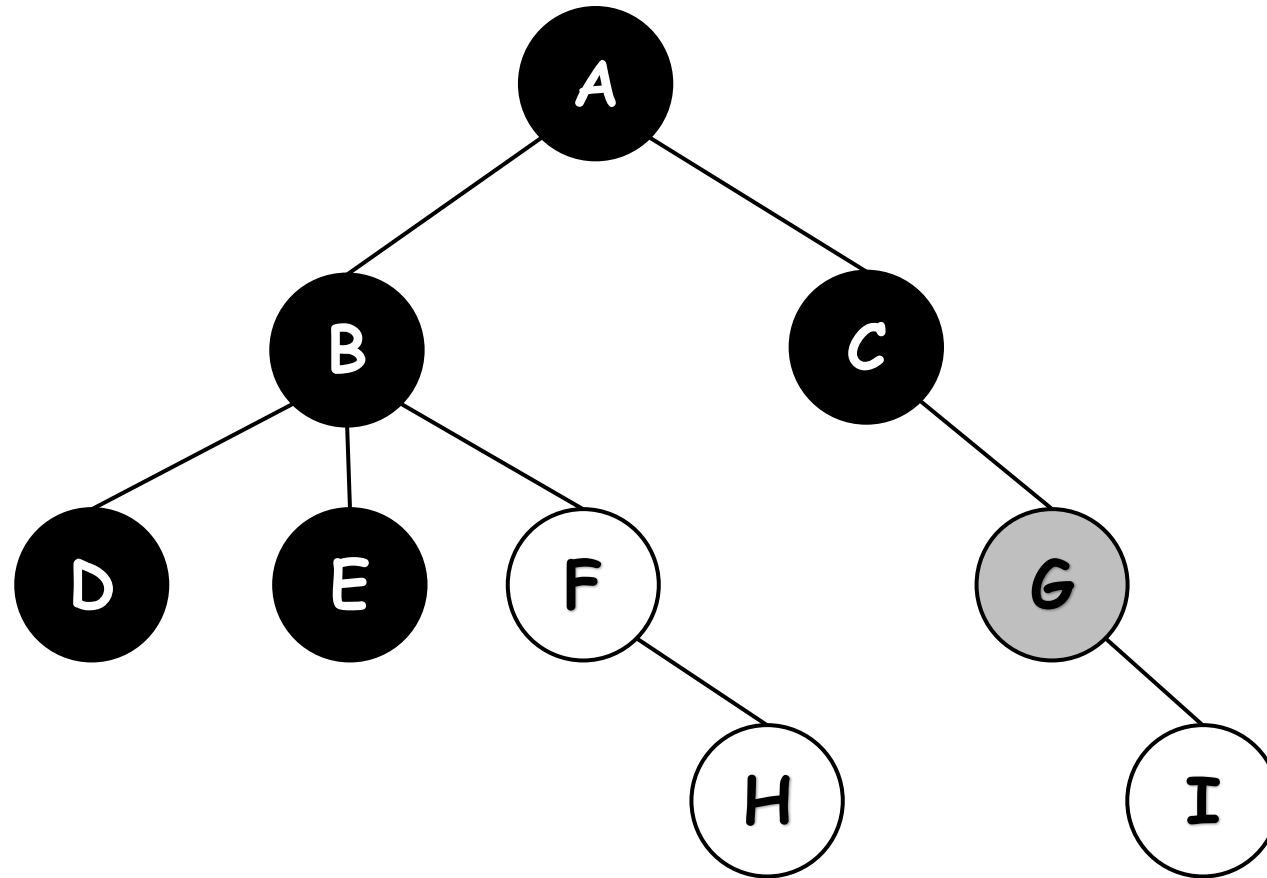
# Ilustrasi Breadth-First Search

[ F, G ]  
↑  
Enqueue



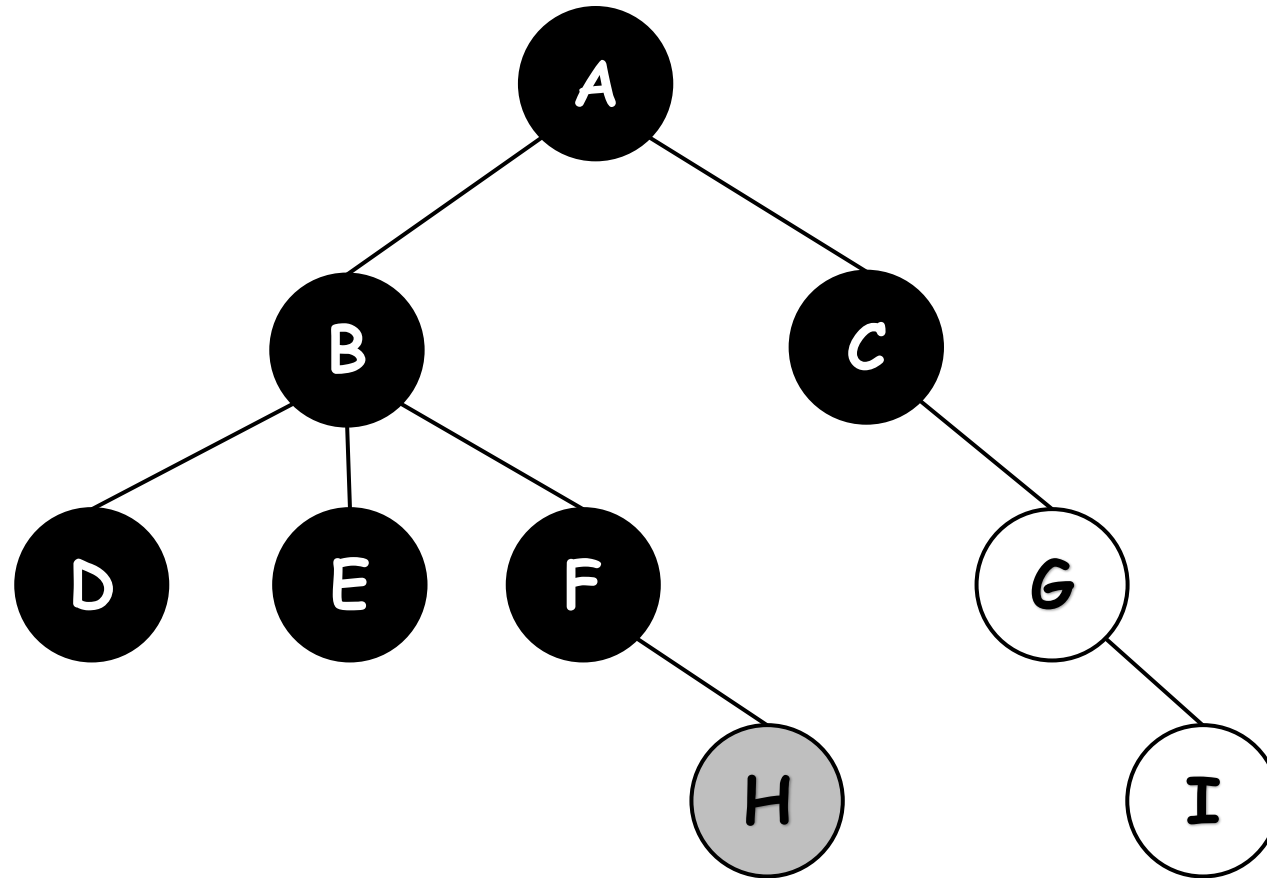
# Ilustrasi Breadth-First Search

[ G, H ]  
↑  
Enqueue



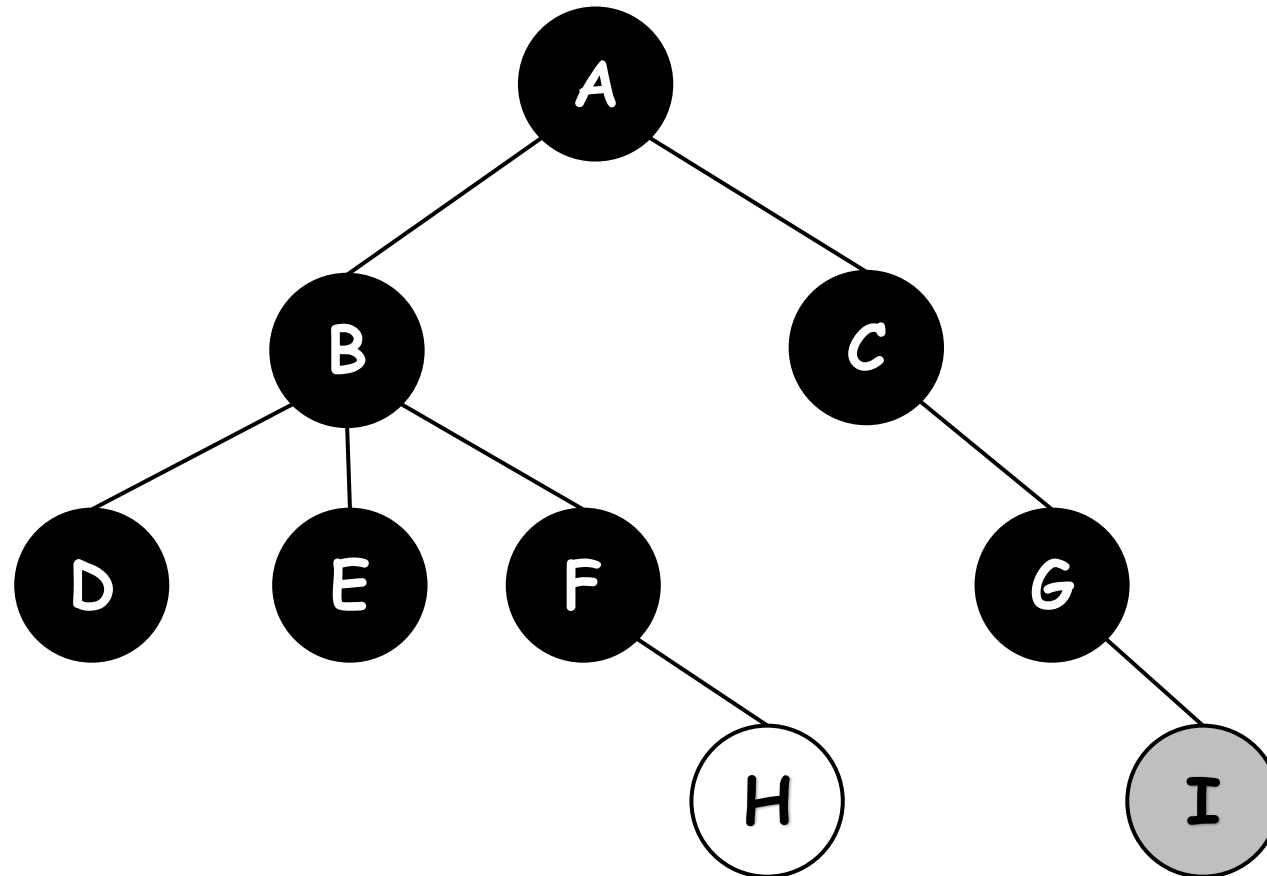
# Ilustrasi Breadth-First Search

[ H, I ]  
↑  
Enqueue



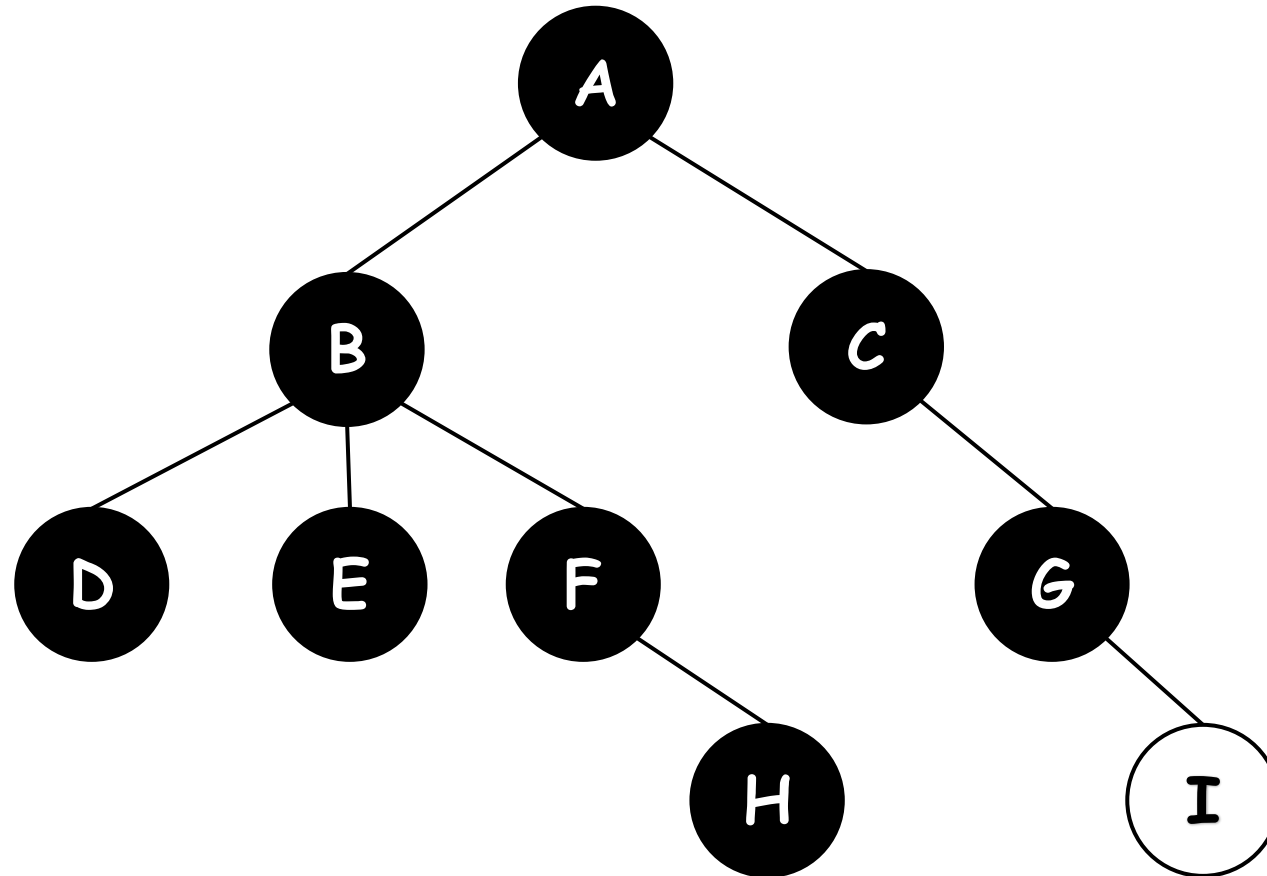
# Ilustrasi Breadth-First Search

[ I ]  
↑  
Enqueue



# Ilustrasi Breadth-First Search

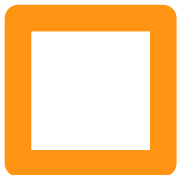
[ ]



# Implementasi Breadth-First Search

```
1 def bfs(graph, node): # Mendefinisikan fungsi bfs yang menerima dua parameter: graph dan node
2     visited = [] # Inisialisasi list kosong untuk menyimpan node yang sudah dikunjungi
3     queue = [] # Inisialisasi list kosong untuk menyimpan node yang akan dikunjungi
4
5     visited.append(node) # Menambahkan node awal ke daftar visited
6     queue.append(node) # Menambahkan node awal ke queue
7
8     while queue: # Selama masih ada node dalam queue
9         s = queue.pop(0) # Ambil dan hapus node pertama dari queue
10        print(s, end = " ") # Cetak node yang sedang diproses dengan spasi sebagai pemisah
11
12        for n in graph[s]: # Loop melalui semua node tetangga dari node s
13            if n not in visited: # Jika node tetangga belum dikunjungi
14                visited.append(n) # Tambahkan node tetangga ke daftar visited
15                queue.append(n) # Tambahkan node tetangga ke queue untuk diproses nanti
16
```

```
1 graph = {
2     'A' : ['B', 'C'],
3     'B' : ['D', 'E', 'F'],
4     'C' : ['G'],
5     'D' : [],
6     'E' : [],
7     'F' : ['H'],
8     'G' : ['I'],
9     'H' : [],
10    'I' : [],
11 }
12 # bfs(graph, 'A') --> output: A B C D E F G H I
```








# DEPTH-FIRST SEARCH



# DEPTH-FIRST SEARCH

**Depth-first search (DFS)** adalah algoritma yang dimulai dari simpul akar (memilih beberapa simpul sembarang sebagai simpul akar dalam kasus grafik) dan mengeksplorasi sejauh mungkin sepanjang setiap cabang sebelum melakukan backtracking. Algoritma ini menggunakan struktur data stack atau rekursi untuk menyimpan simpul yang akan dikunjungi selanjutnya.



