University of Asia Pacific

Department of Computer Science and Engineering

***Course Title* –**Artificial Intelligence and Expert Systems || Lab

***Course Code* –** CSE 404

Project -Implementation of a small Address Map (from your own home to UAP) using A* Search Algorithm.

Submitted By

Asikur Rahman Sumon

Reg. No-19201055   Section-A2.

Submitted To

Dr. Nasima Begum

Associate Professor, UAP, CSE Department

Problem Title

Implementation of a small Address Map (from your own home to UAP) using A* Search Algorithm.

Problem Description

To implement a small address map from one's own home to the University of Asia Pacific using the A* search algorithm, the following steps can be taken:

1. Create a graph of the area with nodes representing locations and edges representing routes between locations using google maps.

2. Assign weights to the edges based on factors such as distance, traffic, road conditions etc.

3. Use the A* search algorithm with heuristic functions to find the shortest path from the starting node (home) to the destination node (University of Asia Pacific).

4. Implement the algorithm in code using a programming language such as Python, Java or any language.

5. Test the algorithm with sample inputs and fine-tune as necessary. Some key considerations include choosing appropriate heuristic functions to ensure efficient search and handling cases where there are multiple valid paths to the destination. Additionally, the accuracy of the route may be impacted by the quality of the map and the data used for edge weight assignments.

A Algorithm*

A* is a graph traversal and path search algorithm, which is used in many fields of computer science due to its completeness, and optimal efficiency. It works by searching the graph from the starting node using a heuristic function to evaluate the potential cost of moving to different neighboring nodes. The algorithm evaluates these nodes based on two factors: the actual cost of moving to the node and the estimated cost from that node to the destination node.

The steps of the A algorithm*

1. Initialize the algorithm with the starting node and the destination node.

2. Generate a set of potential paths that start with the starting node and move towards the destination node.

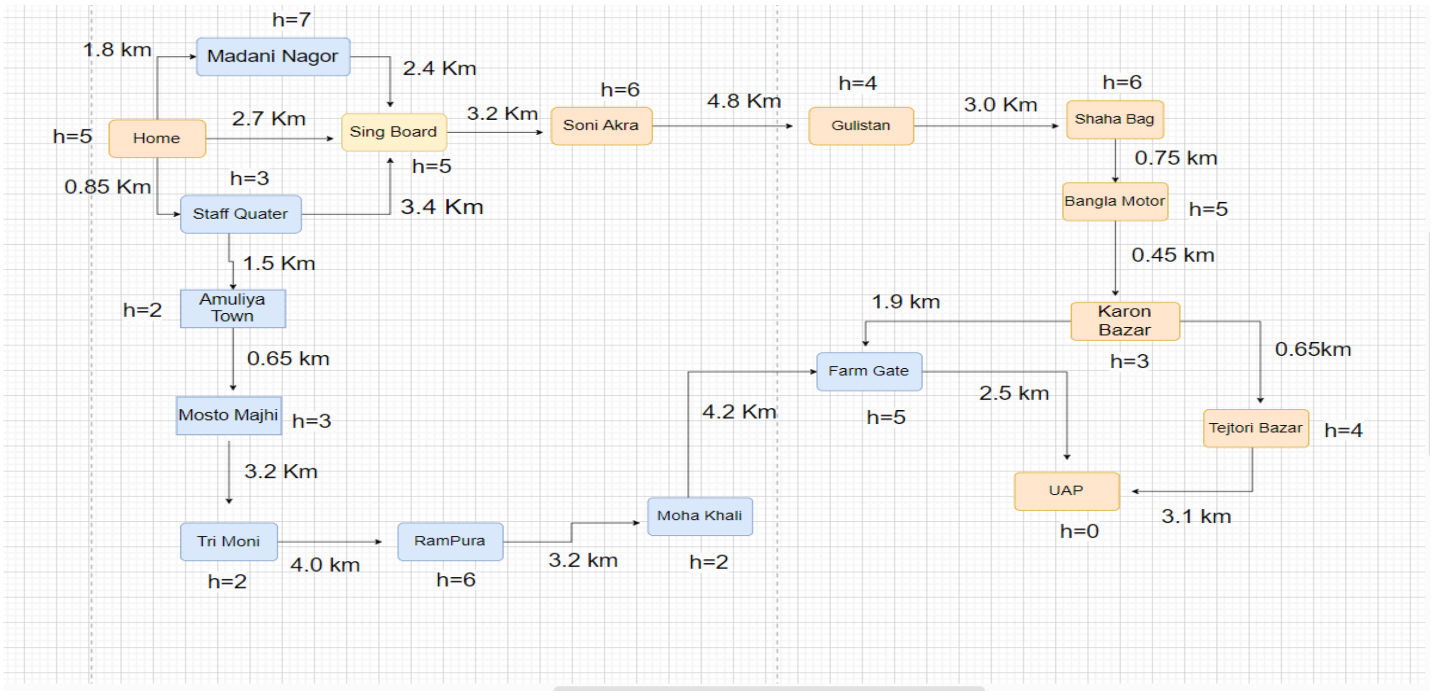
3. Use a heuristic function to determine which path to explore first (i.e., the one that has the highest potential to reach the destination node).

