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.

Date of Submission- 20.03.2023

Submitted By

Rifa Tasfia

Reg. No-19201045 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.

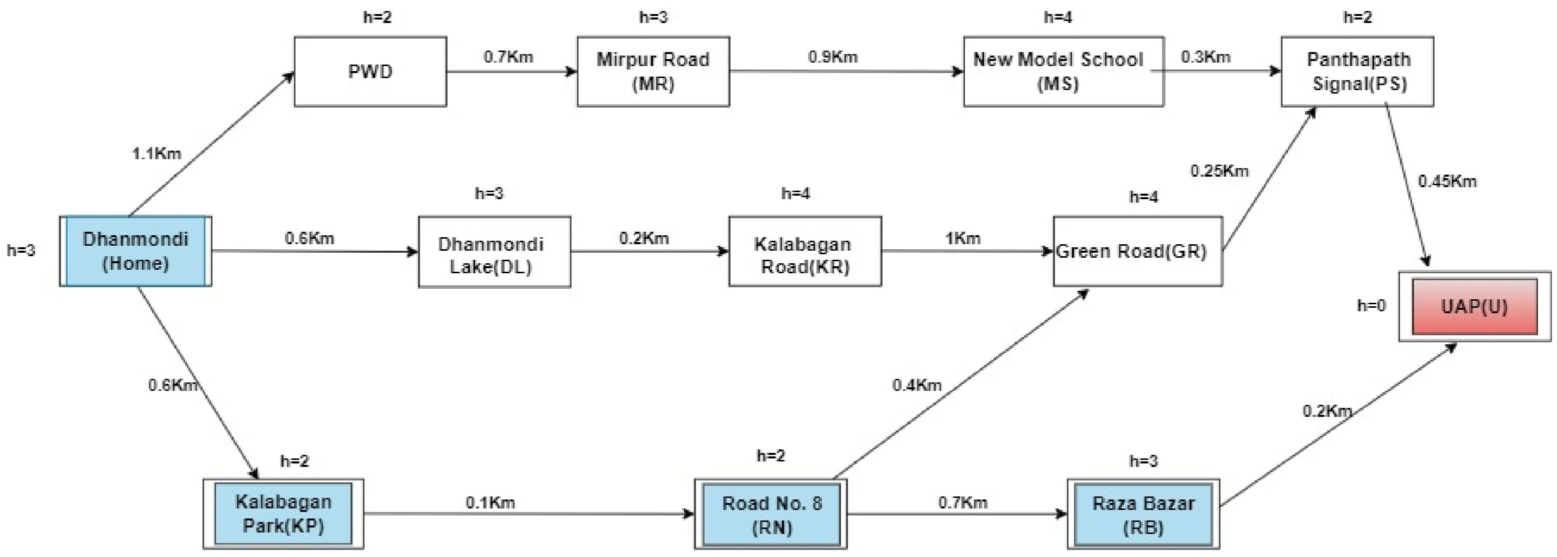
5. Repeat step 4 until either the destination node is reached, or all potential paths have been explored.

By using a heuristic function to prioritize node exploration, the A* algorithm is able to find the shortest path more efficiently than other search algorithms.

Tools & Languages:

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

Graph



The above graph is my small address map.

Heuristic Value:

The heuristic function provides an estimate of the minimum cost between a given node and the target node.