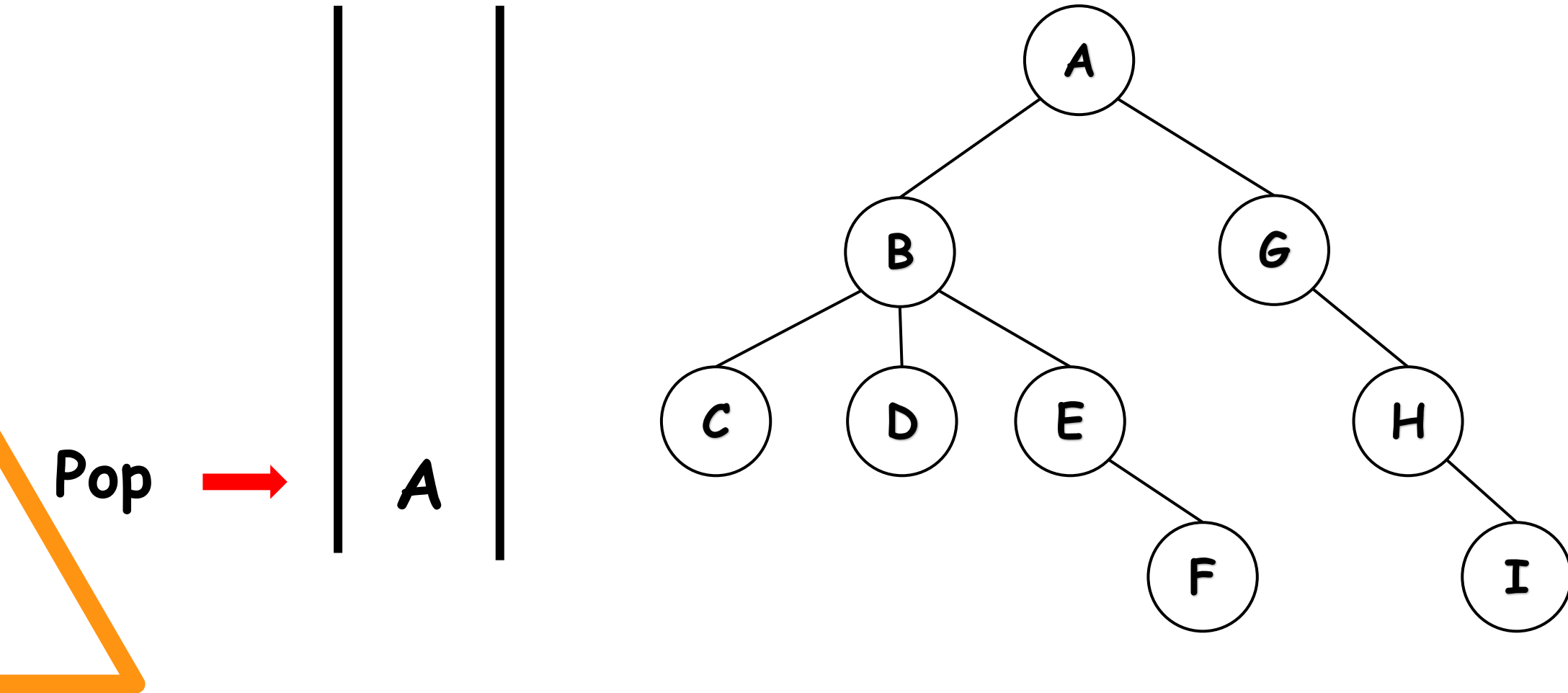


# DEPTH-FIRST SEARCH

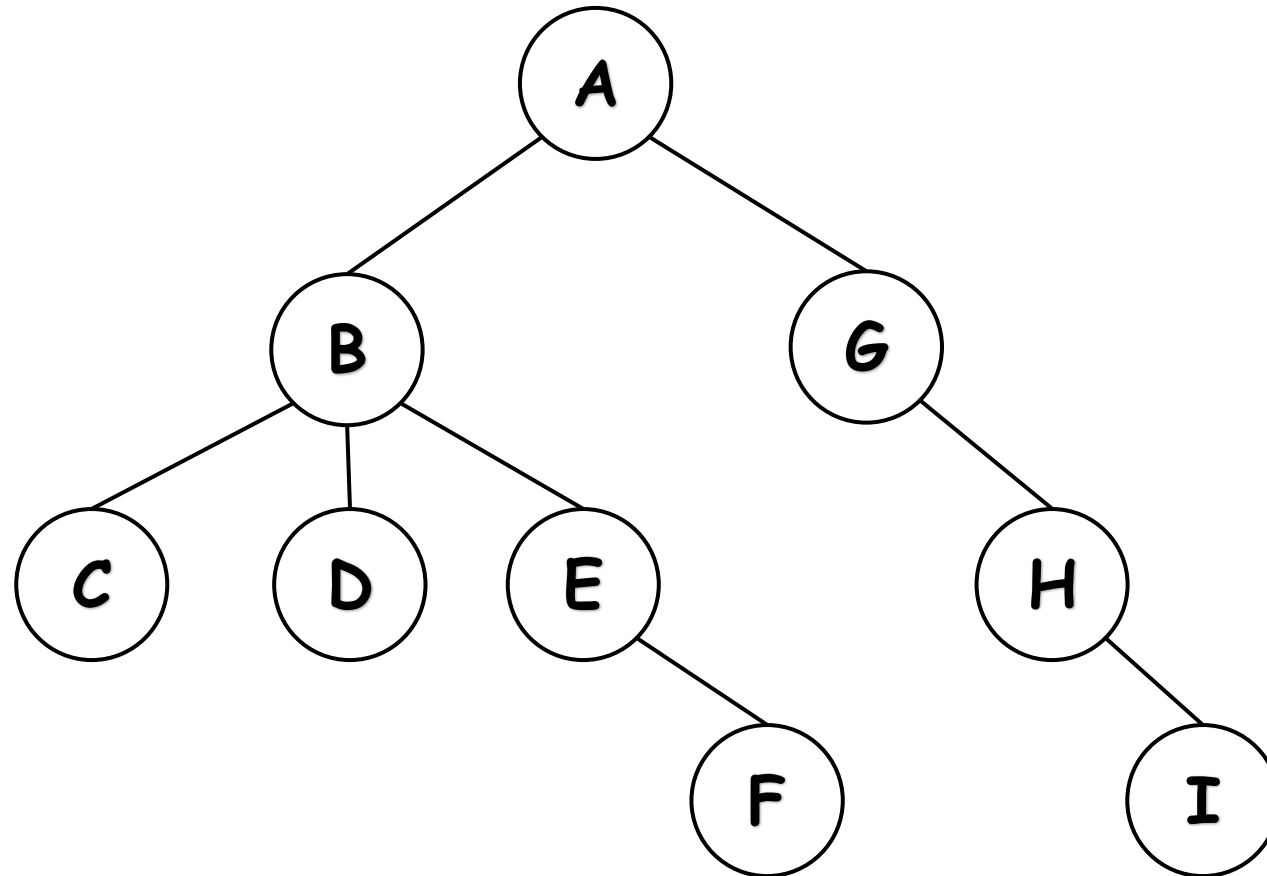
## Beberapa sifat DFS:

- DFS mengeksplorasi graf secara mendalam sebelum meluas;
- Menggunakan stack untuk melacak vertex-vertex yang akan dikunjungi berikutnya;
- Kompleksitas waktunya adalah  $O(V+E)$ , di mana  $V$  adalah jumlah simpul (vertices) dan  $E$  adalah jumlah sisi (edges) dalam graf.

