Department of Computer Science and Engineering

Project: 01

Course Title – Artificial Intelligence and Expert Systems Lab
Course Code- CSE 404
Year- 4th
Semester- 1st

Submitted to:

Dr. Nasima Begum
Associate Professor
University of Asia Pacific

Submitted by:

Name- Md. Abdur Rashid
Section- A1
ID- 19101008

Project Name: Implementation of a small address map (from my own home to UAP) using A* Search Algorithm.

Introduction:

The project problem is to implementation of a small address map from my own home to UAP, using A* search algorithm and then find out the optimal path.

A* algorithm is a searching algorithm that searches for the shortest path between the initial state to the final state. Here, I will find the most optimal path from my home (Mugdapara) to my university (UAP) using A* search algorithm.

Objective:

The objective of this project is to find an optimal path from my home (Mugdapara) to my university (UAP).

Tools And Languages:

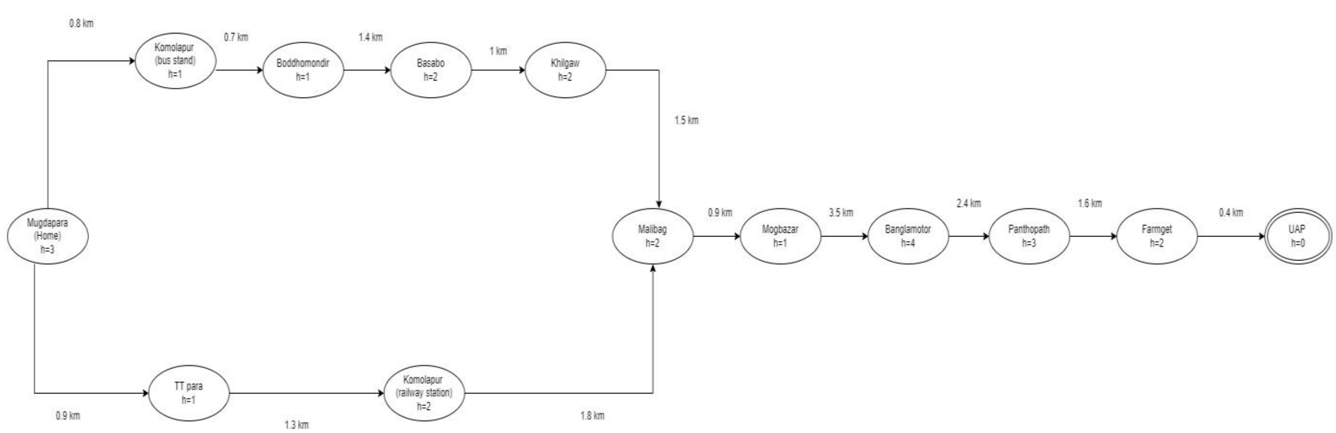
Distance Measurement: Google Maps

Map Designing: app diagram dot net

IDE: PyCharm

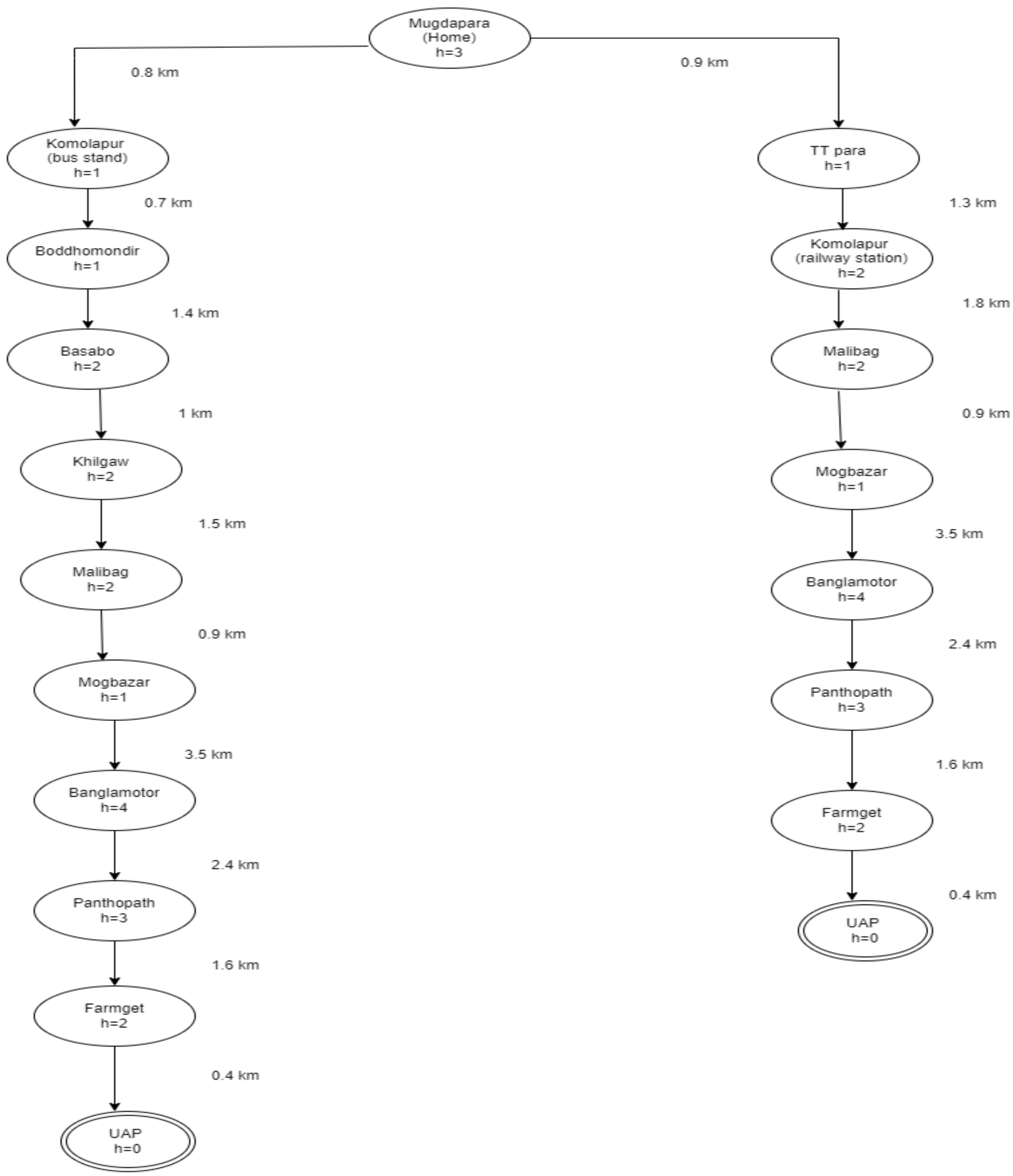
Programing Language: Python

Designed Map:

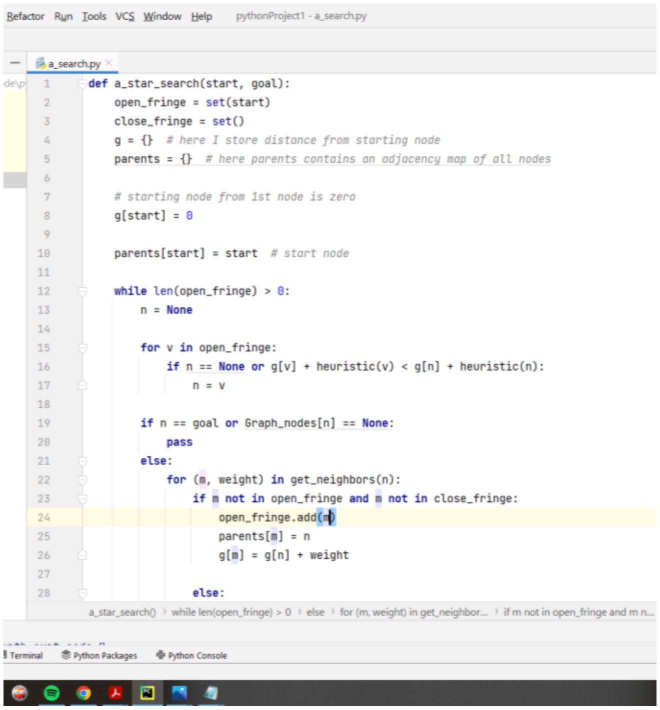


Here,
Start node: Mugdapara
Goal node: UAP
Cost in distance: Kilometer (km)

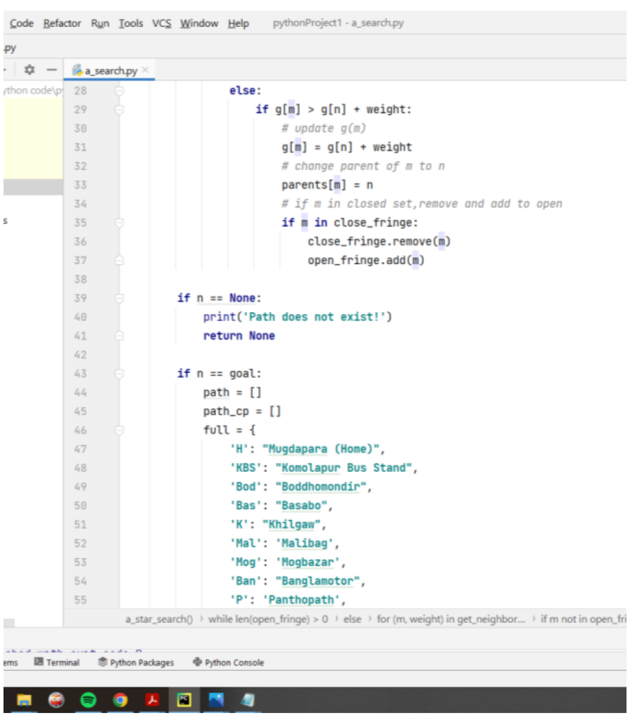
Search tree of designed map:



Implemented code using python:



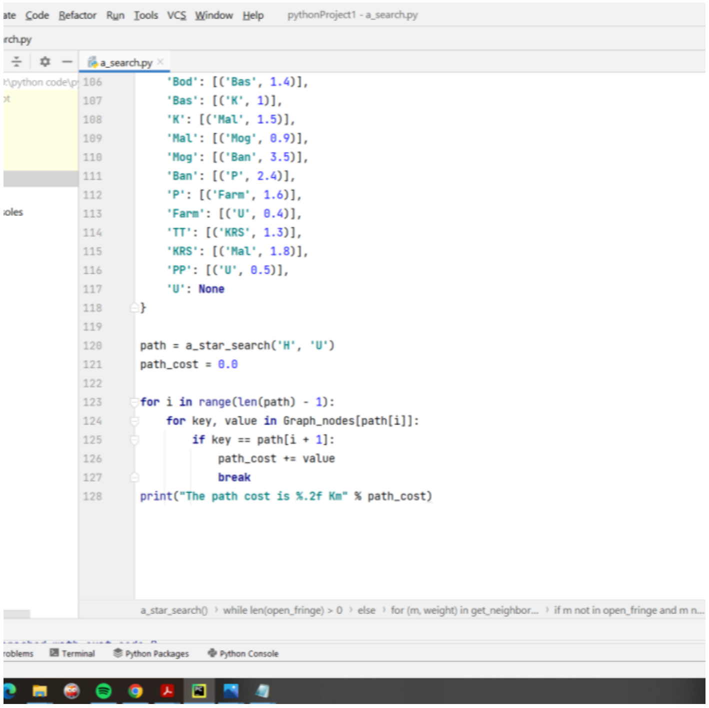
```python
def a_star_search(start, goal):
    open_fringe = set(start)
    close_fringe = set()
    g = {}  # here I store distance from starting node
    parents = {}  # here parents contains an adjacency map of all nodes

    # starting node from 1st node is zero
    g[start] = 0

    parents[start] = start  # start node

    while len(open_fringe) > 0:
        n = None

        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):
                if m not in open_fringe and m not in close_fringe:
                    open_fringe.add(m)
                    parents[m] = n
                    g[m] = g[n] + weight

                else:
```



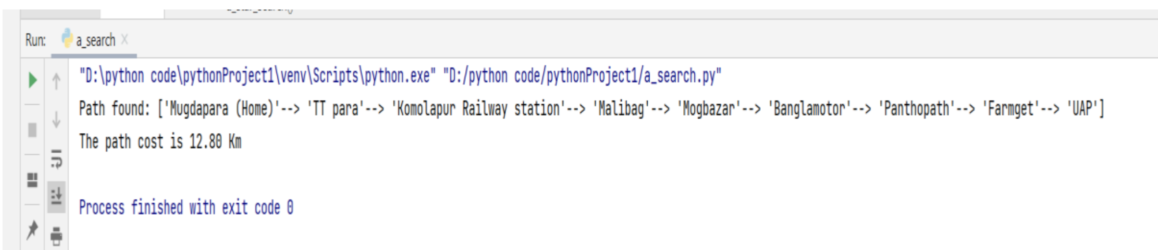
```python
                else:
                    if g[m] > g[n] + weight:
                        # update g(m)
                        g[m] = g[n] + weight
                        # change parent of m to n
                        parents[m] = n
                        # if m in closed set,remove 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 n == goal:
            path = []
            path_cp = []
            full = {
                'H': "Mugdapara (Home)",
                'KBS': "Komolapur Bus Stand",
                'Bod': "Boddhomondir",
                'Bas': "Basabo",
                'K': "Khilgaw",
                'Mal': 'Malibag',
                'Mog': 'Mogbazar',
                'Ban': "Banglamotor",
                'P': 'Panthopath',
```

```python
                'P': 'Panthopath',
                'Farm': 'Farmget',
                'TT': 'TT para',
                'KRS': "Komolapur Railway station",
                '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(",", "-->")))
            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:
```

```python
    else:
        return None

def heuristic(n):
    H_dist = {
        'H': 3,
        'KBS': 1,
        'Bod': 1,
        'Bas': 2,
        'K': 2,
        'Mal': 2,
        'Mog': 1,
        'Ban': 4,
        'P': 3,
        'Farm': 2,
        'TT': 1,
        'KRS': 2,
        'U': 0
    }
    return H_dist[n]

Graph_nodes = {
    'H': [('KBS', 0.8), ('TT', 0.9)],
    'KBS': [('Bod', 0.7)],
    'Bod': [('Bas', 1.4)],
    'Bas': [('K', 1)],
    'K': [('Mal', 1.5)],
    'Mal': [('Mog', 0.9)],
```



Output:



Output Analysis:

After Using A* Search Algorithm on this designed map, on output we can find the shortest path:

Mugdapara (Home) --> TT para --> Komolapur Railway station --> Malibag --> Mogbazar --> Banglamotor --> Panthopath --> Farmget --> UAP.

Path cost: (0.9+1.3+1.8+0.9+3.5+2.4+1.6+0.4) km = 12.80 km

Conclusion:

In this project, after successful implementation, A* search algorithm gives the most optimal path as output. So we can use this algorithm for approximate the shortest path in real-life situation, like - in maps, games etc.