**Assignment for Search Strategies**

**Level 1 – Basic Understanding**

**🔹 Task 1: Define a Search Problem**

Define :

* Possible actions
* Goal test
* Path cost

For -

* Maze
* City map (from A to B)
* Robot vacuum cleaner

➡️ **Classify** if it is:

* Goal-based?
* Deterministic?
* Observable?

1. **Maze –**

**Possible Actions :** The possible actions in the maze are only four directions –

i] Up: Move one cell up from the current position.

ii] Down: Move one cell down from the current position.

iii] Left: Move one cell left from the current position.

iv] Right: Move one cell right from the current position.

**Goal Test :** The goal test is the condition where the current position is equal to the goal or destination position. It can be a predefine position as a target position.

**Path Cost :** The Path Cost is defined as the number of cells or distance travelled from starting to the current position is known as Path Cost.

Number of steps: The number of movements (up, down, left, right) taken to reach the current position from the starting position.

Distance: The Euclidean distance or Manhattan distance between the starting position and the current position.

**Classification**

**Goal-based :** Yes, the maze problem is goal-based because it has a clear objective (reaching the target position).

**Deterministic :** Yes, the maze problem is deterministic because the outcome of each action (up, down, left, right) is predictable and certain. The maze layout and rules are fixed, and there are no random elements.

**Observable :** Yes, the maze problem is observable because the agent has complete knowledge of the current state (position) and can perceive the environment (maze layout). The agent can see the entire maze and knows its current position.

1. City Map (from A to B)

**Possible Actions**

The possible actions in a City Map problem are typically movements or transitions between connected locations:

- Move to adjacent intersections or nodes: Move from one intersection or node to another adjacent intersection or node.

- Take a specific route or road: Take a specific route or road that connects two nodes.

**Goal Test**

The goal test is a condition that determines whether the current location is the destination (point B):

- Reached the destination node: The current node or location is the same as the predefined destination node (point B).

**Path Cost**

The path cost is a measure of the cost or distance traveled from the starting point (point A) to the current location:

- Distance: The Euclidean distance or Manhattan distance between the starting point and the current location.

- Travel time: The estimated time it takes to travel from the starting point to the current location, considering factors like traffic, road type, and speed limits.

- Other costs: Other costs like tolls, fuel consumption, or road conditions can also be considered.

**Classification :**

**Goal-based**

- Yes, the City Map problem is goal-based because it has a clear objective: to find a route from point A (starting point) to point B (destination).

**Deterministic**

- Yes, the City Map problem can be considered deterministic if we assume that the map is static and the traffic patterns are predictable. However, in real-world scenarios, traffic patterns can be dynamic and unpredictable, which might introduce non-determinism.

**Observable**

- Partially Observable, the City Map problem can be partially observable in real-world scenarios because while we can see the map and our current location, we might not have complete knowledge of dynamic factors like traffic congestion, road closures, or construction. However, if we assume a static map with no dynamic factors, it could be considered fully observable.

1. **Robot vacuum cleaner**

**Possible Actions:**

The possible actions for a Robot Vacuum Cleaner are:

- Move forward: Move the robot forward in a straight line.

- Turn: Turn the robot left or right to change direction.

- Clean: Activate the vacuum cleaner to clean the current area.

**Goal Test :**

The goal test for a Robot Vacuum Cleaner is:

- All areas cleaned: The robot has cleaned all accessible areas of the environment.

**Path Cost :**

The path cost for a Robot Vacuum Cleaner can be:

- Distance traveled: The total distance the robot travels while cleaning.

- Time taken: The total time the robot takes to clean the area.

- Energy consumption: The amount of energy the robot consumes while cleaning.

**Classification :**

**Goal-based :**

- Yes, the Robot Vacuum Cleaner problem is goal-based because it has a clear objective: to clean the entire area or reach a specific state (e.g., all areas cleaned).

**Deterministic :**

- Partially Deterministic, the Robot Vacuum Cleaner problem is partially deterministic because while the robot's movements and cleaning actions are predictable, the environment can be dynamic (e.g., people moving around, new obstacles appearing). The robot's sensors and navigation system can also introduce uncertainty.

**Observable :**

- Partially Observable, the Robot Vacuum Cleaner problem is partially observable because while the robot has sensors to detect its surroundings and map the environment, it might not have complete knowledge of the entire area, especially if it's a large or complex space.

**🔹 Task 2: BFS vs DFS Comparison Table**

Fill in a table comparing:

* Data structure used
* Time and space complexity
* Completeness
* Optimality
* When to use

| **Criteria** | **BFS (Breadth-First Search)** | **DFS (Depth-First Search)** |
| --- | --- | --- |
| **Data Structure Used** | Queue | Stack (explicit or via recursion call stack) |
| **Time Complexity** | O(V + E) | O(V + E) |
| **Space Complexity** | O(V) (due to queue storing all frontier nodes) | O(V) (in worst case — for stack/visited nodes) |
| **Completeness** | Yes (if the branching factor is finite) | No (may get stuck in infinite paths in some graphs) |
| **Optimality** | Yes (if all edge costs are equal) | No (does not guarantee shortest path) |
| **When to Use** | When you need to find the shortest/least-cost path or explore nodes level-wise (like in unweighted graphs) | When searching for any path quickly, or when memory is limited, or in scenarios like topological sorting or connected components |

**🔹 Task 3: Manual BFS and DFS Tracing**

Draw a **binary tree (depth = 3)**.

