**UNIVERSITY OF ASIA PACIFIC** DEPARTMENT OF CSE



**ASSIGNMENT 2**

**CSE 404**

DATE OF SUBMISSION: OCTOBER 11, 2021

|  |
| --- |
| SUBMITTED BY  HASAN TAHSIN RAFSAN  18101009  4TH YEAR 1ST SEMESTER  CSE, UAP |
| SUBMITTED TO  MOLLA RASHIED HUSSEIN  ASSISTANT PROFESSOR  CSE, UAP |

A\* SEARCH ALGORITHM

**PROBLEM STATEMENT**

As per the discussion in class, please create your own map from your home district to University of Asia Pacific and write in any programming language to implement the A\* search to reach the destination from the starting point by using the path costs and straight-line distances.

**TOOLS**

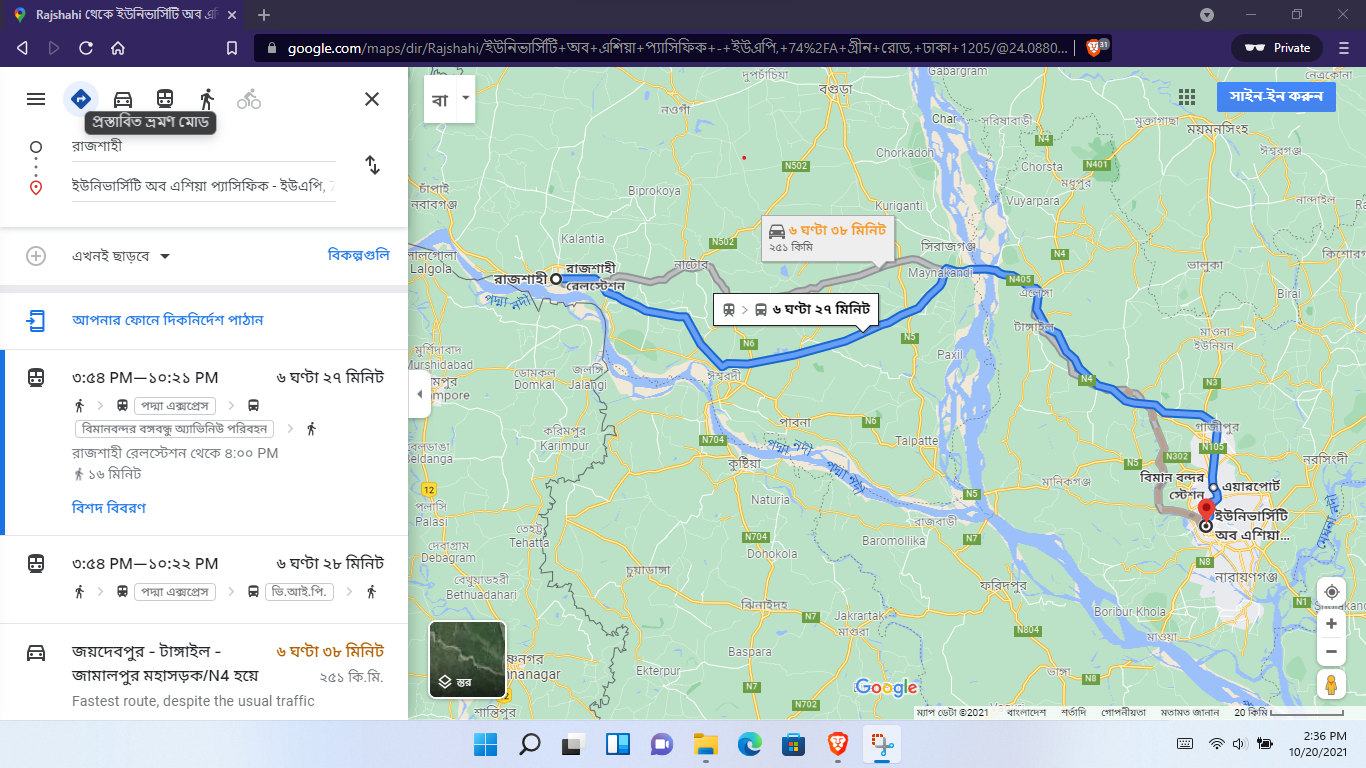
Google Maps, Google Colaboratory, Python Programming Language

