



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:

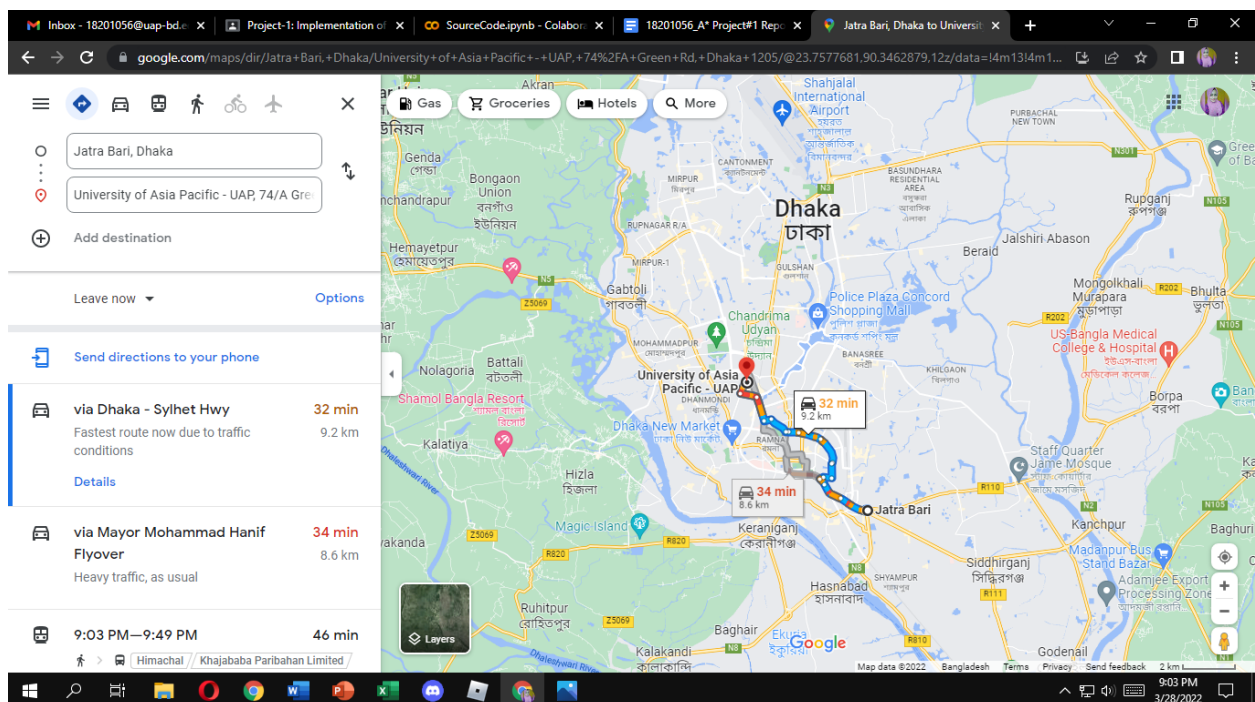
Google Maps, Google Colaboratory, Python Programming Language.

