# Assignment-1: Maze Solver using BFS and DFS

Objective: Implement BFS and DFS to solve a maze.

Problem Statement: Given a grid-based maze where 0 represents walls and 1 represents walkable paths, find the shortest path from a start cell to an end cell.

Tasks:

- Use BFS to find the shortest path.
- Use DFS to explore all possible paths and report one valid path (not necessarily the shortest).
- Compare the number of nodes explored by BFS and DFS.

# AI Search

Artificial Intelligence can be referred to as an agent that can act rationally, it can perform some kind of search algorithm to achieve its tasks in solving a search problem such as a Maze problem.

# A search problem in Artificial Intelligence consists of:

- **State Space** refers to all possible states where the agent can be.
- **Start State** refers to the state where the search begins.
- **Goal State** refers to the state where the search ends.
- **Solution** refers to the sequence of actions that shows how an agent searches from the start state to the goal state.
- **Cost** refers to the effort of the agent in doing the searching task.

# There are several algorithms can be used to solve a search problem, such as:

- Breadth First Search (BFS)
- Depth First Search (DFS)
- Greedy Best First Search (GBFS)
- A* Search

# Breadth First Search (BFS)

The Breadth First Search (BFS) algorithm is used to search a graph data structure for a node that meets a set of criteria. It starts at the root of the graph and visits all nodes at the current depth level before moving on to the nodes at the next depth level.

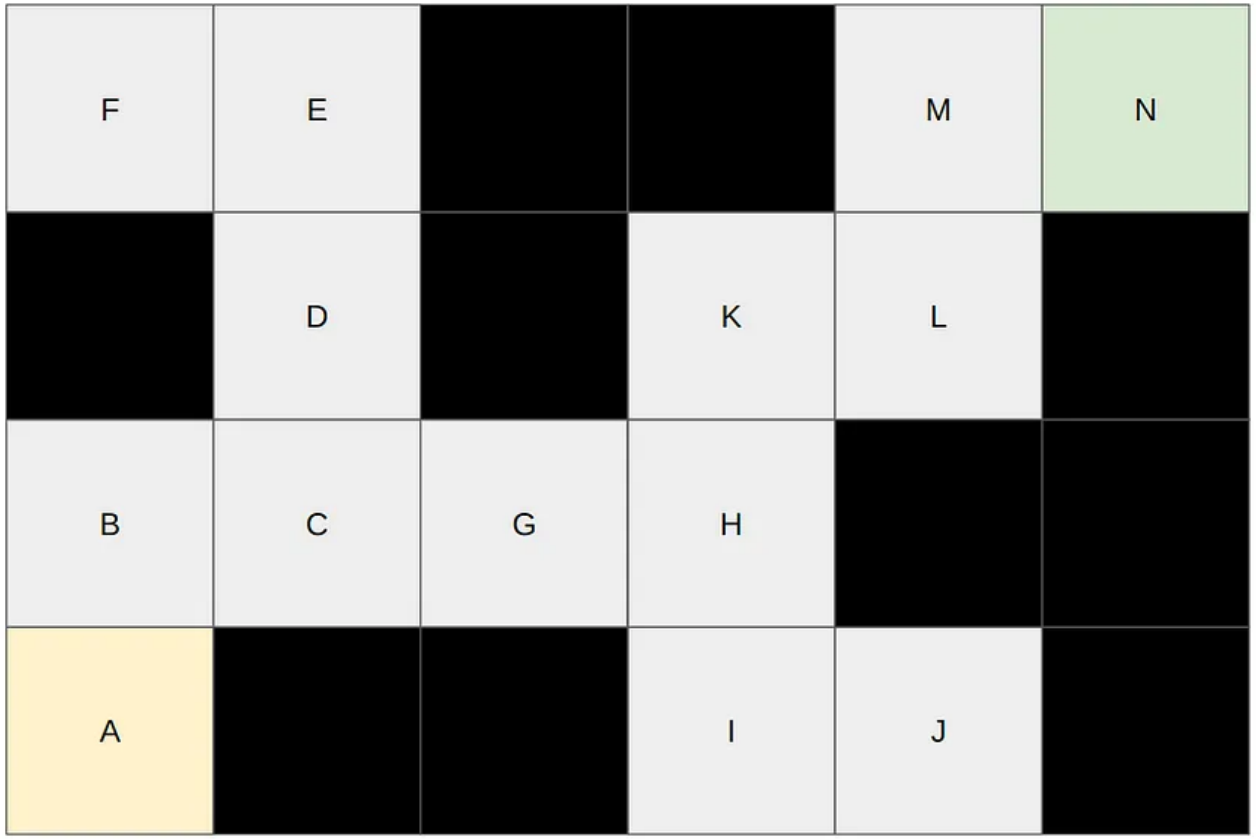
In short, BFS will visit all current depth level nodes before proceeding to the nodes at the next level of depth.

