



Department of Computer Science and Engineering (Data Science)

Subject: Artificial Intelligence (DJ19DSC502)

AY: 2023-24

60009210105

Amitesh Sawarkar

D 12

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.



Department of Computer Science and Engineering (Data Science)

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.

$\text{successor.g} = \text{q.g} + \text{the calculated distance between the successor and the q.}$



Department of Computer Science and Engineering (Data Science)

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.



Department of Computer Science and Engineering (Data Science)

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

```
df=pd.read_csv("/content/Lab 6 A star - Sheet.csv")
df.tail()
```

	Location	Latitude	Longitude
29	Girgaon chowpaty	18.953888	72.814558
30	Wankhede Stadium	18.938276	72.825746
31	Trident	18.927725	72.821198
32	Churchgate Station	18.933876	72.827663
33	Pizza By the Bay	18.933419	72.823988

```
import matplotlib.pyplot as plt
plt.figure(figsize=(15, 10))
plt.scatter(df['Longitude'], df['Latitude'])

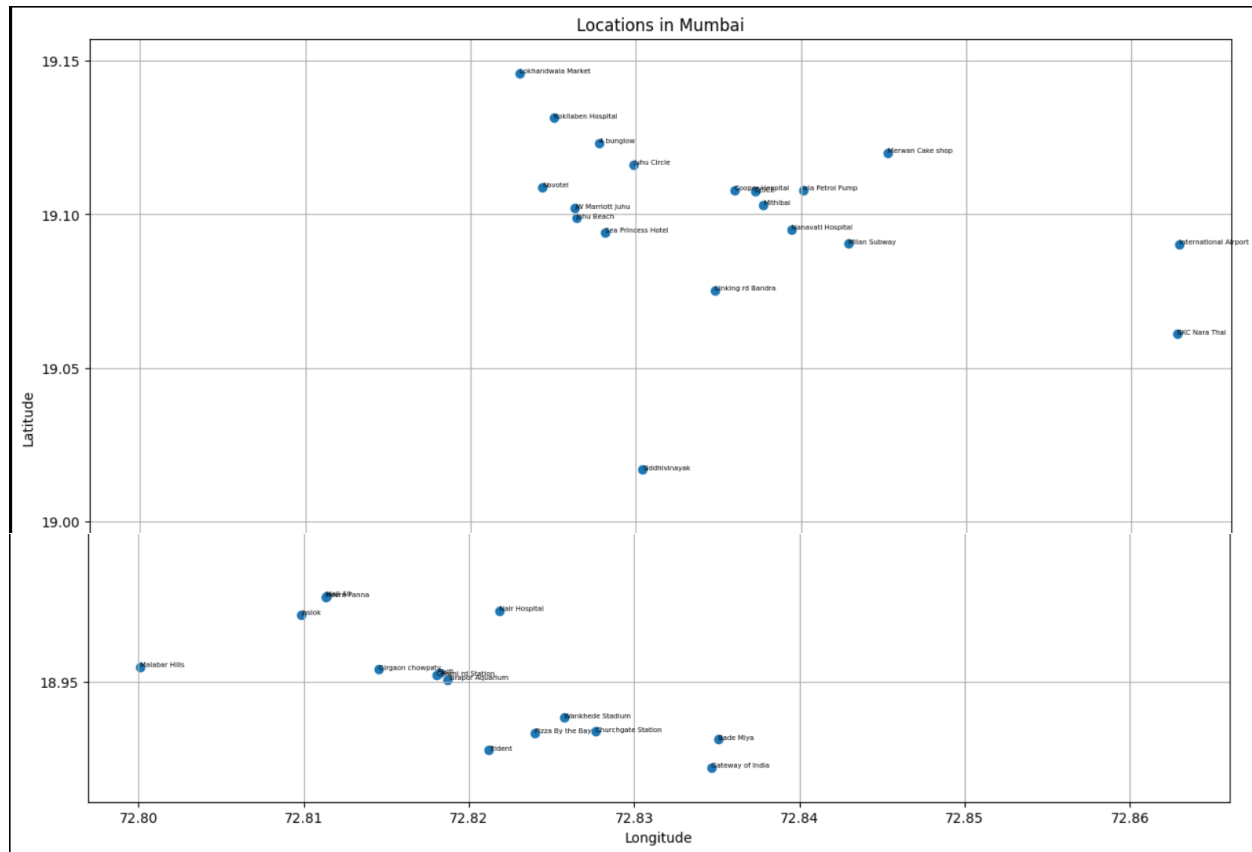
for i, row in df.iterrows():
    plt.annotate(row['Location'], (row['Longitude'], row['Latitude']), fontsize=5)

plt.xlabel('Longitude')
plt.ylabel('Latitude')
plt.title('Locations in Mumbai')

plt.grid(True)
plt.show()
```