# Ilustrasi Depth-First Search

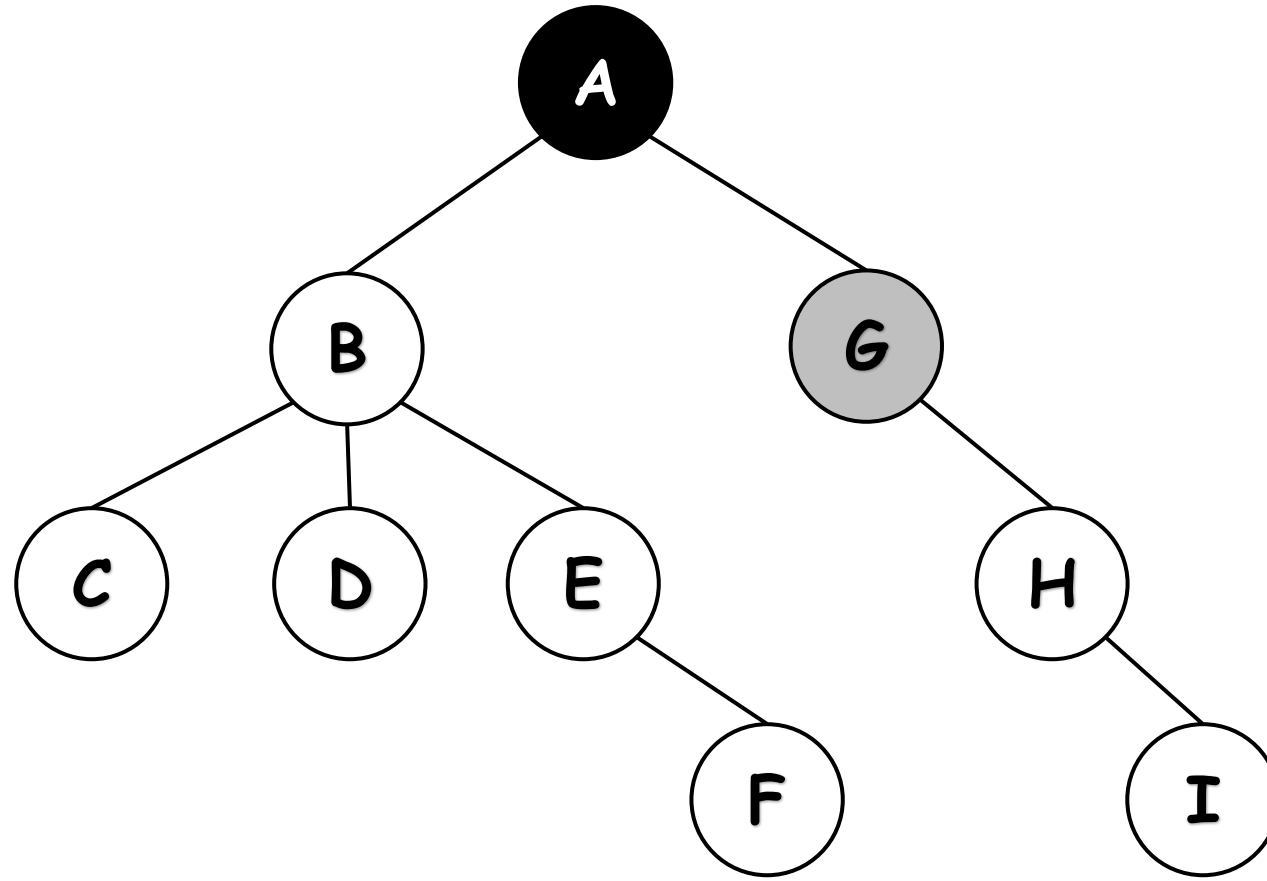


# Ilustrasi Depth-First Search



# Ilustrasi Depth-First Search

***G***

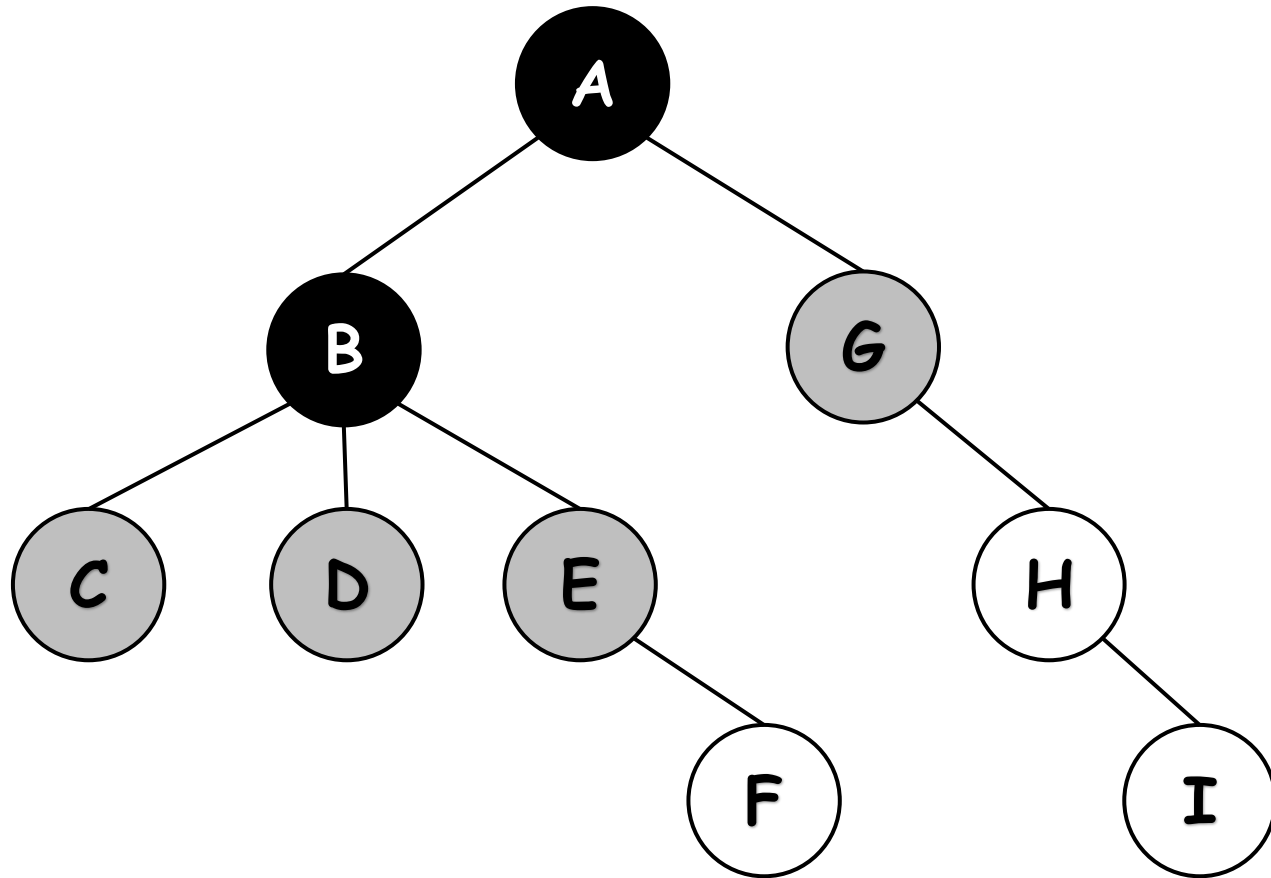


# Ilustrasi Depth-First Search

Pop



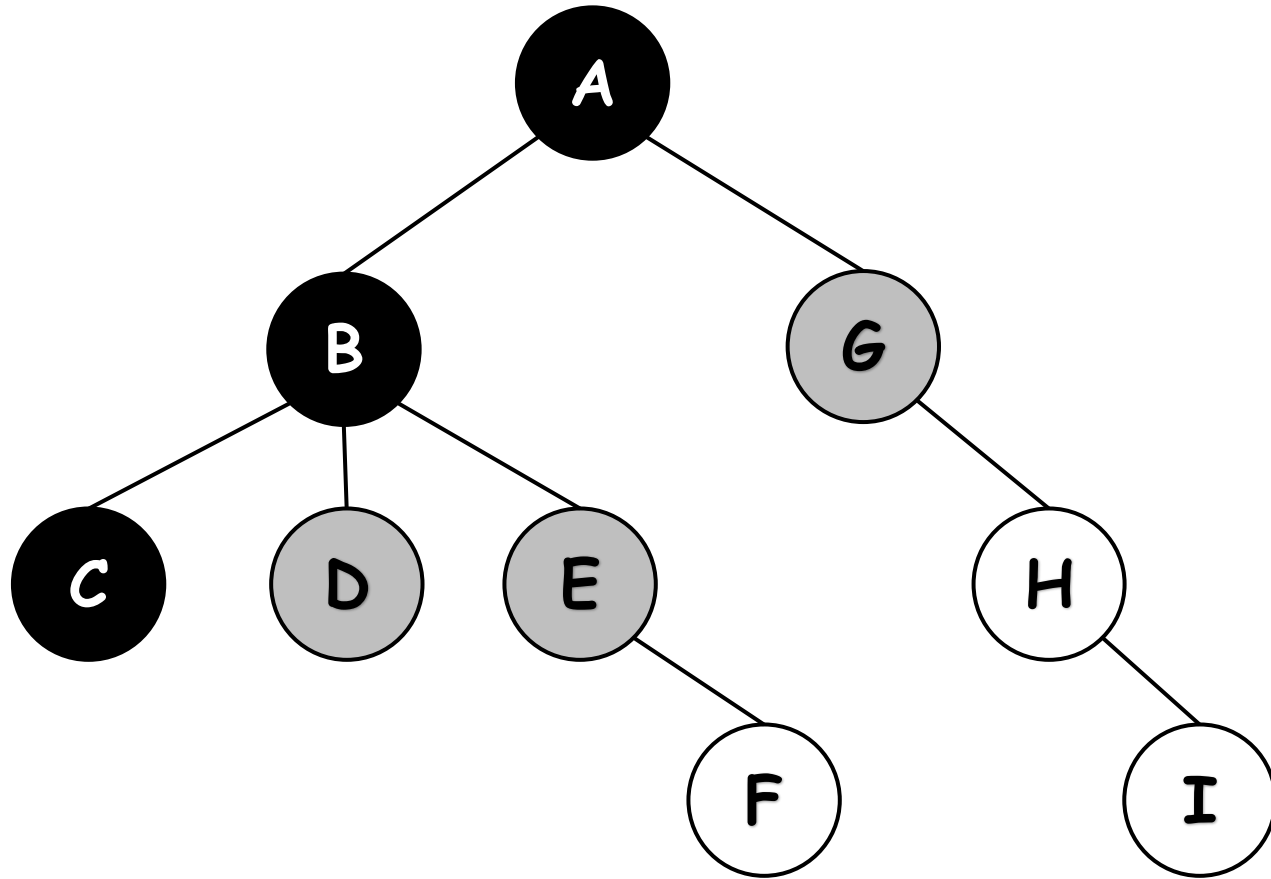
C  
D  
E  
G



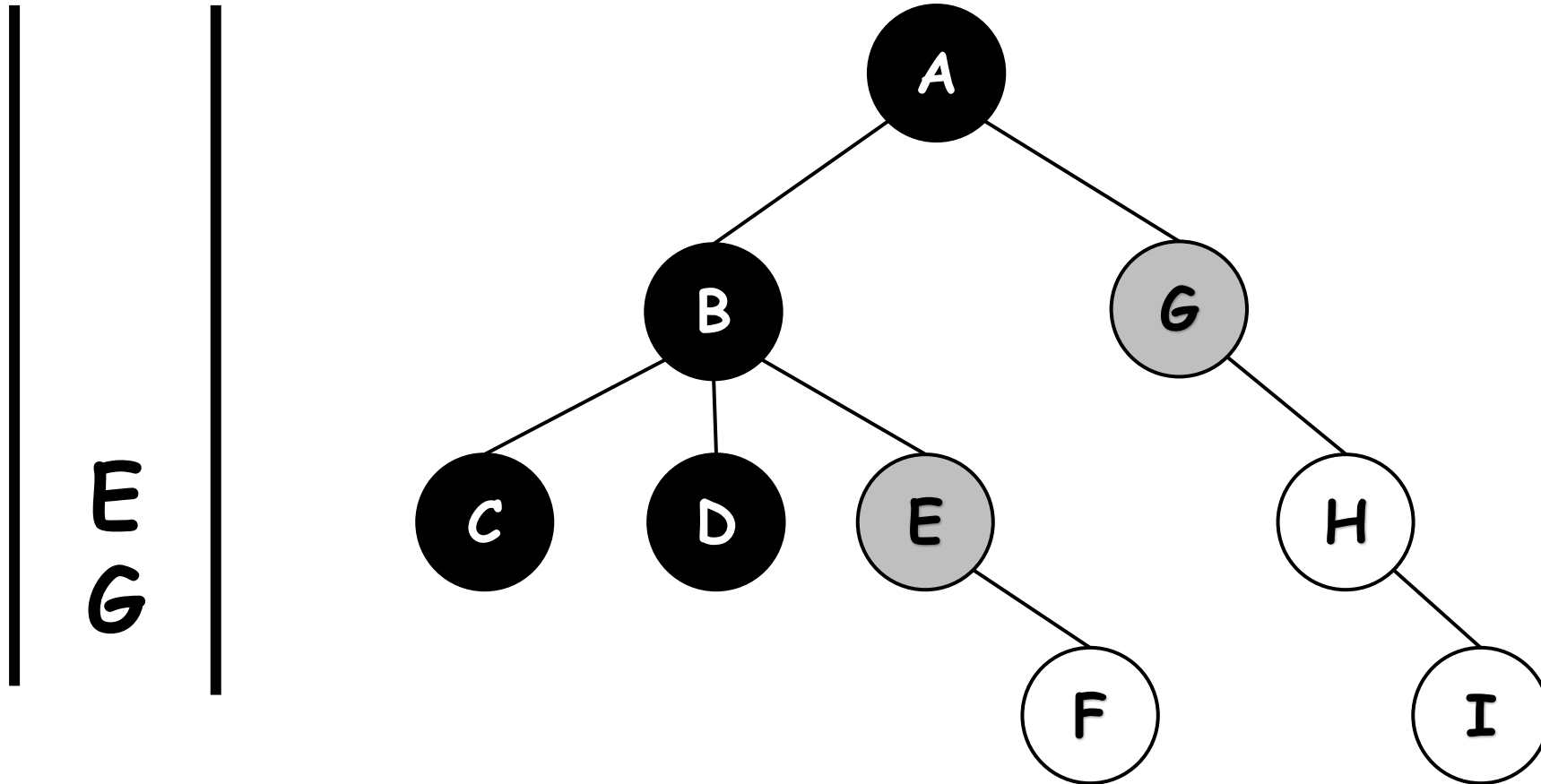
# Ilustrasi Depth-First Search

Pop →

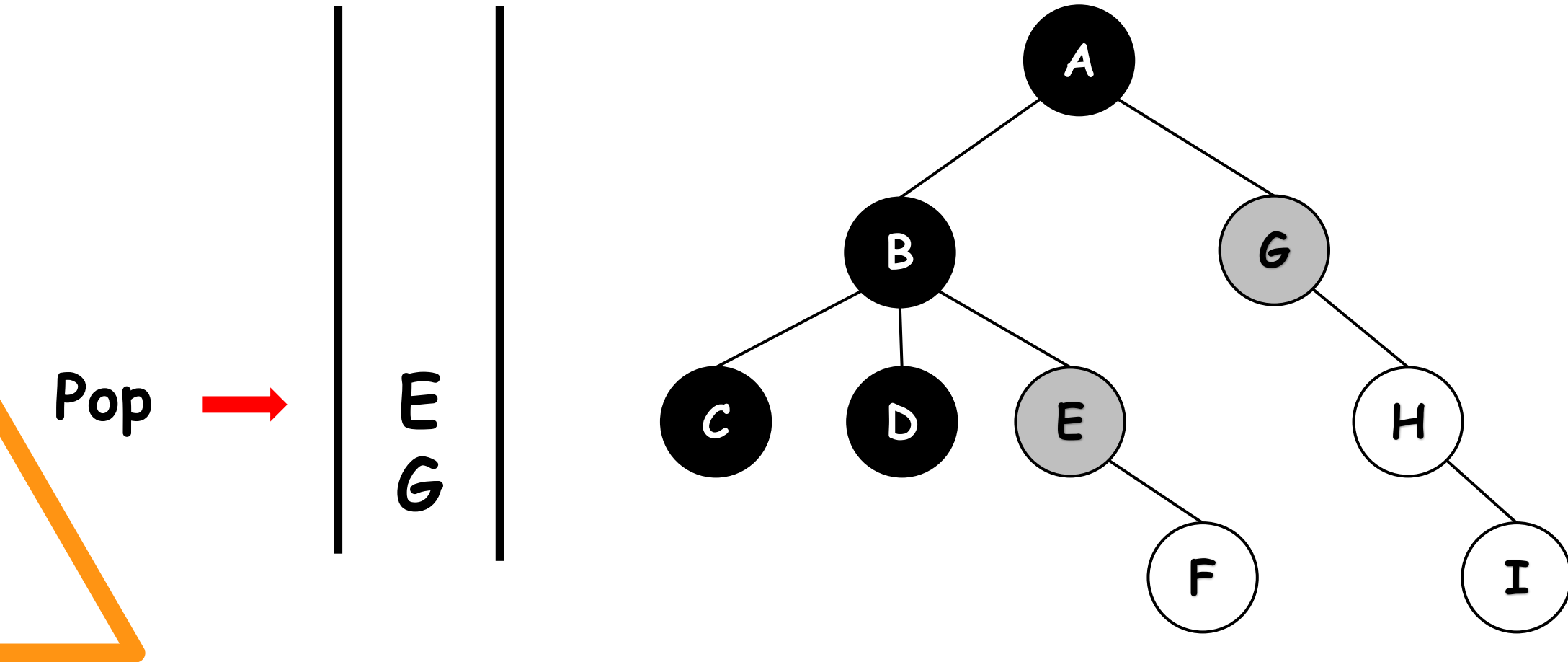
D  
E  
G



# Ilustrasi Depth-First Search

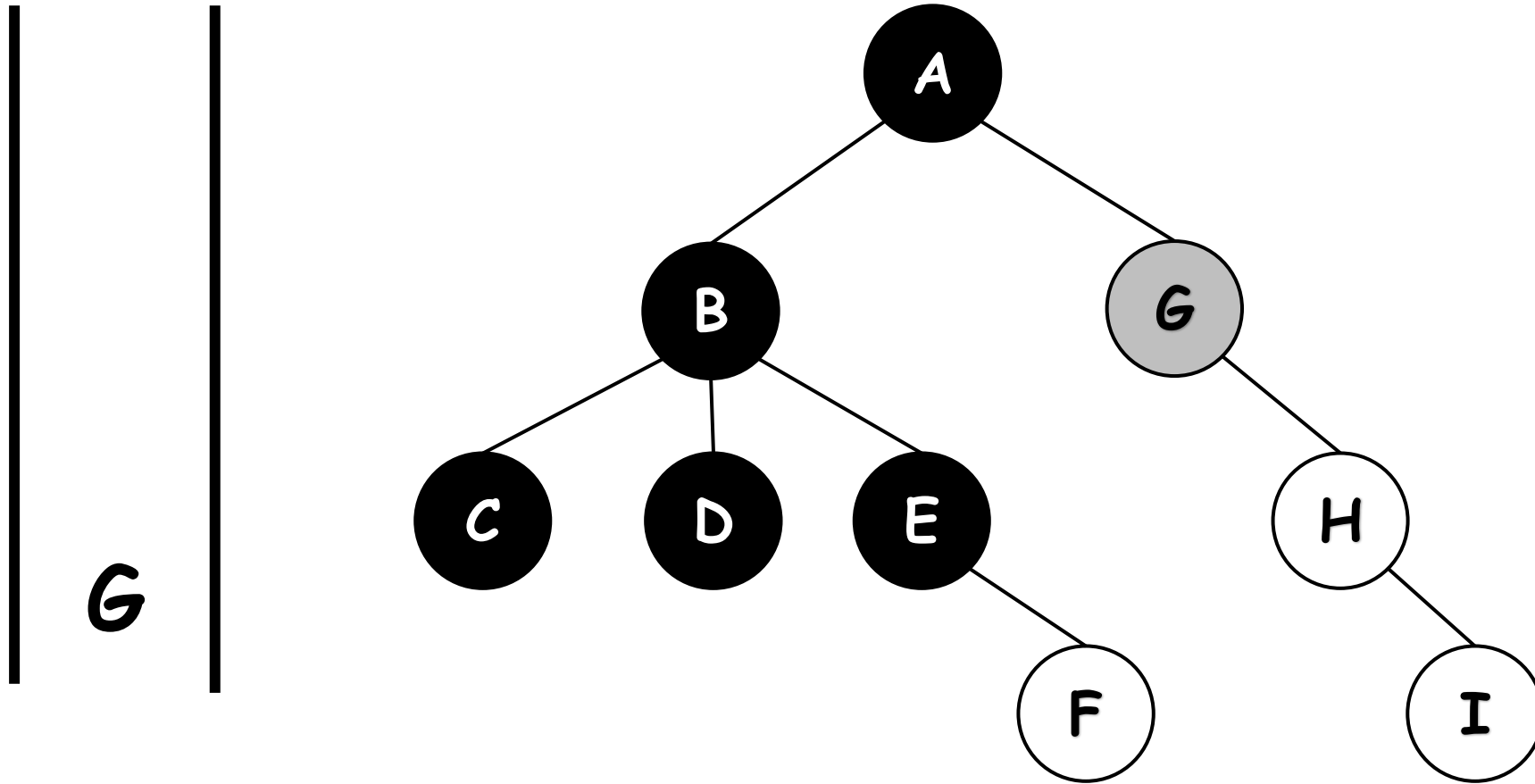