# Pseudocode of the algorithm:

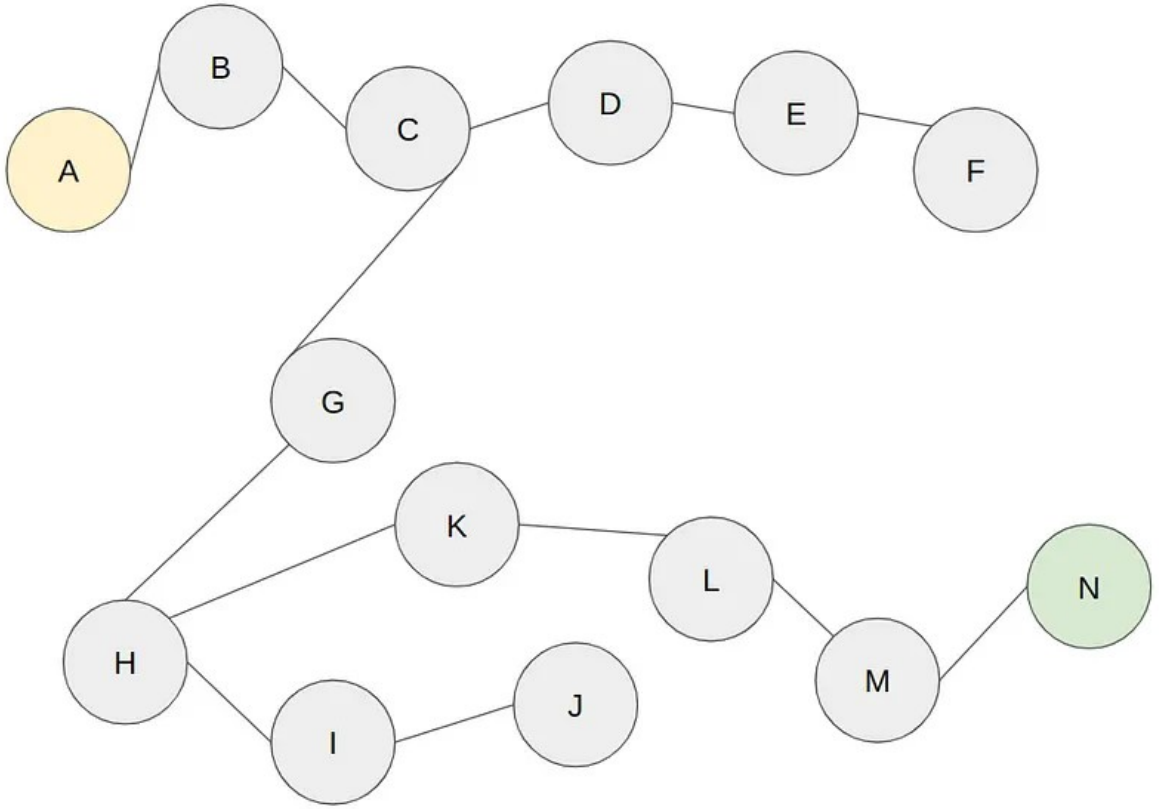
1. Input **graph**, **start node**, and **goal node**
2. Define **solution** as a list
3. Define **frontier** as a queue
4. Define **visited** as a list
5. Append **start node** to **frontier** and **visited**
6. While the **frontier** is not empty:
7. Dequeue the **frontier** and store the value in the **selected node**
8. If the **selected node** is equal to the **goal node**, append the **selected node** to the **solution**, break from the loop, and return the **solution**.
9. If not, just append the **selected node** to the **solution**.
10. Loop through all **neighbors** of the **selected node** from the **graph**, append each **neighbor** to the **frontier** and **visited** if the **neighbor** is not yet in the **visited** list.

# Solve Maze Problem with BFS

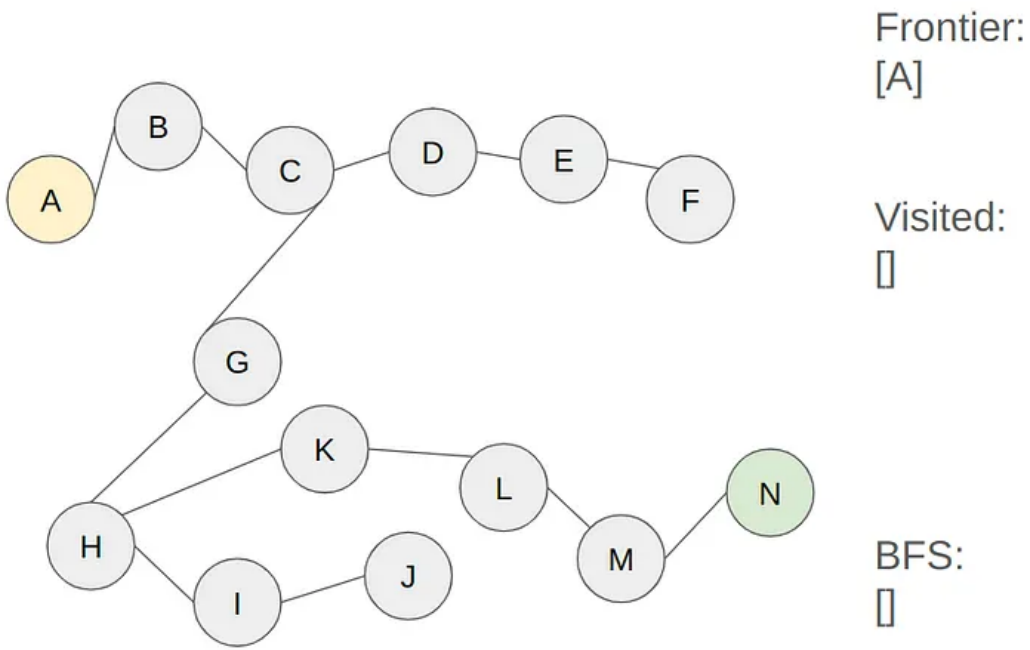
What is the **solution** and **cost** of the above Maze problem if we want to move from **A** to **N**?



