University of Asia Pacific

Artificial Intelligence and Expert Systems

CSE 404

Project – 1

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

Name: Md. Azim Islam

Registration: 19201026

Section: A2

Course Code: CSE 404

Semester: Fall 2022

Dr. Nasima Begum

Associate Professor, University of Asia Pacific, Dhaka

Table of Content

1. Problem Title

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

1.1 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.

Note: This description assumes that University of Asia Pacific (UAP) refers to a specific location and not a general concept.

2. A* Algorithm:

A* search algorithm is a pathfinding algorithm that can be used to find the shortest path between two nodes on a weighted graph.
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. A* tries to minimize the sum of these two factors.

Due to its efficiency and accuracy in finding the shortest path, A* is commonly used in applications such as maps, video games, and robotics.

2.1 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.

During this process, the algorithm keeps track of the shortest path found so far and updates it if a shorter path is found later. Additionally, it uses a priority queue to keep track of the nodes that it needs to explore next in the search process.

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.

2.2 Using google map to generate the path graph:

Google Maps uses various data sources to determine routes between two or more points. The data sources include real-time and historical traffic data, road network data, and user data such as location and search history.

In order to find the best route, Google Maps uses a routing algorithm based on the A* search algorithm. This algorithm takes into account factors such as the distance and estimated *travel time*, as well as real-time and historical *traffic conditions*.

Additionally, Google Maps offers multiple options for route optimization, including alternate routes and the ability to add multiple stops to a route. Users can also customize their preferences such as avoiding highways or tolls.

Overall, the pathfinding and route-planning capabilities of Google Maps is a complex system that takes into account various factors and data sources to provide users with the most efficient and accurate routes possible.

For example, the shortest route from my home to University of Asia Pacific based on distance stands as.