# Ilustrasi Depth-First Search

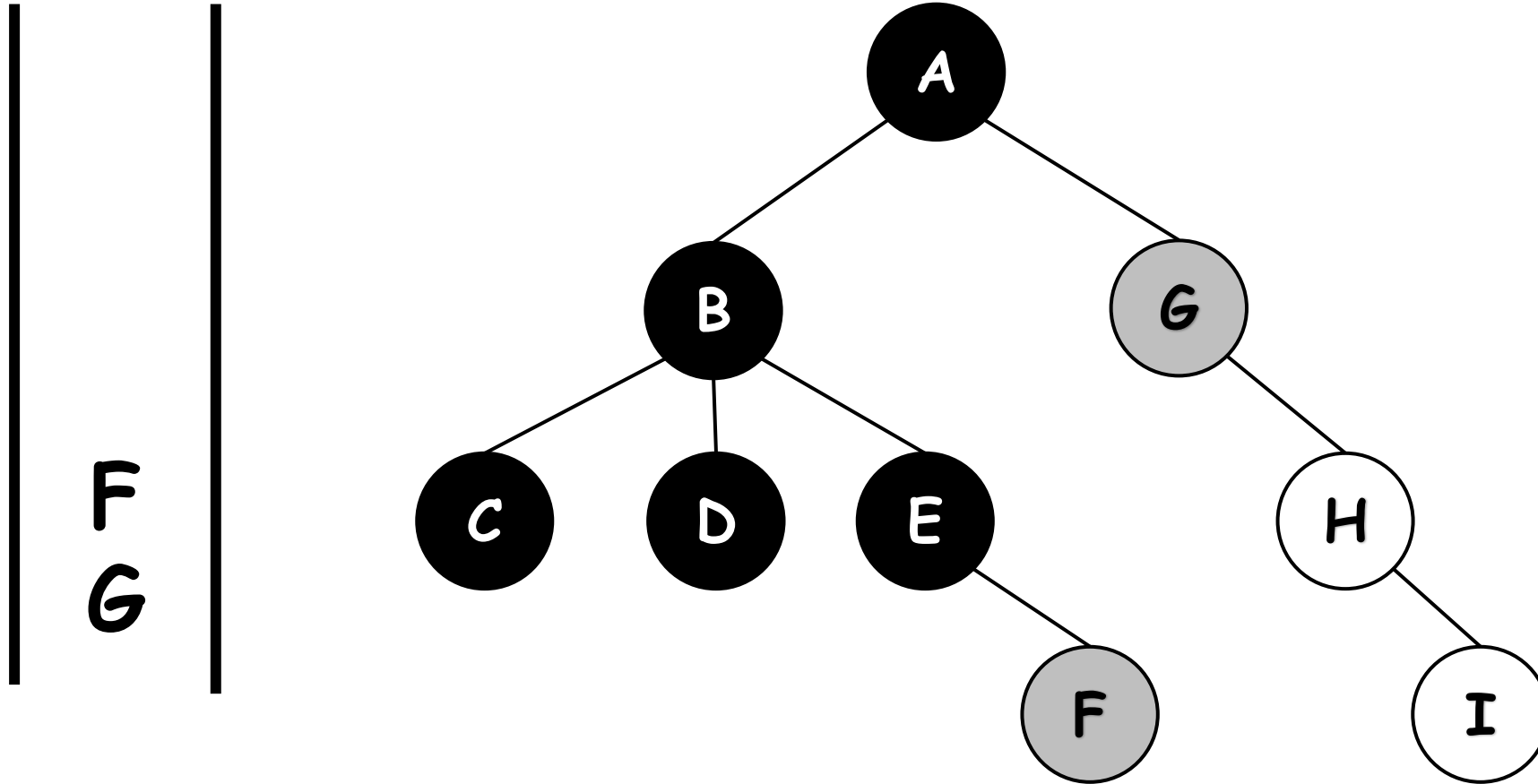




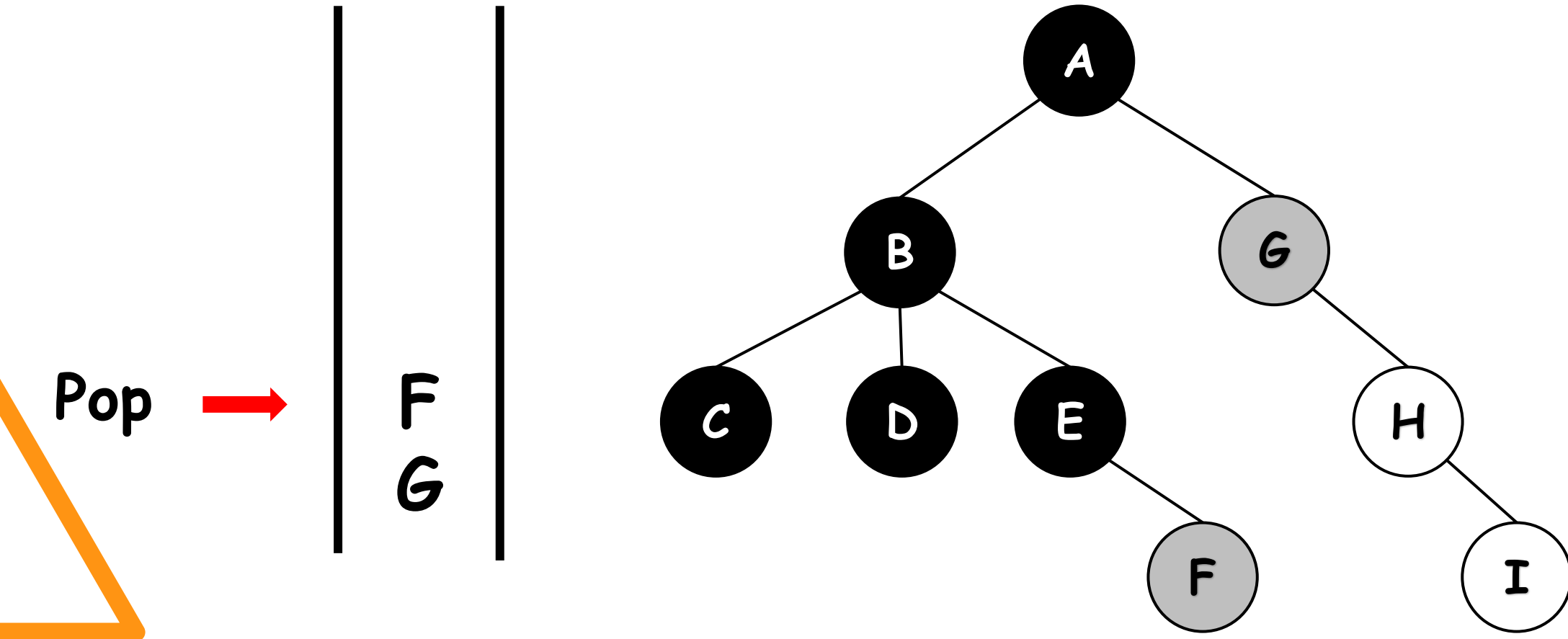
# Ilustrasi Depth-First Search



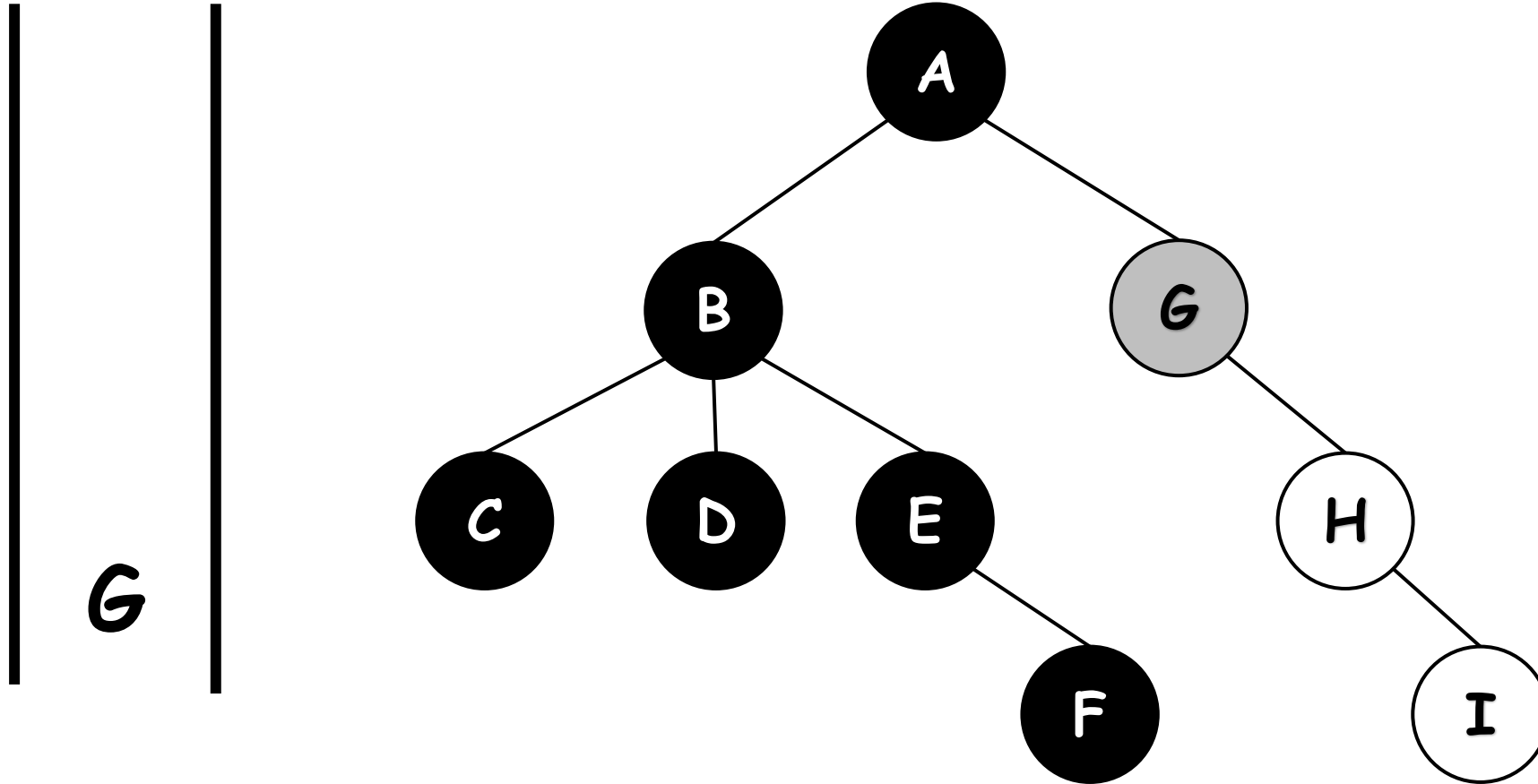
# Ilustrasi Depth-First Search



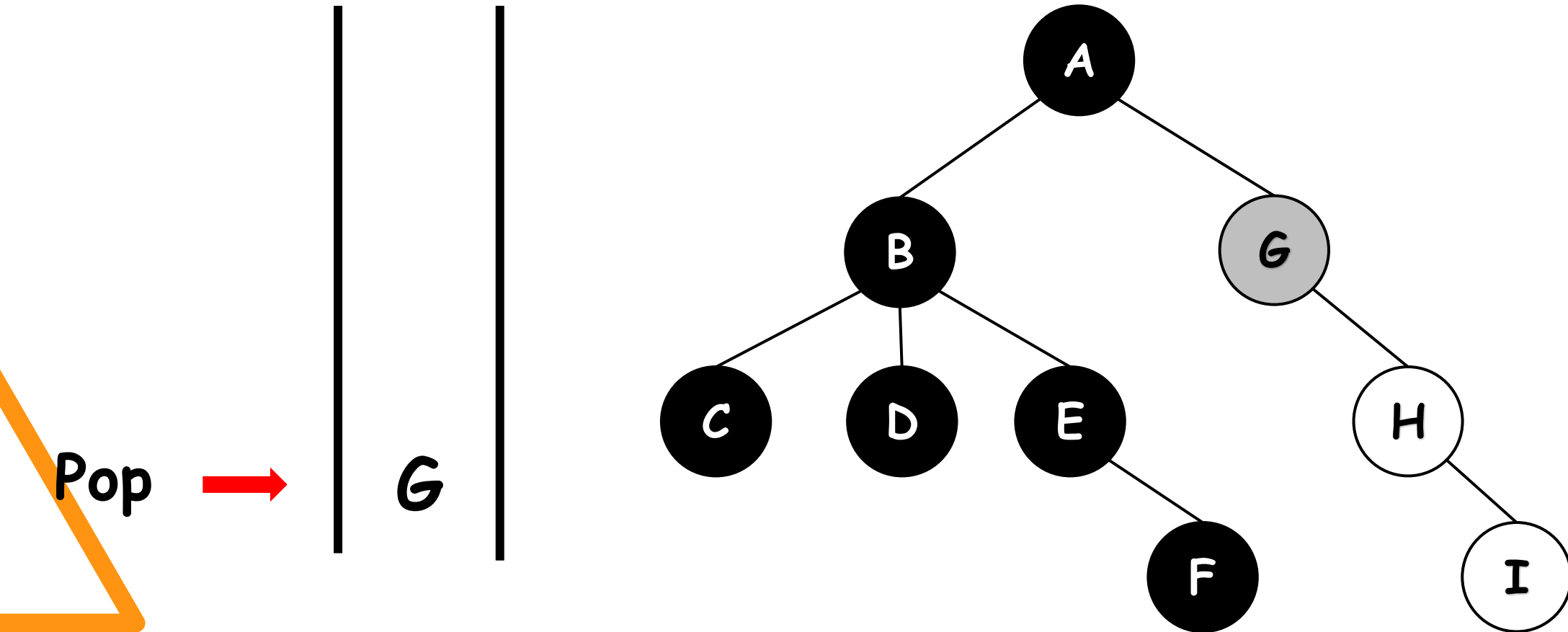
# Ilustrasi Depth-First Search



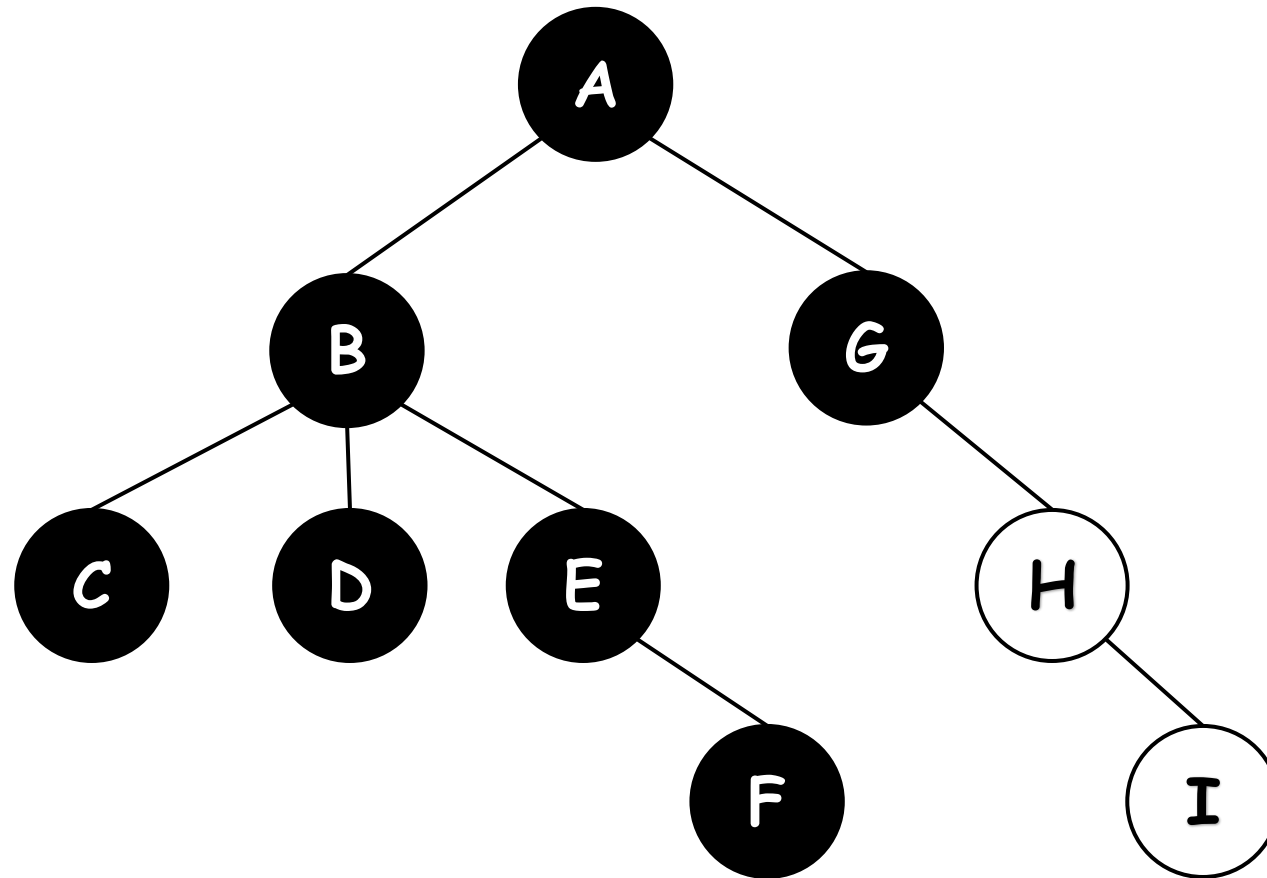
# Ilustrasi Depth-First Search



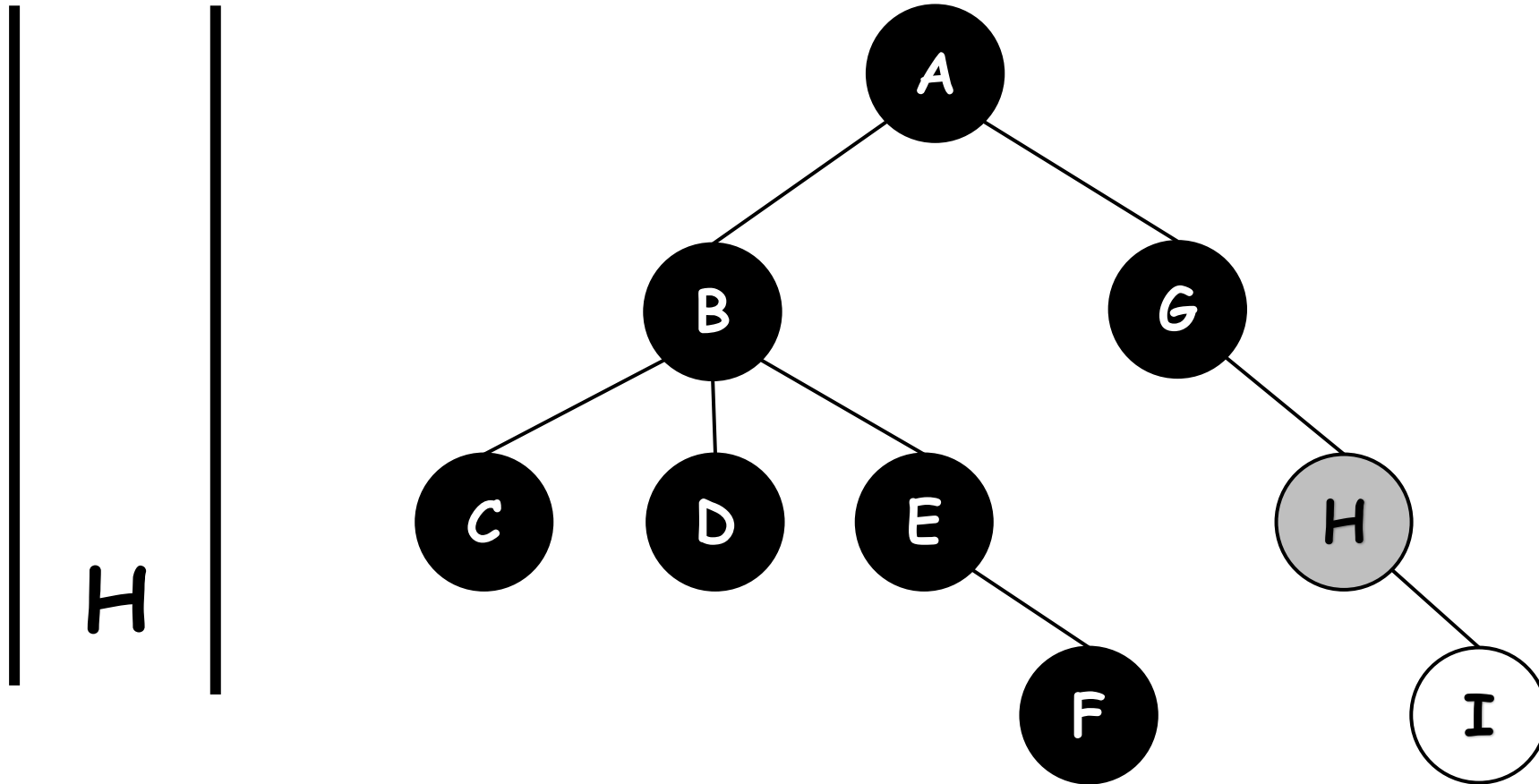
# Ilustrasi Depth-First Search



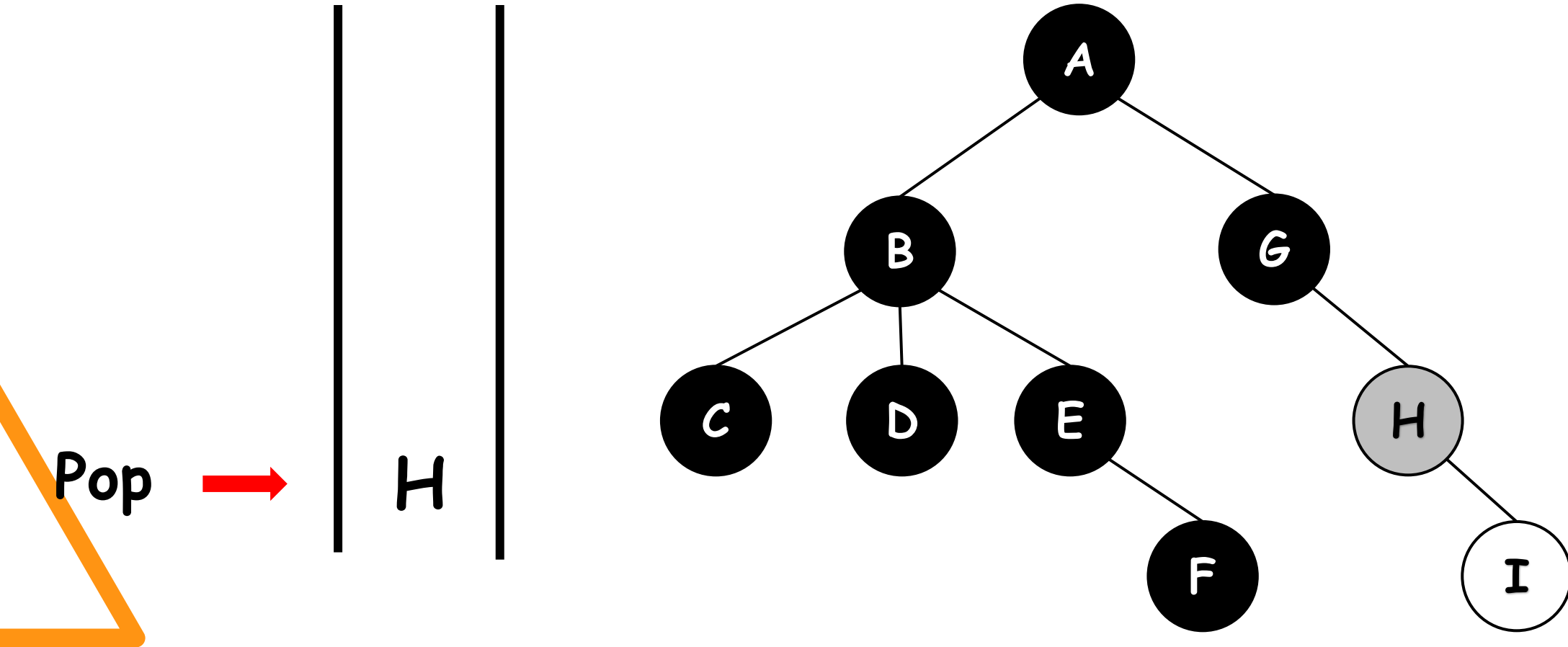
# Ilustrasi Depth-First Search



# Ilustrasi Depth-First Search

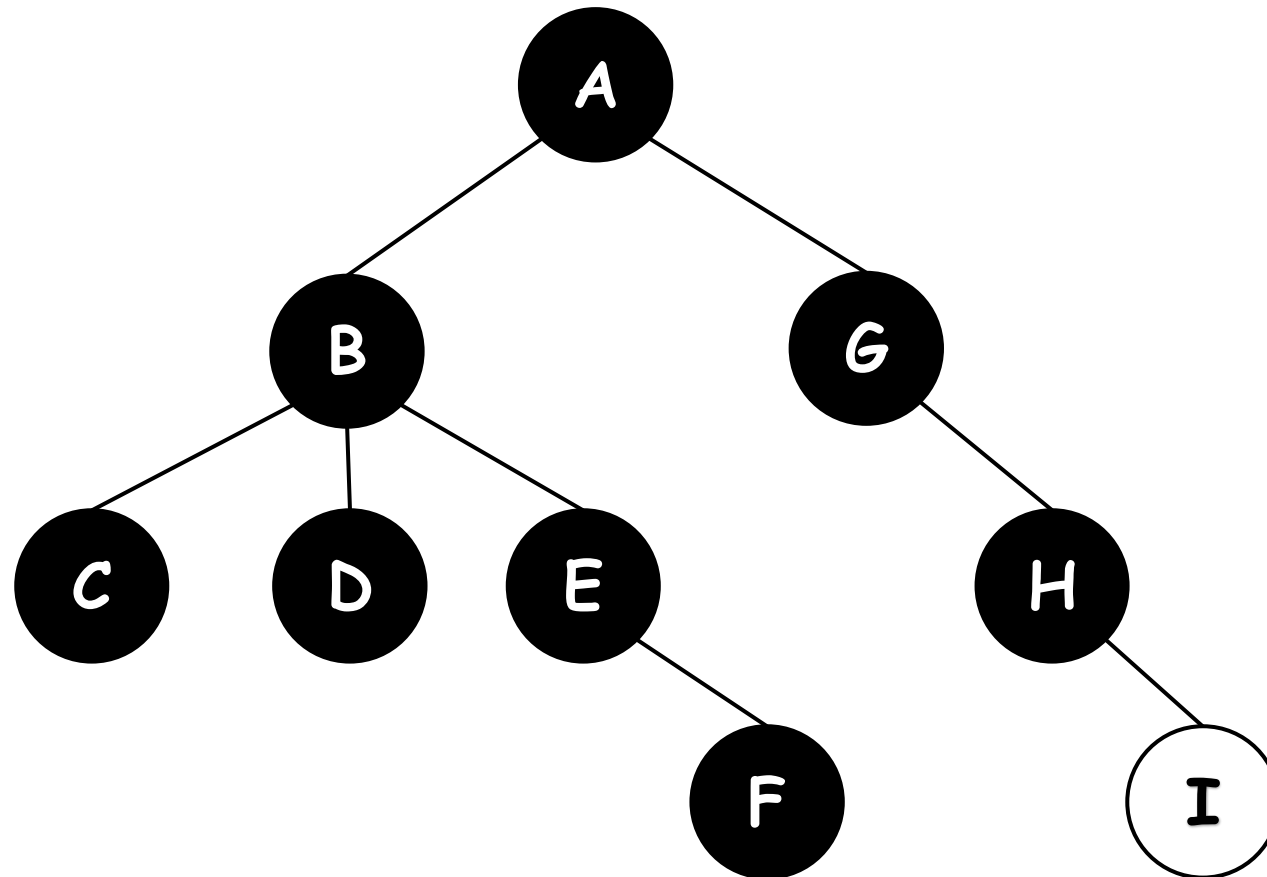


