University of Asia Pacific

Department of Computer Science & Engineering

Course Title: Artificial Intelligence and Expert System Lab

Course Code : CSE 404

A* Algorithm Project#1 Report

Submitted By
Sumaiya Akter
Department of CSE
ID: 18201056
Section: B1

Submitted To
Dr. Nasima Begum
Associate Professor,
Department of CSE

A* SEARCH ALGORITHM

PROBLEM STATEMENT:

As per the discussion in class, please create your own address map from your home to University of Asia Pacific (UAP) and write in any programming language to implement the A* search algorithm to reach the destination from the starting point.

TOOLS:

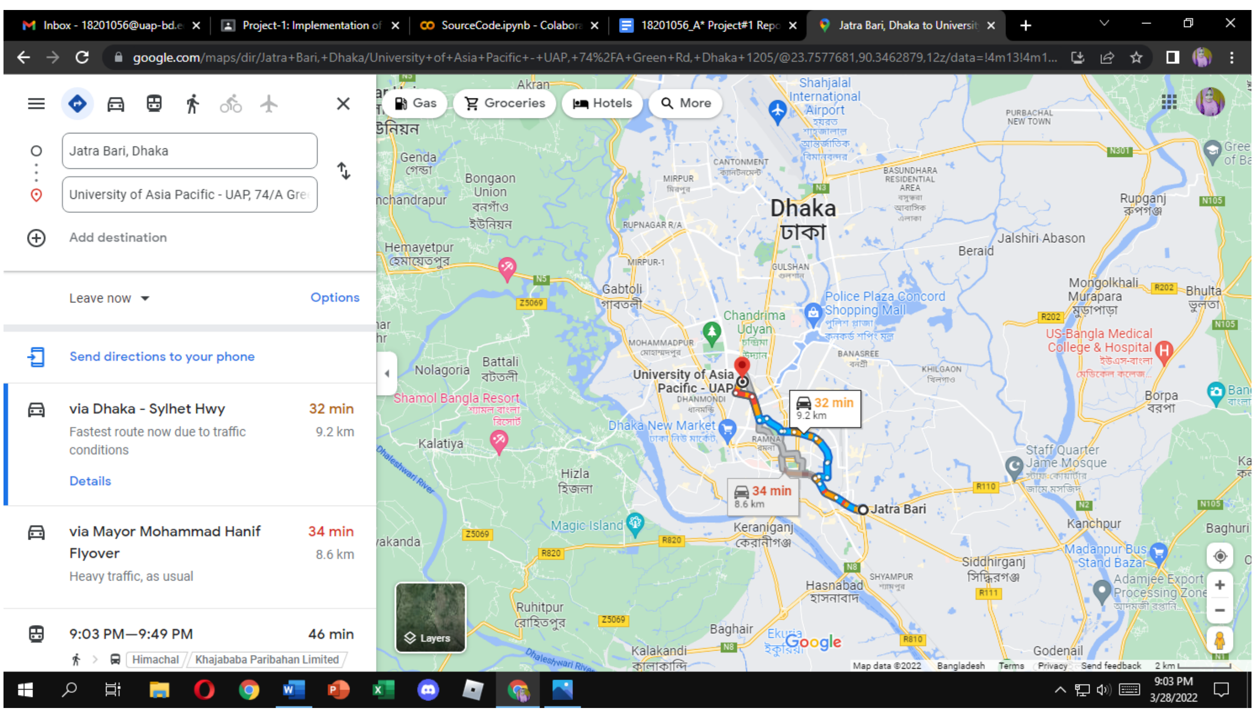
Google Maps, Google Colaboratory, Python Programming Language.