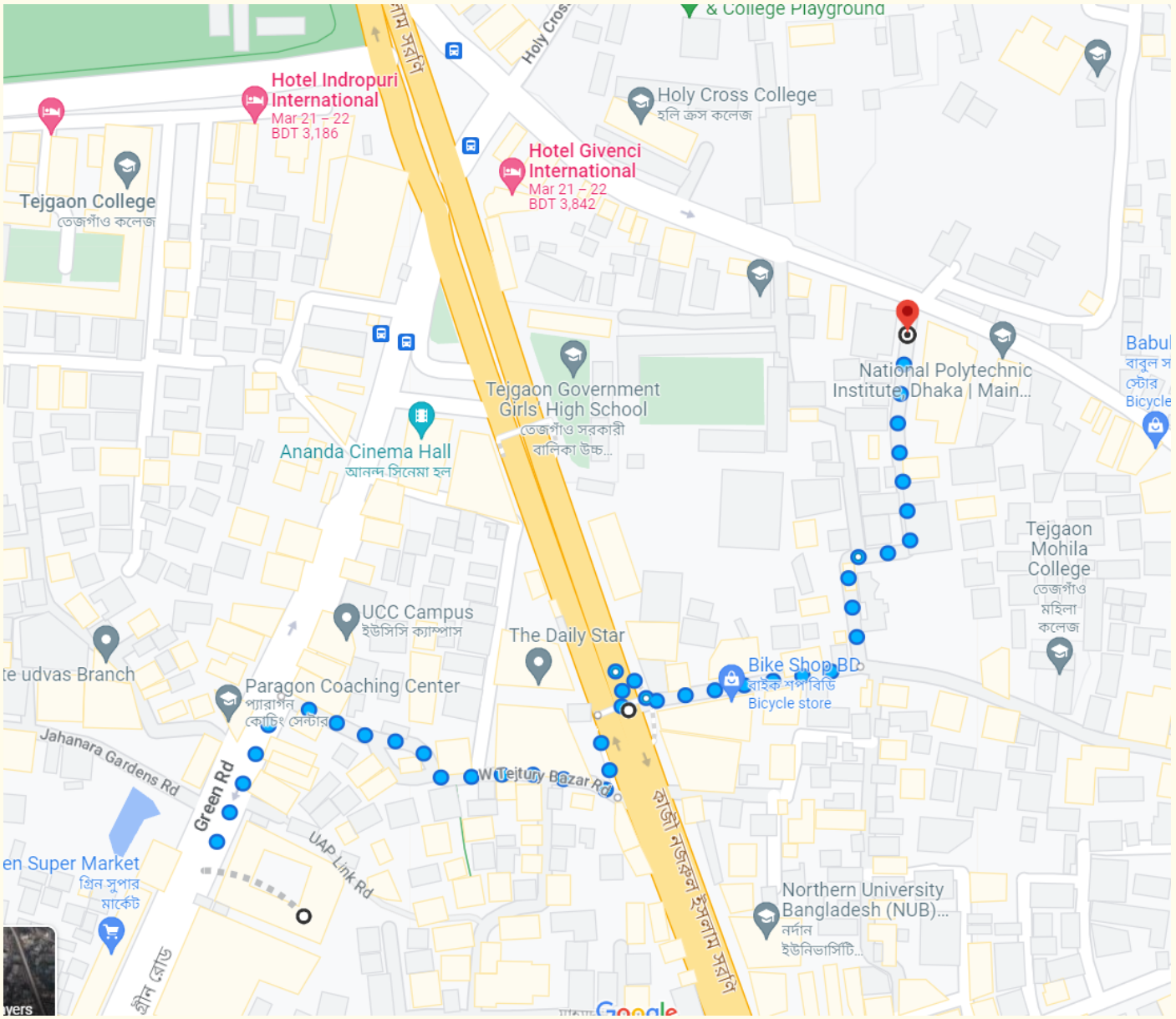

Figure 1: Shortest path from my home to University of Asia Pacific.

2.3 Path Graph Nodes:

Location Name	Latitude, Longitude
Tejturi Bazar	23.757262, 90.392418
K.B. Rail Gate	23.756152, 90.394041
Farmgate	23.758343, 90.390466
The Daily Star	23.755619, 90.390917
SAARC Fountain	23.750060, 90.393046
Panthapath Signal	23.751092, 90.387233
Karwan Bazar	23.750377, 90.394330
Ananda Cinema Hall	23.756876, 90.389878
Chourangi Mor	23.755872, 90.392188
University of Asia Pacific	23.755039, 90.389071

Table 1:Path Graph Nodes Latitude, Longitude. Start Node, Goal Node

2.4 Calculating the Heuristic Values of the nodes:

We can use the Haversine formula to find the distance between two nodes.

The Haversine formula is a mathematical formula used to determine the great-circle distance between two points on the surface of a sphere , such as the Earth.

Great-circle distance is the shortest distance between two points on the surface of a sphere , and is commonly used in navigation and aviation. The formula takes into account the curvature of the Earth's surface and the fact that it is not a perfect sphere. It relies on the longitude and latitude of the two points being compared, and calculates the distance between them by accounting for the spherical shape of the Earth. The formula has many practical applications, including in mapping, control systems, and more.