**MAPS**

Home: Rajshahi

Destination: UAP

Fig-1 Rajshahi>Sirajganj>UAP



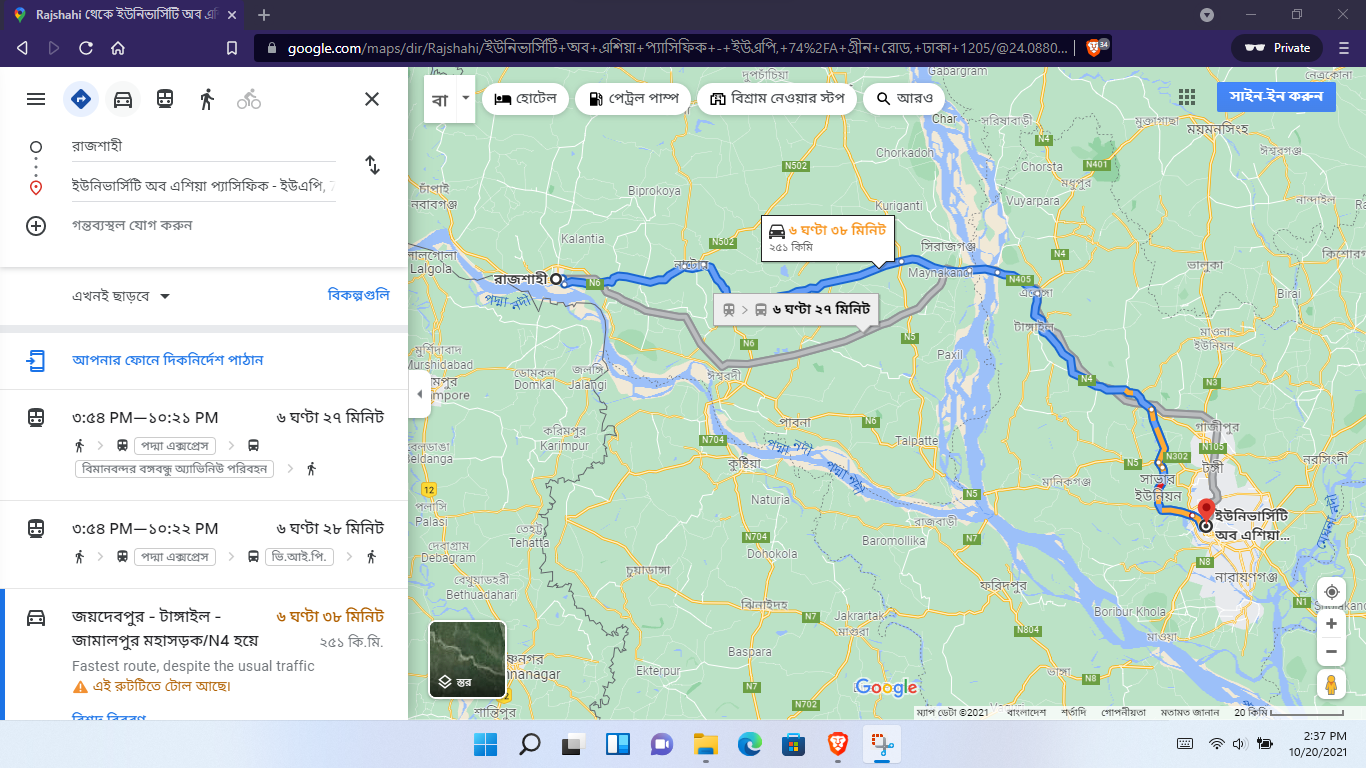


