**Assignment 4**

**Title:**

Implementation of the A\* Algorithm for Optimal Pathfinding in a Grid-based Environment

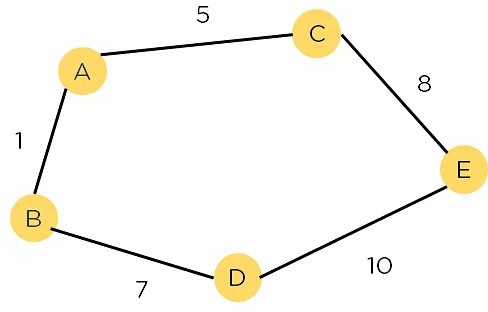
**Aim:**

To implement the A\* (A-star) algorithm to find the optimal path from a start node to a goal node in a grid-based environment, and evaluate its efficiency in comparison to other search algorithms.

**Objectives:**

1. To understand the *A algorithm*\*, including its heuristic-based search mechanism.
2. To implement A\* for **pathfinding** in a grid-based environment.
3. To demonstrate the use of **heuristic functions** in guiding the search towards the goal.
4. To analyze the efficiency of the A\* algorithm and compare it with other pathfinding algorithms like Dijkstra’s Algorithm and BFS.
5. To apply the A\* algorithm to a real-world application such as **robot navigation** or **game development**.

**Theory:**

****

**A Algorithm Overview:**

The *A algorithm*\* is a search algorithm widely used for finding the shortest path between two points, especially in grid-based environments like maps or mazes. It combines elements of both the **Dijkstra’s algorithm** (which finds the shortest path) and **Greedy Best-First Search** (which uses heuristics to prioritize exploration).

A\* ensures that it finds the shortest path efficiently by using two main components:

* **g(n)**: The cost of the path from the start node to the current node.
* **h(n)**: The heuristic estimate of the cost from the current node to the goal node.
* **f(n) = g(n) + h(n)**: The total estimated cost of the path through the current node.

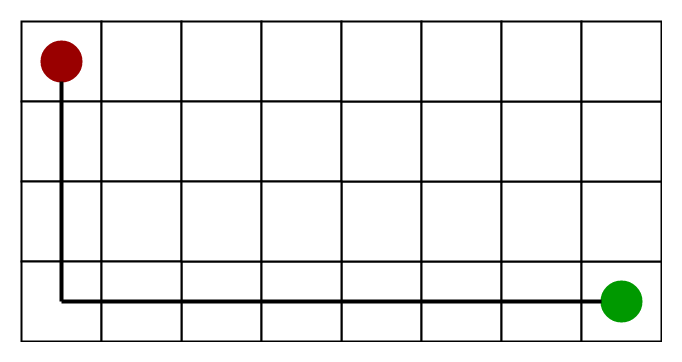
The algorithm continues exploring nodes with the lowest **f(n)** until the goal node is reached.

**Heuristics:**

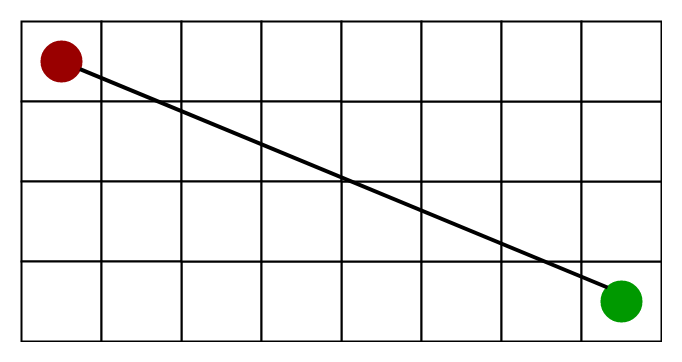


The most commonly used heuristic function for A\* in a grid-based environment is the **Manhattan Distance** (for movement along grid edges) or the **Euclidean Distance** (for diagonal and straight-line movement).