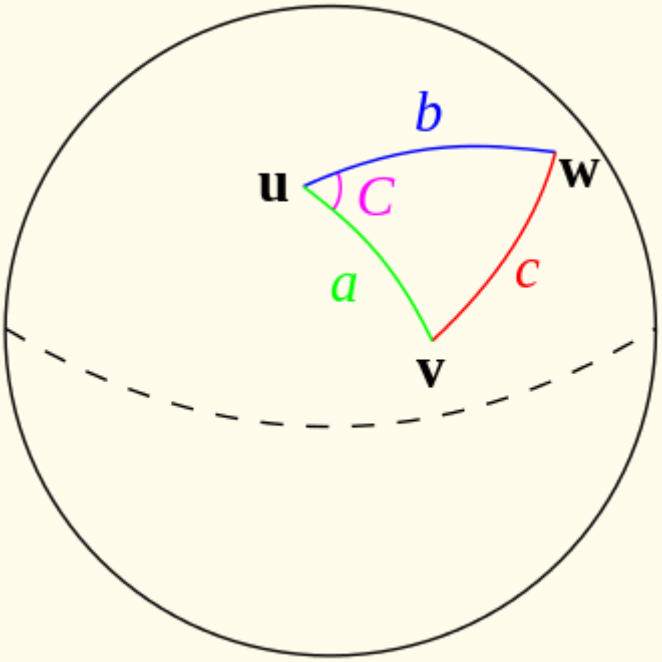

Figure 2: Spherical triangle solved by the law of haversines

The formula which was used to calculate the distance between two longitude and latitudes:

```
    lon1 = radians(lon1)
    lon2 = radians(lon2)
    lat1 = radians(lat1)
    lat2 = radians(lat2)
    # Haversine formula
    dlon = lon2 - lon1
    dlat = lat2 - lat1
    a = sin(dlat / 2)**2 + cos(lat1) * cos(lat2) * sin(dlon / 2)**2
    c = 2 * asin(sqrt(a))
    # Radius of earth in kilometers. Use 3956 for miles
    r = 6371
    # calculate the result
    return(c * r)
```

2.5 Heuristic Values of The Nodes:

Location Name	Heuristic Values
Tejturi Bazar	0.4209 KM
K.B. Rail Gate	0.5207 KM
Farmgate	0.3939 KM
The Daily Star	0.1986 KM
SAARC Fountain	0.6857 KM
Panthapath Signal	0.4771 KM
Karwan Bazar	0.7451 KM
Ananda Cinema Hall	0.2202 KM
Chourangi Mor	0.3305 KM
University of Asia Pacific	0.0000 KM

Table 2: Heuristic Values of The Nodes. Start Node, Goal Node

2.6 Path Graph:

Edges:

Tejturi Bazar -> K.B. Rail Gate -> 0.23 KM
Tejturi Bazar -> Chourangi Mor -> 0.21 KM
Tejturi Bazar -> Farmgate -> 0.29 KM
K.B. Rail Gate -> Chourangi Mor -> 0.28 KM
K.B. Rail Gate -> Karwan Bazar -> 0.75 KM
Chourangi Mor -> Karwan Bazar -> 0.7 KM
Chourangi Mor -> The Daily Star -> 0.24 KM
Farmgate -> The Daily Star -> 0.4 KM
Farmgate -> Ananda Cinema Hall -> 0.35 KM
The Daily Star -> SAARC Fountain -> 0.9 KM
The Daily Star -> UAP -> 0.27 KM
SAARC Fountain -> Panthapath Signal -> 0.65 KM
Panthapath Signal -> UAP -> 0.5 KM
Karwan Bazar -> SAARC Fountain -> 0.22 KM
Ananda Cinema Hall -> UAP -> 0.25 KM

