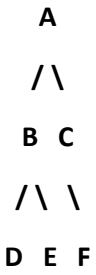
Uninformed Search Algorithms

1. Depth First Search (DFS):



DFS Algorithm

1. Start from the root node.
2. Mark the current node as visited.
3. Explore each adjacent node (neighbor) recursively.
4. Backtrack when there are no unvisited neighbors.

Steps:

1. Start DFS from node **A**.
2. Explore node **A**, then go to **B** (neighbor of A).
3. From **B**, explore its neighbors, first **D**, then **E**.
4. After visiting all neighbors of **B**, backtrack to **A**.
5. From **A**, go to **C**, and then explore **F** (neighbor of C).
6. DFS ends after all nodes are visited.

```
# Depth First Search (DFS) Algorithm Implementation

# Define the graph
graph = {
    'A': ['B', 'C'],
    'B': ['A', 'D', 'E'],
    'C': ['A', 'F'],
    'D': ['B'],
    'E': ['B'],
    'F': ['C']
}
```

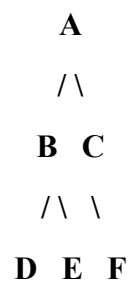
```
# DFS function
def dfs(graph, node, visited=None):
    if visited is None:
        visited = set()        # Set to track visited nodes

    visited.add(node)          # Mark the current node as visited
    print(node)                # Print the current node

    # Recur for all the adjacent nodes (neighbors)
    for neighbor in graph[node]:
        if neighbor not in visited:
            dfs(graph, neighbor, visited)  # Visit the unvisited
neighbor

# Start DFS from node 'A'
print ("Depth-First Search (DFS) traversal starting from node A:")
dfs(graph, 'A')
```

2. Breadth First Search (BFS):



BFS Algorithm:

1. Start at the root
2. Mark the node as visited and add it to a queue.
3. Explore all neighbors of the current node, mark them as visited, and enqueue them.
4. Dequeue the next node from the queue and repeat the process until the queue is empty.

Steps in BFS:

- Start from **A**.
- Visit **A** and enqueue its neighbors: **B** and **C**.
- Dequeue **B**, visit **B**, and enqueue **B's** neighbors: **D** and **E**.
- Dequeue **C**, visit **C**, and enqueue **C's** neighbor: **F**.
- Dequeue **D**, visit **D** (no new neighbors to enqueue).
- Dequeue **E**, visit **E** (no new neighbors to enqueue).
- Dequeue **F**, visit **F** (no new neighbors to enqueue).
- BFS ends when the queue is empty.

```
from collections import deque

# Graph representation
graph = {
    'A': ['B', 'C'],
    'B': ['A', 'D', 'E'],
    'C': ['A', 'F'],
    'D': ['B'],
    'E': ['B'],
    'F': ['C']
}

# BFS function
def bfs(graph, start):
    visited = set()  # Set to keep track of visited nodes
    queue = deque([start]) # Queue to hold nodes to be explored

    while queue:
        node = queue.popleft()  # Dequeue a node
        if node not in visited:
            visited.add(node) # Mark the node as visited
            print(node)  # Print the current node

            # Enqueue all unvisited neighbors of the current node
            for neighbor in graph[node]:
                if neighbor not in visited:
                    queue.append(neighbor)
```

```
# Start BFS from node 'A'
print ("Breadth-First Search (BFS) traversal starting from node
A:")
bfs(graph, 'A')
```