4. While the current node is not the destination node and there are still potential paths to explore, select the path that has the highest potential to reach the destination and explore the next node on that path.

Tools & Languages:

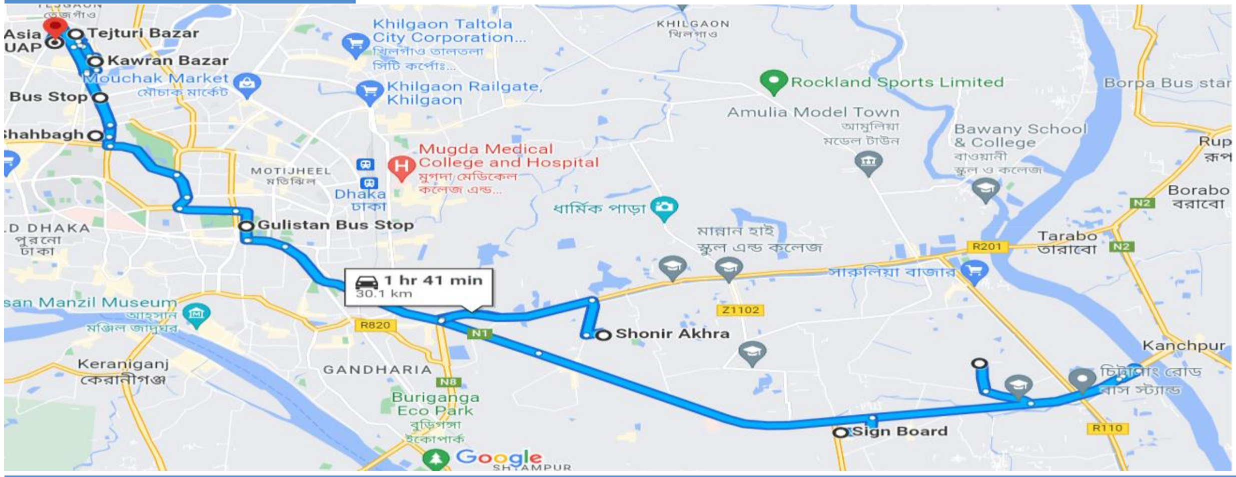
- Diagram.net. (Design Tree).
- Notepad (Write rules & facts).
- PyCharm.

Graph



The above graph is my small address map.

GOOGLE MAP:

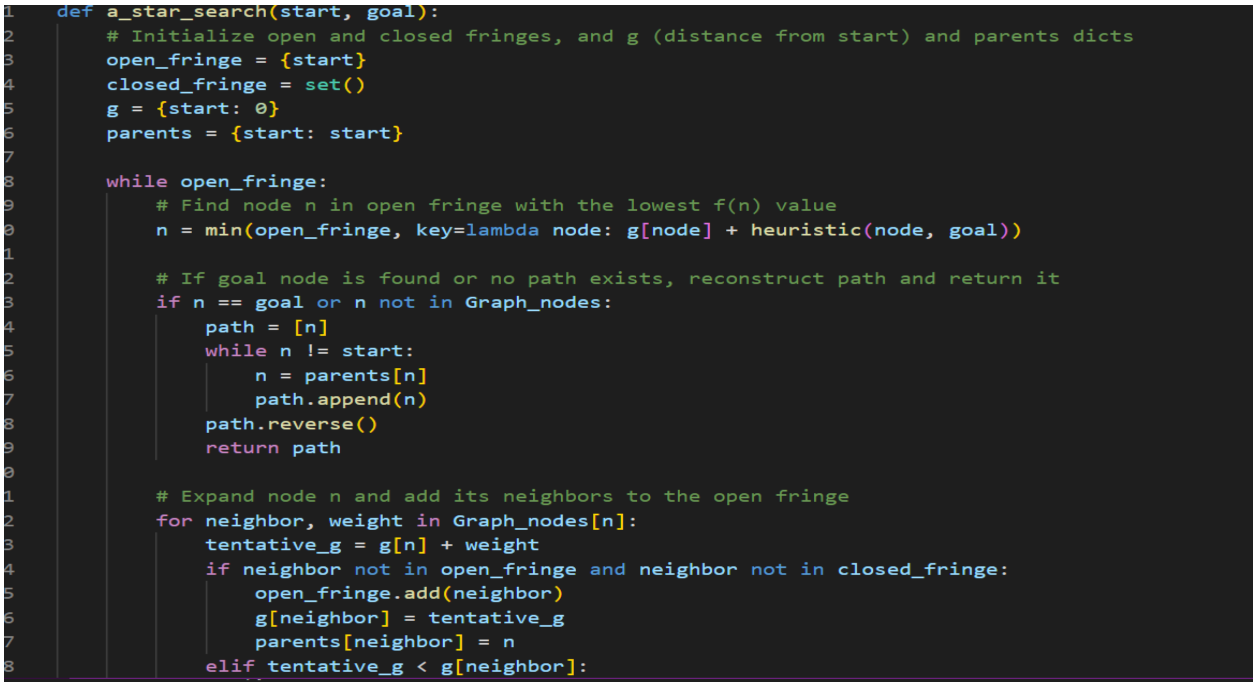


Nodes:

Location Name	Latitude, Longitude
Home	23.698172, 90.508396
madani nagor	23.69721222197322, 90.50219265068066
sing board	23.69383252677157, 90.48133311411327
soni akra	23.70267925598862, 90.45042872427254
Gulistan	23.725247432011304, 90.41187467116401
sha bag	23.738441330255814, 90.39566599120757
bangla motor	23.746967214172802, 90.39493154011586
karon bazar	23.750184202676092, 90.39324059230391
Farmgate	23.758506870657218, 90.38990857169617
Tejtori bazar	23.7563648430853 5, 90.39248523488524
Mohakhali	23.77318443242795, 90.4009900948624

Mosto majhi	23.73911634730412, 90.48085935806495
uap	23.754894602408545, 90.38954351205821