Department of Computer Science and Engineering (Data Science)





Department of Computer Science and Engineering (Data Science)

```
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

```
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']
```

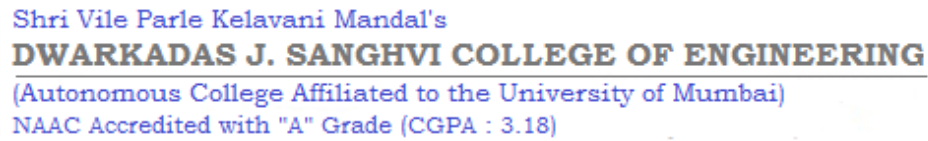


Department of Computer Science and Engineering (Data Science)

```
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()
```

0.00	0.14	0.49	0.31	1.64	1.24	2.02	2.95	4.53	1.29	1.48	1.76	1.36	3.59	14.71	1.40	1.97	10.09	14.68	15.36	5.81	19.59	20.61
0.14	0.00	0.57	0.43	1.66	1.12	1.91	2.85	4.44	1.21	1.42	1.74	1.23	3.64	14.74	1.48	2.06	10.13	14.71	15.39	5.91	19.64	20.66
0.49	0.57	0.00	0.59	2.05	1.68	2.47	3.41	5.01	1.21	1.28	1.41	1.54	3.11	14.24	0.91	1.49	9.60	14.21	14.89	5.35	19.11	20.12
0.31	0.43	0.59	0.00	1.46	1.42	2.14	3.05	4.60	1.59	1.76	1.98	1.66	3.67	14.82	1.43	1.95	10.16	14.78	15.47	5.71	19.65	20.66
1.64	1.66	2.05	1.46	0.00	1.67	1.86	2.46	3.71	2.82	3.08	3.40	2.53	5.10	16.25	2.85	3.29	11.57	16.22	16.90	6.80	21.02	22.03
1.24	1.12	1.68	1.42	1.67	0.00	0.82	1.76	3.39	1.62	1.96	2.46	1.01	4.58	15.55	2.56	3.16	11.03	15.52	16.20	7.03	20.57	21.58
2.02	1.91	2.47	2.14	1.86	0.82	0.00	0.94	2.57	2.37	2.72	3.24	1.65	5.39	16.31	3.37	3.97	11.82	16.27	16.95	7.82	21.36	22.37
2.95	2.85	3.41	3.05	2.46	1.76	0.94	0.00	1.64	3.26	3.61	4.15	2.51	6.32	17.17	4.31	4.91	12.73	17.14	17.82	8.75	22.27	23.28
4.53	4.44	5.01	4.60	3.71	3.39	2.57	1.64	0.00	4.89	5.25	5.79	4.13	7.96	18.78	5.92	6.50	14.36	18.74	19.42	10.31	23.90	24.91
1.29	1.21	1.21	1.59	2.82	1.62	2.37	3.26	4.89	0.00	0.35	0.90	0.78	3.11	13.94	1.58	2.16	9.47	13.91	14.59	5.94	19.01	20.02
1.48	1.42	1.28	1.76	3.08	1.96	2.72	3.61	5.25	0.35	0.00	0.56	1.13	2.77	13.60	1.43	1.96	9.12	13.56	14.24	5.67	18.66	19.67
1.76	1.74	1.41	1.98	3.40	2.46	3.24	4.15	5.79	0.90	0.56	0.00	1.68	2.21	13.09	1.19	1.59	8.58	13.06	13.74	5.16	18.12	19.13
1.36	1.23	1.54	1.66	2.53	1.01	1.65	2.51	4.13	0.78	1.13	1.68	0.00	3.89	14.67	2.20	2.81	10.23	14.64	15.31	6.66	19.78	20.78
