**Artificial Intelligent (Lab)**

**Task # 07**

****

**Submitted To: Sir Rasikh Ali**

**Submitted By: Shumaila Maryam**

**Roll no: SU92-BSDSM-F24-062**

**Submitted On: 16 September 2025**

**Section: BSDS-3A**

**Department of Software Engineering, Superior University, Lahore**

**Introduction:**

The **A\* Path Finder Program** is a Python tool designed to find the shortest path between nodes in a graph using the A\* search algorithm. It allows the user to dynamically create a graph by adding nodes and weighted edges, set heuristic values for each node, and then calculate the most efficient path from a start node to a goal node. The program also provides step-by-step output, showing which nodes are visited and the cost calculations involved in the pathfinding process.

**Why I Made This Program:**

The main idea behind this program is to practice and demonstrate concepts of **graphs, algorithms, loops, user input, and conditional statements** in Python. Graph traversal and pathfinding are essential concepts in computer science, and the A\* algorithm is a widely used heuristic-based approach for finding optimal paths. By creating this program, I wanted to:

* Learn how to represent graphs in Python using dictionaries and lists.
* Understand the A\* algorithm, including concepts like g(n) (cost from start), h(n) (heuristic), and f(n) (total estimated cost).
* Practice taking dynamic user input to build a graph and assign heuristics.
* Gain experience in using loops and conditionals to manage open and closed nodes, calculate costs, and reconstruct paths.
* Visualize the pathfinding process through step-by-step console outputs.

This program serves both as an educational exercise and a practical tool for understanding and testing the A\* algorithm on small graphs.

**How It Works:**

**1. Starting with Data**

The program starts by allowing the user to input the number of edges in the graph. For each edge, the user provides:

* The starting node
* The ending node
* The cost (weight) of traveling between the nodes

This creates a dynamic graph structure using a dictionary of adjacency lists.

**2. Adding Heuristics**

After the graph is created, the user enters heuristic values for each node. The heuristic represents an estimate of the cost from that node to the goal node and is essential for the A\* algorithm to make informed decisions.

**3. Graph Summary**

The program prints a summary of the graph showing each node and its neighbors along with their edge weights. It also displays the heuristic values for each node and the chosen start and goal nodes. This ensures the user knows the dataset being used for the search.

**4. Running the A\* Algorithm**

The program performs the A\* search as follows:

* It initializes the **open list** with the start node and an empty **closed list**.
* It maintains a dictionary g for the cost from the start node to each node.
* It maintains a dictionary parent to reconstruct the path after reaching the goal.
* At each step, it selects the node with the smallest total estimated cost f = g + h.
* It visits this node, prints its status, and adds neighbors to the open list if they are not already visited or if a shorter path is found.
* This process repeats until the goal node is reached or no nodes are left to explore.

**5. Path Reconstruction**

Once the goal node is reached, the program reconstructs the path by tracing back from the goal node to the start node using the parent dictionary. The path and its total cost are then displayed.

**6. Handling Special Cases**

* If the start and goal nodes are the same, the program recognizes this immediately.
* If a node has no neighbors, it prints a message to indicate that.
* If no path exists between the start and goal, the program prints a clear message.

**Summary**

1. The **A\* Path Finder Program** is a Python project designed to find the shortest path between nodes in a weighted graph using the A\* algorithm. It allows dynamic creation of the graph, heuristic assignment, and step-by-step visualization of the search process.
2. This project was created to practice Python programming concepts such as dictionaries, lists, loops, conditionals, user input, and algorithm implementation. It demonstrates how the A\* search algorithm works in a clear, educational, and interactive way.