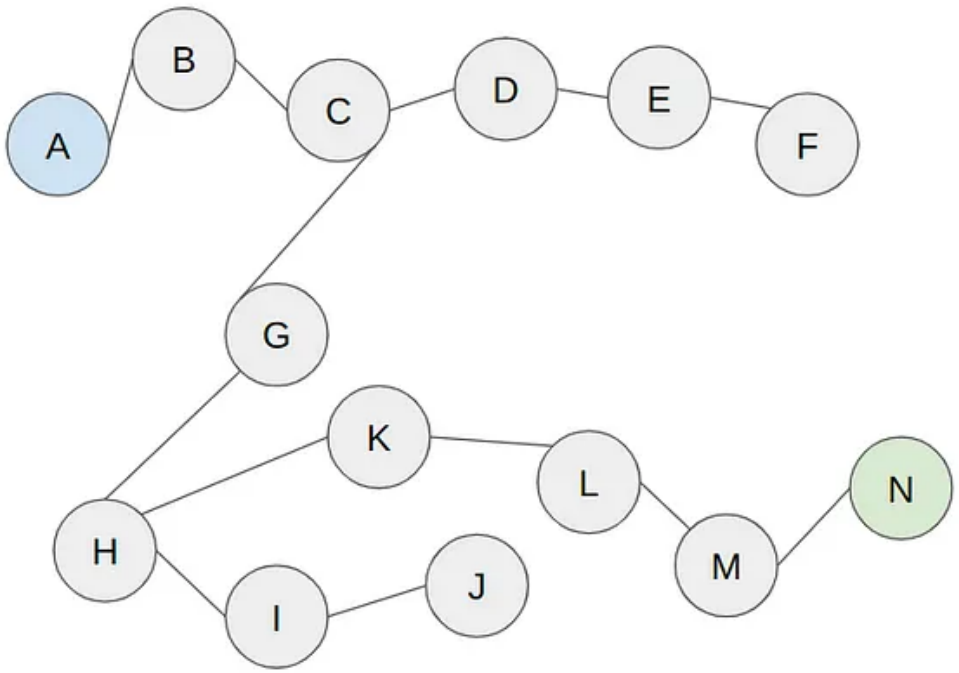
# Graph representation of Maze



# Below is the step-to-step visualization of solving the search problem using the BFS algorithm:




Append ***A*** *to the* ***Frontier.***

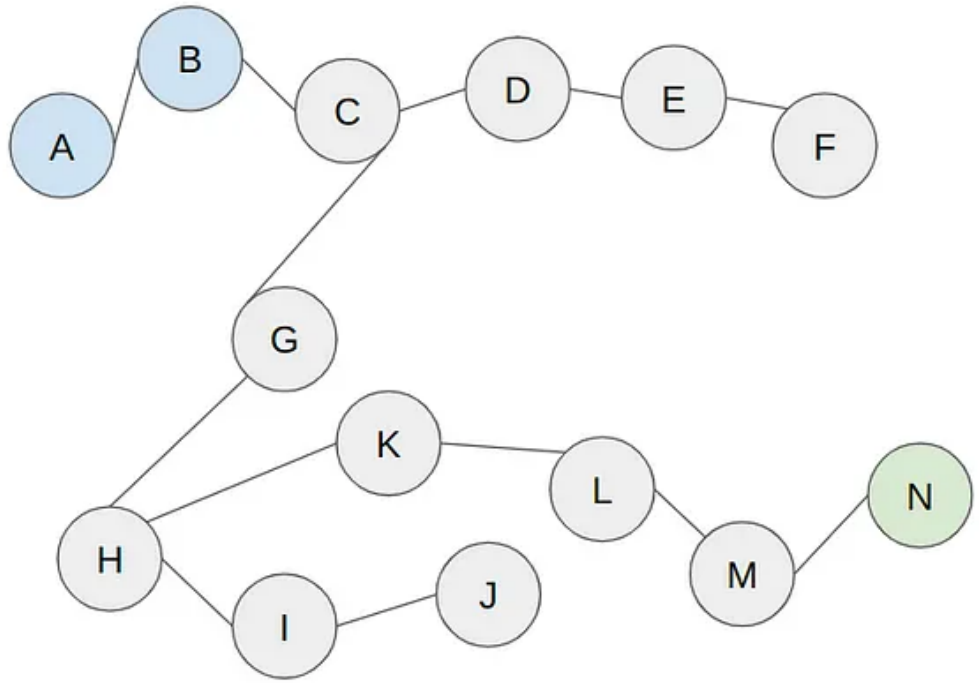



Frontier:
[B]

Visited:
[A]

BFS:
[A]

Append **A** to the **Visited** and **BFS**, then append **A**'s neighbor (**B**) to the **Frontier**. Finally, pop **A** from the **Frontier**.

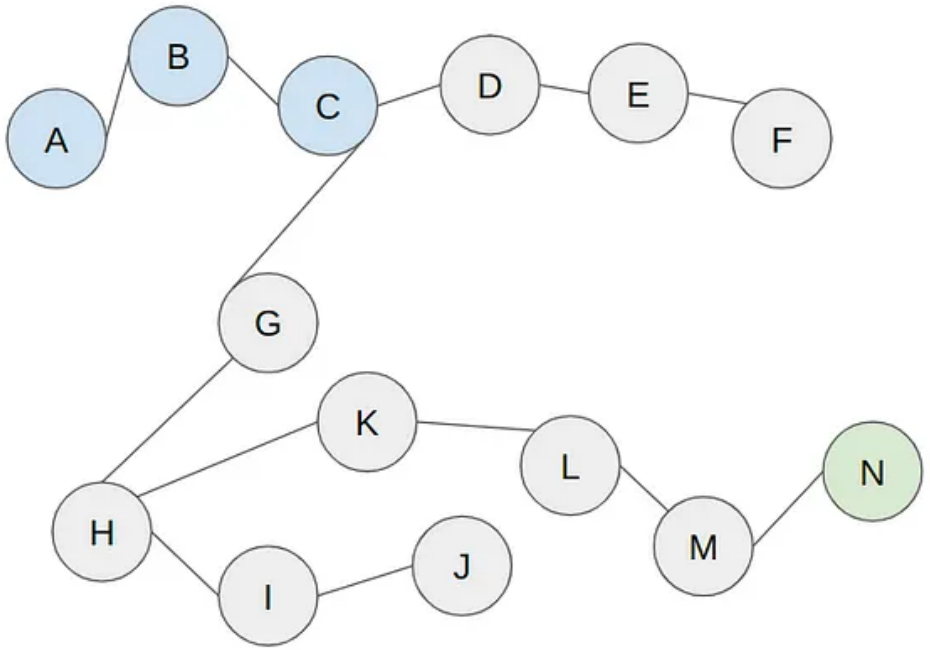



Frontier:
[C]

Visited:
[A, B]

BFS:
[A, B]

Append **B** to the **Visited** and **BFS**, then append **B**'s neighbor (**C**) to the **Frontier**. Finally, pop **B** from the **Frontier**.