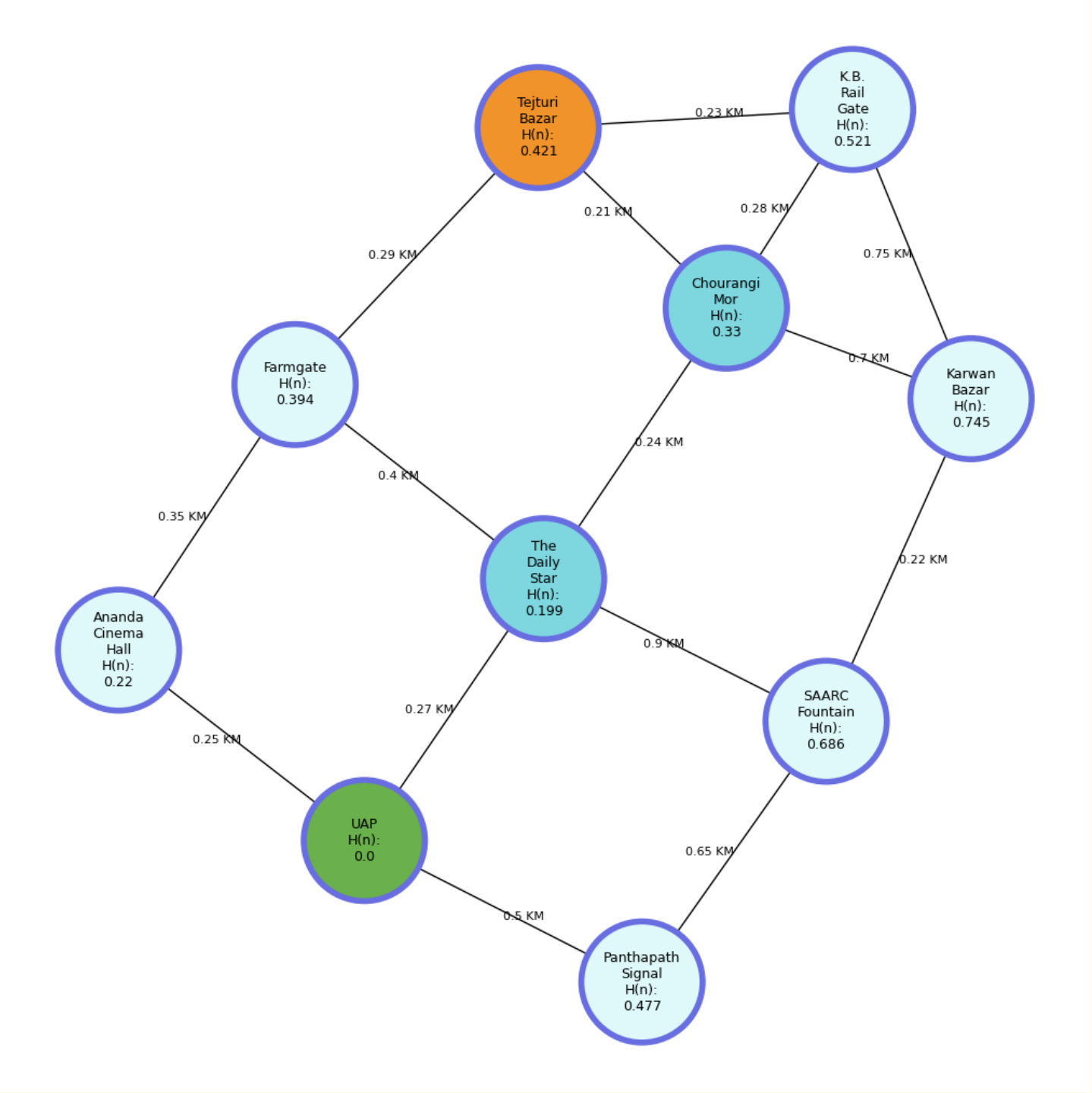

Figure 3: Total path graph (bi-directional). Start Node, Goal Node, Intermediate Node, Shortest Path Node. H(n) values are in KM.

3. Implementation:

We use a PriorityQueue to get the shortest path based on their stored f(n).

[f(n) = g(n) + h(n)]

