Department of Computer Science and Engineering (Data Science)

Subject: Artificial Intelligence (DJ19DSC502)

AY: 2023-24

Experiment 6

(Optimization)

Name: Jhanvi Parekh

SAPID: 60009210033

Aim: Find the shortest path between two places using A* Algorithm.

Theory:

A* is a searching algorithm that is used to find the shortest path between an initial and a final point.

It is a handy algorithm that is often used for map traversal to find the shortest path to be taken. A* was initially designed as a graph traversal problem, to help build a robot that can find its own course. It still remains a widely popular algorithm for graph traversal.

It searches for shorter paths first, thus making it an optimal and complete algorithm. An optimal algorithm will find the least cost outcome for a problem, while a complete algorithm finds all the possible outcomes of a problem.

Another aspect that makes A* so powerful is the use of weighted graphs in its implementation. A weighted graph uses numbers to represent the cost of taking each path or course of action. This means that the algorithms can take the path with the least cost, and find the best route in terms of distance and time.

Explanation

In the event that we have a grid with many obstacles and we want to get somewhere as rapidly as possible, the A* Search Algorithms are our savior. From a given starting cell, we can get to the target cell as quickly as possible. It is the sum of two variables' values that determines the node it picks at any point in time.
At each step, it picks the node with the smallest value of 'f' (the sum of 'g' and 'h') and processes that node/cell. 'g' and 'h' is defined as simply as possible below:

- 'g' is the distance it takes to get to a certain square on the grid from the starting point, following the path we generated to get there.
- 'h' is the heuristic, which is the estimation of the distance it takes to get to the finish line from that square on the grid.

Heuristics are basically educated guesses. It is crucial to understand that we do not know the distance to the finish point until we find the route since there are so many things that might get in the way (e.g., walls, water, etc.). In the coming sections, we will dive deeper into how to calculate the heuristics.

Algorithm

Initial condition - we create two lists - Open List and Closed List.

Now, the following steps need to be implemented -

- The open list must be initialized.
- Put the starting node on the open list (leave its f at zero). Initialize the closed list.
- Follow the steps until the open list is non-empty:

1. Find the node with the least f on the open list and name it "q".
2. Remove Q from the open list.
3. Produce q's eight descendants and set q as their parent.
4. For every descendant:

i) If finding a successor is the goal, cease looking ii)Else, calculate g and h for

the successor. successor.g = q.g + the calculated distance between the

successor and the q. successor.h = the calculated distance between the

successor and the goal. We will cover three heuristics to do this: the

Diagonal, the Euclidean, and the Manhattan heuristics.

successor.f = successor.g plus successor.h iii) Skip this successor if a node in the OPEN list with the same location as it but a lower f value than the successor is present.

iv) Skip the successor if there is a node in the CLOSED list with the same position as the successor but a lower f value; otherwise, add the node to the open list end (for loop).

- Push Q into the closed list and end the while loop.

Lab Assignment to do:

Solve the following Shortest Path Algorithm:

1. Consider 40 -45 geo locations between Juhu beach till Gateway of India. 5-8 locations not in the route and choice of location should keep in mind two different possible path.
2. Find the shortest path between Juhu beach till Gateway of India.
3. Use Havesine formula for distance calculation.

Code:

Link : https://colab.research.google.com/drive/11Hyrob73W_Z6TJY9stoqodADI1evxL7u?usp=sharing

