Shri Vile Parle Kelavani Mandal's
DWARKADAS J. SANGHVI COLLEGE OF ENGINEERING
(Autonomous College Affiliated to the University of Mumbai)
NAAC Accredited with "A" Grade (CGPA : 3.18)

Department of Computer Science and Engineering (Data Science)

Subject: Artificial Intelligence (DJ19DSC502)

AY: 2023-24

NAME:DIVYESH KHUNT **SAPID:60009210116** **BATCH:D12**

Experiment 6

(Optimization)

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.

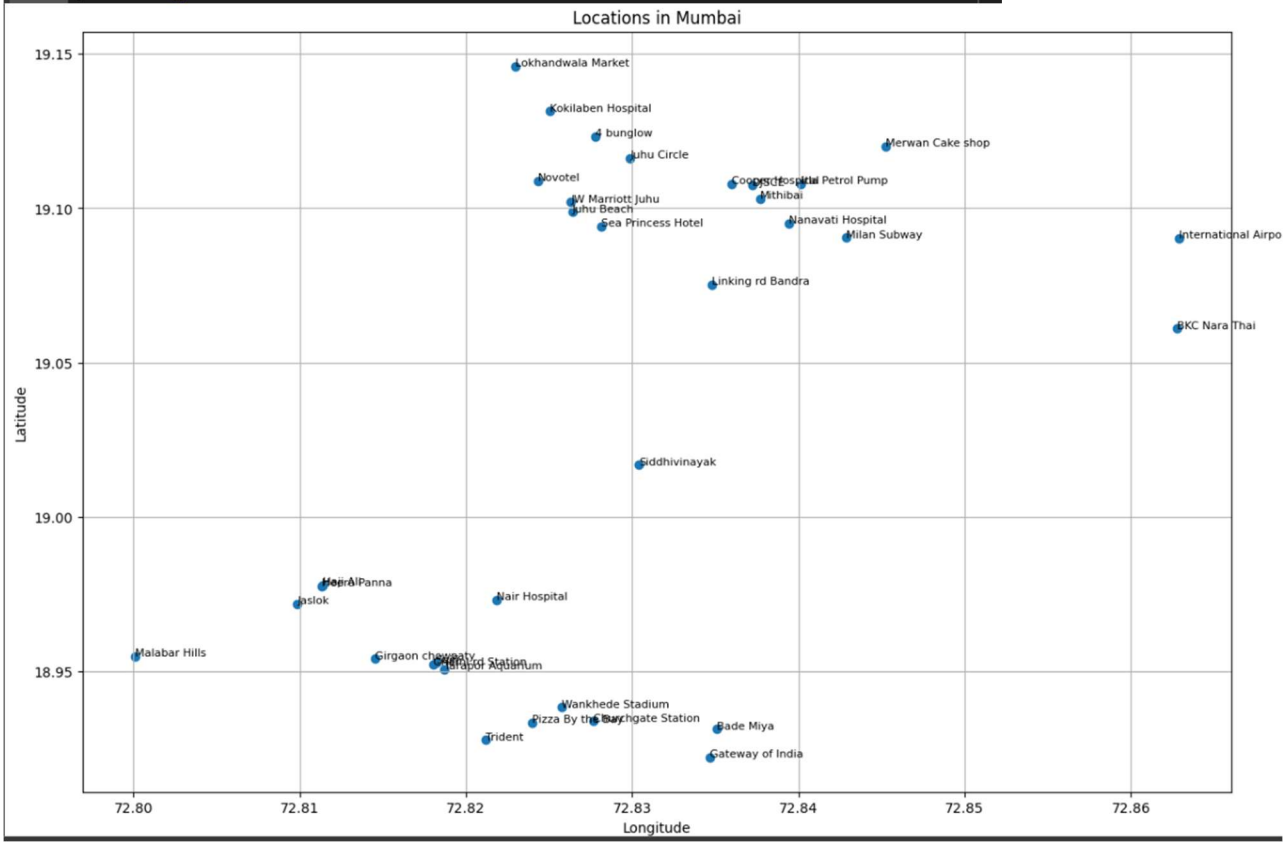
Lab Assignment to do:
Solve the following Shortest Path Algorithm:
1. Consider 40 -45 geo locations between Juhu beach till Gateway of India. 5-8 locations not in the route and choice of location should keep in mind two different possible path.
2. Find the shortest path between Juhu beach till Gateway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

```
[2] import csv
    import pandas as pd
    import numpy as np
```

```
[3] df=pd.read_csv("/content/Lab 6 A star - Sheet.csv")
    df.head(5)
```