3.59	3.64	3.11	3.67	5.10	4.58	5.39	6.32	7.96	3.11	2.77	2.21	3.89	0.00	11.15	2.26	1.90	6.50	11.12	11.80	3.33	16.01	17.02
14.71	14.74	14.24	14.82	16.25	15.55	16.31	17.17	18.78	13.94	13.60	13.09	14.67	11.15	0.00	13.41	13.01	4.83	0.03	0.65	10.76	5.71	6.62
1.40	1.48	0.91	1.43	2.85	2.56	3.37	4.31	5.92	1.58	1.43	1.19	2.20	2.26	13.41	0.00	0.62	8.73	13.37	14.06	4.50	18.22	19.23
1.97	2.06	1.49	1.95	3.29	3.16	3.97	4.91	6.50	2.16	1.96	1.59	2.81	1.90	13.01	0.62	0.00	8.29	12.97	13.66	3.88	17.73	18.74
10.09	10.13	9.60	10.16	11.57	11.03	11.82	12.73	14.36	9.47	9.12	8.58	10.23	6.50	4.83	8.73	8.29	0.00	4.80	5.47	5.97	9.54	10.55
14.68	14.71	14.21	14.78	16.22	15.52	16.27	17.14	18.74	13.91	13.56	13.06	14.64	11.12	0.03	13.37	12.97	4.80	0.00	0.68	10.73	5.74	6.65
15.36	15.39	14.89	15.47	16.90	16.20	16.95	17.82	19.42	14.59	14.24	13.74	15.31	11.80	0.65	14.06	13.66	5.47	0.68	0.00	11.39	5.23	6.10
5.81	5.91	5.35	5.71	6.80	7.03	7.82	8.75	10.31	5.94	5.67	5.16	6.66	3.33	10.76	4.50	3.88	5.97	10.73	11.39	0.00	14.73	15.73
19.59	19.64	19.11	19.65	21.02	20.57	21.36	22.27	23.90	19.01	18.66	18.12	19.78	16.01	5.71	18.22	17.73	9.54	5.74	5.23	14.73	0.00	1.01
20.61	20.66	20.12	20.66	22.03	21.58	22.37	23.28	24.91	20.02	19.67	19.13	20.78	17.02	6.62	19.23	18.74	10.55	6.65	6.10	15.73	1.01	0.00
17.33	17.36	16.85	17.41	18.83	18.22	19.00	19.88	21.50	16.63	16.28	15.76	17.37	13.74	2.85	15.99	15.55	7.26	2.88	2.30	12.94	2.97	3.81
3.30	3.43	3.00	3.08	3.79	4.50	5.19	6.04	7.47	4.06	3.95	3.67	4.54	3.40	13.68	2.52	2.10	8.85	13.64	14.32	3.25	17.93	18.94
...																						
18.85	18.90	18.37	18.92	20.32	19.79	20.57	21.47	23.09	18.21	17.86	17.33	18.96	15.26	4.61	17.49	17.03	8.76	4.64	4.08	14.21	1.26	2.02
20.06	20.10	19.58	20.13	21.54	20.98	21.75	22.65	24.26	19.39	19.04	18.52	20.14	16.47	5.62	18.70	18.25	9.97	5.66	5.04	15.47	1.52	1.55
19.33	19.37	18.84	19.39	20.79	20.27	21.06	21.96	23.58	18.70	18.35	17.82	19.46	15.74	5.14	17.97	17.49	9.24	5.17	4.61	14.63	0.84	1.50
19.40	19.45	18.92	19.47	20.87	20.33	21.11	22.01	23.63	18.75	18.40	17.87	19.50	15.81	5.07	18.05	17.58	9.32	5.10	4.52	14.78	1.19	1.68



Python