Department of Computer Science and Engineering (Data Science)

```
import numpy as np
import pandas as pd
```

```
cost_matrix=pd.DataFrame(data=[
    [0, 3, 6, 4, 3, 5, 2, 1, 2, 1, 5, 4, 3, 10, 10, 8, 7, 7, 8, 8, 9, 10, 10, 9, 10, 12, 10, 15],
    [3, 0, 6, 3, 3, 4, 4, 2, 2, 2, 5, 4, 4, 10, 9, 7, 6, 6, 7, 7, 8, 9, 9, 8, 10, 11, 9, 14],
    [6, 6, 0, 9, 8, 2, 9, 8, 7, 8, 6, 5, 8, 12, 7, 9, 10, 12, 12, 12, 12, 13, 14, 12, 14, 14, 7, 4],
    [4, 3, 9, 0, 2, 7, 1, 4, 5, 4, 5, 5, 3, 8, 3, 1, 2, 3, 4, 4, 5, 5, 5, 4, 6, 8, 6, 11],
    [3, 3, 8, 2, 0, 6, 2, 3, 4, 3, 4, 3, 2, 8, 3, 2, 2, 3, 3, 3, 4, 5, 5, 4, 6, 7, 5, 10],
    [5, 4, 2, 7, 6, 0, 7, 6, 5, 6, 3, 2, 5, 10, 5, 7, 8, 10, 10, 10, 10, 11, 12, 10, 12, 12, 5, 8],
    [2, 4, 9, 1, 2, 7, 0, 3, 4, 3, 6, 6, 3, 8, 4, 2, 3, 4, 5, 4, 6, 6, 6, 6, 8, 9, 7, 12],
    [1, 2, 8, 4, 3, 6, 3, 0, 1, 1, 4, 3, 1, 7, 4, 4, 5, 6, 6, 6, 7, 7, 8, 6, 8, 9, 7, 12],
    [2, 2, 7, 5, 4, 5, 4, 1, 0, 1, 4, 3, 2, 8, 5, 5, 6, 7, 6, 7, 7, 8, 8, 7, 9, 10, 8, 13],
    [1, 2, 8, 4, 3, 6, 3, 1, 1, 0, 4, 3, 2, 8, 5, 5, 6, 7, 6, 7, 7, 7, 8, 6, 8, 9, 7, 12],
    [5, 5, 6, 5, 4, 3, 6, 4, 4, 4, 0, 1, 4, 9, 6, 6, 7, 9, 8, 9, 9, 10, 10, 9, 11, 11, 2, 7],
    [4, 4, 5, 5, 3, 2, 6, 3, 3, 3, 1, 0, 3, 8, 5, 5, 6, 8, 7, 8, 8, 9, 9, 8, 10, 10, 2, 7],
    [3, 4, 8, 3, 2, 5, 3, 1, 2, 2, 4, 3, 0, 6, 5, 4, 5, 6, 5, 6, 6, 7, 8, 6, 8, 9, 5, 10],
    [10, 10, 12, 8, 8, 10, 8, 7, 8, 8, 9, 8, 6, 0, 6, 8, 9, 9, 10, 10, 10, 11, 12, 11, 12, 11, 10, 15],
    [10, 9, 7, 3, 3, 5, 4, 4, 5, 5, 6, 5, 5, 6, 0, 2, 3, 4, 4, 5, 5, 6, 7, 5, 7, 8, 9, 14],
    [8, 7, 9, 1, 2, 7, 2, 4, 5, 5, 6, 5, 4, 8, 2, 0, 1, 2, 3, 2, 4, 4, 4, 4, 6, 8, 7, 12],
    [7, 6, 10, 2, 2, 8, 3, 5, 6, 6, 7, 6, 5, 9, 3, 1, 0, 1, 2, 2, 3, 3, 3, 2, 4, 6, 6, 11],
    [7, 6, 12, 3, 3, 10, 4, 6, 7, 7, 9, 8, 6, 9, 4, 2, 1, 0, 1, 1, 2, 2, 2, 1, 3, 5, 7, 12],
    [8, 7, 12, 4, 3, 10, 5, 6, 6, 6, 8, 7, 5, 10, 4, 3, 2, 1, 0, 1, 1, 1, 2, 1, 2, 4, 8, 13],
    [8, 7, 12, 4, 3, 10, 4, 6, 7, 7, 9, 8, 6, 10, 5, 2, 2, 1, 1, 0, 1, 1, 2, 1, 3, 5, 8, 13],
    [9, 8, 12, 5, 4, 10, 6, 7, 7, 7, 9, 8, 6, 10, 5, 4, 3, 2, 1, 1, 0, 1, 1, 1, 3, 4, 9, 14],
    [10, 9, 13, 5, 5, 11, 6, 7, 8, 7, 10, 9, 7, 11, 6, 4, 3, 2, 1, 1, 1, 0, 1, 1, 3, 3, 10, 15],
    [10, 9, 14, 5, 5, 12, 6, 8, 8, 8, 10, 9, 8, 12, 7, 4, 3, 2, 1, 2, 1, 1, 0, 2, 4, 2, 11, 16],
    [9, 8, 12, 4, 4, 10, 6, 6, 7, 6, 9, 8, 6, 11, 5, 4, 2, 1, 1, 1, 1, 3, 2, 0, 2, 4, 9, 14],
    [10, 10, 14, 6, 6, 12, 8, 8, 9, 8, 11, 10, 8, 12, 7, 6, 4, 3, 2, 3, 3, 3, 4, 2, 0, 4, 9, 14],
    [12, 11, 14, 8, 7, 12, 9, 9, 10, 9, 11, 10, 9, 11, 8, 8, 6, 5, 4, 5, 4, 3, 2, 4, 4, 0, 9, 14],
    [10, 9, 7, 6, 5, 5, 7, 7, 8, 7, 2, 2, 5, 10, 9, 7, 6, 7, 8, 8, 9, 10, 11, 9, 9, 9, 0, 5],
    [15, 14, 4, 11, 10, 8, 12, 12, 13, 12, 7, 7, 10, 15, 14, 12, 11, 12, 13, 13, 14, 15, 16, 14, 14, 14, 5, 0]
]
,columns=['DJ Sanghvi college', 'mithibai', 'cooper hospital', 'juhu circle',
       'irla petrol pump', 'nanavati', 'jw marriot-airport', 'andheri station',
       'santacruz aeroplane park', 'airport-andheri', 'lokhandwaala market',