	Location	Latitude	Longitude
0	DJSCE	19.107462	72.837200
1	Cooper Hospital	19.107928	72.835984
2	Mithibai	19.103075	72.837716
3	Irla Petrol Pump	19.107881	72.840121
4	Merwan Cake shop	19.120065	72.845228

```
import matplotlib.pyplot as plt
plt.figure(figsize=(15, 10))
plt.scatter(df['Longitude'], df['Latitude'])
for i, row in df.iterrows():
    plt.annotate(row['Location'], (row['Longitude'], row['Latitude']), fontsize=8)
plt.xlabel('Longitude')
plt.ylabel('Latitude')
plt.title('Locations in Mumbai')
plt.grid(True)
plt.show()
```



[5]
```
import math
def haversine_distance(lat1, lon1, lat2, lon2):
    lat1, lon1, lat2, lon2 = map(math.radians, [lat1, lon1, lat2, lon2])
    dlat = lat2 - lat1
    dlon = lon2 - lon1
    a = math.sin(dlat/2)**2 + math.cos(lat1) * math.cos(lat2) * math.sin(dlon/2)**2
    c = 2 * math.atan2(math.sqrt(a), math.sqrt(1 - a))
    radius = 6371
    distance = radius * c
    return round(distance,2)
```

```
location=df["Location"].values
lat=df["Latitude"].values
lon=df["Longitude"].values
haversine_distance(lat[0], lon[0], lat[0], lon[0])
```

```
0.0
```

[7]
```
print(location)
```

```
['DJSCE' 'Cooper Hospital' 'Mithibai' 'Irla Petrol Pump'
 'Merwan Cake shop' 'Juhu Circle' '4 bunglow' 'Kokilaben Hospital'
 'Lokhandwala Market' 'JW Marriott Juhu' 'Juhu Beach' 'Sea Princess Hotel'
 'Novotel' 'Linking rd Bandra' 'Heera Panna' 'Nanavati Hospital'
 'Milan Subway' 'Siddhivinayak' 'Haji Ali' 'Jaslok ' 'BKC Nara Thai'
 'Bade Miya' 'Gateway of India' 'Saifi' 'International Airport'
 'Tarapor Aquarium' 'Nair Hospital' 'Malabar Hills' 'Charni rd Station'
 'Girgaon chowpaty' 'Wankhede Stadium' 'Trident' 'Churchgate Station'
 'Pizza By the Bay']
```

```
cost_matrix=[]
for i in range(len(location)):
  a=[]
  for j in range(len(location)):
    a.append(haversine_distance(lat[i],lon[i],lat[j],lon[j]))
  cost_matrix.append(a)

for i in range(len(cost_matrix)):
    for j in range(len(cost_matrix[i])):
        print(f'{cost_matrix[i][j]:.2f}', end='\t')
    print()
```