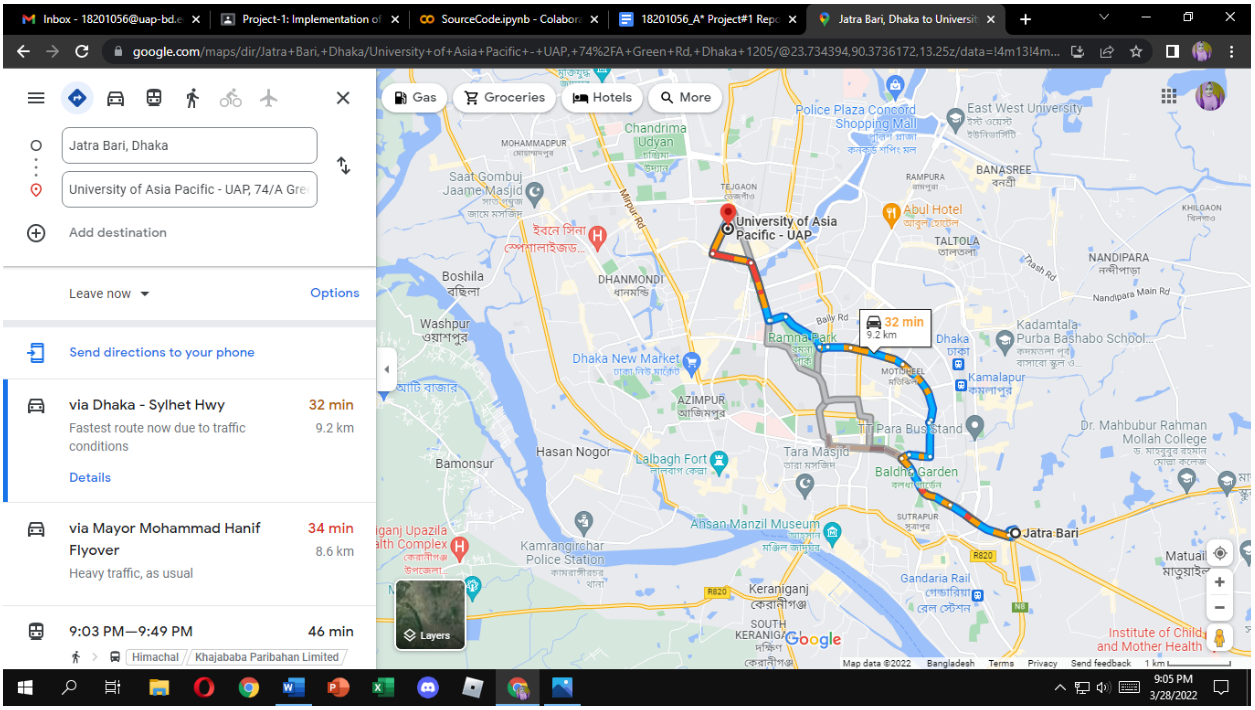
MAPS:

Home: Jatrabari

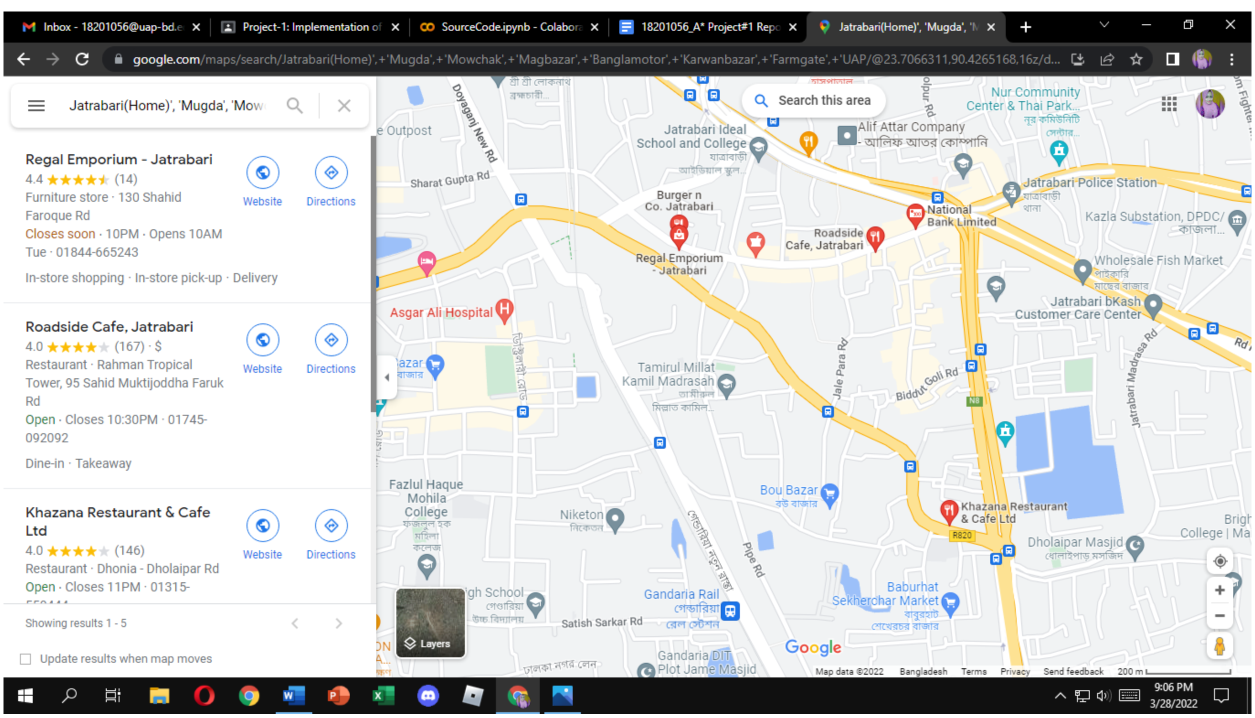
Destination: UAP

Fig-1 :