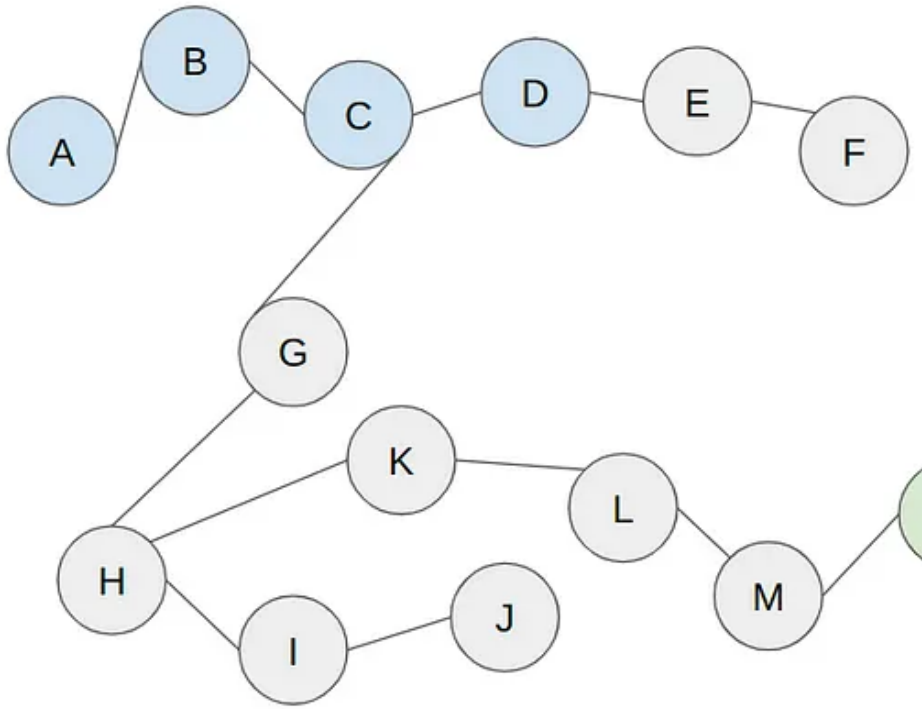



Frontier:
[D, G]

Visited:
[A, B, C]

BFS:
[A, B, C]

Append **C** to the **Visited** and **BFS**, then append **C**'s neighbor (D, G) to the **Frontier**. Finally, pop **C** from the **Frontier**.

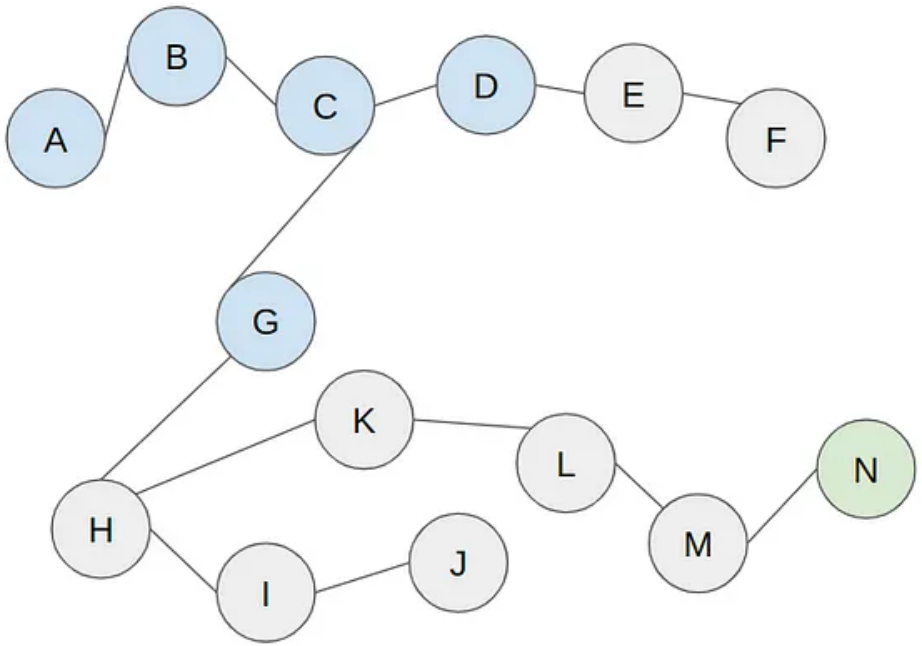



Frontier:
[G, E]

Visited:
[A, B, C, D]

BFS:
[A, B, C, D]

Append **D** to the **Visited** and **BFS**, then append **D**'s neighbor (E) to the **Frontier**. Finally, pop **D** from the **Frontier**.