33X33 MATRIX

5	3.59	14.71	1.40	1.97	10.09	14.68	15.36	5.81	19.59	20.61	17.33	3.30	17.55	15.04	17.42	17.40	17.24	18.85	20.06
2	3.64	14.74	1.48	2.06	10.13	14.71	15.39	5.91	19.64	20.66	17.36	3.43	17.59	15.07	17.45	17.43	17.28	18.90	20.10
1	3.11	14.24	0.91	1.49	9.60	14.21	14.89	5.35	19.11	20.12	16.85	3.00	17.07	14.56	16.96	16.92	16.77	18.37	19.58
5	3.67	14.82	1.43	1.95	10.16	14.78	15.47	5.71	19.65	20.66	17.41	3.08	17.63	15.12	17.54	17.48	17.33	18.92	20.13
8	5.10	16.25	2.85	3.29	11.57	16.22	16.90	6.80	21.02	22.03	18.83	3.79	19.05	16.53	18.98	18.90	18.76	20.32	21.54
1	4.58	15.55	2.56	3.16	11.03	15.52	16.20	7.03	20.57	21.58	18.22	4.50	18.45	15.94	18.22	18.29	18.12	19.79	20.98
5	5.39	16.31	3.37	3.97	11.82	16.27	16.95	7.82	21.36	22.37	19.00	5.19	19.22	16.72	18.96	19.06	18.89	20.57	21.75
1	6.32	17.17	4.31	4.91	12.73	17.14	17.82	8.75	22.27	23.28	19.88	6.04	20.11	17.61	19.81	19.95	19.76	21.47	22.65
2	7.96	18.78	5.92	6.50	14.36	18.74	19.42	10.31	23.90	24.91	21.50	7.47	21.72	19.23	21.39	21.56	21.37	23.09	24.26
3	3.11	13.94	1.58	2.16	9.47	13.91	14.59	5.94	19.01	20.02	16.63	4.06	16.86	14.35	16.61	16.70	16.52	18.21	19.39
2	2.77	13.60	1.43	1.96	9.12	13.56	14.24	5.67	18.66	19.67	16.28	3.95	16.51	14.00	16.26	16.35	16.17	17.86	19.04
3	2.21	13.09	1.19	1.59	8.58	13.06	13.74	5.16	18.12	19.13	15.76	3.67	15.99	13.48	15.78	15.83	15.66	17.33	18.52
0	3.89	14.67	2.20	2.81	10.23	14.64	15.31	6.66	19.78	20.78	17.37	4.54	17.60	15.10	17.32	17.44	17.26	18.96	20.14
0	0.00	11.15	2.26	1.90	6.50	11.12	11.80	3.33	16.01	17.02	13.74	3.40	13.96	11.45	13.89	13.81	13.66	15.26	16.47
67	11.15	0.00	13.41	13.01	4.83	0.03	0.65	10.76	5.71	6.62	2.85	13.68	3.08	1.21	2.79	2.91	2.64	4.61	5.62
0	2.26	13.41	0.00	0.62	8.73	13.37	14.06	4.50	18.22	19.23	15.99	2.52	16.21	13.69	16.14	16.06	15.91	17.49	18.70
1	1.90	13.01	0.62	0.00	8.29	12.97	13.66	3.88	17.73	18.74	15.55	2.10	15.77	13.25	15.76	15.62	15.48	17.03	18.25
23	6.50	4.83	8.73	8.29	0.00	4.80	5.47	5.97	9.54	10.55	7.26	8.85	7.48	4.97	7.62	7.33	7.21	8.76	9.97
64	11.12	0.03	13.37	12.97	4.80	0.00	0.68	10.73	5.74	6.65	2.88	13.64	3.11	1.22	2.82	2.94	2.67	4.64	5.66
31	11.80	0.65	14.06	13.66	5.47	0.68	0.00	11.39	5.23	6.10	2.30	14.32	2.53	1.27	2.15	2.35	2.05	4.08	5.04
5	3.33	10.76	4.50	3.88	5.97	10.73	11.39	0.00	14.73	15.73	12.94	3.25	13.14	10.70	13.54	13.01	12.96	14.21	15.47
78	16.01	5.71	18.22	17.73	9.54	5.74	5.23	14.73	0.00	1.01	2.97	17.93	2.76	4.85	4.52	2.93	3.32	1.26	1.52
78	17.02	6.62	19.23	18.74	10.55	6.65	6.10	15.73	1.01	0.00	3.81	18.94	3.58	5.82	5.14	3.76	4.11	2.02	1.55
47	13.74	2.85	15.99	15.55	7.26	2.88	2.30	12.94	2.97	3.81	0.00	16.02	0.23	2.30	1.92	0.07	0.40	1.79	2.79
3	3.40	13.68	2.52	2.10	8.85	13.64	14.32	3.25	17.93	18.94	16.02	0.00	16.22	13.74	16.46	16.09	16.00	17.36	18.61
60	13.96	3.08	16.21	15.77	7.48	3.11	2.53	13.14	2.76	3.58	0.23	16.22	0.00	2.51	2.01	0.18	0.57	1.56	2.56
10	11.45	1.21	13.69	13.25	4.97	1.22	1.27	10.70	4.85	5.82	2.30	13.74	2.51	0.00	3.06	2.36	2.26	3.89	5.04