3.1 Code:

```
from collections import defaultdict
from math import radians, cos, sin, asin, sqrt
from queue import PriorityQueue
from time import sleep
import igraph as ig
import matplotlib.pyplot as plt

input = open('input_map_graph.txt', "r").readline

# Returns the distance between two longitude latitudes.
def find_longitude_lattitude_distance(n1, n2):
    # The math module contains a function named
    # radians which converts from degrees to radians.
    lat1 = n1[0]
    lat2 = n2[0]
    lon1 = n1[1]
    lon2 = n2[1]
    lon1 = radians(lon1)
    lon2 = radians(lon2)
    lat1 = radians(lat1)
    lat2 = radians(lat2)

    # Haversine formula
    dlon = lon2 - lon1
    dlat = lat2 - lat1
    a = sin(dlat / 2)**2 + cos(lat1) * cos(lat2) * sin(dlon / 2)**2

    c = 2 * asin(sqrt(a))

    # Radius of earth in kilometers. Use 3956 for miles
    r = 6371

    # calculate the result
    return(c * r)

# Convert a string to a CSM string.
def csm(s):
```

```
    return "\n".join(s.split(" "))

# Show a graph of nodes edges and huristics
def show_graph(nodes, edges, huristics, full_path):
    # Returns a dictionary containing the indexes of all nodes.
    d = defaultdict(int)
    for i,v  in enumerate(nodes.keys()):
        d[v] = i


    es = []
    costs = []
    vertex_color = []

    for e in edges:
        for i in edges[e]:
            es.append((d[e], d[i[0]]))
            costs.append(str(i[1])+" KM")

    # Plots a location graph.
    n_vertices = len(nodes)

    g = ig.Graph(n_vertices, es, directed=False)
    g["title"] = "Location Graph"
    g.vs["name"] = [csm(node+" H(n): "+str(round(huristics[node], 3))) for node in nodes.keys()]
    g.vs["huristic"] = [huristics[node] for node in huristics.keys()]
    g.es["label"] = costs
    fig, ax = plt.subplots(figsize=(20, 20))
    for i in range(len(g.es)):
        g.es[i]["label_size"] = 8

    for node in g.vs["name"]:
        node = node.split("H(n):")
        node = node[0].replace("\n", " ")
        node = node.strip()
        if goal_node in node:
            vertex_color.append("#6ab04c")
        elif start_node in node:
            vertex_color.append("#f0932b")
        elif node in full_path:
            vertex_color.append("#7ed6df")
        else:
            vertex_color.append("#dff9fb")
```

```
    edge_color = []
    # Plots a graph.
    ig.plot(
        g,
        target=ax,
        layout="graphopt",
        # bbox = (200, 200), # print nodes in a circular layout
        vertex_size=13,
        vertex_color=vertex_color,
        vertex_frame_width=4.0,
        vertex_frame_color="#686de0",
        vertex_label=g.vs["name"],
        vertex_label_size=9,
        edge_width=[1 for _ in range(len(edges))],
        edge_color='black',
        edge_label = g.es["label"],
        edge_lable_size = 5
    )
    # print(g.es)
    plt.show()

pc = lambda: print("-"*50)
s = lambda: sleep(0)

no_of_nodes, no_of_edges = map(int, input().split())

# Creates a dictionary of edges and nodes.
edges = defaultdict(list)
nodes = defaultdict(list)
heuristic = defaultdict(float)

start_node, goal_node = map(str.strip, input().split(" -> "))
pc()
print(f"Our start nodes is: {start_node}")
print(f"Our goal nodes is: {goal_node}")
pc()


# Split a string into a list of nodes and coordinates.
for i in range(no_of_nodes):
    node, coordinates = input().split(" = ")
    node = node.strip()
    coordinates = list(map(float, coordinates.split(",")))
```

```
            nodes[node] = coordinates

        #Building the heuristic value of each node

        for node in nodes:
            heuristic[node] = find_longitude_lattitude_distance(nodes[goal_node], nodes[node])
        pc()
        print("The co-ordinate values of the nodes are")
        [(s(), print(f"NODE: {k} ".ljust(40)+f"Longitude: {v[1]:.4f}, Lattitude {v[0]:.4f}".rjust(25))) for k,v in
nodes.items()]
        pc()
        print("The Heuristic values of the nodes are")
        [(s(), print(f"NODE: {k} ".ljust(40)+f"Heuristic: {v:.4f} KM".rjust(0))) for k,v in heuristic.items()]

        input()
        for q in range(no_of_edges):
            node1, node2, path_cost = map(str.strip, input().split(" -> "))
            path_cost = float(path_cost)
            edges[node1].append((node2, path_cost))



        #A* algorithm

        #First we start from the starting node.
        open_fringe = PriorityQueue()
        closed_fringe = defaultdict(list)

        # Finds a goal within a closed fringe.
        #we add [g(n)+h(n), g(n), node, full_path]
        open_fringe.put([heuristic[start_node]+0, 0, start_node, [start_node]])

        # Returns a tuple of nodes and directories in the closed_fringe.
        while not open_fringe.empty():
            f_n, g_n, node, full_path = open_fringe.get()
            # print(g_n, node, full_path)
            closed_fringe[node] = [f_n, g_n, full_path]
            print("Goin into the closed fringe ", node)
            if node == goal_node:
                print("Reached goal node! Stopping the iteration here")
                break
            for adj_node in edges[node]:
```

```
            if (adj_node[0] in closed_fringe and closed_fringe[adj_node[0]][0] >
g_n+heuristic[adj_node[0]]+adj_node[1]) or (adj_node[0] not in closed_fringe):
                #[fn, gn, node, full_path]
                cp = full_path[:]
                cp.append(adj_node[0])
                # print("goin into closed fringe ", node)
                print(cp)
                open_fringe.put([g_n+adj_node[1]+heuristic[adj_node[0]], g_n+adj_node[1],
adj_node[0], cp])


    print("The final path:" + "->".join(closed_fringe[goal_node][2]))
    print("Total path cost:", closed_fringe[goal_node][1], "KM")
    show_graph(nodes, edges, heuristic, closed_fringe[goal_node][2])
```