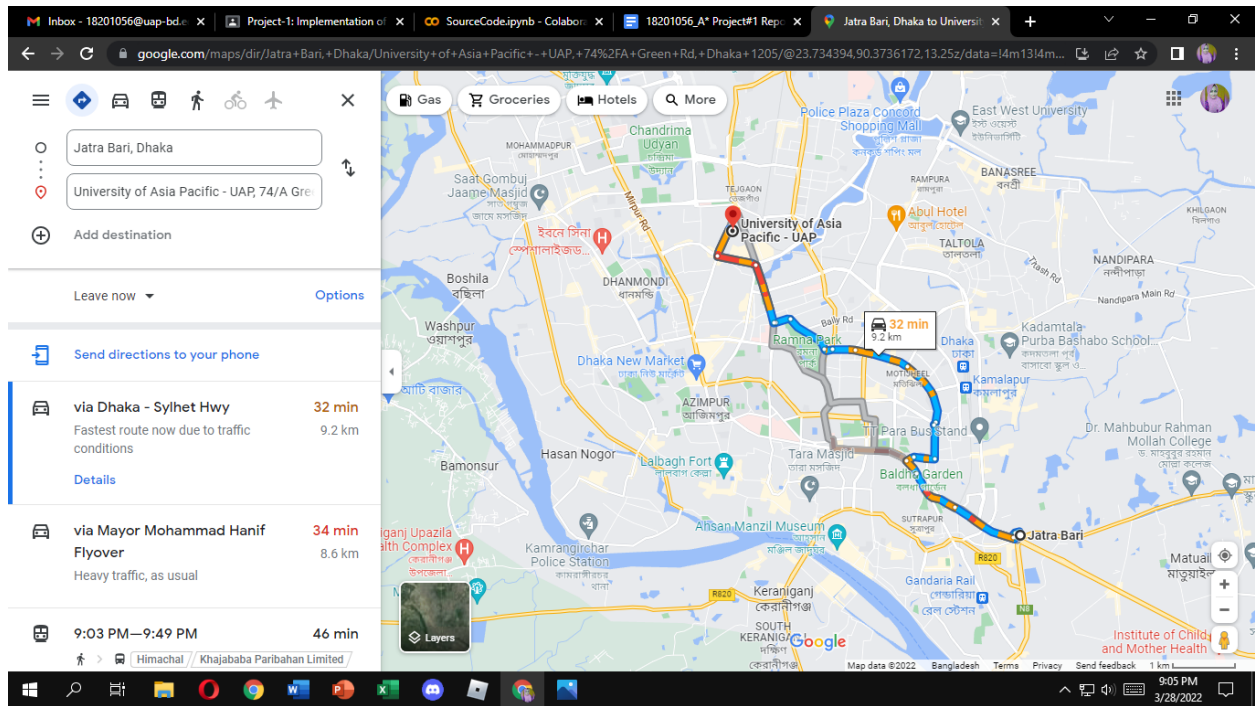
MAPS:

Home: Jatrabari

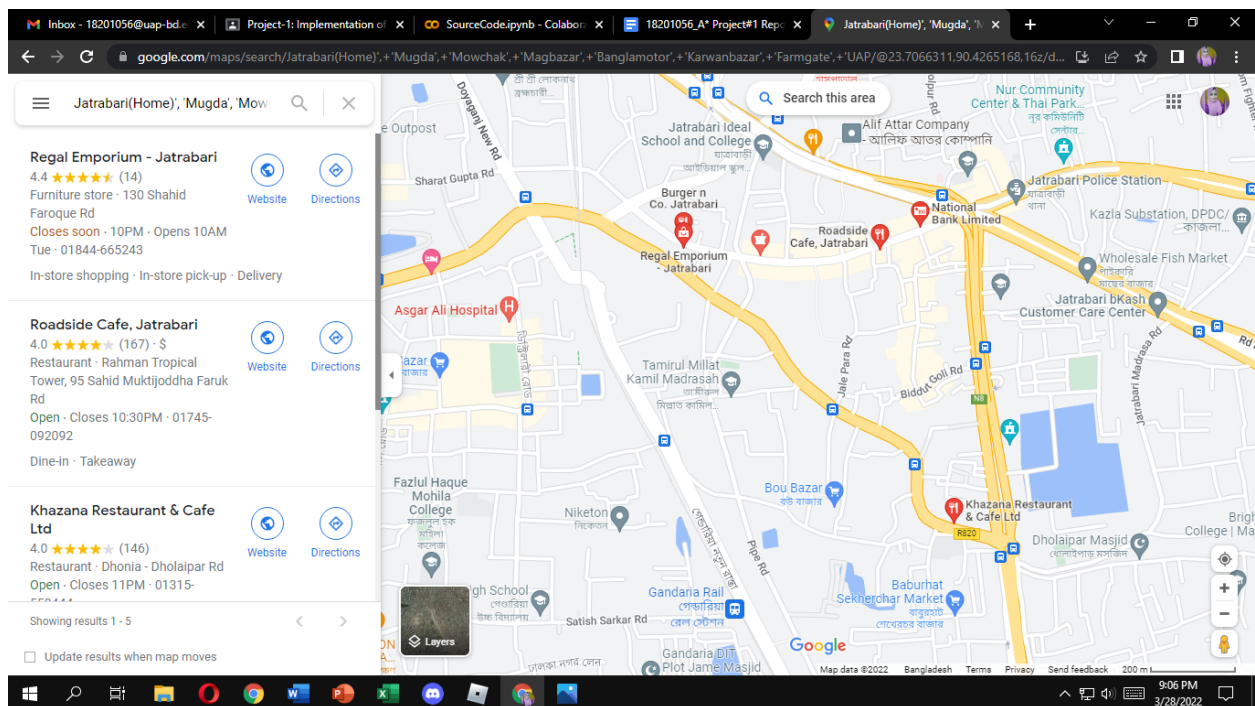
Destination: UAP

Fig-1 :