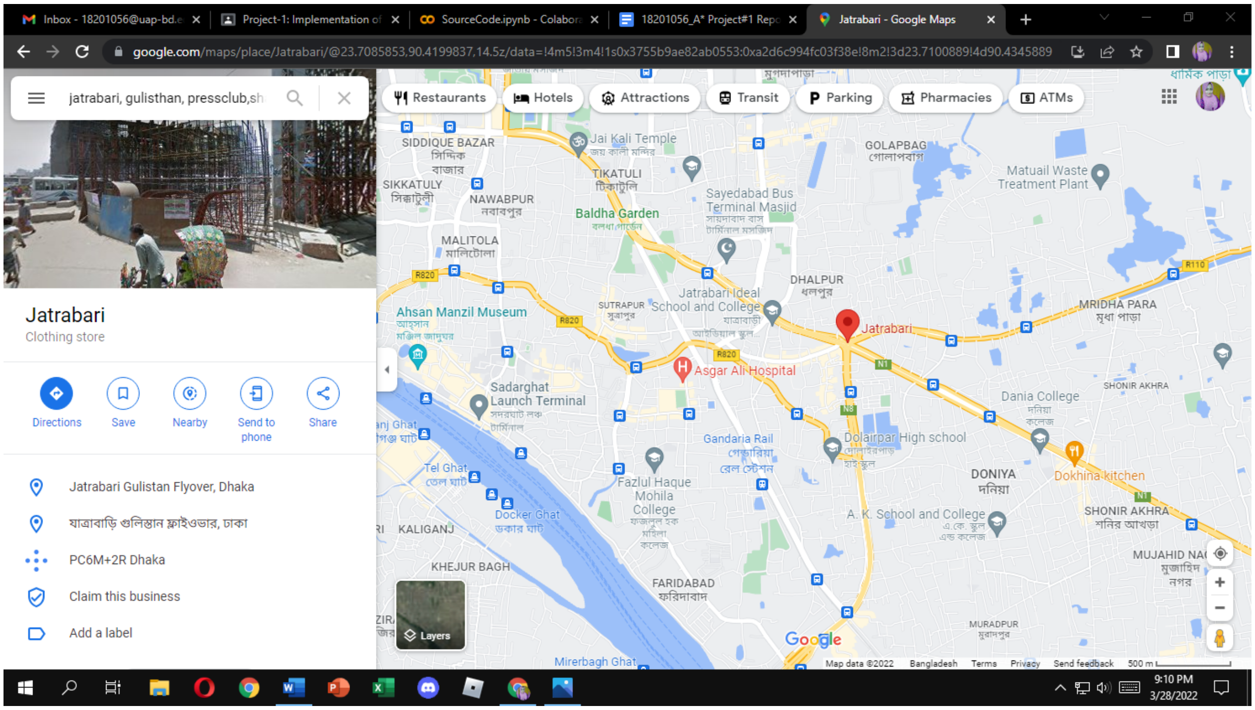




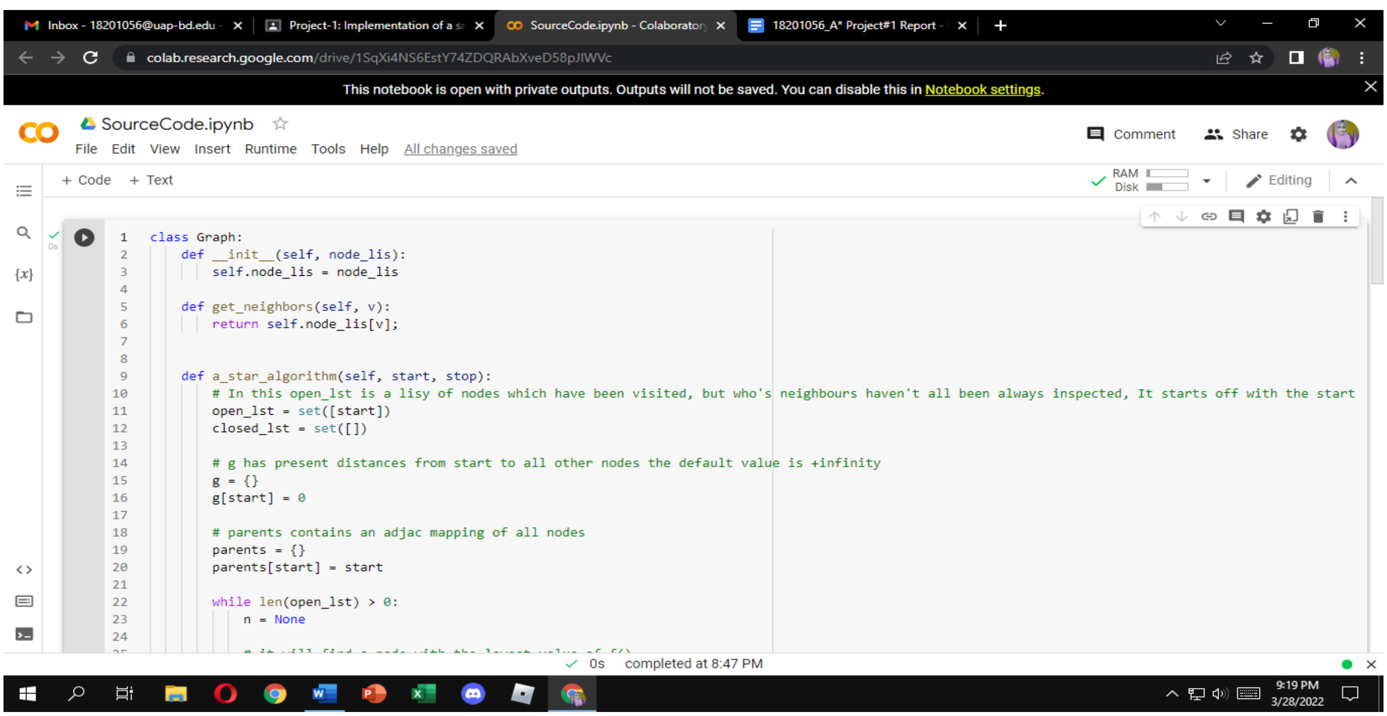
Jatrabari(Home)> Mugdha > Mowchak > Moghbazar > Banglamotor > Karwan Bazar > Farmgate > UAP(Destination)