       'versova telephone exchange', 'The Club-andheri', 'BKC-JIO mall',
       'Worli Sealink', 'Bandra-linking road', 'Jaslok hospital',
       'siddhivinayak temple', 'mahalaxmi railway station', 'saifee hosptial',
       'hanging garden-mumbai', 'tarapore aquarium', ' wankhede stadium',
       'churney road railwat station', 'trident hotel', 'kokilaben hospital',
       'borivali national park', 'Nesco-goregaon '],
                      index=['DJ Sanghvi college', 'mithibai', 'cooper hospital', 'juhu circle',
       'irla petrol pump', 'nanavati', 'jw marriot-airport', 'andheri station',
       'santacruz aeroplane park', 'airport-andheri', 'lokhandwaala market',
       'versova telephone exchange', 'The Club-andheri', 'BKC-JIO mall',
       'Worli Sealink', 'Bandra-linking road', 'Jaslok hospital',
       'siddhivinayak temple', 'mahalaxmi railway station', 'saifee hosptial',
       'hanging garden-mumbai', 'tarapore aquarium', ' wankhede stadium',
       'churney road railwat station', 'trident hotel', 'kokilaben hospital',
       'borivali national park', 'Nesco-goregaon ']
)
```

```
heuristic_matrix=pd.DataFrame(data=hs
,columns=['DJ Sanghvi college', 'mithibai', 'cooper hospital', 'juhu circle',
       'irla petrol pump', 'nanavati', 'jw marriot-airport', 'andheri station',
       'santacruz aeroplane park', 'airport-andheri', 'lokhandwaala market',
       'versova telephone exchange', 'The Club-andheri', 'BKC-JIO mall',
       'Worli Sealink', 'Bandra-linking road', 'Jaslok hospital',
       'siddhivinayak temple', 'mahalaxmi railway station', 'saifee hosptial',
       'hanging garden-mumbai', 'tarapore aquarium', ' wankhede stadium',
       'churney road railwat station', 'trident hotel', 'kokilaben hospital',
       'borivali national park', 'Nesco-goregaon '],
                      index=['DJ Sanghvi college', 'mithibai', 'cooper hospital', 'juhu circle',
       'irla petrol pump', 'nanavati', 'jw marriot-airport', 'andheri station',
       'santacruz aeroplane park', 'airport-andheri', 'lokhandwaala market',
       'versova telephone exchange', 'The Club-andheri', 'BKC-JIO mall',
       'Worli Sealink', 'Bandra-linking road', 'Jaslok hospital',
       'siddhivinayak temple', 'mahalaxmi railway station', 'saifee hosptial',
       'hanging garden-mumbai', 'tarapore aquarium', ' wankhede stadium',
       'churney road railwat station', 'trident hotel', 'kokilaben hospital',
       'borivali national park', 'Nesco-goregaon ']
)
```

```
cols=['DJ Sanghvi college', 'mithibai', 'cooper hospital', 'juhu circle',
       'irla petrol pump', 'nanavati', 'jw marriot-airport', 'andheri station',
       'santacruz aeroplane park', 'airport-andheri', 'lokhandwaala market',
       'versova telephone exchange', 'The Club-andheri', 'BKC-JIO mall',
       'Worli Sealink', 'Bandra-linking road', 'Jaslok hospital',
       'siddhivinayak temple', 'mahalaxmi railway station', 'saifee hosptial',
       'hanging garden-mumbai', 'tarapore aquarium', ' wankhede stadium',