A* implement with code

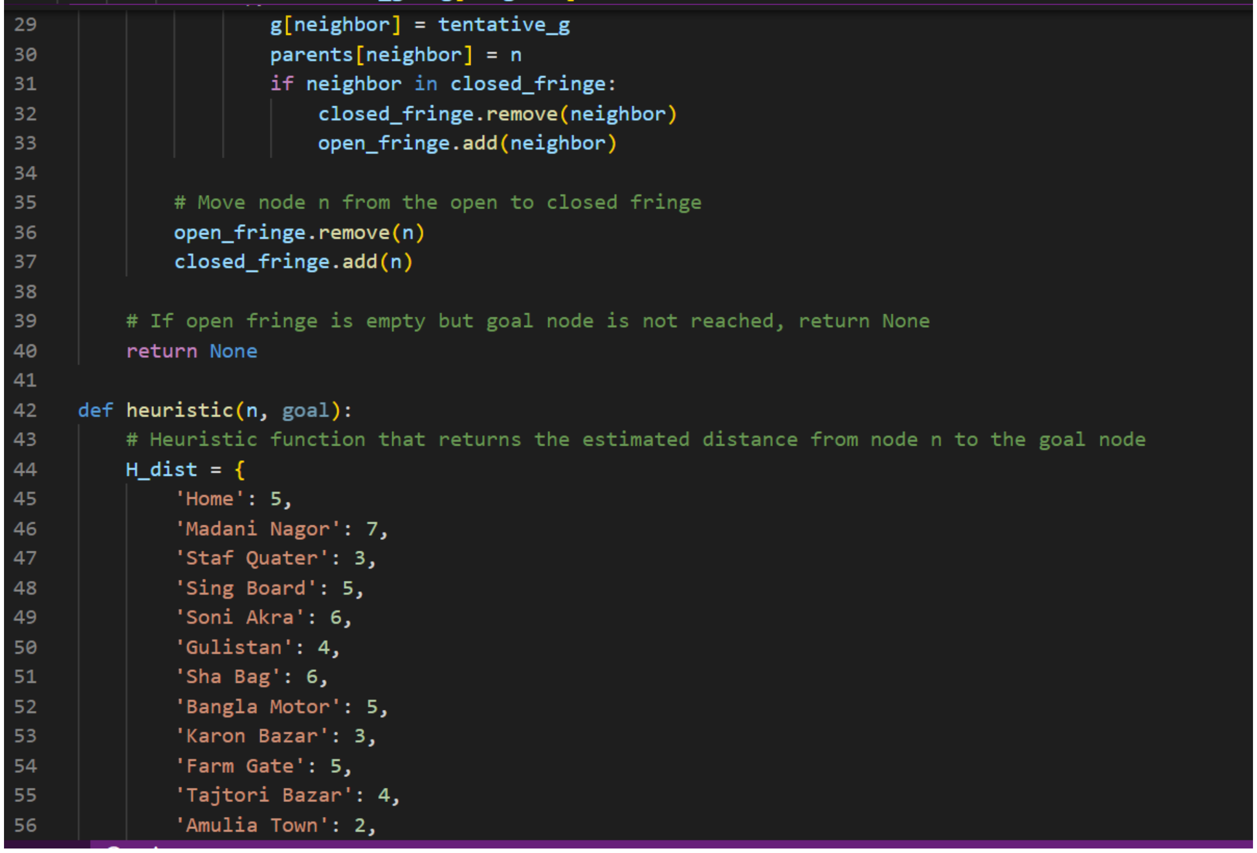


```python
def a_star_search(start, goal):
    # Initialize open and closed fringes, and g (distance from start) and parents dicts
    open_fringe = {start}
    closed_fringe = set()
    g = {start: 0}
    parents = {start: start}

    while open_fringe:
        # Find node n in open fringe with the lowest f(n) value
        n = min(open_fringe, key=lambda node: g[node] + heuristic(node, goal))

        # If goal node is found or no path exists, reconstruct path and return it
        if n == goal or n not in Graph_nodes:
            path = [n]
            while n != start:
                n = parents[n]
                path.append(n)
            path.reverse()
            return path

        # Expand node n and add its neighbors to the open fringe
        for neighbor, weight in Graph_nodes[n]:
            tentative_g = g[n] + weight
            if neighbor not in open_fringe and neighbor not in closed_fringe:
                open_fringe.add(neighbor)
                g[neighbor] = tentative_g
                parents[neighbor] = n
            elif tentative_g < g[neighbor]:
```