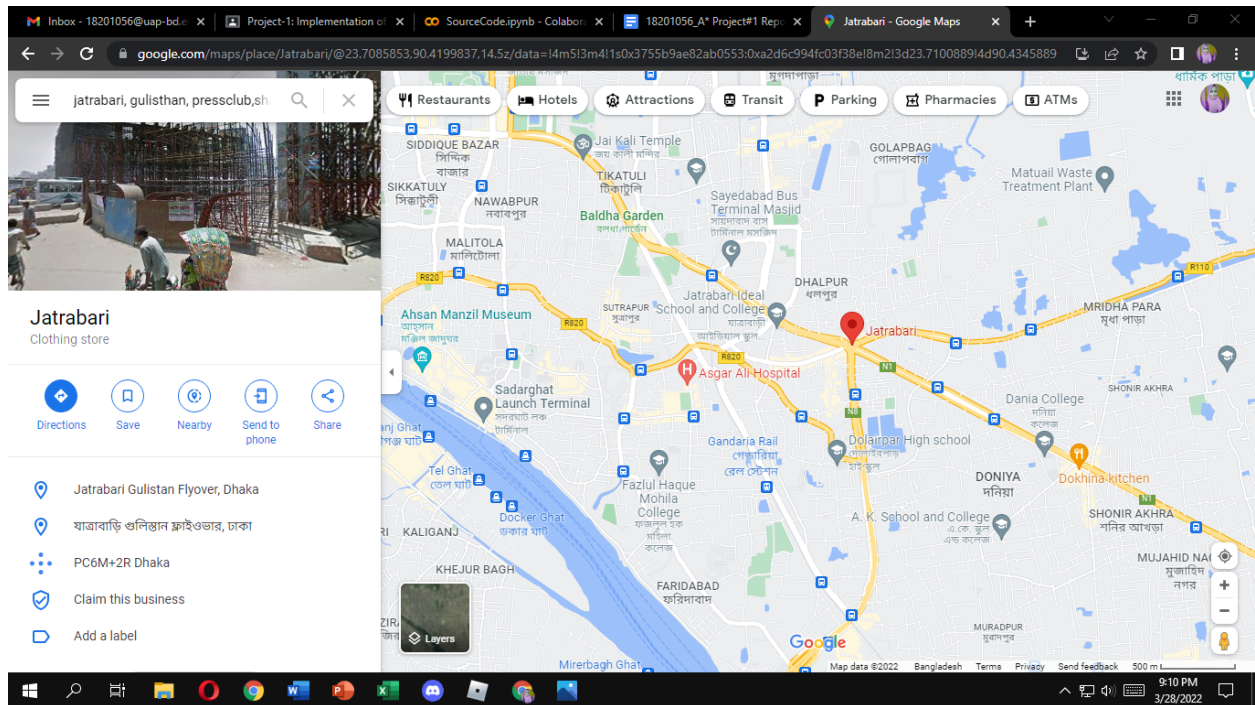




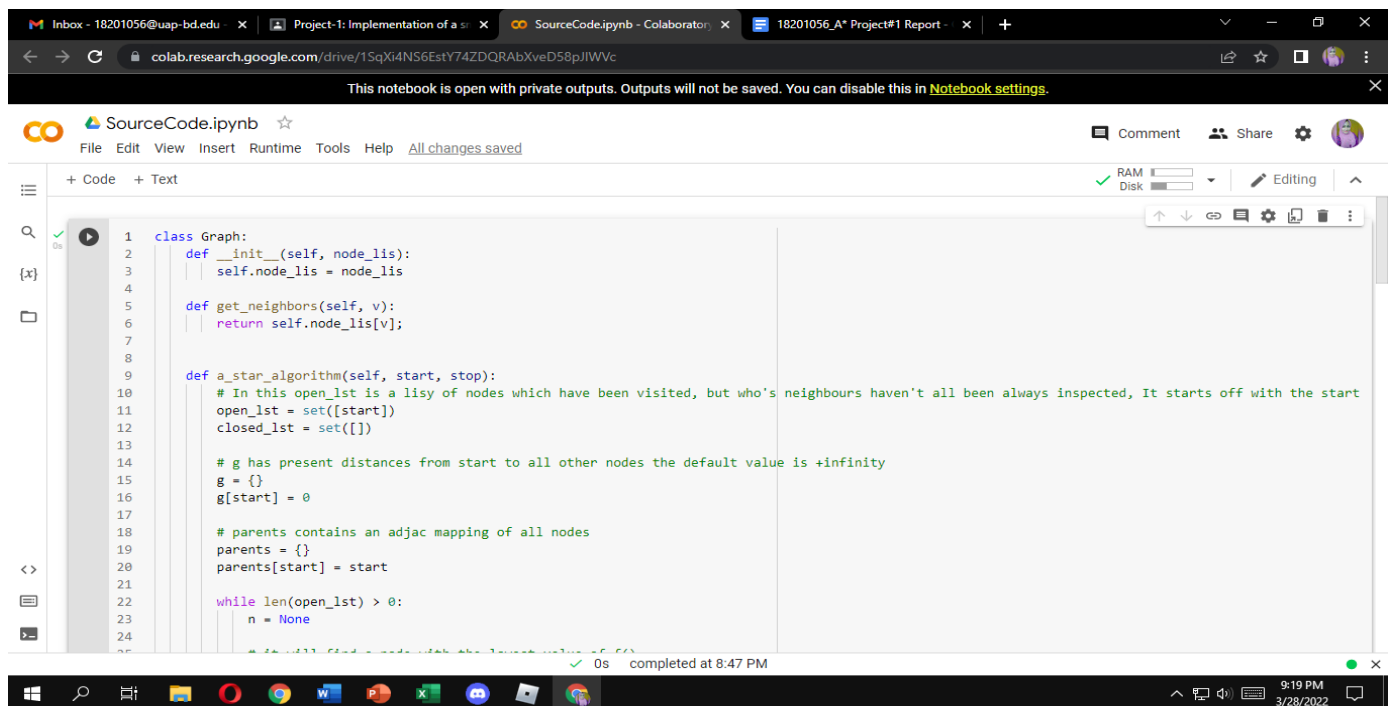
Jatrabari(Home)> Mugdha > Mowchak > Moghbazar > Banglamotor > Karwan Bazar > Farmgate > UAP(Destination)



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



CODE SCREENSHOTS:



Inbox - 18201056@uap-bd.edu x Project-1: Implementation of a s SourceCode.ipynb - Collaborator x 18201056_A* Project#1 Report x +

colab.research.google.com/drive/1SqXi4NS6EstY74ZDQRABXveD58pJIWVc

This notebook is open with private outputs. Outputs will not be saved. You can disable this in [Notebook settings](#).

SourceCode.ipynb ☆

File Edit View Insert Runtime Tools Help All changes saved

+ Code + Text

```
20 parents[start] = start
21
22 while len(open_lst) > 0:
23     n = None
24
25     # it will find a node with the lowest value of f() -
26     for v in open_lst:
27         if n == None or g[v] + self.heuristic(v) < g[n] + self.heuristic(n):
28             n = v;
29
30     if n == None:
31         print('Path does not exist!')
32         return None
33
34     # if the current node is the stop then we start again from start
35     if n == stop:
36         reconst_path = []
37
38         while parents[n] != n:
39             reconst_path.append(n)
40             n = parents[n]
41
42         reconst_path.append(start)
43
44         reconst_path.reverse()
45
46         print('Optimal Path: {}'.format(reconst_path))
```

0s completed at 8:47 PM

9:19 PM 3/28/2022

Inbox - 18201056@uap-bd.edu x Project-1: Implementation of a s SourceCode.ipynb - Collaborator x 18201056_A* Project#1 Report x +

colab.research.google.com/drive/1SqXi4NS6EstY74ZDQRABXveD58pJIWVc

This notebook is open with private outputs. Outputs will not be saved. You can disable this in [Notebook settings](#).