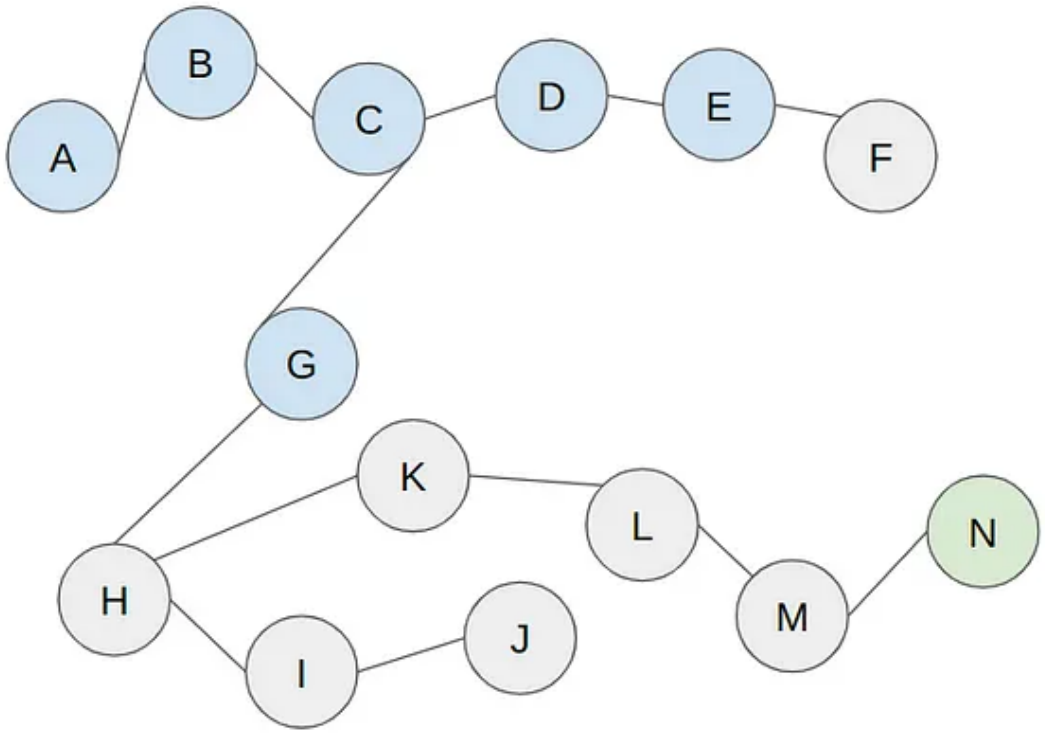



Frontier:
[E, H]

Visited:
[A, B, C, D, G]

BFS:
[A, B, C, D, G]

Append **G** to the **Visited** and **BFS**, then append **G**'s neighbor (H) to the **Frontier**. Finally, pop **G** from the **Frontier**.

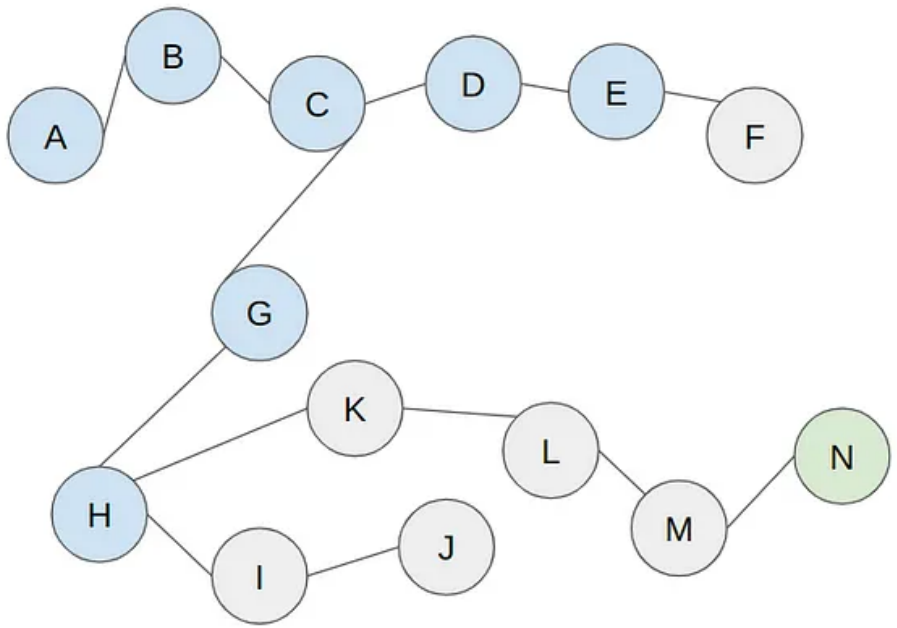



Frontier:
[H, F]

Visited:
[A, B, C, D, G, E]

BFS:
[A, B, C, D, G, E]

Append **E** to the **Visited** and **BFS**, then append **E**'s neighbor (F) to the **Frontier**. Finally, pop **E** from the **Frontier**.

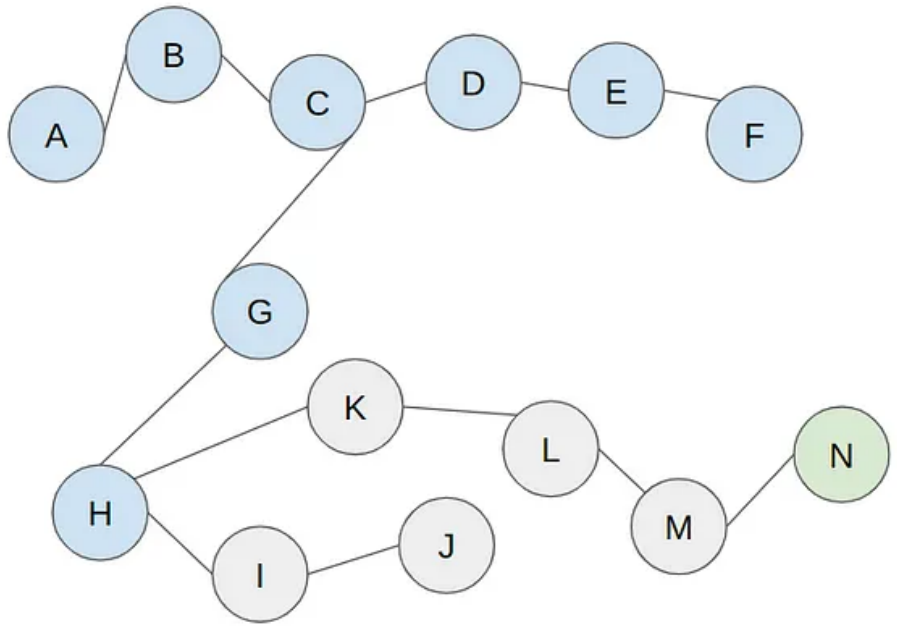



Frontier:
[F, K, I]

Visited:
[A, B, C, D, G, E, H]

BFS:
[A, B, C, D, G, E, H]

Append **H** to the **Visited** and **BFS**, then append **H**'s neighbor (K, I) to the **Frontier**. Finally, pop **H** from the **Frontier**.