       'churney road railwat station', 'trident hotel', 'kokilaben hospital',
       'borivali national park', 'Nesco-goregaon ']
```

```
availabledf=pd.DataFrame(data=np.zeros((28,28)),columns=cols,index=cols)
```

```
for col in cols:
    for row in cols:
        if(heuristic_matrix[col][row]<6.5):
            availabledf[col][row]=1
        else:
            availabledf[col][row]=0
        availabledf[col][col]=0
```

```
def sortit(a):
    x=sorted(a,key=last)
    return x
```

```
def last(tup):
    return tup[2]
```

```
def head(iterable):
    return iterable[0]
```

```
def GoalTestNode(node,Goal):
    if node==Goal:
        return True
    return False
```

```
def ReconstructPath(node,parent,parentarr):
    child=node
    while parent!=None:
        print(f'{parent} <----- {child}\n')
        child=parent
        parent=parentarr[child]
```

```
def movegen(node,cols,df):
    tempcols=[]
    for i in cols:
        if(df[node][i]==1):
            tempcols.append(i)
    return tempcols
```

```
def Propogate_improvement(M,closed,g,k,h,parent,f):
    for X in movegen(M,cols,d):
        if((g[M]+k[M][X])<g[X]):
            parent[X]=M
            g[X]=g[M]+k[M][X]
            f[X]=g[X]+h[X]
            if(X in list(zip(*closed))[0]):
                g,f,parent=Propogate_improvement(X,closed,g,k,h,parent,f)
            return g,f,parent
```

```
def A_star(cols,k,h,S,goal,df):
    parent={}
    for i in cols:
        parent[i]=None
    g={}
    for i in cols:
        g[i]=99999
    f_matrix={}
    parent[S]=None
    g[S]=0
    f_matrix[S]=g[S]+h[S][goal]
    openlist=[(S,parent[S],f_matrix[S])]
    closed=[]
    print(g)

    while openlist!=[]:
        node=(sortit(openlist)).pop(0)
        #print(head(node))
        closed.append(node)
        #print(closed)
        if GoalTestNode(head(node),goal):
            print(parent)
            return ReconstructPath(head(node),node[1],parent)
        for neighbour in movegen(head(node),cols,df):
            #print(g[head(node)]+k[head(node)][neighbour])
            if((neighbour not in list(zip(*closed))[0]) and (neighbour not in list(zip(*openlist))[0])):
                parent[neighbour]=head(node)
                g[neighbour]=g[head(node)]+k[head(node)][neighbour]
                f_matrix[neighbour]=g[neighbour]+h[neighbour][goal]
                openlist.append((neighbour,parent[neighbour],f_matrix[neighbour]))
                #print('new')

            elif(neighbour in list(zip(*closed))[0]):
                if((g[head(node)]+k[head(node)][neighbour])<g[neighbour]):
                    parent[neighbour]=head(node)
                    g[neighbour]=g[head(node)]+k[head(node)][neighbour]
                    f_matrix[neighbour]=g[neighbour]+h[neighbour][goal]
                    g,f_matrix,parent=Propogate_improvement(neighbour,closed,g,k,h,parent,f_matrix)
                #print('impro')
            elif(neighbour in list(zip(*openlist))[0]):
                if((g[head(node)]+k[head(node)][neighbour])<g[neighbour]):
                    parent[neighbour]=head(node)
