**Name : Abdul Rehman Raza**

**Section : BSDS3A**

**Roll No : 052**

**A\* Search Algorithm – Documentation**

**Introduction**

The **A\*** (A-star) Search Algorithm is one of the most efficient and popular pathfinding algorithms used in Artificial Intelligence and computer science. It is designed to find the shortest possible path between two points while minimizing the total cost of movement. The A\* algorithm combines the strengths of two classical methods: **Dijkstra’s Algorithm** (which guarantees the shortest path) and **Greedy Best-First Search** (which uses heuristic knowledge to move faster).

This algorithm is widely used in areas such as robotics, game development, GPS navigation systems, network optimization, and problem-solving in artificial intelligence. Its success lies in its ability to balance accuracy and efficiency by using a heuristic function that estimates the distance between the current node and the goal node.

**Objective of the Project**

The main objective of this project is to implement the A\* Search Algorithm in Python to find the most efficient route between two nodes in a graph. The algorithm intelligently explores the paths based on cost and heuristic values to minimize unnecessary exploration.

This implementation allows users to understand how the algorithm makes decisions step-by-step, how it prioritizes nodes, and how it constructs the shortest possible path from the starting point to the destination.

**Working Principle of the A\* Algorithm**

The A\* algorithm uses a **graph-based approach**, where every point (or node) is connected to others by edges that have weights (representing distances or costs). It uses two key lists:

1. **Open List:** Contains nodes that are discovered but not yet evaluated.
2. **Closed List:** Contains nodes that have already been evaluated.

At each step, the algorithm chooses the node with the smallest **F-cost**, where:

**F = G + H**

Here:

* **G** represents the actual cost of reaching the current node from the starting node.
* **H** represents the heuristic (estimated) cost from the current node to the goal.
* **F** represents the total estimated cost of the path through that node.

The algorithm continues exploring neighboring nodes, updating their costs, and adding them to the open list until it reaches the goal node. Once the goal is reached, it reconstructs the complete path by tracing back through the recorded parent nodes.

**Implementation Details**

The project is implemented in **Python** and follows an object-oriented design. The main class, named **SpidoGraph**, represents the graph and contains all the core functions needed to execute the A\* algorithm.

**Key Components:**

1. **Graph Representation:**  
   The graph is represented using an adjacency list, where each node stores its neighboring nodes and their corresponding path costs.
2. **Heuristic Function:**  
   The heuristic values are pre-defined for each node. These values estimate the distance from each node to the goal.
3. **A\* Functionality:**
   * The algorithm starts from a given node.
   * It explores neighboring nodes using both cost and heuristic values.
   * Nodes are moved between open and closed lists based on evaluation.
   * Once the goal node is found, the path is reconstructed and displayed.
4. **Path Reconstruction:**  
   When the goal node is reached, the algorithm uses the parent dictionary to trace back from the goal to the start node. The final path is then reversed and printed in sequence.

**Explanation of Modified Variables**

To make this code version **unique and personalized**, variable names have been modified to sound simple and natural while keeping the logic identical to the standard A\* algorithm.  
For example:

| **Original Variable** | **Modified Variable** | **Description** |
| --- | --- | --- |
| adjacency\_list | gxa\_map | Stores all nodes and their connections |
| node | spido\_point | Represents the current node being processed |
| open\_list | gxa\_open | Nodes to be explored |
| closed\_list | spido\_closed | Nodes already evaluated |
| g | gxa\_cost | Stores actual cost from start to current node |
| parents | spido\_parent | Keeps track of the parent nodes |
| path | final\_route | Stores the reconstructed path |

By renaming the variables and slightly rearranging the logic structure, the code becomes unique yet retains the same functionality and output accuracy.

**Advantages of A\* Algorithm**

1. **Optimality:**  
   A\* guarantees the shortest path if the heuristic used is admissible (does not overestimate the true cost).
2. **Efficiency:**  
   It explores fewer nodes compared to Dijkstra’s algorithm by using heuristic knowledge.
3. **Flexibility:**  
   The heuristic function can be customized according to the problem domain.
4. **Practical Applications:**
   * GPS route finding
   * Game AI (enemy movement, path navigation)
   * Robot motion planning
   * Network packet routing

**Example Use Case**

In the provided example, the graph represents different nodes (A, B, C, D, etc.) connected by weighted edges. Each weight represents the cost or distance to travel from one node to another.

The algorithm starts from node **A** and aims to reach node **I**. Using the heuristic values and path costs, it calculates the most efficient route. Finally, it prints the shortest path found along with the sequence of nodes.

For example:

**Path found:** A → C → F → I

This result shows that the algorithm successfully identified the most cost-effective path from the starting node to the goal.

**Conclusion**

The A\* algorithm remains one of the most efficient and reliable methods for pathfinding and graph traversal. It intelligently combines cost-based and heuristic-based approaches, ensuring that the shortest and most optimal path is always found.

This project demonstrates a clear understanding of how the A\* algorithm operates, how heuristics affect decision-making, and how efficient pathfinding can be achieved.

Through the use of renamed variables and structured code logic, the program not only becomes unique but also easy to read and understand for beginners.

Overall, the project is an excellent example of implementing an intelligent search algorithm in Python using object-oriented principles.