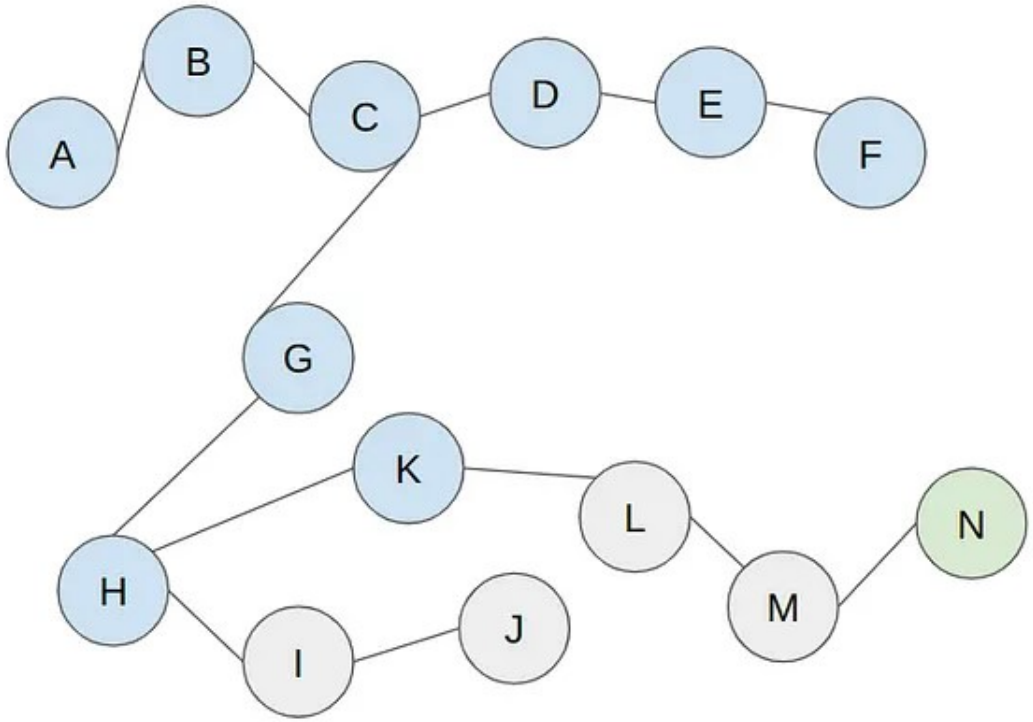



Frontier:
[K, I]

Visited:
[A, B, C, D, G, E, H, F]

BFS:
[A, B, C, D, G, E, H, F]

Append **F** to the **Visited** and **BFS**. Finally, pop **F** from the **Frontier**.

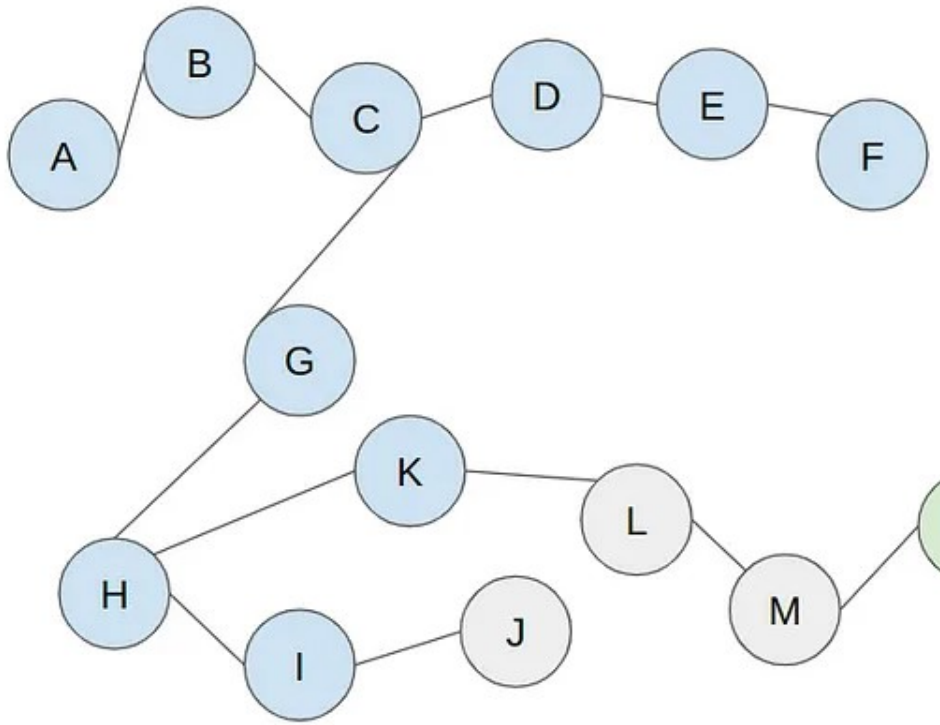



Frontier:
[I, L]

Visited:
[A, B, C, D, G, E, H, F, K]

BFS:
[A, B, C, D, G, E, H, F, K]

Append **K** to the **Visited** and **BFS**, then append **K**'s neighbor (L) to the **Frontier**. Finally, pop **K** from the **Frontier**.