Jatrabari > Gulistan > Press Club > shahbag > Banglamotor > karwan bazar > Farmgate > UAP



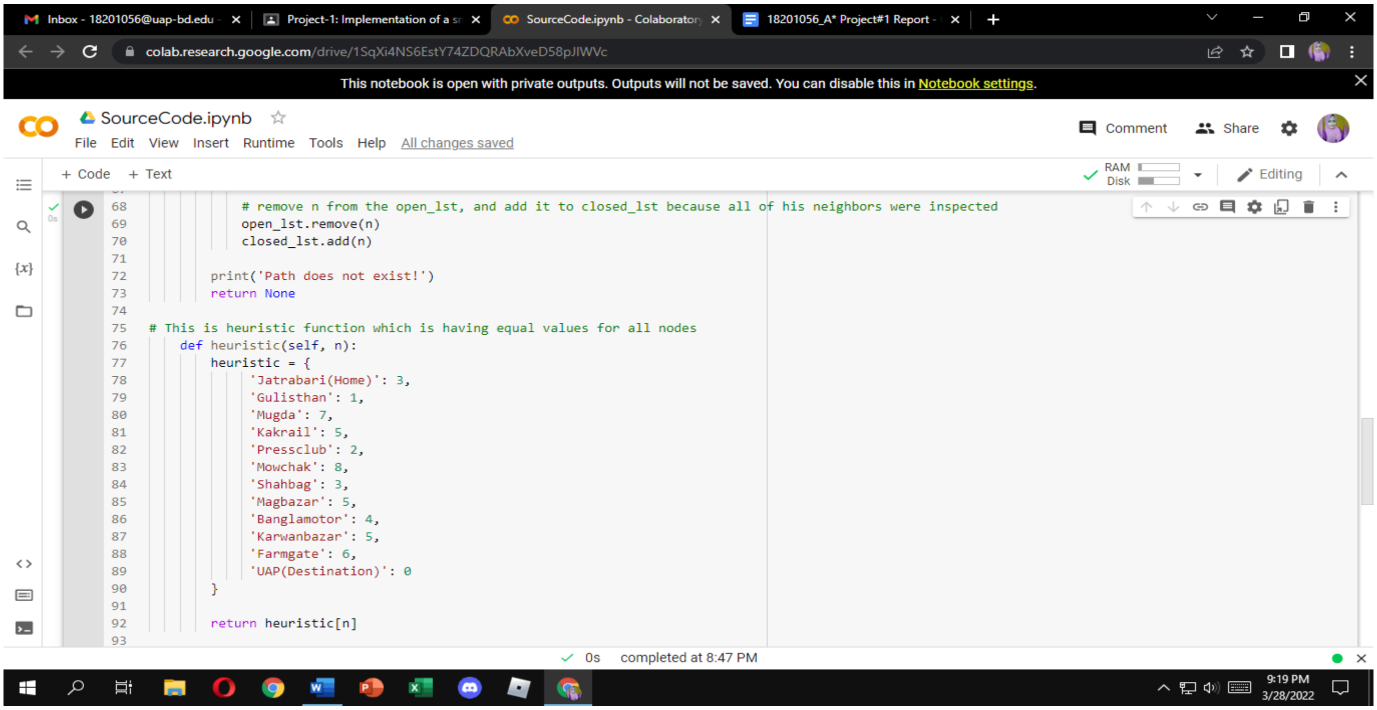
<u>CODE SCREENSHOTS:</u>



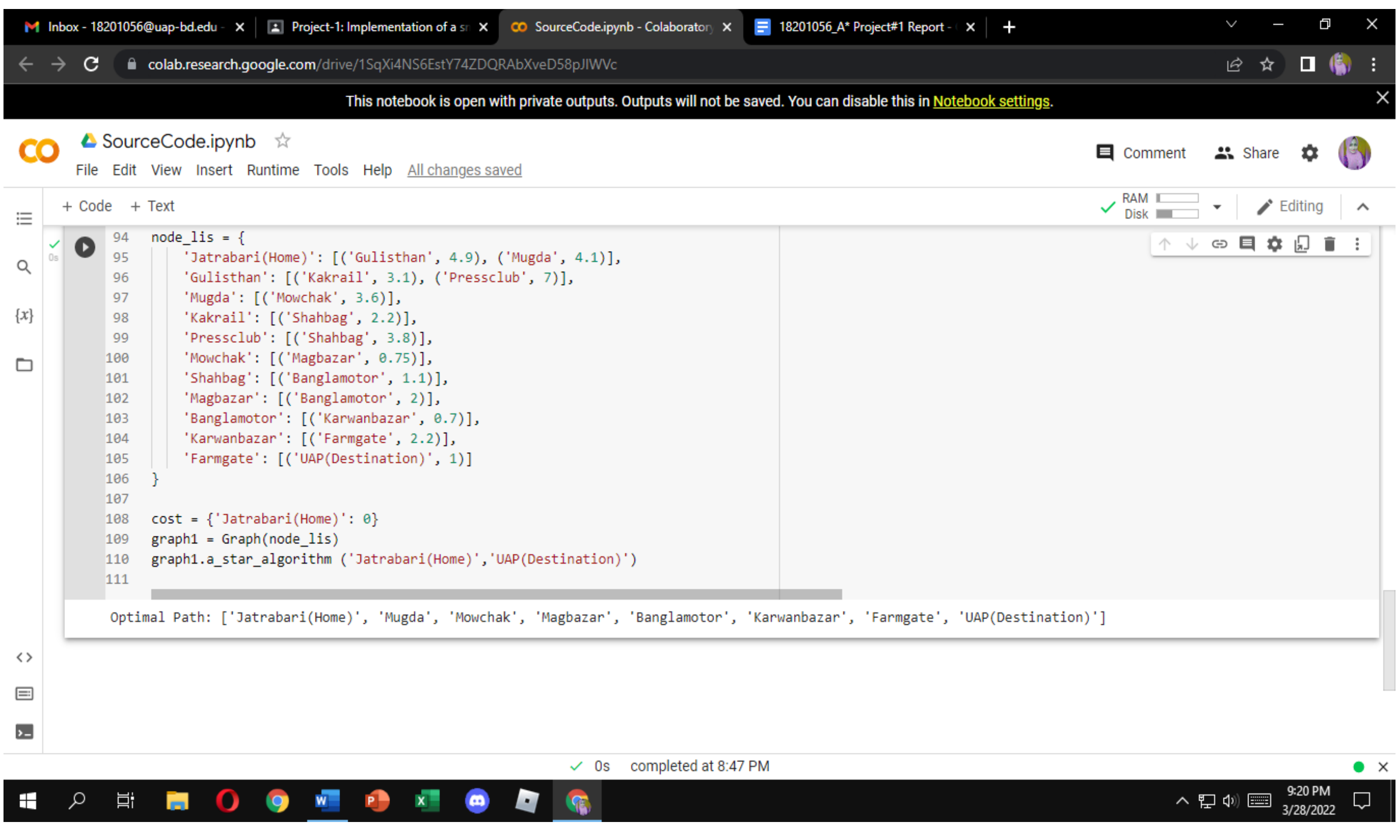

```python
class Graph:
    def __init__(self, node_lis):
        self.node_lis = node_lis

    def get_neighbors(self, v):
        return self.node_lis[v];


    def a_star_algorithm(self, start, stop):
        # In this open_lst is a lisy of nodes which have been visited, but who's neighbours haven't all been always inspected, It starts off with the start
        open_lst = set([start])
        closed_lst = set([])

        # g has present distances from start to all other nodes the default value is +infinity
        g = {}
        g[start] = 0

        # parents contains an adjac mapping of all nodes
        parents = {}
        parents[start] = start

        while len(open_lst) > 0:
            n = None

```

```python
        parents[start] = start

        while len(open_lst) > 0:
            n = None

            # it will find a node with the lowest value of f() -
            for v in open_lst:
                if n == None or g[v] + self.heuristic(v) < g[n] + self.heuristic(n):
                    n = v;

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

            # if the current node is the stop then we start again from start
            if n == stop:
                reconst_path = []

                while parents[n] != n:
                    reconst_path.append(n)
                    n = parents[n]

                reconst_path.append(start)

                reconst_path.reverse()

```

```python
                reconst_path.append(start)

                reconst_path.reverse()

                print('Optimal Path: {}'.format(reconst_path))
