**Lab Assignment 4: Implementation of A\* Algorithm for Pathfinding in an Application**

**1. Objective:**

The objective of this lab assignment is to implement the **A\*** (A-star) algorithm for an application, such as pathfinding on a grid, where the goal is to find the shortest path between two points. A\* is an efficient algorithm that combines elements of Dijkstra’s Algorithm and Greedy Best-First Search. It uses a heuristic to guide the search, making it faster in finding the shortest path.

**2. Problem Statement:**

You are tasked with implementing the A\* algorithm to find the shortest path between a start and a goal location in a 2D grid-based map (or a graph). The grid may contain obstacles, which the path should avoid, and some grid cells may have varying movement costs (for example, different terrains).

**3. Theory:**

**3.1. A\* Algorithm Overview:**

The **A\*** algorithm is a popular pathfinding and graph traversal algorithm. It uses a combination of cost to reach a node and an estimated cost to the goal (heuristic) to find the shortest path.

A\* works by maintaining two lists:

* **Open List**: Nodes that have been discovered but not yet evaluated.
* **Closed List**: Nodes that have already been evaluated.

At each step, A\* picks the node from the open list with the lowest *f* score, which is defined as:

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

**3.2. Heuristic Function:**

The **heuristic (h)** guides the search towards the goal by estimating the cost from the current node to the goal. Common heuristics include:

* **Manhattan Distance**: Used when movement is restricted to horizontal and vertical directions (e.g., in grids).
* **Euclidean Distance**: Used when movement is possible in all directions (e.g., in open spaces).

**3.3. Grid Representation:**

Each cell in a grid can have a cost, and some cells may be blocked or impassable. The A\* algorithm needs to account for this when calculating the cost and choosing the path.

**4. Algorithm Design:**

**4.1. Steps of the A\* Algorithm:**

1. **Initialize**:
   * Place the start node in the open list.
   * Set the g-cost of the start node to 0 and calculate its f-cost.
2. **Loop**:
   * While the open list is not empty:
     + Select the node with the lowest f-cost from the open list (this is the current node).
     + If the current node is the goal node, the path has been found.
     + Move the current node to the closed list.
     + For each neighboring node:
       - If the neighbor is in the closed list, ignore it.
       - Calculate the g-cost of the neighbor.
       - If the neighbor is not in the open list, add it and compute its f-cost.
       - If the neighbor is already in the open list, update its g-cost and f-cost if the new path is better.
3. **Termination**:
   * If the goal is reached, reconstruct the path by backtracking from the goal node to the start node.
   * If the open list is empty and the goal has not been reached, return failure (no path).

**4.2. Pseudocode:**

sql

Copy code

function A\_star(start, goal):

open\_list = priority\_queue containing start node

closed\_list = empty set

g\_score[start] = 0

f\_score[start] = heuristic(start, goal)

while open\_list is not empty:

current = node in open\_list with lowest f\_score

if current == goal:

return reconstruct\_path(came\_from, current)

remove current from open\_list

add current to closed\_list

for each neighbor of current:

if neighbor in closed\_list:

continue

tentative\_g\_score = g\_score[current] + distance(current, neighbor)

if neighbor not in open\_list:

add neighbor to open\_list

elif tentative\_g\_score >= g\_score[neighbor]:

continue

came\_from[neighbor] = current

g\_score[neighbor] = tentative\_g\_score

f\_score[neighbor] = g\_score[neighbor] + heuristic(neighbor, goal)

return failure

function reconstruct\_path(came\_from, current):

total\_path = [current]

while current in came\_from:

current = came\_from[current]

total\_path.prepend(current)

return total\_path

**5. Application: Pathfinding in a Grid**

Consider a 2D grid with some cells that are blocked or have higher movement costs. A\* will be used to find the shortest path from the start position to the goal, avoiding blocked cells and minimizing movement costs.

**5.1. Grid Representation:**

* **Start (S)**: The starting point.
* **Goal (G)**: The destination.
* **Empty Cell**: Walkable cells (normal movement cost).
* **Blocked Cell (X)**: Cells that cannot be passed through.
* **Cost Cells**: Cells that have additional movement costs.

**Example Grid:**

Copy code

S . . X . G

. X . X . .

. . . . . X

. X X X . .

. . . . . .

* S: Start (0, 0)
* G: Goal (5, 0)
* X: Blocked cell (impassable)

**5.2. Pathfinding Example:**

* A\* will start at S, expand its neighbors, and move toward the goal G, avoiding blocked cells (X). It will use the heuristic to prioritize nodes closer to G while minimizing the path cost.

**6. Heuristic Function:**

In this grid-based application, you can use the **Manhattan Distance** as the heuristic, calculated as:

scss

Copy code

h(n) = abs(current\_x - goal\_x) + abs(current\_y - goal\_y)

This heuristic works well for grid-based pathfinding where movement is restricted to four directions.

**7. Example Execution:**

Assume the start node is (0, 0) and the goal node is (5, 0). A\* will calculate the g, h, and f scores for each node it processes. It will move in the direction of the node with the lowest f score, while avoiding blocked cells.

**Steps:**

* Start at (0, 0).
* Explore neighbors: (1, 0), (0, 1).
* Calculate g, h, and f for each.
* Continue exploring and updating the open and closed lists until the goal is reached.

**8. Expected Output:**

The program should output the shortest path from the start to the goal, avoiding obstacles and minimizing movement costs. The final path should be displayed as a sequence of grid coordinates or visually on the grid.

Example Output:

mathematica

Copy code

Path found: [(0, 0), (1, 0), (2, 0), (3, 0), (4, 1), (5, 0)]

Total cost: 6

**9. Conclusion:**

This lab assignment demonstrates the implementation of the A\* algorithm in a pathfinding application. A\* efficiently finds the shortest path in a grid by balancing the cost of reaching a node and the estimated cost to the goal. It is applicable in real-world scenarios like robotics, game development, and GPS systems.

**10. References:**

* Hart, P. E., Nilsson, N. J., & Raphael, B. (1968). A Formal Basis for the Heuristic Determination of Minimum Cost Paths. *IEEE Transactions on Systems Science and Cybernetics*.
* Russell, S. J., & Norvig, P. (2020). *Artificial Intelligence: A Modern Approach* (4th ed.). Pearson.