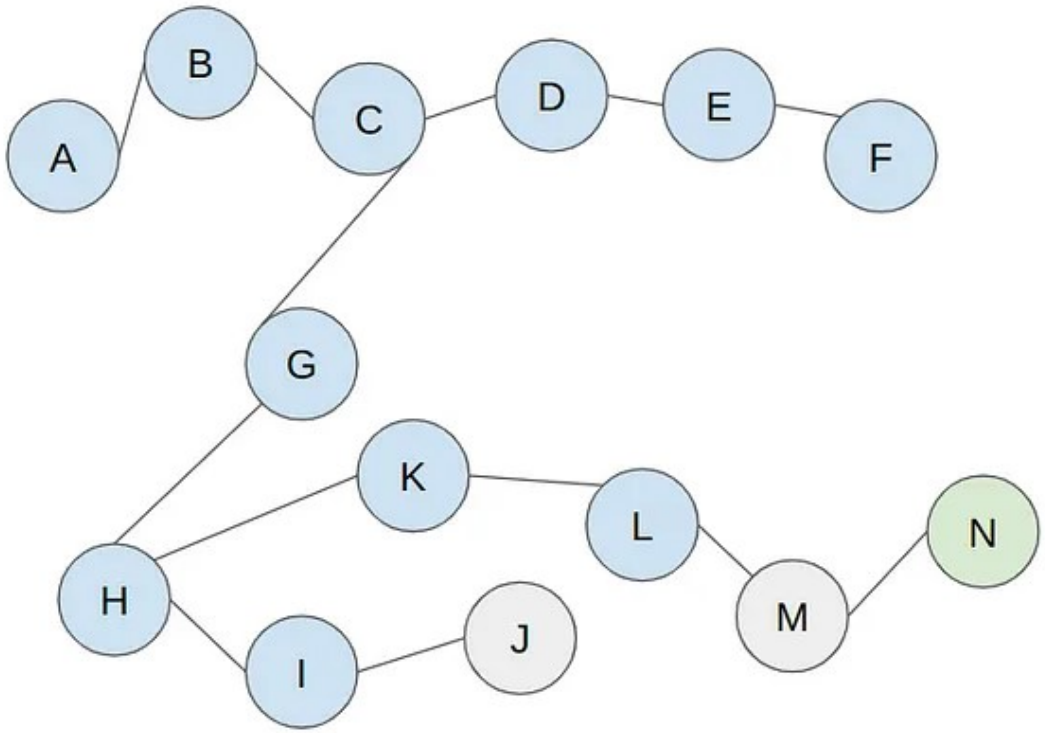



Frontier:
[L, J]

Visited:
[A, B, C, D, G, E, H, F, K, I]

BFS:
[A, B, C, D, G, E, H, F, K, I]

Append **I** to the **Visited** and **BFS**, then append **I**'s neighbor (J) to the **Frontier**. Finally, pop **I** from the **Frontier**.

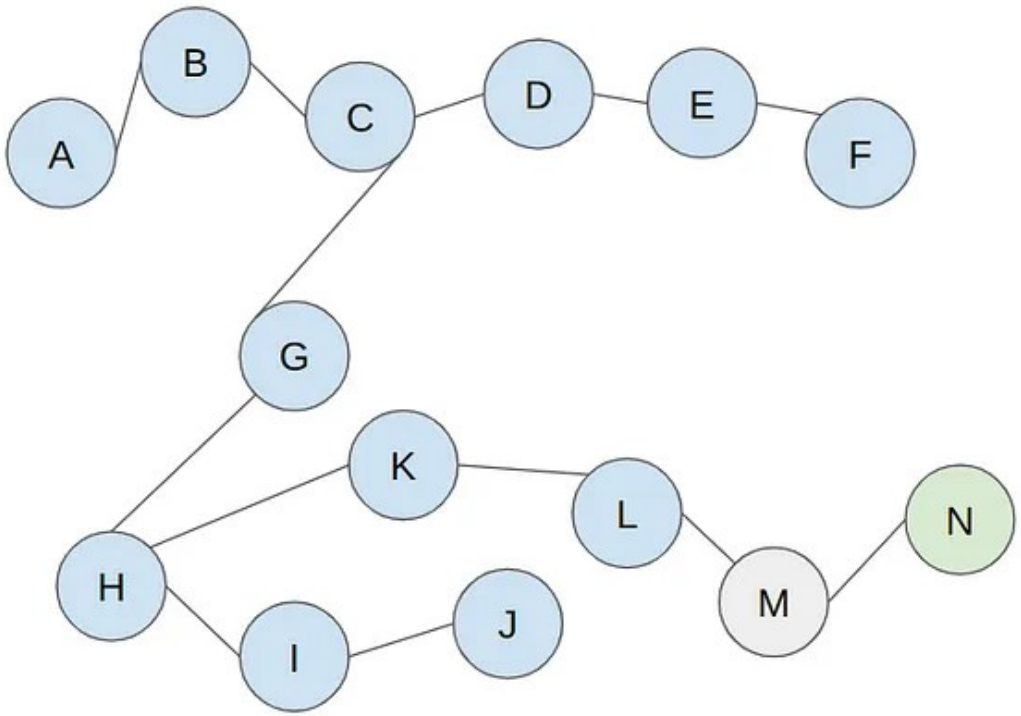



Frontier:
[J, M]

Visited:
[A, B, C, D, G, E, H, F, K, I, L]

BFS:
[A, B, C, D, G, E, H, F, K, I, L]

Append **L** to the **Visited** and **BFS**, then append **L**'s neighbor (M) to the **Frontier**. Finally, pop **L** from the **Frontier**.