                #print('Optimal Path cost: 25 km')
                return

            # for all the neighbors of the current node do
            for (m, cost) in self.get_neighbors(n):
              # if the current node is not presentin both open_lst and closed_lst add it to open_lst and note n as it's path find the path cost
                if m not in open_lst and m not in closed_lst:
                    open_lst.add(m)
                    parents[m] = n
                    g[m] = g[n] + cost

                # otherwise, check if it's quicker to first visit n, then m and if it is, update parents data and poo data and if the node was in the closed
                else:
                    if g[m] > g[n] + cost:
                        g[m] = g[n] + cost
                        parents[m] = n

                        if m in closed_lst:
                            closed_lst.remove(m)
                            open_lst.add(m)
```



```python
                # remove n from the open_lst, and add it to closed_lst because all of his neighbors were inspected
                open_lst.remove(n)
                closed_lst.add(n)

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

    # This is heuristic function which is having equal values for all nodes
        def heuristic(self, n):
            heuristic = {
                'Jatrabari(Home)': 3,
                'Gulisthan': 1,
                'Mugda': 7,
                'Kakrail': 5,
                'Pressclub': 2,
                'Mowchak': 8,
                'Shahbag': 3,
                'Magbazar': 5,
                'Banglamotor': 4,
                'Karwanbazar': 5,
                'Farmgate': 6,
                'UAP(Destination)': 0
            }

            return heuristic[n]
```