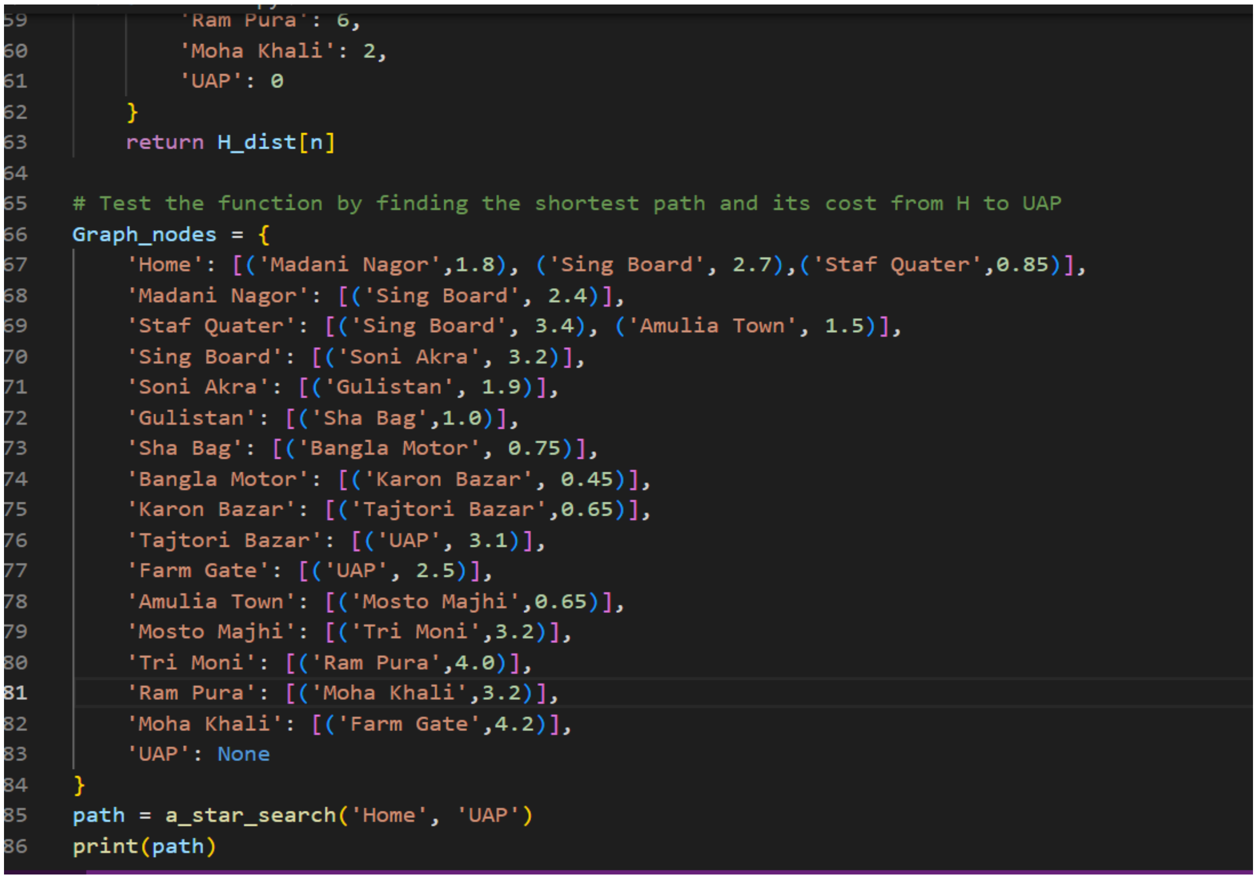


```python
                g[neighbor] = tentative_g
                parents[neighbor] = n
                if neighbor in closed_fringe:
                    closed_fringe.remove(neighbor)
                    open_fringe.add(neighbor)

        # Move node n from the open to closed fringe
        open_fringe.remove(n)
        closed_fringe.add(n)

    # If open fringe is empty but goal node is not reached, return None
    return None

def heuristic(n, goal):
    # Heuristic function that returns the estimated distance from node n to the goal node
    H_dist = {
        'Home': 5,
        'Madani Nagor': 7,
        'Staf Quater': 3,
        'Sing Board': 5,
        'Soni Akra': 6,
        'Gulistan': 4,
        'Sha Bag': 6,
        'Bangla Motor': 5,
        'Karon Bazar': 3,
        'Farm Gate': 5,
        'Tajtori Bazar': 4,
        'Amulia Town': 2,
```


```
        'Ram Pura': 6,
        'Moha Khali': 2,
        'UAP': 0
    }
    return H_dist[n]

# Test the function by finding the shortest path and its cost from H to UAP
Graph_nodes = {
    'Home': [('Madani Nagor',1.8), ('Sing Board', 2.7),('Staf Quater',0.85)],
    'Madani Nagor': [('Sing Board', 2.4)],
    'Staf Quater': [('Sing Board', 3.4), ('Amulia Town', 1.5)],
    'Sing Board': [('Soni Akra', 3.2)],
    'Soni Akra': [('Gulistan', 1.9)],
    'Gulistan': [('Sha Bag',1.0)],
    'Sha Bag': [('Bangla Motor', 0.75)],
    'Bangla Motor': [('Karon Bazar', 0.45)],
    'Karon Bazar': [('Tajtori Bazar',0.65)],
    'Tajtori Bazar': [('UAP', 3.1)],
    'Farm Gate': [('UAP', 2.5)],
    'Amulia Town': [('Mosto Majhi',0.65)],
    'Mosto Majhi': [('Tri Moni',3.2)],
    'Tri Moni': [('Ram Pura',4.0)],
    'Ram Pura': [('Moha Khali',3.2)],
    'Moha Khali': [('Farm Gate',4.2)],
    'UAP': None
}
path = a_star_search('Home', 'UAP')
print(path)
```

OUTPUT :

```
Path found: ['Home', 'Sing Board', 'Soni Akra', 'Gulistan', 'Sha Bag', 'Bangla Motor', 'Karon Bazar', 'Tajtori Bazar', 'UAP']

The path cost is 18.65 Km
```

CONCLUSION:

The A* algorithm, utilized in computer science and game development, is a highly efficient path finding method. It incorporates heuristics and combines two other algorithms, the greedy best-first search and breadth-first search, to determine the optimal path in a network or graph.

Particularly in scenarios where determining the shortest path is crucial, such as in GPS systems, maps, and robotics, the A* algorithm is highly advantageous. In general, this algorithm is an indispensable and versatile tool for addressing a multitude of real-world problems in computer science and beyond.