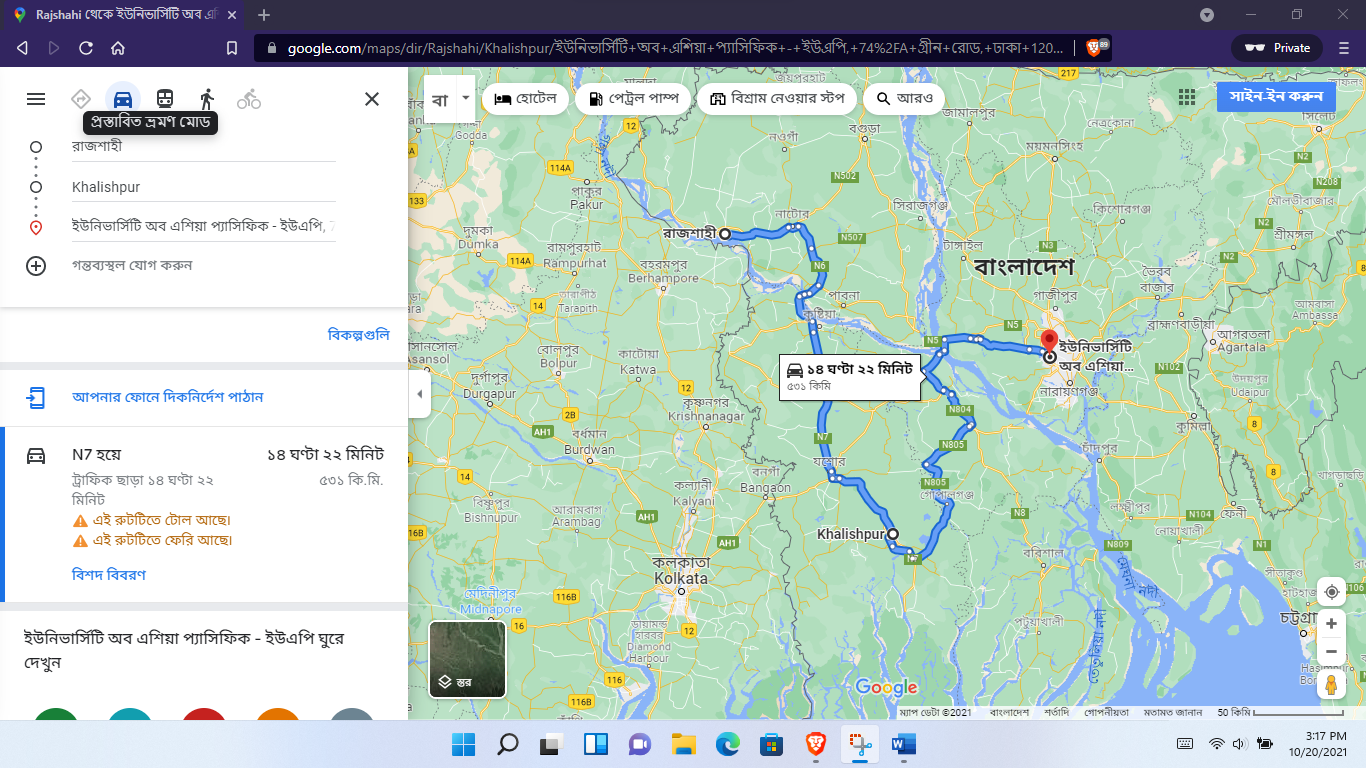
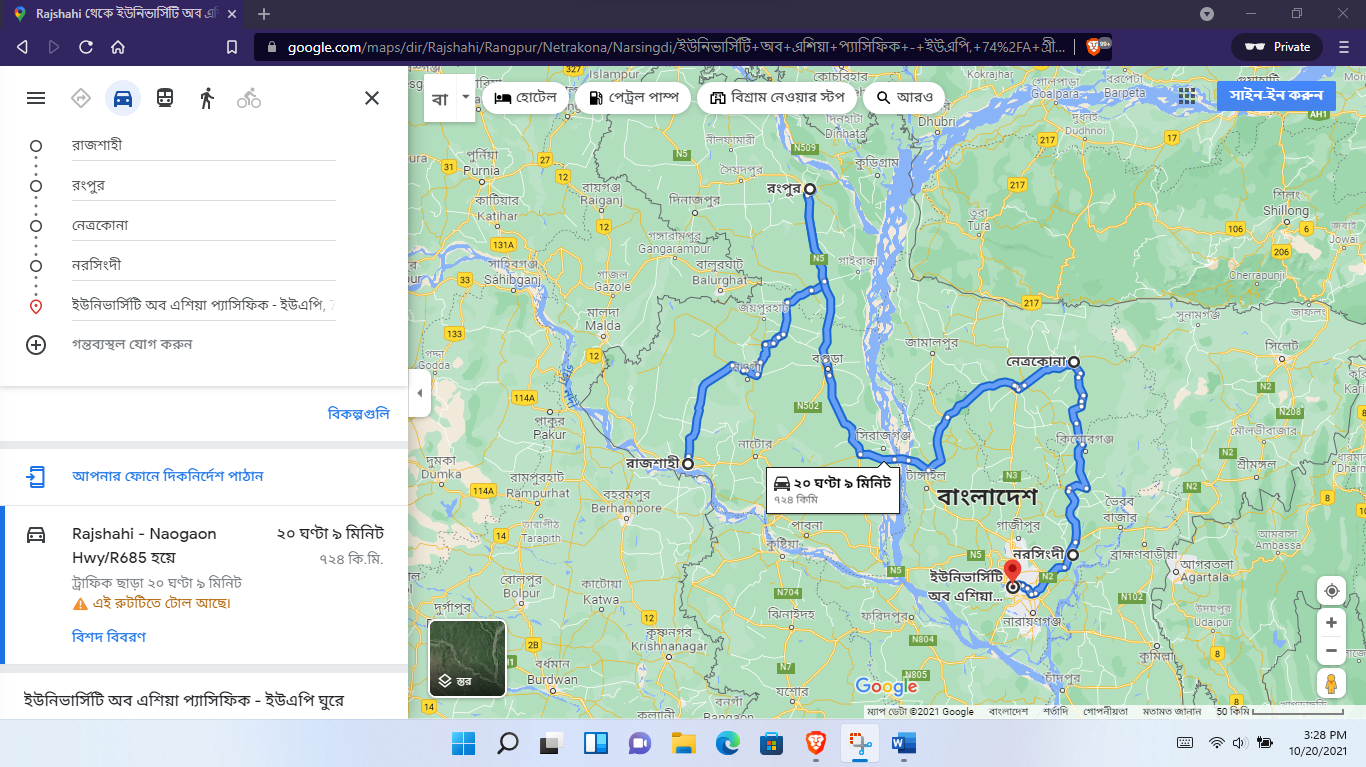