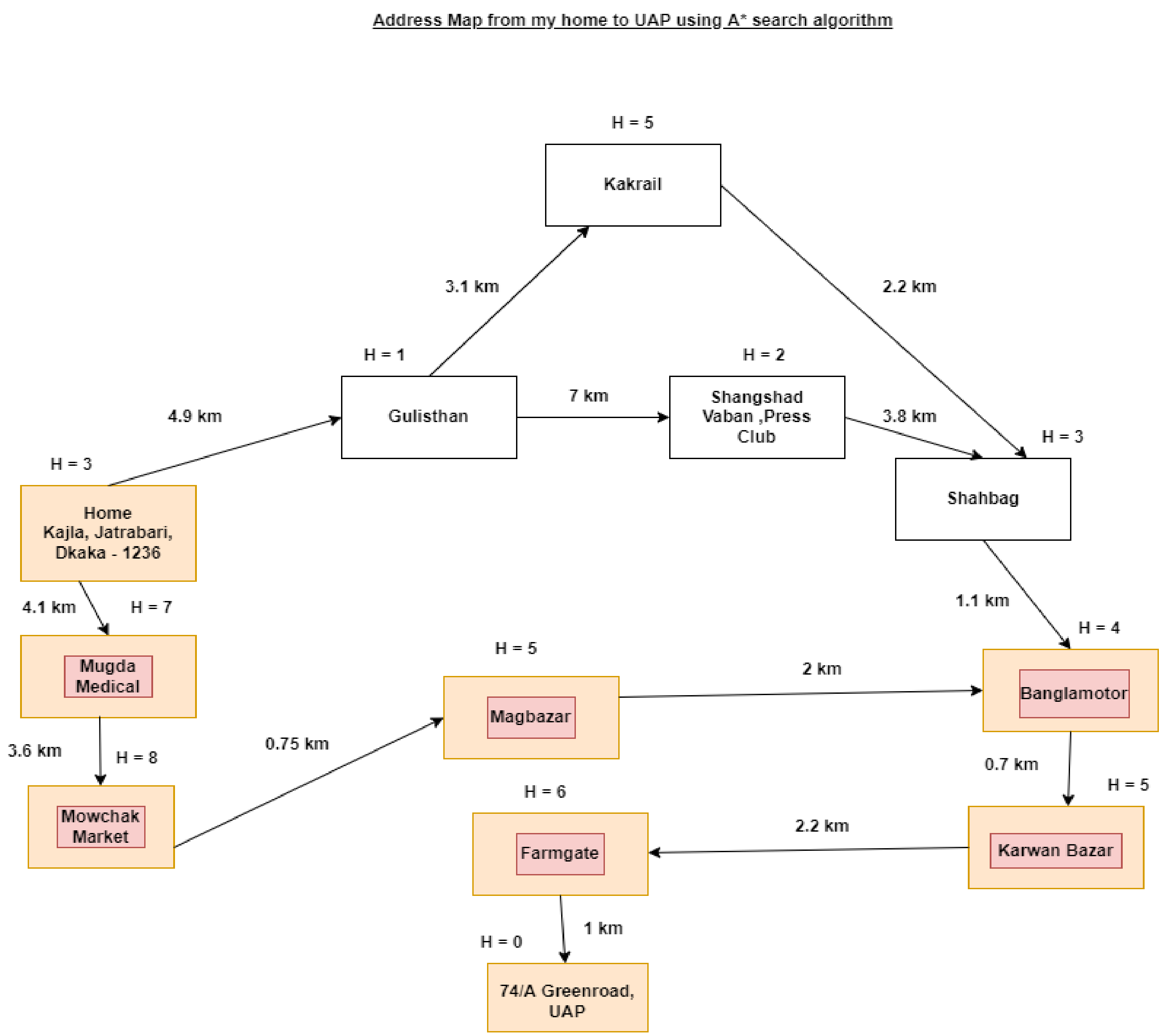
```python
node_lis = {
    'Jatrabari(Home)': [('Gulisthan', 4.9), ('Mugda', 4.1)],
    'Gulisthan': [('Kakrail', 3.1), ('Pressclub', 7)],
    'Mugda': [('Mowchak', 3.6)],
    'Kakrail': [('Shahbag', 2.2)],
    'Pressclub': [('Shahbag', 3.8)],
    'Mowchak': [('Magbazar', 0.75)],
    'Shahbag': [('Banglamotor', 1.1)],
    'Magbazar': [('Banglamotor', 2)],
    'Banglamotor': [('Karwanbazar', 0.7)],
    'Karwanbazar': [('Farmgate', 2.2)],
    'Farmgate': [('UAP(Destination)', 1)]
}

cost = {'Jatrabari(Home)': 0}
graph1 = Graph(node_lis)
graph1.a_star_algorithm ('Jatrabari(Home)','UAP(Destination)')
```

```
Optimal Path: ['Jatrabari(Home)', 'Mugda', 'Mowchak', 'Magbazar', 'Banglamotor', 'Karwanbazar', 'Farmgate', 'UAP(Destination)']
```