# Ilustrasi Depth-First Search

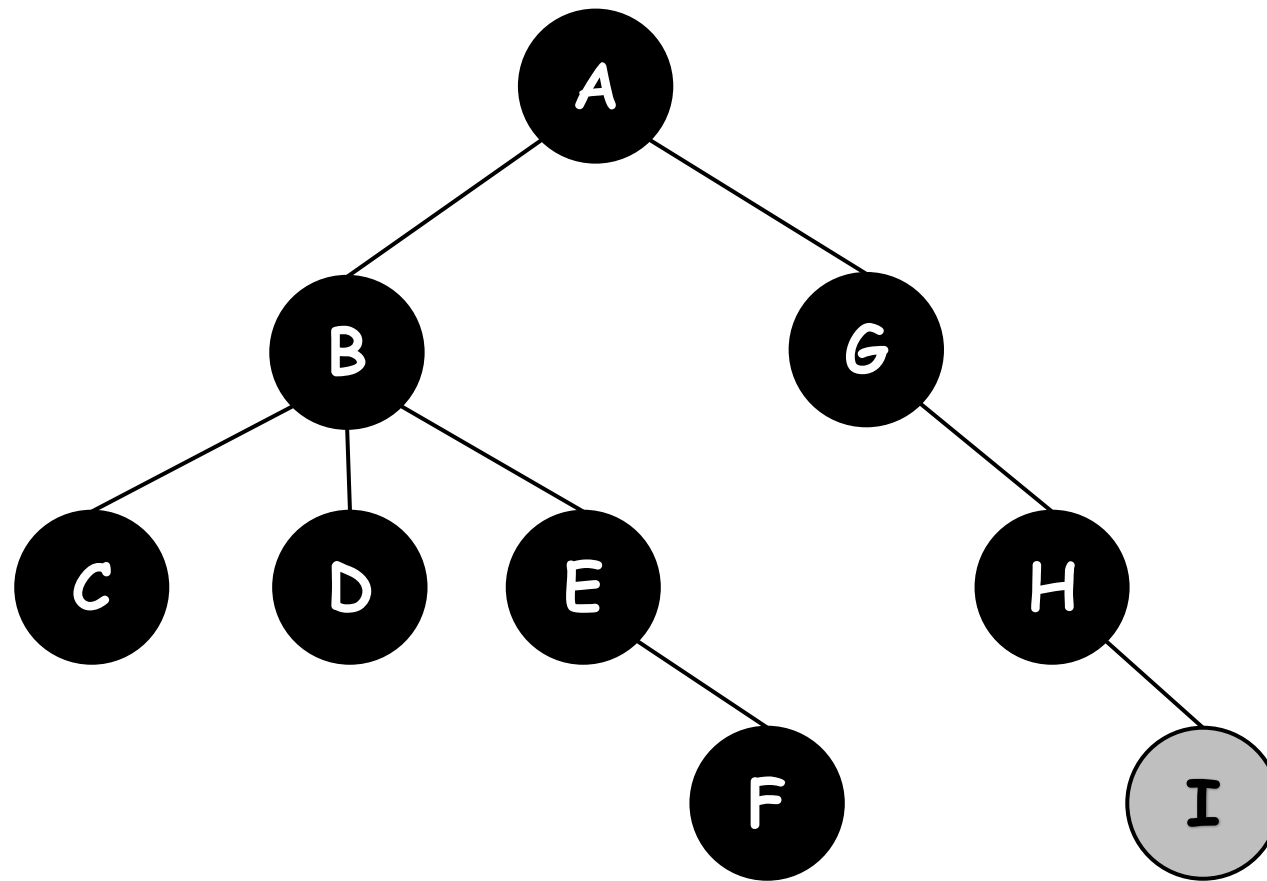




# Ilustrasi Depth-First Search

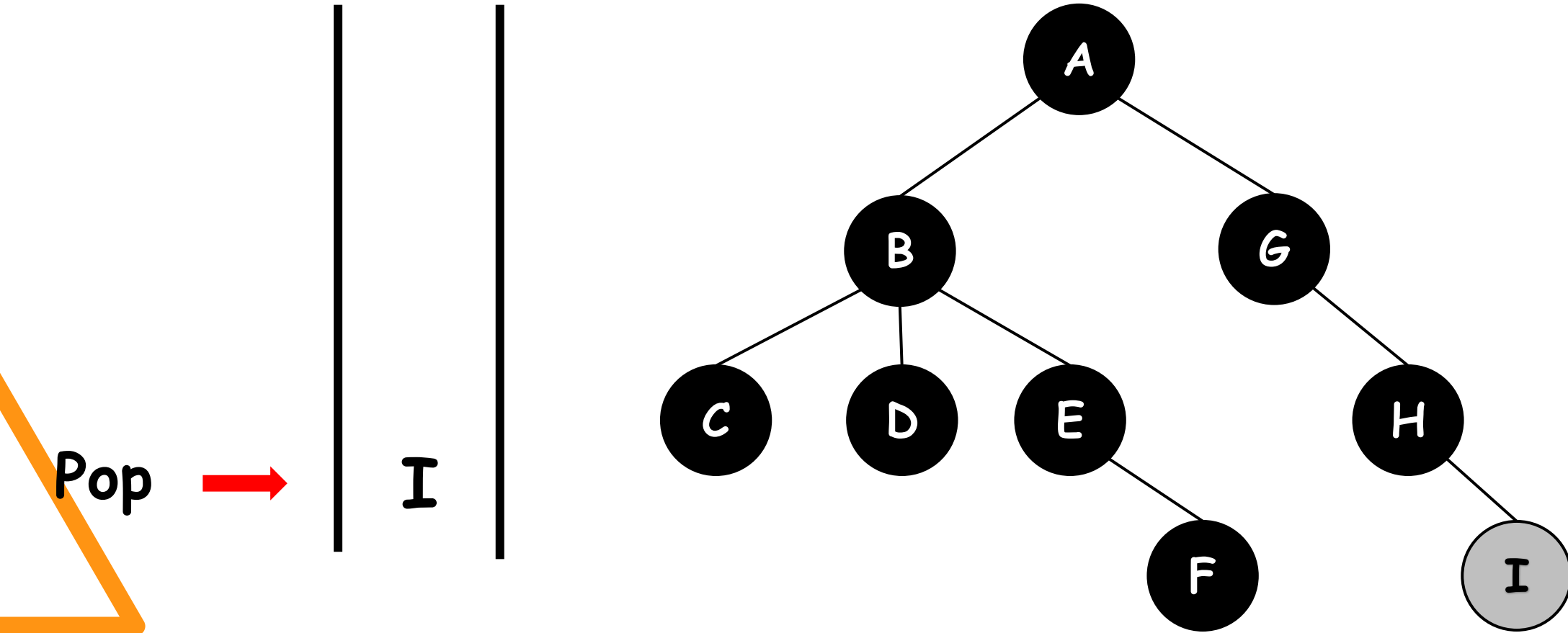


# Ilustrasi Depth-First Search

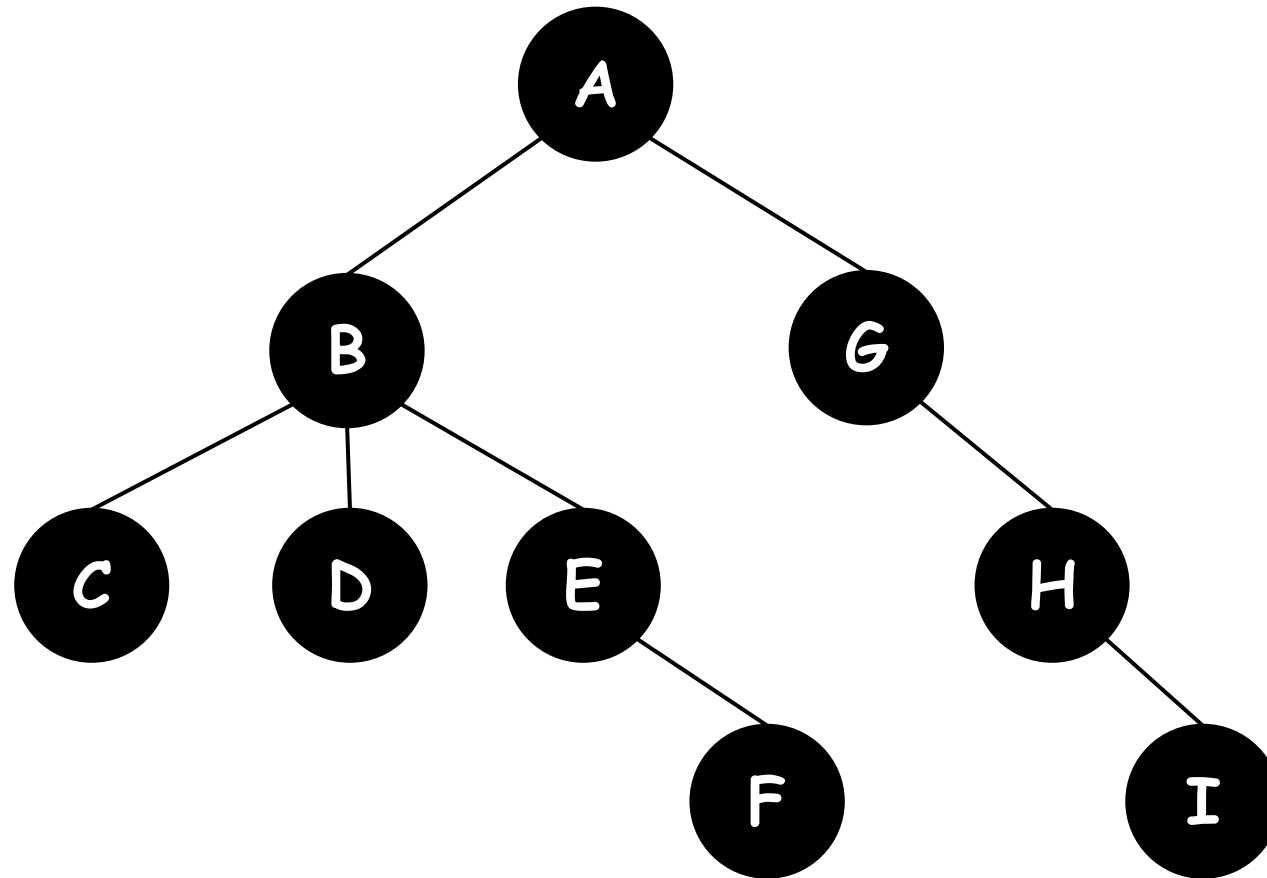


I

# Ilustrasi Depth-First Search



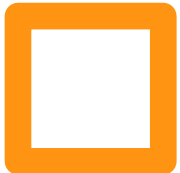
# Ilustrasi Depth-First Search



# Implementasi Depth-First Search

```
1 from collections import deque # Import deque from collections module
2
