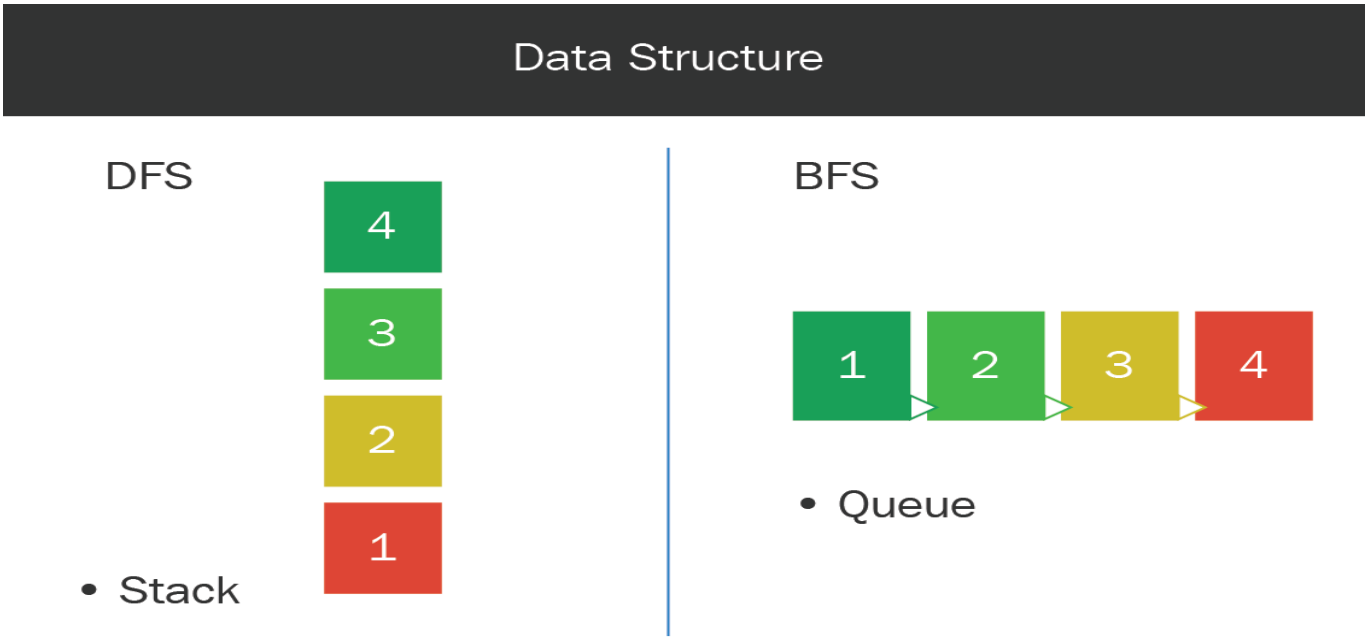
# Task1

## Depth First Search:

- DFS explores as far as possible along one branch of the graph before backtracking. It starts at the root (or an arbitrary node) and explores as deeply as possible along each branch before backtracking.
- It uses a stack (either explicitly or through recursion) to keep track of the nodes to visit. The recursive implementation naturally uses the call stack for this purpose.
- DFS is often used in problems involving finding paths, topological sorting, and connected components in graphs.

## Breadth-First Search:

- BFS explores all the neighbors of a node before moving on to the next level of neighbors. It starts at the root (or an arbitrary node) and explores all the nodes at the current depth before moving on to the nodes at the next depth level.
- It uses a queue data structure to keep track of the nodes to visit. The nodes are processed in the order they are encountered.
- BFS is often used in problems involving finding the shortest path, minimum spanning tree, and level-order traversal of a tree or graph.