* **Manhattan Distance**: h(n)=∣xgoal−xcurrent∣+∣ygoal−ycurrent∣h(n) = |x\_{\text{goal}} - x\_{\text{current}}| + |y\_{\text{goal}} - y\_{\text{current}}|h(n)=∣xgoal​−xcurrent​∣+∣ygoal​−ycurrent​∣



* **Euclidean Distance**: h(n)=(xgoal−xcurrent)2+(ygoal−ycurrent)2h(n) = \sqrt{(x\_{\text{goal}} - x\_{\text{current}})^2 + (y\_{\text{goal}} - y\_{\text{current}})^2}h(n)=(xgoal​−xcurrent​)2+(ygoal​−ycurrent​)2​



**Procedure:**

**1. Define the Environment:**

* Create a **grid-based environment** where each cell can either be free (walkable) or blocked (obstacle).
* The grid is represented as a 2D array where:
  + 0 indicates free space (walkable).
  + 1 indicates a blocked cell (obstacle).

Example grid:

0 0 0 0 0

1 1 0 1 0

0 0 0 1 0

0 1 0 0 0

**2. Define the Start and Goal Nodes:**

* Specify the coordinates for the **start** node and **goal** node.
* The algorithm will search for the optimal path between these two nodes.

**3. Implement the A Algorithm:**

* **Initialize Open and Closed Lists**:
  + **Open List**: Contains the nodes that are yet to be evaluated.
  + **Closed List**: Contains nodes that have already been evaluated.
* **Start from the initial node**:
  + Calculate the **f(n)**, **g(n)**, and **h(n)** values for the start node.
  + Add it to the open list.
* **Loop**:
  + While the goal is not reached:
    1. Select the node with the lowest **f(n)** from the open list.
    2. Explore its neighbors (up, down, left, right, and diagonal if applicable).
    3. For each neighbor:
       - If it is not already in the closed list, calculate its **f(n)**, **g(n)**, and **h(n)** values.
       - Add it to the open list if it’s a valid move (not an obstacle).
    4. Add the current node to the closed list.
* **Backtrack** from the goal node to the start node to reconstruct the optimal path.

**4. Heuristic Selection:**

* **Manhattan Distance** can be used for grid-based movement where only up/down/left/right moves are allowed.
* **Euclidean Distance** can be used if diagonal movement is allowed.

**5. Implement Path Visualization:**

* Visualize the grid with the obstacles, start, goal, and the optimal path found by A\*.
* Mark the path on the grid for easy identification.

**6. Performance Analysis:**

* Compare the performance of A\* with **Dijkstra’s Algorithm** (which uses only the g(n) cost) and **BFS** (which does not use heuristics).
* Measure:
  + The number of nodes explored.
  + Time taken to find the solution.
  + The efficiency of different heuristics (Manhattan vs. Euclidean).

**Expected Output:**

1. **Grid Environment**:
   * A visual representation of the grid where obstacles, the start node, goal node, and the final path are clearly marked.
2. **Optimal Path**:
   * The shortest path from the start node to the goal node as found by A\*.
   * The algorithm should return the path length, and the list of nodes (coordinates) in the path.
3. **Performance Metrics**:
   * The total number of nodes evaluated.
   * The total cost of the path (sum of g(n) values).
   * Time taken to find the solution.

**Procedure for Example Problem (Grid Pathfinding):**

1. **Define the Grid**:
   * Represent a 5x5 grid where cells can be walkable or blocked.
   * Example:

Grid:

S 0 1 0 G

0 1 1 0 0

0 0 1 0 1

1 0 0 0 1

* Start node (S) is at (0,0), goal node (G) is at (0,4). Start node (S) is at (0,0), goal node (G) is at (0,4).

1. Run the A \* Algorithm:

* Initialize with the start node and calculate the f(n), g(n), and h(n) values.
* Traverse the grid, explore neighbours, and use the heuristic to guide the search.
* Identify the optimal path, which avoids obstacles.

1. **Visualize and Output Results**:

* The algorithm should return the following information:
  + The grid with the optimal path marked:

Path: S -> (0,1) -> (1,2) -> (2,3) -> G

* Performance details:
* Number of nodes explored.
* Total path cost.
* Time taken to find the path.

**Conclusion:**

In this lab, the A\* algorithm was implemented for pathfinding in a grid-based environment. The use of heuristic functions (Manhattan or Euclidean distance) allowed efficient exploration of the grid, leading to the discovery of the optimal path between the start and goal nodes. A\* was found to be more efficient than other algorithms like BFS and Dijkstra’s Algorithm in terms of the number of nodes explored and time taken. This exercise demonstrated the practical application of A\* in navigation and pathfinding problems.