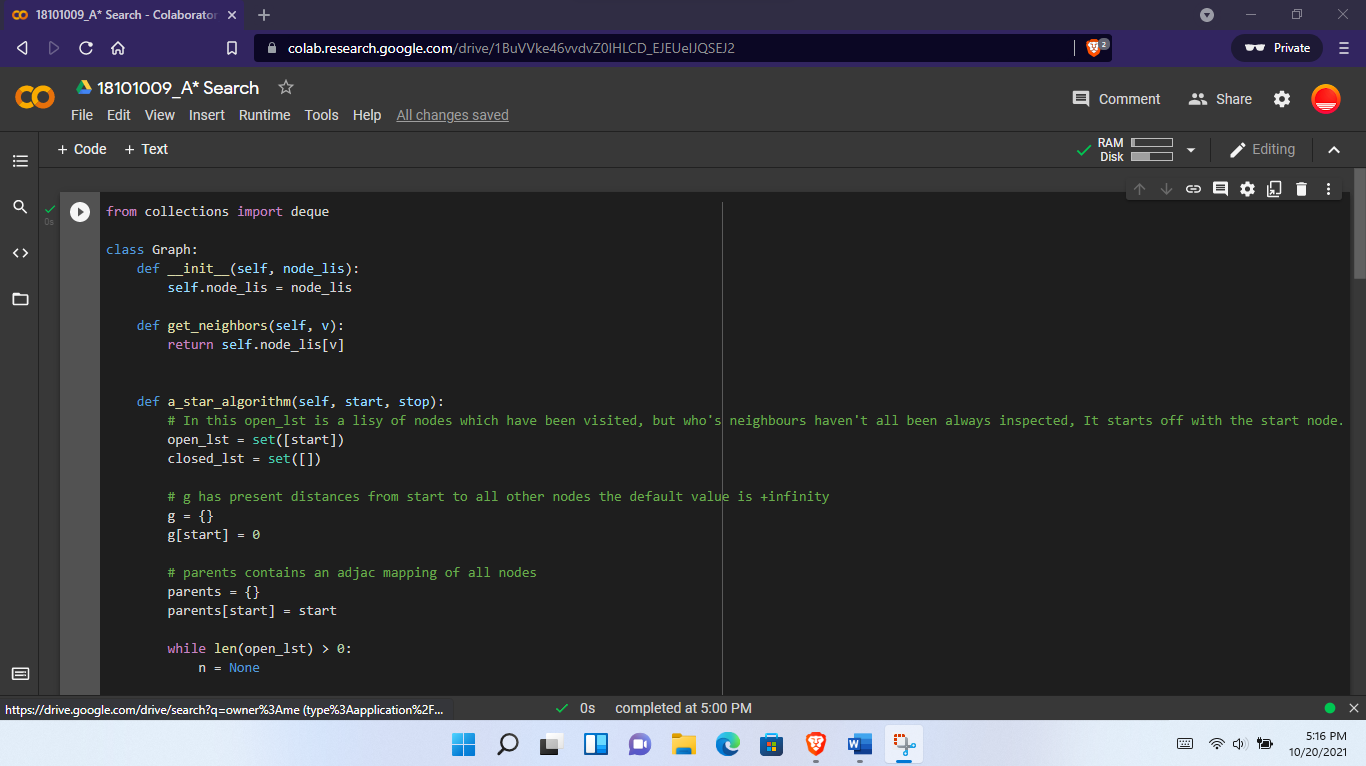