                    g[neighbour]=g[head(node)]+k[head(node)][neighbour]
                    f_matrix[neighbour]=g[neighbour]+h[neighbour][goal]
                #print('alre')
            else:
                pass
                #print('else')
        temp=openlist.pop(0)
        print(temp)
    return []
```

```
import math

# Function to calculate the Haversine distance between two points given their latitude and longitude
def haversine_distance(lat1, lon1, lat2, lon2):
    R = 6371.0  # approximate radius of the Earth in kilometers

    lat1_rad = math.radians(lat1)
    lon1_rad = math.radians(lon1)
    lat2_rad = math.radians(lat2)
    lon2_rad = math.radians(lon2)

    dlon = lon2_rad - lon1_rad
    dlat = lat2_rad - lat1_rad

    a = math.sin(dlat / 2)**2 + math.cos(lat1_rad) * math.cos(lat2_rad) * math.sin(dlon / 2)**2
    c = 2 * math.atan2(math.sqrt(a), math.sqrt(1 - a))

    distance = R * c
    return distance

# Locations and their coordinates
locations = {
    'DJ Sanghvi college': (19.1048, 72.8375),
    'mithibai': (19.1075, 72.8377),
    'cooper hospital': (19.0598, 72.8387),
    'juhu circle': (19.1073, 72.8276),
    'irla petrol pump': (19.1068, 72.8381),
    'nanavati': (19.1026, 72.8385),
    'jw marriot-airport': (19.1123, 72.8614),
    'andheri station': (19.1197, 72.8464),
    'santacruz aeroplane park': (19.0994, 72.8539),
    'airport-andheri': (19.1191, 72.8482),
    'lokhandwaala market': (19.1308, 72.8297),
    'versova telephone exchange': (19.1317, 72.8141),
    'The Club-andheri': (19.1155, 72.8424),
    'BKC-JIO mall': (19.0671, 72.8679),
    'Worli Sealink': (19.0227, 72.8174),
    'Bandra-linking road': (19.0596, 72.8295),
    'Jaslok hospital': (18.9730, 72.8126),
    'siddhivinayak temple': (19.0176, 72.8308),
    'mahalaxmi railway station': (18.9827, 72.8235),
    'saifee hosptial': (18.9587, 72.8162),
    'hanging garden-mumbai': (18.9593, 72.8134),
    'tarapore aquarium': (18.9274, 72.8260),
    'wankhede stadium': (18.9388, 72.8255),
    'churney road railwat station': (18.9575, 72.8040),
    'trident hotel': (18.9253, 72.8207),
    'kokilaben hospital': (19.1318, 72.8294),
    'borivali national park': (19.2295, 72.8543),
    'Nesco-goregaon': (19.1553, 72.8556)
}

# Initialize an empty matrix
adjacency_matrix_haversine = [[0 for _ in range(len(locations))] for _ in range(len(locations))]

# Calculate Haversine distances and populate the matrix
for i, (place1, (lat1, lon1)) in enumerate(locations.items()):
    for j, (place2, (lat2, lon2)) in enumerate(locations.items()):
        if i != j:
            distance = haversine_distance(lat1, lon1, lat2, lon2)
            adjacency_matrix_haversine[i][j] = distance
hs=[]
# Print the adjacency matrix
for row in adjacency_matrix_haversine:
    hs.append(row)
```

Output:

```
Jaslok hospital <----- trident hotel

Worli Sealink <----- Jaslok hospital

Bandra-linking road <----- Worli Sealink

santacruz aeroplane park <----- Bandra-linking road

DJ Sanghvi college <----- santacruz aeroplane park
```