18201056 (Map):

Optimal Path Cost: 52.35km

Optimal Path: **Jatrabari(Home)> Mugdha > Mowchak > Moghbazar > Banglamotor > Karwan Bazar > Farmgate > UAP(Destination)**



Heuristic Values:

h(home) = 3

h(mugda) = 7

h(gulisthan) = 1

h(kakrail) = 5

h(shahbag) = 3

h(mowchak) = 8

h(magbazar) = 5

h(banglamotor) = 4

h(karwan bazar) = 5

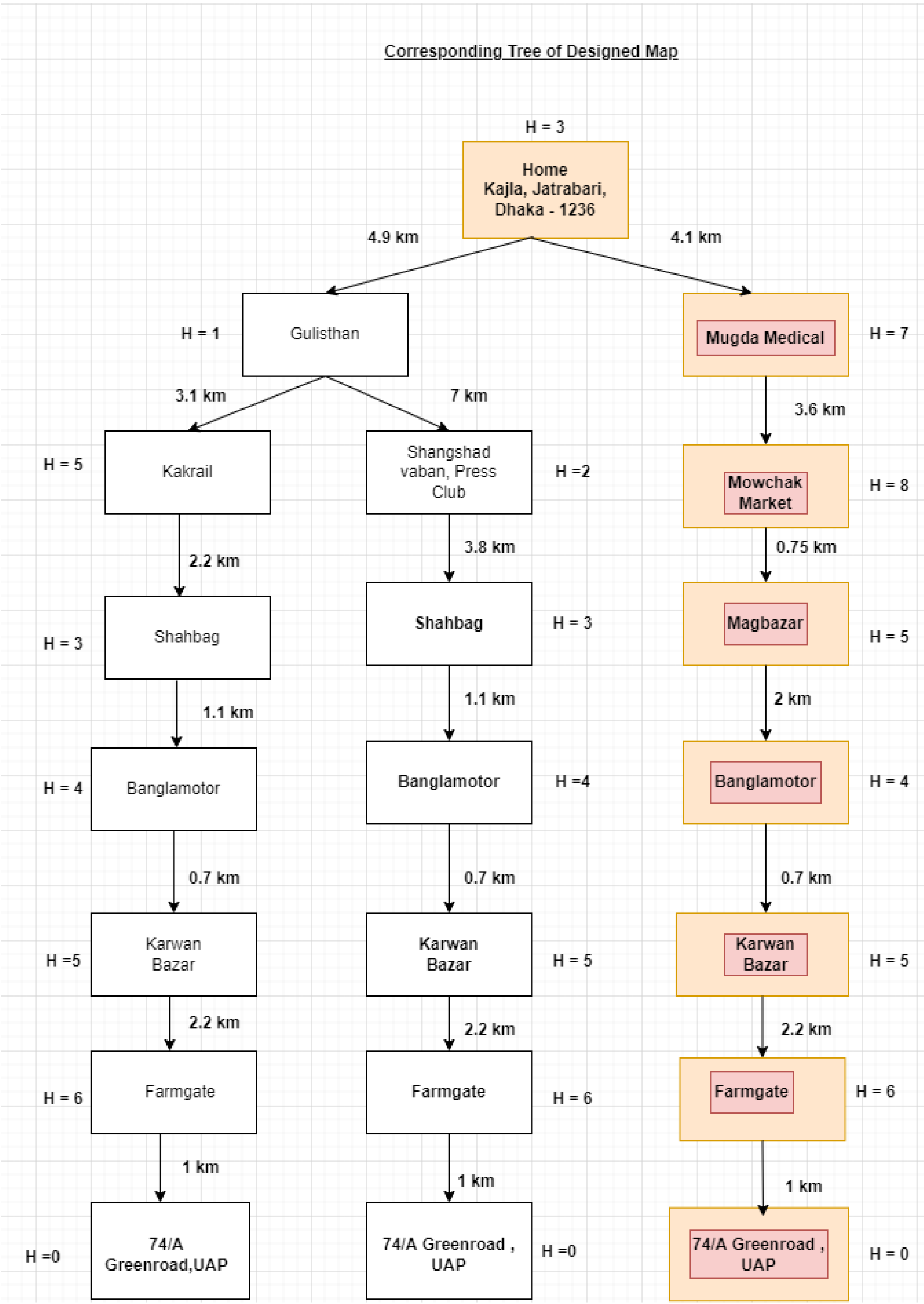
h(farmgate) = 6

h(pressclub) = 2

h(destination) = 0

Search Tree: 18201056

Here my initial state is Jatrabari which is home & goal state is UAP which is destination.