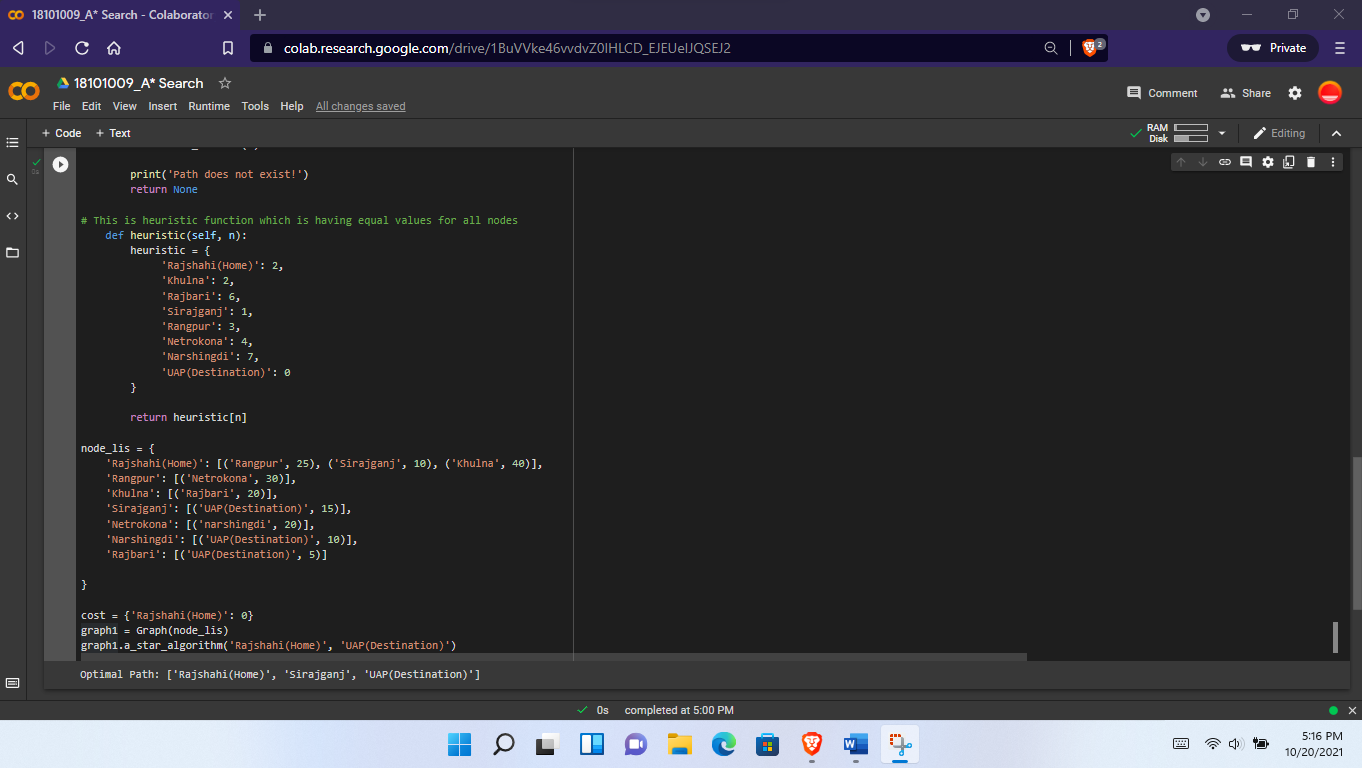
Fig-2: Rajshahi> Khalishpur(Khulna)>UAP