32	13.89	2.79	16.14	15.76	7.62	2.82	2.15	13.54	4.52	5.14	1.92	16.46	2.01	3.06	0.00	1.91	1.53	3.26	3.74
44	13.81	2.91	16.06	15.62	7.33	2.94	2.35	13.01	2.93	3.76	0.07	16.09	0.18	2.36	1.91	0.00	0.42	1.74	2.73
26	13.66	2.64	15.91	15.48	7.21	2.67	2.05	12.96	3.32	4.11	0.40	16.00	0.57	2.26	1.53	0.42	0.00	2.10	2.99
06	15.26	4.61	17.49	17.03	8.76	4.64	4.08	14.21	1.26	2.02	1.79	17.36	1.56	3.89	3.26	1.74	2.10	0.00	1.27
14	16.47	5.62	18.70	18.25	9.97	5.66	5.04	15.47	1.52	1.55	2.79	18.61	2.56	5.04	3.74	2.73	2.99	1.27	0.00
46	15.74	5.14	17.97	17.49	9.24	5.17	4.61	14.63	0.84	1.50	2.32	17.79	2.09	4.40	3.71	2.26	2.62	0.53	0.96
50	15.81	5.07	18.05	17.58	9.32	5.10	4.52	14.78	1.19	1.68	2.23	17.93	1.99	4.41	3.46	2.17	2.48	0.57	0.70

```
[9] location_costs = {}
    for i, locations in enumerate(location):
        location_costs[locations] = {}
        for j, cost in enumerate(cost_matrix[i]):
            location_costs[locations][location[j]] = cost
    print(location_costs)
```

```
{'DJSCE': {'DJSCE': 0.0, 'Cooper Hospital': 0.14, 'Mithibai': 0.49, 'Irla Petrol Pump': 0.31, 'Merwan Cake shop': 1.64, 'Juhu Circle': 1.24, '4 bung
```

```
heuristic_values = []
def cal_hn(distance ,goal):
    for i in range(len(distance)):
            heuristic_value = haversine_distance(lat[goal], lon[goal], lat[i], lon[i])
            heuristic_values.append(round(heuristic_value,2))
    return (heuristic_values)

goal=int(input("Enter row no. of goal node"))
cal_hn(cost_matrix,goal)
print("loc\t\th(n)")
for i in range(len(location)):
  print(f"{location[i]}\t\t|{heuristic_values[i]}")
```

```
Enter row no. of goal node31
loc             h(n)
DJSCE           |20.06
Cooper Hospital         |20.1
Mithibai                |19.58
Irla Petrol Pump                |20.13
Merwan Cake shop                |21.54
Juhu Circle             |20.98
4 bunglow               |21.75
Kokilaben Hospital              |22.65
Lokhandwala Market              |24.26
JW Marriott Juhu                |19.39
Juhu Beach              |19.04
Sea Princess Hotel              |18.52
Novotel         |20.14
Linking rd Bandra               |16.47
Heera Panna             |5.62
Nanavati Hospital               |18.7
Milan Subway            |18.25
Siddhivinayak           |9.97
Haji Ali                |5.66
Jaslok          |5.04
BKC Nara Thai           |15.47
Bade Miya               |1.52
Gateway of India                |1.55
Saifi           |2.79
International Airport           |18.61
Tarapor Aquarium                |2.56
Nair Hospital           |5.04
Malabar Hills           |3.74
Charni rd Station               |2.73
Girgaon chowpaty                |2.99
Wankhede Stadium                |1.27
Trident         |0.0
Churchgate Station              |0.96
Pizza By the Bay                |0.7
```

```
[11] h_n = dict(zip(location, heuristic_values))
     print(h_n)
```

```
{'DJSCE': 20.06, 'Cooper Hospital': 20.1, 'Mithibai': 19.58, 'Irla Petrol Pump': 20.13, 'Merwan Cake sh
```

```
[12] openlist=[]
     closed=[]
```