SourceCode.ipynb ☆

File Edit View Insert Runtime Tools Help All changes saved

+ Code + Text

```
41
42 reconst_path.append(start)
43
44 reconst_path.reverse()
45
46 print('Optimal Path: {}'.format(reconst_path))
47 #print('Optimal Path cost: 25 km')
48 return
49
50 # for all the neighbors of the current node do
51 for (m, cost) in self.get_neighbors(n):
52     # if the current node is not present in both open_lst and closed_lst add it to open_lst and note n as it's path find the path cost
53     if m not in open_lst and m not in closed_lst:
54         open_lst.add(m)
55         parents[m] = n
56         g[m] = g[n] + cost
57
58     # otherwise, check if it's quicker to first visit n, then m and if it is, update parents data and pool data and if the node was in the closed
59     else:
60         if g[m] > g[n] + cost:
61             g[m] = g[n] + cost
62             parents[m] = n
63
64         if m in closed_lst:
65             closed_lst.remove(m)
66             open_lst.add(m)
```

0s completed at 8:47 PM

9:19 PM 3/28/2022

Inbox - 18201056@uap-bd.edu x Project-1: Implementation of a SourceCode.ipynb - Collaborator x 18201056_A* Project#1 Report x +

colab.research.google.com/drive/1SqXi4NS6EstY74ZDQRAbXveD58pJIWVc

This notebook is open with private outputs. Outputs will not be saved. You can disable this in [Notebook settings](#).

SourceCode.ipynb

File Edit View Insert Runtime Tools Help All changes saved

RAM Disk Editing

```
68 # remove n from the open_lst, and add it to closed_lst because all of his neighbors were inspected
69 open_lst.remove(n)
70 closed_lst.add(n)
71
72 print('Path does not exist!')
73 return None
74
75 # This is heuristic function which is having equal values for all nodes
76 def heuristic(self, n):
77     heuristic = {
78         'Jatrabari(Home)': 3,
79         'Gulistan': 1,
80         'Mugda': 7,
81         'Kakrail': 5,
82         'Pressclub': 2,
83         'Mowchak': 8,
84         'Shahbag': 3,
85         'Magbazar': 5,
86         'Banglamotor': 4,
87         'Karwanbazar': 5,
88         'Farmgate': 6,
89         'UAP(Destination)': 0
90     }
91
92     return heuristic[n]
93
```

0s completed at 8:47 PM

9:19 PM 3/28/2022

Inbox - 18201056@uap-bd.edu x Project-1: Implementation of a SourceCode.ipynb - Collaborator x 18201056_A* Project#1 Report x +

colab.research.google.com/drive/1SqXi4NS6EstY74ZDQRAbXveD58pJIWVc

This notebook is open with private outputs. Outputs will not be saved. You can disable this in [Notebook settings](#).

SourceCode.ipynb

File Edit View Insert Runtime Tools Help All changes saved

RAM Disk Editing

```
94 node_lis = {
95     'Jatrabari(Home)': [('Gulistan', 4.9), ('Mugda', 4.1)],
96     'Gulistan': [('Kakrail', 3.1), ('Pressclub', 7)],
97     'Mugda': [('Mowchak', 3.6)],
98     'Kakrail': [('Shahbag', 2.2)],
99     'Pressclub': [('Shahbag', 3.8)],
100    'Mowchak': [('Magbazar', 0.75)],
101    'Shahbag': [('Banglamotor', 1.1)],
102    'Magbazar': [('Banglamotor', 2)],
103    'Banglamotor': [('Karwanbazar', 0.7)],
104    'Karwanbazar': [('Farmgate', 2.2)],
105    'Farmgate': [('UAP(Destination)', 1)]
106 }
107
108 cost = {'Jatrabari(Home)': 0}
109 graph1 = Graph(node_lis)
110 graph1.a_star_algorithm ('Jatrabari(Home)', 'UAP(Destination)')
111
```