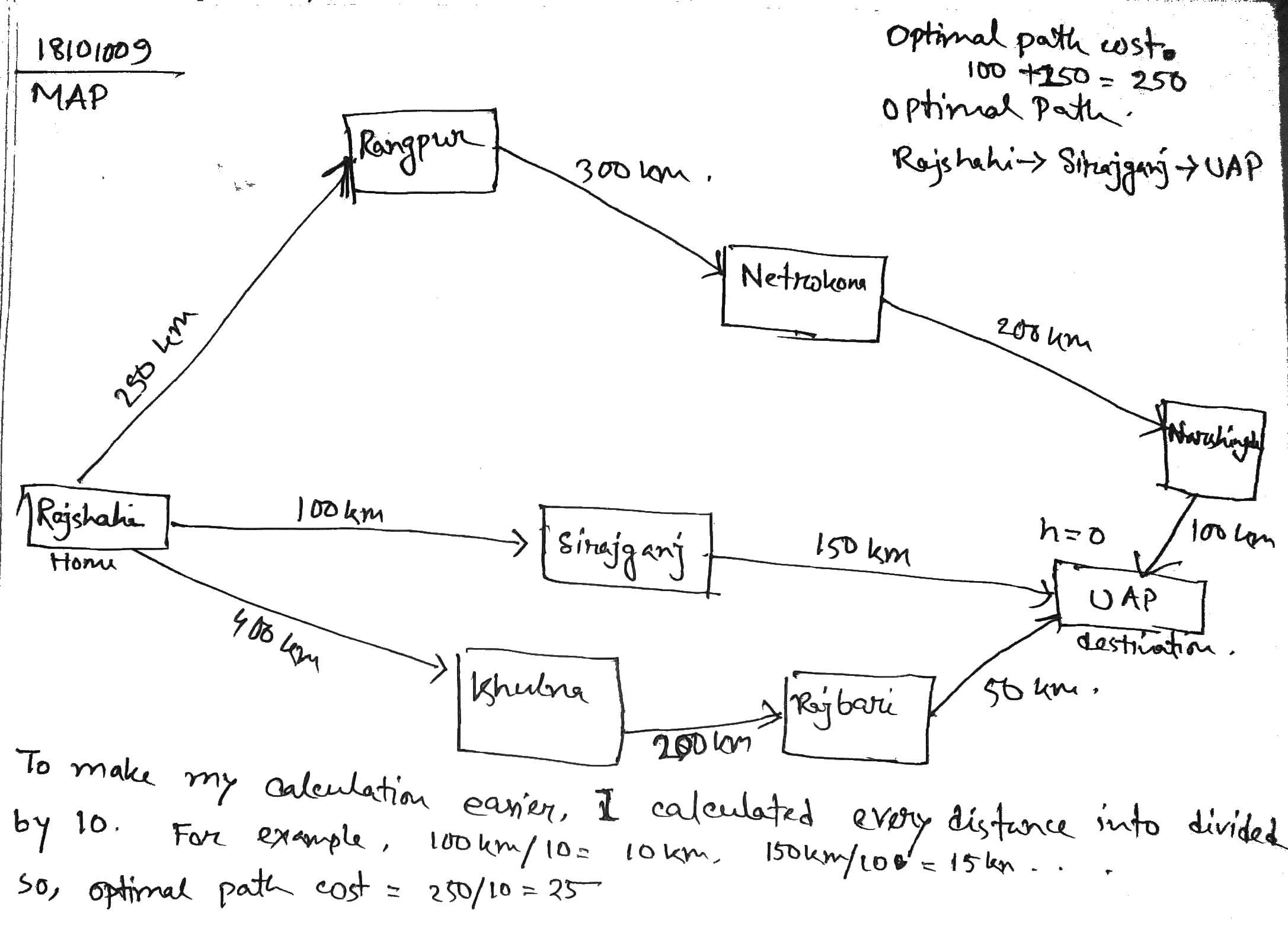
Fig-3: Rajshahi>Rangpur>Netrokona>Narshingdi>UAP 