3.2 Input:

Number of nodes, edges.
Start node -> Goal Node.
*Nodes with coordinates.
*Edges with weights (distance).

10 15
Tejturi Bazar -> UAP
Tejturi Bazar = 23.757262, 90.392418
K.B. Rail Gate = 23.756152, 90.394041
Farmgate = 23.758343, 90.390466
The Daily Star = 23.755619, 90.390917
SAARC Fountain = 23.750060, 90.393046
Panthapath Signal = 23.751092, 90.387233
Karwan Bazar = 23.750377, 90.394330
Ananda Cinema Hall = 23.756876, 90.389878
Chourangi Mor = 23.755872, 90.392188
UAP = 23.755039, 90.389071

Tejturi Bazar -> K.B. Rail Gate -> 0.23
Tejturi Bazar -> Chourangi Mor -> 0.21
Tejturi Bazar -> Farmgate -> 0.29
K.B. Rail Gate -> Chourangi Mor -> 0.28

K.B. Rail Gate -> Karwan Bazar -> 0.75
Chourangi Mor -> Karwan Bazar -> 0.7
Chourangi Mor -> The Daily Star -> 0.24
Farmgate -> The Daily Star -> 0.4
Farmgate -> Ananda Cinema Hall -> 0.35
The Daily Star -> SAARC Fountain -> 0.9
The Daily Star -> UAP -> 0.27
SAARC Fountain -> Panthapath Signal -> 0.65
Panthapath Signal -> UAP -> 0.5
Karwan Bazar -> SAARC Fountain -> 0.22
Ananda Cinema Hall -> UAP -> 0.25