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

0s completed at 8:47 PM

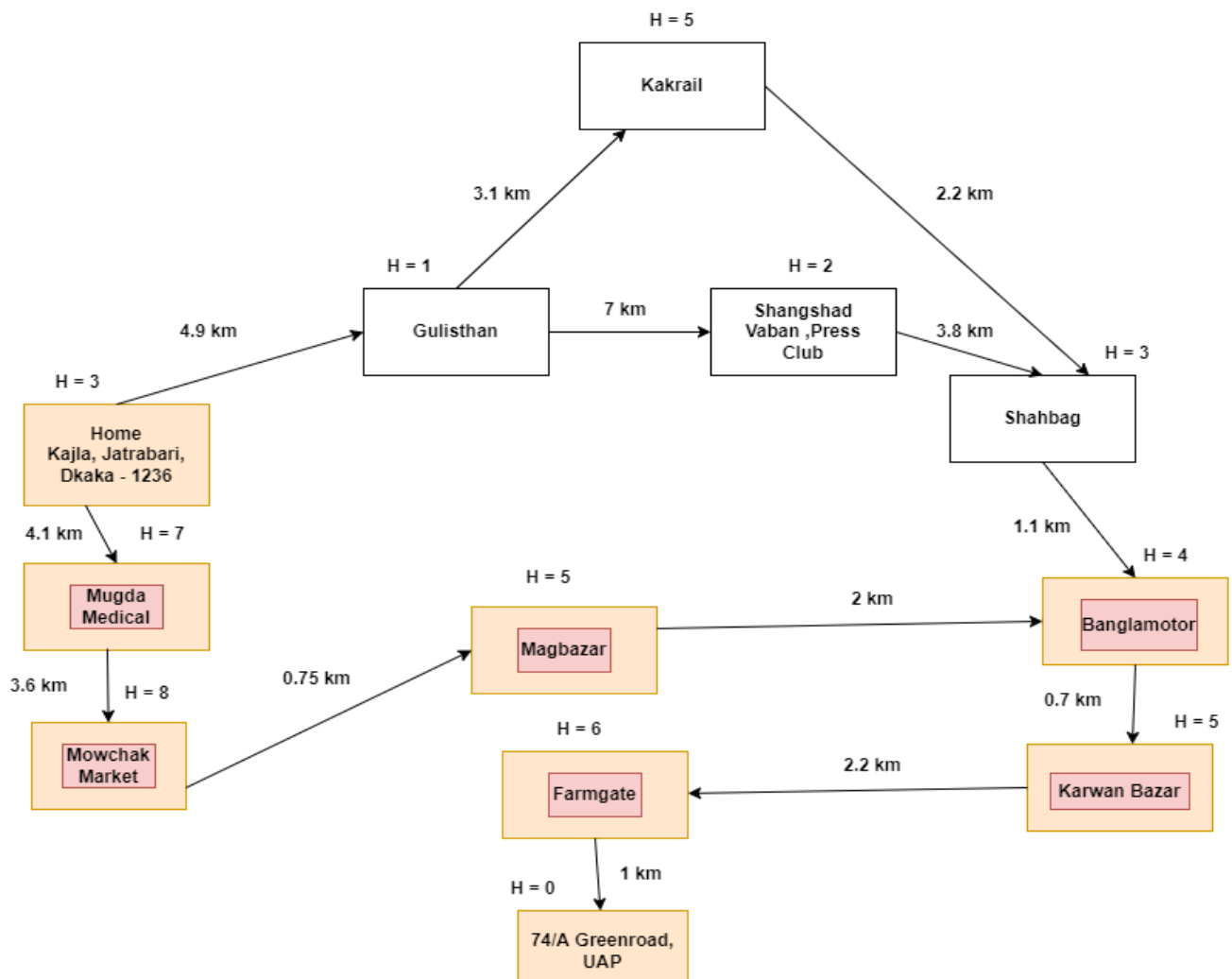
9:20 PM 3/28/2022

18201056 (Map):

Optimal Path Cost: 52.35km

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

Address Map from my home to UAP using A* search algorithm



Heuristic Values:

$h(\text{home}) = 3$

$h(\text{mugda}) = 7$

$h(\text{gulisthan}) = 1$

$h(\text{kakrail}) = 5$

$h(\text{shahbag}) = 3$

$h(\text{mowchak}) = 8$

$h(\text{magbazar}) = 5$

$h(\text{banglamotor}) = 4$

$h(\text{karwan bazar}) = 5$

$h(\text{farmgate}) = 6$

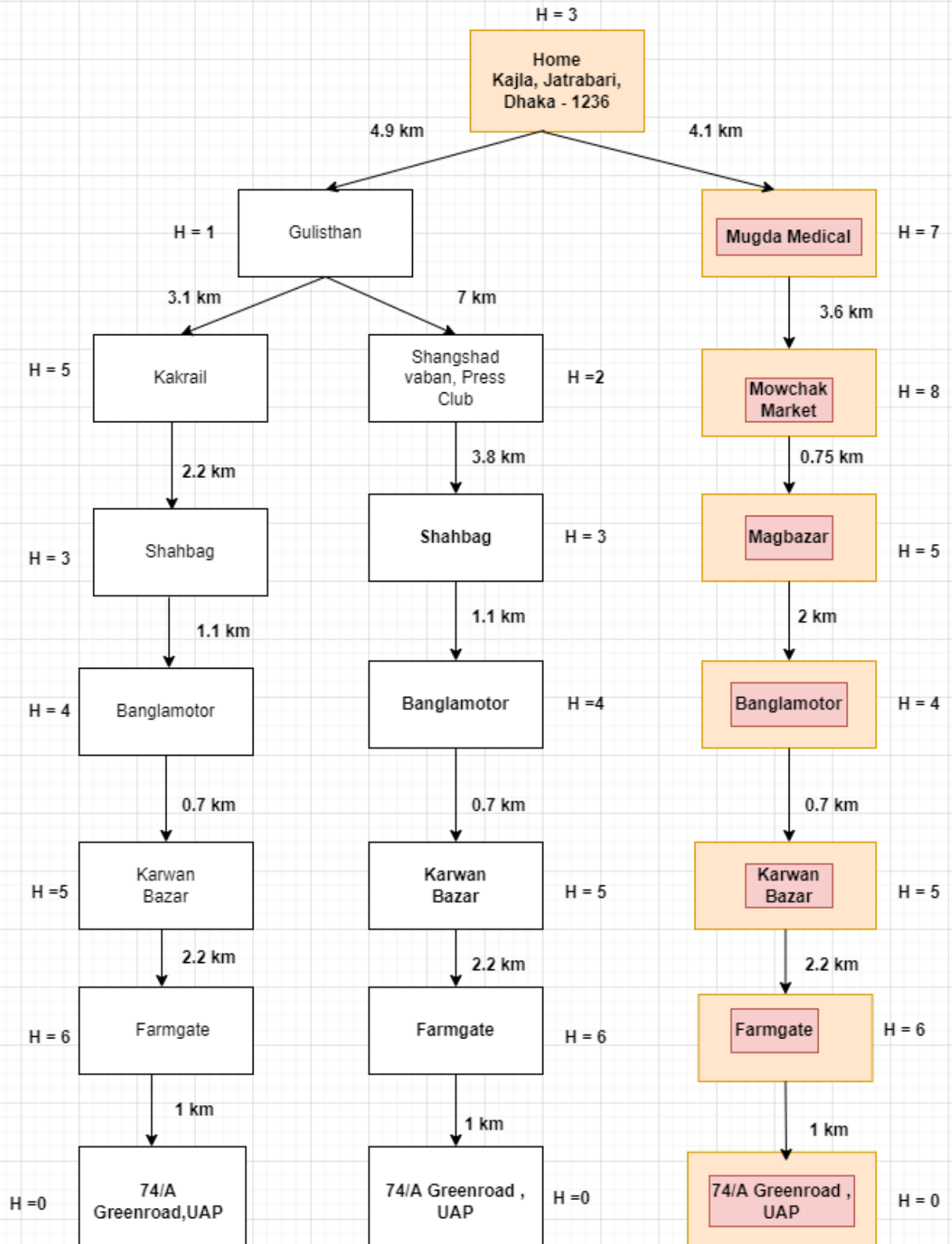
$h(\text{pressclub}) = 2$

$h(\text{destination}) = 0$

Search Tree: 18201056

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

Corresponding Tree of Designed Map



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 present in 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,

        'Gulistan': 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*****
```