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

```
def generate_successor_nodes(current_node, distance_matrix, heuristic_matrix):
    successors = []

    current_node_index = current_node
    for i in range(len(distance_matrix[current_node_index])):
        g_value = distance_matrix[current_node_index][i]
        h_value = heuristic_matrix[i]
        f_value = g_value + h_value
        successor_node = (i, g_value, h_value, f_value)
        successors.append(successor_node)
    return successors

current_node = 0
successors = generate_successor_nodes(current_node, cost_matrix, heuristic_values)

for successor in successors:
    print(f"Node {successor[0]} - g(n): {successor[1]}, h(n): {successor[2]}, f(n): {successor[3]}")
```

```
Node 0 - g(n): 0.0, h(n): 20.06, f(n): 20.06
Node 1 - g(n): 0.14, h(n): 20.1, f(n): 20.240000000000002
Node 2 - g(n): 0.49, h(n): 19.58, f(n): 20.069999999999997
Node 3 - g(n): 0.31, h(n): 20.13, f(n): 20.439999999999998
Node 4 - g(n): 1.64, h(n): 21.54, f(n): 23.18
Node 5 - g(n): 1.24, h(n): 20.98, f(n): 22.22
Node 6 - g(n): 2.02, h(n): 21.75, f(n): 23.77
Node 7 - g(n): 2.95, h(n): 22.65, f(n): 25.599999999999998
Node 8 - g(n): 4.53, h(n): 24.26, f(n): 28.790000000000003
Node 9 - g(n): 1.29, h(n): 19.39, f(n): 20.68
Node 10 - g(n): 1.48, h(n): 19.04, f(n): 20.52
Node 11 - g(n): 1.76, h(n): 18.52, f(n): 20.28
Node 12 - g(n): 1.36, h(n): 20.14, f(n): 21.5
Node 13 - g(n): 3.59, h(n): 16.47, f(n): 20.06
Node 14 - g(n): 14.71, h(n): 5.62, f(n): 20.330000000000002
Node 15 - g(n): 1.4, h(n): 18.7, f(n): 20.099999999999998
Node 16 - g(n): 1.97, h(n): 18.25, f(n): 20.22
Node 17 - g(n): 10.09, h(n): 9.97, f(n): 20.060000000000002
Node 18 - g(n): 14.68, h(n): 5.66, f(n): 20.34
Node 19 - g(n): 15.36, h(n): 5.04, f(n): 20.4
Node 20 - g(n): 5.81, h(n): 15.47, f(n): 21.28
Node 21 - g(n): 19.59, h(n): 1.52, f(n): 21.11
Node 22 - g(n): 20.61, h(n): 1.55, f(n): 22.16
Node 23 - g(n): 17.33, h(n): 2.79, f(n): 20.119999999999997
Node 24 - g(n): 3.3, h(n): 18.61, f(n): 21.91
Node 25 - g(n): 17.55, h(n): 2.56, f(n): 20.11
Node 26 - g(n): 15.04, h(n): 5.04, f(n): 20.08
Node 27 - g(n): 17.42, h(n): 3.74, f(n): 21.160000000000004
Node 28 - g(n): 17.4, h(n): 2.73, f(n): 20.13
Node 29 - g(n): 17.24, h(n): 2.99, f(n): 20.229999999999997
Node 30 - g(n): 18.85, h(n): 1.27, f(n): 20.12
Node 31 - g(n): 20.06, h(n): 0.0, f(n): 20.06
Node 32 - g(n): 19.33, h(n): 0.96, f(n): 20.29
Node 33 - g(n): 19.4, h(n): 0.7, f(n): 20.099999999999998
```