3.3 Output: (From terminal)

——————————————————————————

Our start nodes is: Tejturi Bazar
Our goal nodes is: UAP

```
(CSE 404 LAB) PS E:\prog\CSE 404 LAB> py .\A_STAR_ALGORITHM.py
--------------------------------------------------
Our start nodes is: Tejturi Bazar
Our goal nodes is: UAP
--------------------------------------------------
--------------------------------------------------
The co-ordinate values of the nodes are
NODE: Tejturi Bazar                     Longitude: 90.3924, Lattitude 23.7573
NODE: K.B. Rail Gate                    Longitude: 90.3940, Lattitude 23.7562
NODE: Farmgate                          Longitude: 90.3905, Lattitude 23.7583
NODE: The Daily Star                    Longitude: 90.3909, Lattitude 23.7556
NODE: SAARC Fountain                    Longitude: 90.3930, Lattitude 23.7501
NODE: Panthapath Signal                 Longitude: 90.3872, Lattitude 23.7511
NODE: Karwan Bazar                      Longitude: 90.3943, Lattitude 23.7504
NODE: Ananda Cinema Hall                Longitude: 90.3899, Lattitude 23.7569
NODE: Chourangi Mor                     Longitude: 90.3922, Lattitude 23.7559
NODE: UAP                               Longitude: 90.3891, Lattitude 23.7550
--------------------------------------------------
The Heuristic values of the nodes are
NODE: Tejturi Bazar                     Heuristic: 0.4209 KM
NODE: K.B. Rail Gate                    Heuristic: 0.5207 KM
NODE: Farmgate                          Heuristic: 0.3939 KM
NODE: The Daily Star                    Heuristic: 0.1986 KM
NODE: SAARC Fountain                    Heuristic: 0.6857 KM
NODE: Panthapath Signal                 Heuristic: 0.4771 KM
NODE: Karwan Bazar                      Heuristic: 0.7451 KM
NODE: Ananda Cinema Hall                Heuristic: 0.2202 KM
NODE: Chourangi Mor                     Heuristic: 0.3305 KM
NODE: UAP                               Heuristic: 0.0000 KM
Goin into the closed fringe  Tejturi Bazar
['Tejturi Bazar', 'K.B. Rail Gate']
['Tejturi Bazar', 'Chourangi Mor']
['Tejturi Bazar', 'Farmgate']
Goin into the closed fringe  Chourangi Mor
['Tejturi Bazar', 'Chourangi Mor', 'Karwan Bazar']
['Tejturi Bazar', 'Chourangi Mor', 'The Daily Star']
Goin into the closed fringe  The Daily Star
['Tejturi Bazar', 'Chourangi Mor', 'The Daily Star', 'SAARC Fountain']
['Tejturi Bazar', 'Chourangi Mor', 'The Daily Star', 'UAP']
Goin into the closed fringe  Farmgate
```

```
--------------------------------------------------
The Heuristic values of the nodes are
NODE: Tejturi Bazar                     Heuristic: 0.4209 KM
NODE: K.B. Rail Gate                    Heuristic: 0.5207 KM
NODE: Farmgate                          Heuristic: 0.3939 KM
NODE: The Daily Star                    Heuristic: 0.1986 KM
NODE: SAARC Fountain                    Heuristic: 0.6857 KM
NODE: Panthapath Signal                 Heuristic: 0.4771 KM
NODE: Karwan Bazar                      Heuristic: 0.7451 KM
NODE: Ananda Cinema Hall                Heuristic: 0.2202 KM
NODE: Chourangi Mor                     Heuristic: 0.3305 KM
NODE: UAP                               Heuristic: 0.0000 KM
Goin into the closed fringe  Tejturi Bazar
['Tejturi Bazar', 'K.B. Rail Gate']
['Tejturi Bazar', 'Chourangi Mor']
['Tejturi Bazar', 'Farmgate']
Goin into the closed fringe  Chourangi Mor
['Tejturi Bazar', 'Chourangi Mor', 'Karwan Bazar']
['Tejturi Bazar', 'Chourangi Mor', 'The Daily Star']
Goin into the closed fringe  The Daily Star
['Tejturi Bazar', 'Chourangi Mor', 'The Daily Star', 'SAARC Fountain']
['Tejturi Bazar', 'Chourangi Mor', 'The Daily Star', 'UAP']
Goin into the closed fringe  Farmgate
['Tejturi Bazar', 'Farmgate', 'Ananda Cinema Hall']
Goin into the closed fringe  UAP
--------------------------------------------------
Reached goal node! Stopping the iteration here
--------------------------------------------------
The final path:Tejturi Bazar->Chourangi Mor->The Daily Star->UAP
--------------------------------------------------
Total path cost: 0.72 KM
--------------------------------------------------
```

The output is very close (+-1%) of the shortest path returned by the google map based on the distance.