h (Dhanmondi) = (45 % 4) + 2 = 3

h (PWD) = (45 % 5) +2 = 0 + 2 = 2

h (Mirpur Road) = (45 % 5) + 3 = 0 + 3 =3

h (New Model School) = (45 % 6) + 1 = 3 +1 =4

h (Panthapath Signal) = (45 % 4) +1 = 1 + 1 =2

h (Dhanmondi Lake) = (45 % 4) +2 = 1+2 =3

h (Kalabagan Road) = (45 % 4) + 3 = 1+ 3 =4

h (Green Road) = h (Dhanmondi) + 1 = 3 + 1 =4

h (Kalabagan Park) = (45 % 2) + 1 = 1 + 1 =2

h(Road No 8) = h(Kalabagan) + 1 = 1 + 1 = 2

h (Raza Bazar) = h(Panthapath) + 1 = 2 + 1= 3

h (UAP) = 0

Source Code: https://github.com/tasfiarifa/CSE-404-4.1-AI-lab-

A implementation with Code*

```
def a_star_search(start, goal):
    open_fringe = set(start)
    close_fringe = set()
    g = {}  # store distance from starting node
    parents = {}  # parents contains an adjacency map of all nodes

    # ditance of starting node from itself is zero
    g[start] = 0

    # start is root node i.e it has no parent nodes
    # so start is set to its own parent node
    parents[start] = start  # start node

    while len(open_fringe) > 0:
        n = None
        # node with lowest f() is found
        for v in open_fringe:
            if n == None or g[v] + heuristic(v) < g[n] + heuristic(n):
                n = v

        if n == goal or Graph_nodes[n] == None:
            pass
        else:
            for (m, weight) in get_neighbors(n):
                # nodes 'm' not in first and last set are added to first
                # n is set its parent
                if m not in open_fringe and m not in close_fringe:
                    open_fringe.add(m)
                    parents[m] = n
                    g[m] = g[n] + weight


                # for each node m,compare its distance from start i.e g(m) to
the
                # from start through n node
                else:
                    if g[m] > g[n] + weight:
                        # update g(m)
                        g[m] = g[n] + weight
                        # change parent of m to n
                        parents[m] = n

                        # if m in closed set,remove and add to open
                        if m in close_fringe:
                            close_fringe.remove(m)
                            open_fringe.add(m)

        if n == None:
            print('Path does not exist!')
            return None

        # if the current node is the goal
        # then  begin reconstructing the path from it to the start
        if n == goal:
            path = []
            path_cp = []
            full = {
                'H': "Dhanmondi (Home)",
                'PWD': "PWD",
                'MR': "Mirpur Road",
                'MS': "New Model School",
                'PS': "Panthapath Signal",
                'DL': "Dhanmondi Lake",
                'KR': "Kalabagan Road",
                'GR': "Green Road",
                'KP': "Kalabagan Park",
                'RN': "Road No 8",
                'RB': "Raza Bazar",
                'U': "UAP"
            }
            while parents[n] != n:
                path.append(n)
                path_cp.append(full[n])
                n = parents[n]

            path.append(start)
            path_cp.append(full[start])
            path.reverse()
            path_cp.reverse()
            print('Path found: {}'.format(str(path_cp).replace(",", "-->")))
```

```python
            return path

        open_fringe.remove(n)
        close_fringe.add(n)

    print('Path does not exist!')
    return None


def get_neighbors(v):
    if v in Graph_nodes:
        return Graph_nodes[v]
    else:
        return None


def heuristic(n):
    H_dist = {
        'H': 3,
        'PWD': 2,
        'MR': 3,
        'MS': 4,
        'PS': 2,
        'DL': 3,
        'KR': 4,
        'GR': 4,
        'KP': 2,
        'RN': 2,
        'RB': 3,
        'U': 0
    }
    return H_dist[n]


Graph_nodes = {
    'H': [('PWD', 1.1), ('DL', 0.6), ('KP', 0.8)],
    'PWD': [('MR', 0.7)],
    'MR': [('MS', 0.9)],
    'MS': [('PS', 0.3)],
    'PS': [('U', 0.45)],
    'DL': [('KR', 0.2)],
    'KR': [('GR', 1)],
    'GR': [('PS', 0.25)],
    'KP': [('RN', 0.1)],
    'RN': [('GR', 0.4), ('RB', 0.7)],
    'RB': [('U', 0.2)],
    'U': None
}

path = a_star_search('H', 'U')

path_cost = 0.0

for i in range(len(path) - 1):
    for key, value in Graph_nodes[path[i]]:
        if key == path[i + 1]:
            path_cost += value
            break
print("The path cost is %.2f Km" % path_cost)
```

Sample Input

```
Graph_nodes = {
    'H': [('PWD', 1.1), ('DL', 0.6), ('KP', 0.8)],
    'PWD': [('MR', 0.7)],
    'MR': [('MS', 0.9)],
    'MS': [('PS', 0.3)],
    'PS': [('U', 0.45)],
    'DL': [('KR', 0.2)],
    'KR': [('GR', 1)],
    'GR': [('PS', 0.25)],
    'KP': [('RN', 0.1)],
    'RN': [('GR', 0.4), ('RB', 0.7)],
    'RB': [('U', 0.2)],
    'U': None
}
def heuristic(n):
    H_dist = {
        'H': 3,
        'PWD': 2,
        'MR': 3,
        'MS': 4,
        'PS': 2,
        'DL': 3,
        'KR': 4,
        'GR': 4,
        'KP': 2,
        'RN': 2,
        'RB': 3,
        'U': 0
    }
```

Sample Output

Path found: ['Dhanmondi (Home)'--> 'Kalabagan Park'--> 'Road No 8'--> 'Raza Bazar'--> 'UAP']

The path cost is 1.80 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.