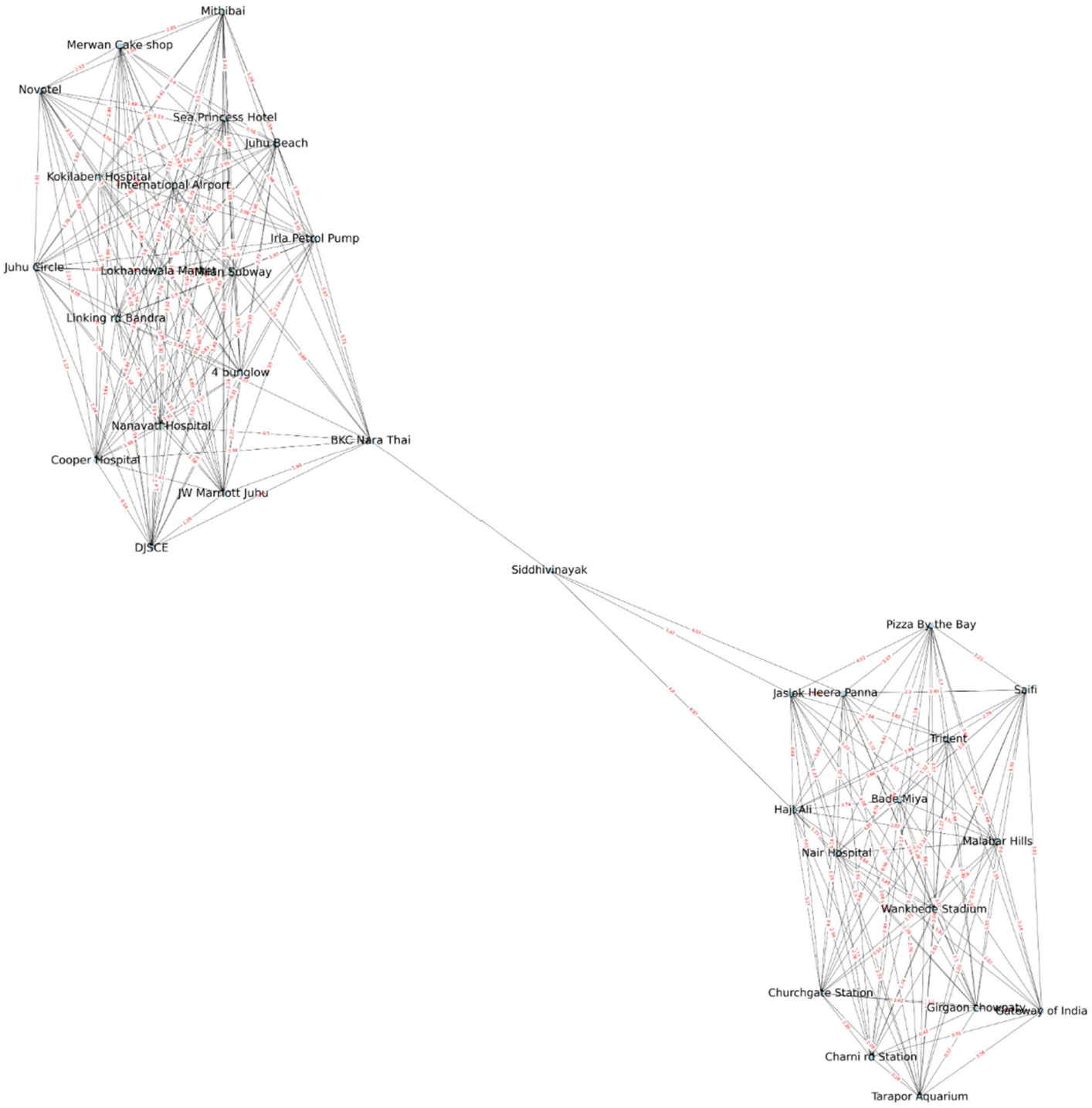
```
graph = {}
def create_graph(cost_matrix, locations, threshold_distance):
    for i, location in enumerate(locations):
        neighbors = {}
        for j in range(len(cost_matrix[i])):
            if i != j and cost_matrix[i][j] <= threshold_distance:
                neighbor_location = locations[j]
                neighbor_cost = cost_matrix[i][j]
                neighbors[neighbor_location] = neighbor_cost
        graph[location] = neighbors
    return graph
threshold_distance = 6.0
graph = create_graph(cost_matrix, location, threshold_distance)
graph
```

```
{'DJSCE': {'Cooper Hospital': 0.14,
  'Mithibai': 0.49,
  'Irla Petrol Pump': 0.31,
  'Merwan Cake shop': 1.64,
  'Juhu Circle': 1.24,
  '4 bunglow': 2.02,
  'Kokilaben Hospital': 2.95,
  'Lokhandwala Market': 4.53,
  'JW Marriott Juhu': 1.29,
  'Juhu Beach': 1.48,
  'Sea Princess Hotel': 1.76,
  'Novotel': 1.36,
  'Linking rd Bandra': 3.59,
  'Nanavati Hospital': 1.4,
  'Milan Subway': 1.97,
  'BKC Nara Thai': 5.81,
  'International Airport': 3.3},
 'Cooper Hospital': {'DJSCE': 0.14,
  'Mithibai': 0.57,
  'Irla Petrol Pump': 0.43,
  'Merwan Cake shop': 1.66,
  'Juhu Circle': 1.12,
  '4 bunglow': 1.91
```

```python
def a_star_movegen(graph, start_location, n):
    if start_location not in graph:
        print("Start location not found in the graph.")
        return

    open_list = [(0, start_location)]
    closed_list = set()
    parent = {start_location: None}

    while open_list:
        open_list.sort()  # Sort
        current_cost, current_location = open_list.pop(0)
        if current_location in closed_list:
            continue
        print(f"Current Location: {current_location}, Cost: {current_cost}, Parent: {parent[current_location]}")
        if len(closed_list) >= n:
            break
        for neighbor, cost in graph[current_location].items():
            if neighbor not in closed_list:
                total_cost = current_cost + cost
                open_list.append((total_cost, neighbor))
                parent[neighbor] = current_location
        closed_list.add(current_location)

start_location = 'DJSCE'
n = 2
a_star_movegen(graph, start_location, n)
```

```
Current Location: DJSCE, Cost: 0, Parent: None
Current Location: Cooper Hospital, Cost: 0.14, Parent: DJSCE
Current Location: Irla Petrol Pump, Cost: 0.31, Parent: Cooper Hospital
```

```python
import networkx as nx
import matplotlib.pyplot as plt

plt.figure(figsize=(100,100))
G = nx.DiGraph()
for node, edges in graph.items():
    G.add_node(node)
    for neighbor, weight in edges.items():
        G.add_edge(node, neighbor, weight=weight)

pos = nx.spring_layout(G, seed=1)
edge_weights = nx.get_edge_attributes(G, 'weight')
nx.draw(G, pos, with_labels=True, node_size=500, node_color='lightblue', font_size=50, font_color='black')
nx.draw_networkx_edge_labels(G, pos, edge_labels=edge_weights, font_size=20, font_color='red')
plt.title("Weighted Directed Graph")
plt.axis('off')
plt.show()
```