3 def dfs(graph, node): # Mendefinisikan fungsi dfs dengan parameter graph dan node awal
4     visited = []      # Inisialisasi daftar kosong untuk menyimpan node yang sudah dikunjungi
5     stack = deque()    # Inisialisasi stack dengan deque untuk menyimpan node yang akan dikunjungi
6
7     visited.append(node) # Menandai node awal sebagai dikunjungi dengan menambahkannya ke daftar visited
8     stack.append(node)   # Menambahkan node awal ke stack
9
10    while stack:         # Loop berlanjut selama stack tidak kosong
11        s = stack.pop()  # Mengambil node dari atas stack
12        print(s, end=" ") # Mencetak node yang diambil
13
14        for n in reversed(graph[s]): # Untuk setiap tetangga dari node s (diperoleh dari graf), dalam urutan terbalik
15            if n not in visited:     # Jika tetangga tersebut belum dikunjungi
16                visited.append(n)    # Menandai tetangga tersebut sebagai dikunjungi
17                stack.append(n)      # Menambahkan tetangga tersebut ke stack
```

```
1 graph = {
2     'A' : ['B', 'G'],
3     'B' : ['C', 'D', 'E'],
4     'C' : [],
5     'D' : [],
6     'E' : ['F'],
7     'F' : [],
8     'G' : ['H'],
9     'H' : ['I'],
10    'I' : [],
11 }
12 # dfs(graph, 'A') --> output: A B C D E F G H I
```



The background is white with several decorative elements: a large blue circle on the left containing the text 'Thank you'; a purple circle in the top-left corner; a green circle on the left edge; an orange L-shaped line in the top-right corner; an orange L-shaped line in the bottom-left corner; and a blue semi-circle at the bottom center.

Thank you