**CODE SCREENSHOT**





**CALCULATION & HANDWRITTEN IMAGES**



Heuristic Values

Here maps distance is calculated based on 60km in scale distance

h(home) = 60%4+2 = 2

h(rangpur) = 60%5+3 = 3

h(sirajganj) = 60%6+1 = 1

h(khulna) = 60%5+2 = 2

h(netrokona) = 60%4+2 = 4

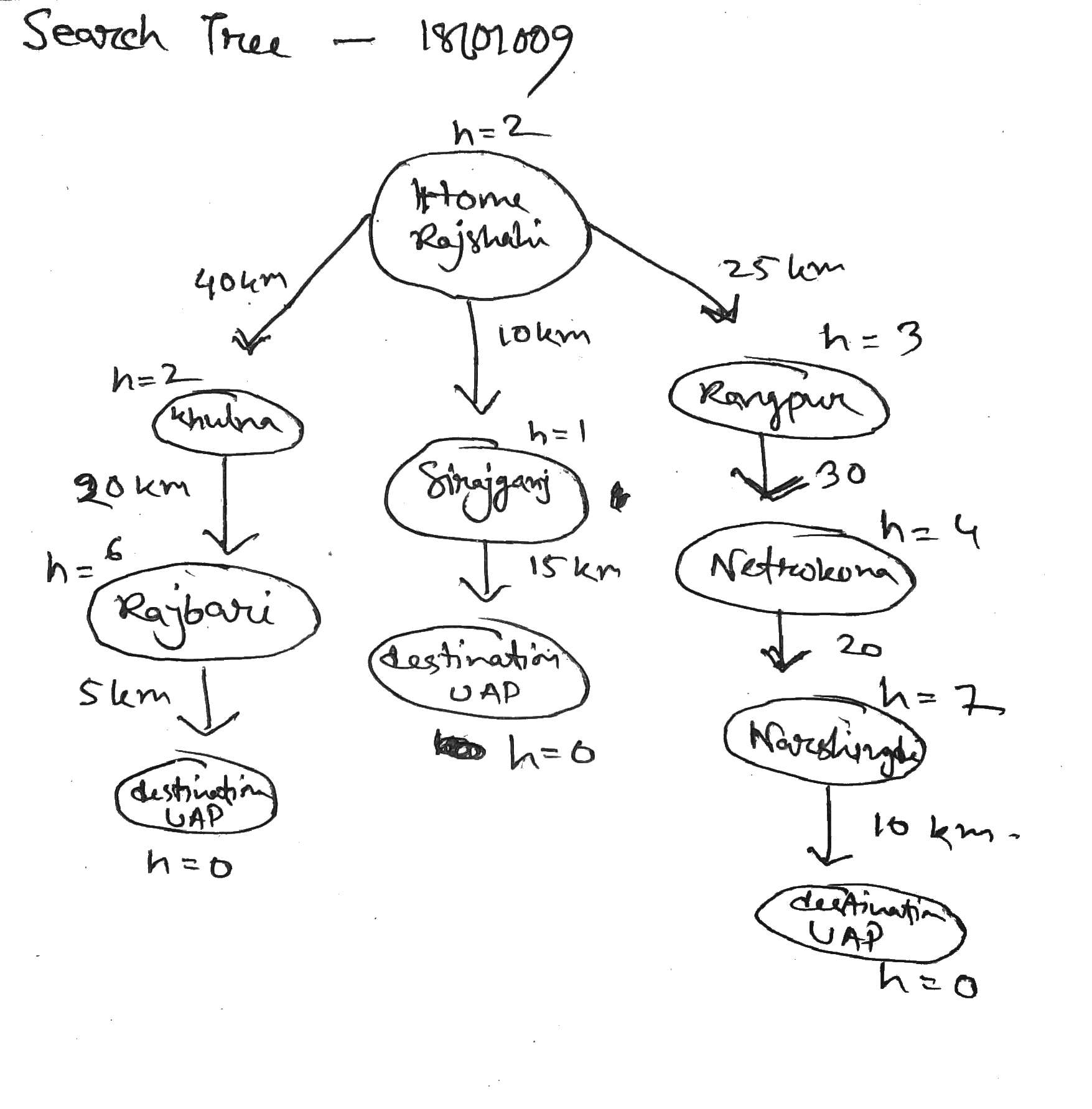
h(narshingdi) = 3+4 = 7

h(rajbari) = 4+2 = 6

h(destination) = 0

Search Tree

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



**CODE**

**from collections import deque**

**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 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 = {**

**'Rajshahi(Home)': 2,**

**'Khulna': 2,**

**'Rajbari': 6,**

**'Sirajganj': 1,**

**'Rangpur': 3,**

**'Netrokona': 4,**

**'Narshingdi': 7,**

**'UAP(Destination)': 0**

**}**

**return heuristic[n]**

**node\_lis = {**

**'Rajshahi(Home)': [('Rangpur', 25), ('Sirajganj', 10), ('Khulna', 40)],**

**'Rangpur': [('Netrokona', 30)],**

**'Khulna': [('Rajbari', 20)],**

**'Sirajganj': [('UAP(Destination)', 15)],**

**'Netrokona': [('narshingdi', 20)],**

**'Narshingdi': [('UAP(Destination)', 10)],**

**'Rajbari': [('UAP(Destination)', 5)]**

**}**

**cost = {'Rajshahi(Home)': 0}**

**graph1 = Graph(node\_lis)**

**graph1.a\_star\_algorithm('Rajshahi(Home)', 'UAP(Destination)')**