Mithibai
Merwan Cake shop
Novotel
Sea Princess Hotel
Juhu Beach
Kokilaben Hospital
International Airport
Irla Petrol Pump
Juhu Circle
Lokhandwala Market
Mean Subway
Linking rd Bandra
4 bunglow
Nanavati Hospital
BKC Nara Thai
Cooper Hospital
JW Marriott Juhu
DJSCE
Siddhivinayak
Pizza By the Bay
Jaslok Heera Panna
Saifi
Trident
Bade Miya
Haji Ali
Malabar Hills
Nair Hospital
Wankhede Stadium
Churchgate Station
Girgaon chowpaty
Gateway of India
Charni rd Station
Tarapor Aquarium

```
def Propogate_improvement(M,closed,g,k,h,parent,f):
    for X in graph(M,location,df):
        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(graph, start, goal, heuristic_dict):
    open_list = [(0, start)]
    closed_list = set()
    parent = {start: None}
    g_score = {node: float('inf') for node in graph}
    g_score[start] = 0

    while open_list:
        open_list.sort()
        f, current = open_list.pop(0)
        if current == goal:
            path = []
            while current is not None:
                path.append(current)
                current = parent[current]
            path_cost = 0
            for i in range(1, len(path)):
                path_cost += graph[path[i-1]][path[i]]
            return path[::-1], path_cost
        closed_list.add(current)

        for neighbor, cost in graph[current].items():
            if neighbor in closed_list:
                continue

            tentative_g = g_score[current] + cost
            if tentative_g < g_score[neighbor]:
                parent[neighbor] = current
                g_score[neighbor] = tentative_g
                h = heuristic_dict[neighbor]
```

```
                f = tentative_g + h

                in_open_list = False
                for i, (f_existing, existing) in enumerate(open_list):
                    if existing == neighbor:
                        in_open_list = True
                        if f < f_existing:
                            open_list[i] = (f, neighbor)
                        break
                if not in_open_list:
                    open_list.append((f, neighbor))
    return None, None

start = "DJSCE"
goal = "Trident"
path, path_cost = a_star(graph, start, goal, h_n)

if path:
    print("Path found:", path)
    print("Path cost:", path_cost)
else:
    print("No path found")
```

```
Path found: ['DJSCE', 'BKC Nara Thai', 'Siddhivinayak', 'Nair Hospital', 'Trident']
Path cost: 21.79
```