```

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\tth(n)")
for i in range(len(location)):
    print(f"{location[i]}\t\t\t{heuristic_values[i]}")

```


**Department of Computer Science and Engineering (Data Science)**

```

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
...
Wankhede Stadium      |1.27
Trident              |0.0
Churchgate Station     |0.96
Pizza By the Bay       |0.7

```

```

h_n = dict(zip(location, heuristic_values))
print(h_n)

```

Python

```
{'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,
```



Department of Computer Science and Engineering (Data Science)

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

```
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]}")
```



Department of Computer Science and Engineering (Data Science)

```
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 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
```



Department of Computer Science and Engineering (Data Science)

```
movegen = {}  
def create_movegen(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  
        movegen[location] = neighbors  
    return movegen  
  
threshold_distance = 6.0  
movegen = create_movegen(cost_matrix, location, threshold_distance)  
movegen
```



Department of Computer Science and Engineering (Data Science)

```
{'DJSCE': {'Cooper Hospital': 0.14,
'Mithibai': 0.49,
'Irla Petrol Pump': 0.31,
'Merwan Cake shop': 1.64,
'Juhu Circle': 1.24,
'4 bungalow': 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 bungalow': 1.91,
'Kokilaben Hospital': 2.85,
'Lokhandwala Market': 4.44,
...
'Charni rd Station': 2.17,
'Girgaon chowpaty': 2.48,
'Wankhede Stadium': 0.57,
'Trident': 0.7,
'Churchgate Station': 0.39}}
```



Department of Computer Science and Engineering (Data Science)

```
def Propagate_improvement(M,closed,g,k,h,parent,f):
    for X in movegen(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=Propagate_improvement(X,closed,g,k,h,parent,f)
            return g,f,parent
```

```
import heapq

def astar(graph, start, goal, heuristics):
    def GoalTestNode(node):
        return node == goal

    def ReconstructPath(came_from, current):
        path = []
        while current is not None:
            path.insert(0, current)
            current = came_from.get(current, None)
        return path

    def movegen(node):
        return graph[node]

    def PropagateImprovement(current, neighbor, g_score, came_from, open_list):
        tentative_g_score = g_score[current] + graph[current][neighbor]
        if tentative_g_score < g_score.get(neighbor, float('inf')):
            came_from[neighbor] = current
            g_score[neighbor] = tentative_g_score
            f_score = tentative_g_score + heuristics[neighbor]
            heapq.heappush(open_list, (f_score, neighbor))

    open_list = [(heuristics[start], start)]
    came_from = {}
    g_score = {node: float('inf') for node in graph}
    g_score[start] = 0
```



Department of Computer Science and Engineering (Data Science)

```
while open_list:
    _, current = heapq.heappop(open_list)

    if GoalTestNode(current):
        path = ReconstructPath(came_from, current)
        cost = g_score[current]+heuristics[current]
        return path, cost

    for neighbor in movegen(current):
        PropagateImprovement(current, neighbor, g_score, came_from, open_list)

return None, None

start_node = 'DJSCE'
goal_node = 'Trident'

path, cost = astar(movegen, start_node, goal_node, h_n)
if path is not None:
    print("Shortest Path:", path)
    print("Cost of Path:", cost)
else:
    print("No path found.")
```

```
Shortest Path: ['DJSCE', 'BKC Nara Thai', 'Siddhivinayak', 'Nair Hospital', 'Trident']
Cost of Path: 21.79
```