# Task 2

## Comparing between Shortest path algorithms speed:

### 1- Dijkstra's Algorithm:

**Time Complexity: O((V + E) * log(V)), where V is the number of vertices and E is the number of edges.**

**Space Complexity: O(V + E) for the data structures used (priority queue or min-heap and adjacency list).**

**Pros:**

1. **Generally, faster on dense graphs with a lot of edges.**
2. **Suitable for graphs with non-negative edge weights.**
3. **Can be more efficient in practice for sparse graphs.**

**Cons:**

1. **Requires non-negative edge weights; doesn't work with negative weights.**
2. **Priority queue operations can be expensive, impacting performance.**
3. **Bellman-Ford Algorithm:**

### 2 - Bellman-Ford Algorithm:

**Time Complexity: O(V * E), where V is the number of vertices and E is the number of edges.**

**Space Complexity: O(V + E) for the data structures used (distance array and adjacency list).**

**Pros:**

1. **Works with graphs containing negative edge weights.**
2. **More versatile in terms of the types of graphs it can handle.**

**Cons:**

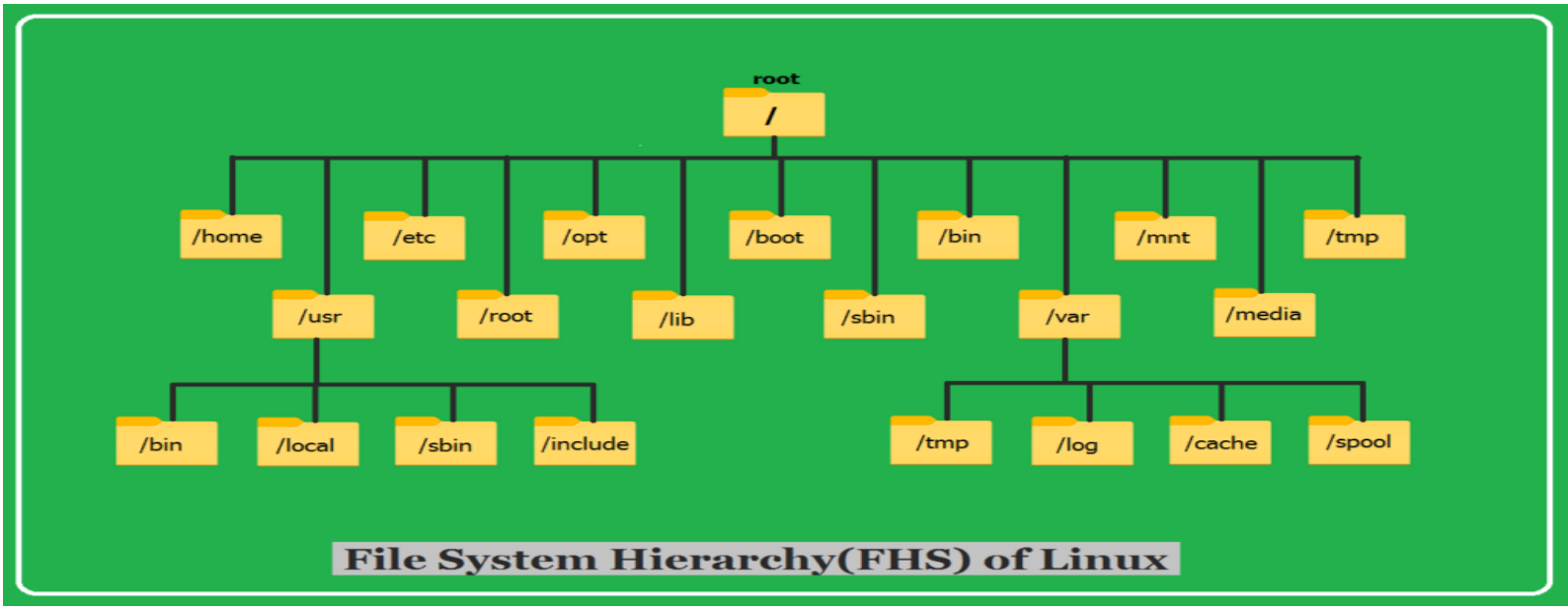
1. **Slower than Dijkstra's algorithm, especially on sparse graphs.**
2. **Inefficient on graphs with many edges.**

## Task3

## Real life example on binary tree:

### 1. File System Hierarchy:

**File systems in computers often use a tree structure to organize files and directories. Each directory is a node, and each node can have subdirectories (children) and files. The root of the tree represents the main directory, and the branches represent the subdirectories. This binary tree structure simplifies file organization and navigation.**




### 2. Game Decision Tree:

**In game theory or decision analysis, decision trees are often used to model different possible outcomes based on player choices. Consider a simple binary decision tree for a player deciding whether to attack or defend in a strategy game. The root represents the initial decision, and each subsequent node represents a player's choice, leading to different outcomes.**