Corresponding Tree of Designed Map
H = 3
Home
Kajla, Jatrabari,
Dhaka - 1236
4.9 km
4.1 km
H = 1
Gulisthan
Mugda Medical
H = 7
3.1 km
7 km
3.6 km
H = 5
Kakrail
Shangshad
vaban, Press
Club
H =2
Mowchak
Market
H = 8
3.8 km
0.75 km
2.2 km
Shahbag
H = 3
Magbazar
H = 5
H = 3
Shahbag
1.1 km
2 km
1.1 km
Banglamotor
H =4
Banglamotor
H = 4
H = 4
Banglamotor
0.7 km
0.7 km
0.7 km
H =5
Karwan
Bazar
Karwan
Bazar
H = 5
Karwan
Bazar
H = 5
2.2 km
2.2 km
2.2 km
Farmgate
H = 6
H = 6
Farmgate
Farmgate
H = 6
1 km
1 km
1 km
H =0
74/A
Greenroad,UAP
74/A Greenroad ,
UAP
H =0
74/A Greenroad ,
UAP
H = 0

CODE :

```
class Graph:

    def __init__(self, node_lis):

        self.node_lis = node_lis


    def get_neighbors(self, v):

        return self.node_lis[v];




    def a_star_algorithm(self, start, stop):

        # In this open_lst is a list of nodes which have been visited, but
who's neighbors haven't all been always inspected, It starts off with the
start node. And closed_lst is a list of nodes which have been visited and
who's neighbors have been always inspected

        open_lst = set([start])

        closed_lst = set([])


        # g has present distances from start to all other nodes the
default value is +infinity

        g = {}

        g[start] = 0


        # parents contains an adjac mapping of all nodes

        parents = {}
```

```
        parents[start] = start

        while len(open_lst) > 0:
            n = None

            # it will find a node with the lowest value of f() -
            for v in open_lst:
                if n == None or g[v] + self.heuristic(v) < g[n] + self.heuristic(n):
                    n = v;

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

            # if the current node is the stop then we start again from start
            if n == stop:
                reconst_path = []

                while parents[n] != n:
                    reconst_path.append(n)
                    n = parents[n]
```

```
                reconst_path.append(start)

                reconst_path.reverse()

                print('Optimal Path: {}'.format(reconst_path))
                #print('Optimal Path cost: 25 km')
                return

            # for all the neighbors of the current node do
            for (m, cost) in self.get_neighbors(n):
              # if the current node is not presentin both open_lst and closed_lst add it to open_lst and note n as it's path find the path cost
                if m not in open_lst and m not in closed_lst:
                    open_lst.add(m)
                    parents[m] = n
                    g[m] = g[n] + cost

                # otherwise, check if it's quicker to first visit n, then m and if it is, update parents data and poo data and if the node was in the closed_lst, move it to open_lst
                else:
                    if g[m] > g[n] + cost:
                        g[m] = g[n] + cost
                        parents[m] = n
```

```
                        if m in closed_lst:

                            closed_lst.remove(m)

                            open_lst.add(m)


            # remove n from the open_lst, and add it to closed_lst because all of his neighbors were inspected

            open_lst.remove(n)

            closed_lst.add(n)


        print('Path does not exist!')

        return None


# This is heuristic function which is having equal values for all nodes

    def heuristic(self, n):

        heuristic = {

             'Jatrabari(Home)': 3,

             'Gulisthan': 1,

             'Mugda': 7,

             'Kakrail': 5,

             'Pressclub': 2,

             'Mowchak': 8,

             'Shahbag': 3,

             'Magbazar': 5,
```

```
            'Banglamotor': 4,
            'Karwan Bazar': 5,
            'Farmgate': 6,
            'UAP(Destination)': 0
        }

        return heuristic[n]

node_lis = {
    'Jatrabari(Home)': [('Gulisthan', 4.9), ('Mugda', 4.1)],
    'Gulisthan': [('Kakrail', 3.1), ('Pressclub', 7)],
    'Mugda': [('Mowchak', 3.6)],
    'Kakrail': [('Shahbag', 2.2)],
    'Pressclub': [('Shahbag', 3.8)],
    'Mowchak': [('Magbazar', 0.75)],
    'Shahbag': [('Banglamotor', 1.1)],
    'Magbazar': [('Banglamotor', 2)],
    'Banglamotor': [('Karwan Bazar', 0.7)],
    'Karwan Bazar': [('Farmgate', 2.2)],
    'Farmgate': [('UAP(Destination)', 1)]
}

cost = {'Jatrabari(Home)': 0}
```

```
graph1 = Graph(node_lis)

graph1.a_star_algorithm ('Jatrabari(Home)','UAP(Destination)')
```

******THE END*****