For BFS:

* List visited nodes at each level
* Track queue at each step

For DFS:

* List order of node visits
* Show stack at each step

Excellent — let’s walk through this **manually, step by step**.

**Binary Tree (Depth = 3)**

A

/ \

B C

/ \ / \

D E F G

**Nodes**: A, B, C, D, E, F, G  
**Depth**: 3

**BFS (Breadth-First Search)**

**Algorithm:**

* Use a **queue**
* Visit level-by-level: Root → Left child → Right child

**BFS Steps**

**Queue initially**: [A]  
**Visited**: —

| **Step** | **Visited Node** | **Queue After** |
| --- | --- | --- |
| 1 | A | [B, C] |
| 2 | B | [C, D, E] |
| 3 | C | [D, E, F, G] |
| 4 | D | [E, F, G] |
| 5 | E | [F, G] |
| 6 | F | [G] |
| 7 | G | [] |

**BFS Final Visit Order (Level-wise):**

A → B → C → D → E → F → G

**DFS (Depth-First Search)**

**Algorithm:**

* Use a **stack**
* Visit as deep as possible via left subtree before backtracking

**DFS Steps (Pre-Order: Node → Left → Right)**

**Stack initially**: [A]  
**Visited**: —

| **Step** | **Visited Node** | **Stack After** |
| --- | --- | --- |
| 1 | A | [C, B] |
| 2 | B | [C, E, D] |
| 3 | D | [C, E] |
| 4 | E | [C] |
| 5 | C | [G, F] |
| 6 | F | [G] |
| 7 | G | [] |

**DFS Final Visit Order:**

A → B → D → E → C → F → G

| **Traversal** | **Data Structure** | **Visit Order** |
| --- | --- | --- |
| **BFS** | Queue | A, B, C, D, E, F, G |
| **DFS** | Stack | A, B, D, E, C, F, G |

**LEVEL 2**

**🔹 Task 6: Compare Time and Space Complexity**

For a binary tree of depth 5:

* Count nodes visited in BFS and DFS
* Estimate time complexity O(b^d)
* Estimate space complexity

->

| **Criteria** | **BFS (Breadth-First Search)** | **DFS (Depth-First Search)** |
| --- | --- | --- |
| **Total Nodes (depth=5)** | 63 | 63 |
| **Time Complexity** | O(b^d) = O(2^5) ≈ O(32) | O(b^d) = O(2^5) ≈ O(32) |
| **Space Complexity** | O(b^d) = O(2^5) = 32 (max queue size at deepest level) | O(d) = 6 (max stack size = tree depth + 1) |
| **Max Nodes Visited** | 63 | 63 |

**LEVEL 3**

**🔍 Task 08: Real-World Search Problem**

**📦 Problem: Autonomous Delivery Drone in a City Grid**

**📌 Scenario:**

An AI-powered drone must deliver a package from a warehouse to a customer’s home in a city grid.  
The city has:

* Open airspaces (flyable)
* No-fly zones (restricted)
* Tall buildings (blocked)
* Temporary weather hazards (delays movement)

The drone needs to find an efficient path to deliver the package while avoiding obstacles and delays.

**✅ Problem Definition**

**📖 State Representation**

A state is:

(x, y, battery\_level)

* (x, y) → Drone’s current coordinates on the city grid
* battery\_level → Remaining battery (in units; decreases per move)

**🎛️ Action Model**

Possible actions:

* Move North
* Move South
* Move East
* Move West
* Each move:
  + Consumes 1 battery unit
  + Only valid if within grid and not a no-fly zone or tall building

**🎯 Goal Test**

The drone reaches the **customer’s location (x\_goal, y\_goal)** before running out of battery.

if (x, y) == (x\_goal, y\_goal) and battery\_level ≥ 0 → Goal achieved

**🗺️ Search Strategy**

**A\* Search (A-Star)**  
**Why A\*?**

* Combines **cost so far** (battery used, movement cost)
* With an **estimated cost to goal** (using Manhattan distance as heuristic)
* Guarantees an **optimal path** if heuristic is admissible
* Works efficiently for grid-based navigation problems

**Heuristic:**

h(x, y) = abs(x\_goal - x) + abs(y\_goal - y)

**📊 Summary**

| **Component** | **Definition** |
| --- | --- |
| State | (x, y, battery\_level) |
| Actions | Move North, South, East, West (if not blocked, consumes 1 battery) |
| Goal Test | Drone reaches (x\_goal, y\_goal) with battery ≥ 0 |
| Search Strategy | **A\*** (cost-efficient + heuristic guided) |

**🎨 Possible Extensions:**

* Add recharging stations
* Weather zones (extra movement cost)
* Priority delivery time window
* Multiple packages → route optimization