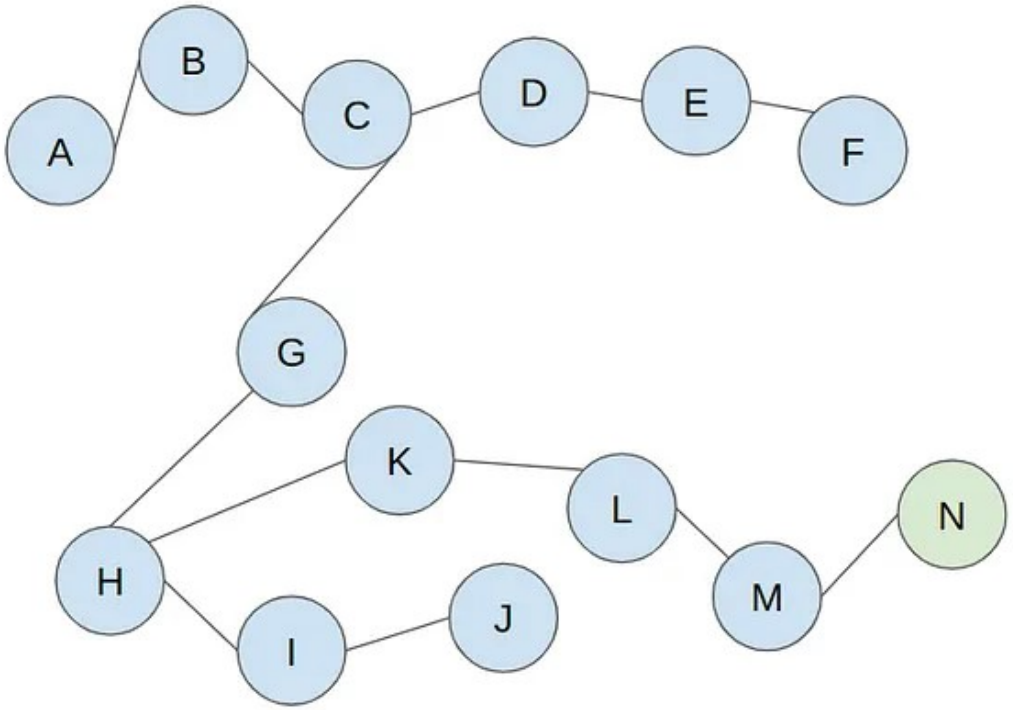



Frontier:
[M]

Visited:
[A, B, C, D, G, E, H, F, K, I, L, J]

BFS:
[A, B, C, D, G, E, H, F, K, I, L, J]

Append **J** to the **Visited** and **BFS**. Finally, pop **J** from the **Frontier**.

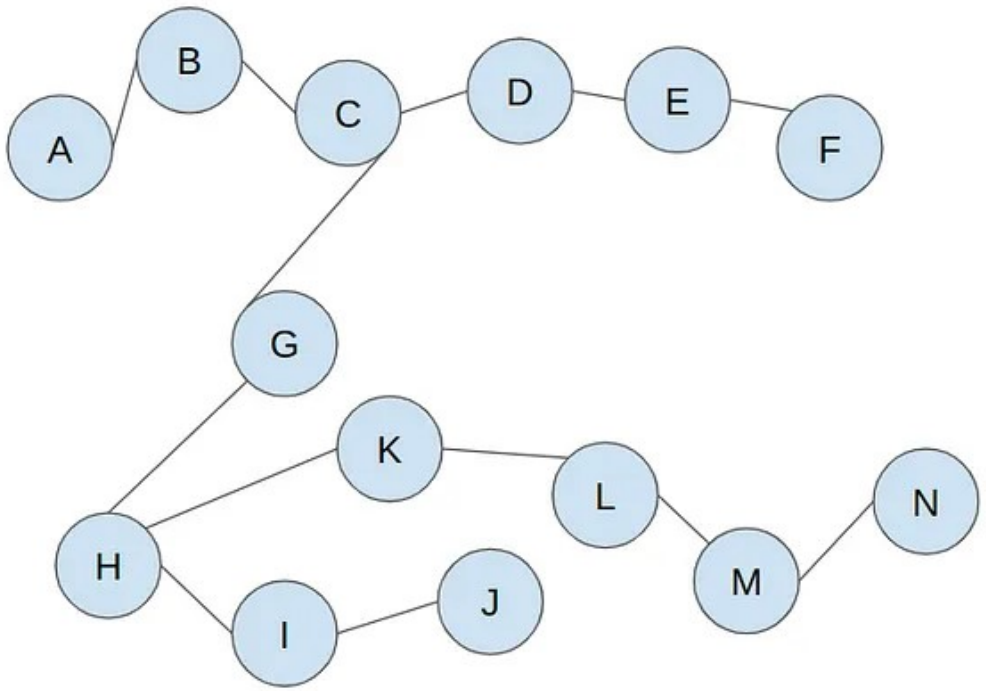



Frontier:
[N]

Visited:
[A, B, C, D, G, E, H, F, K, I, L, J, M]

BFS:
[A, B, C, D, G, E, H, F, K, I, L, J, M]

Append **M** to the **Visited** and **BFS**, then append **M**'s neighbor (N) to the **Frontier**. Finally, pop **M** from the **Frontier**.




Frontier:
[]

Visited:
[A, B, C, D, G, E, H, F, K, I, L, J, M, N]

BFS:
[A, B, C, D, G, E, H, F, K, I, L, J, M, N]

Append **N** to the **Visited** and **BFS**. Finally, pop **N** from the **Frontier**.

# The answer to the above question will be:

- Solution: [A, B, C, D, G, E, H, F, K, I, L, J, M, N]
- Costs: 13 (there are 13 nodes to step to reach the **end node** from the **start node**)