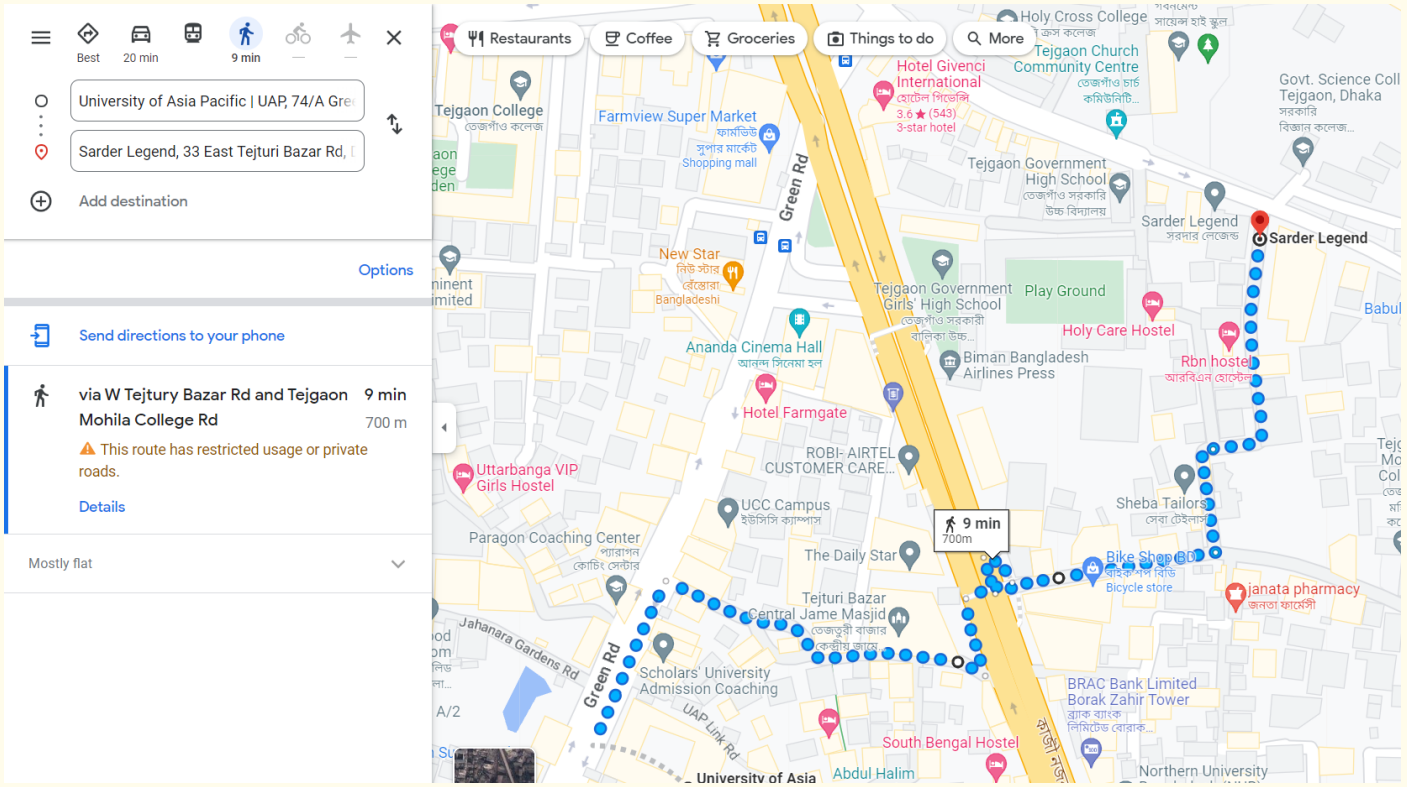

Figure 4: Shortest path by google maps.

4. Conclusion:

The A* algorithm is a very efficient pathfinding algorithm that is widely used in computer science and game development. It is a combination of two other algorithms, breadth-first search and the greedy best-first search, and it uses heuristics to find the optimal path in a graph or a network. The A* algorithm is especially useful in situations where finding the shortest path is of utmost importance, such as in maps, GPS systems, and robotics. Overall, the A* algorithm is a powerful and versatile tool that is essential for solving many real-world problems in computer science and beyond.

5. Citations:

- *"The Haversine Formula," by E. W. Weisstein (MathWorld, 2003)*
- *"The Haversine Formula Revisited," by T. Vincenty (1985)*
- *"The Haversine Formula in JavaScript," by R. Finch (2008)*
- *"A Fast Haversine Algorithm for Short-Range Great-Circle Navigation," by R. W. Sinnott (1984)*
- *Peter Hart, Nils Nilsson, and Bertram Raphael. "A formal basis for the heuristic determination of minimum cost paths." IEEE Transactions on Systems Science and Cybernetics, vol. 4, no. 2, pp. 100-107, July 1968.*

- *Manuela M. Veloso, Ronald A. Brachman, and Paul K. Allen. "A real-time heuristic search algorithm." Artificial Intelligence, vol. 25, no. 1, pp. 97-109, 1985.*

- *Subhrajit Bhattacharya, Jiebo Luo, and Amit K. Roy-Chowdhury. "A fast A* algorithm for video object segmentation." IEEE Transactions on Circuits and Systems for Video Technology, vol. 23, no. 9, pp. 1582-1591, Sept. 2013.*