DEPARTMENT OF COMPUTER & SOFTWARE ENGINEERING

COLLEGE OF E&ME, NUST, RAWALPINDI

EC-350 Artificial Intelligence and Decision Support System

LAB MANUAL – 06

Course Instructor: Assoc Prof Dr Arsalan Shaukat

Lab Engineer: Kashaf Raheem

Student Name: ______________________________

Degree/ Syndicate: ______________________________

LAB # 6: A* SEARCH FOR GRAPHS

Lab Objective:

- To implement optimal A* algorithm for graphs using python

Hardware/Software required:

Hardware: Desktop/ Notebook Computer
Software Tool: Python 3.10.0

Lab Description:

A* is the best form of greedy first search that gives path with optimal cost for reaching goal from the start node.

Greedy Best First Search (GBFS) is based on greedy algorithm in which node that is closest to the goal node is expanded first. GBFS uses heuristic function for node evaluation. Since at any level, BFS expands a node with the lowest cost so we need to implement a priority queue to hold the nodes with their costs as priority.

h(n) = estimated cost of the cheapest path from the state at node n to a goal state

The pseudocode for the BFS is given below:

```
BFS(Graph g, sNode, goal, priority = 0, path = [])
Initialize a Priority Queue ‘PQ’ and insert sNode in it.
while PQ is not empty
    u = PQ.get()
    if u.value is the goal
        return path
    else
        foreach neighbor v of u
            foreach key in v
                if key not in path
                    Insert v in PQ
    Append u.value in path
return path
```

A* search not only checks the heuristic value at each node, but it also adds the path cost for reaching that node as well along with the heuristics through which the global optimal path is achieved. The shortest path is obtained through Dijkstra’s algorithm where the path cost at each node is calculated as:

$$f(n) = g(n) + h(n)$$

Where f(n) is the path cost at nth node, h(n) is the heuristic value of nth node to goal, g(n) is the cost for reaching nth node from start. The C for the A* search is given below:

```
AStar(graph, heuristic, startNode, goal):
    Initialize PriorityQueue
    Add ‘startNode’ in PriorityQueue with minimal priority
    Initialize a dictionary ‘cameFrom’ with the first key as startNode
    Initialize another dictionary ‘costSoFar’ with first key as startNode along with its cost
    while q is not empty:
        currentNode = PriorityQueue.get()
        foreach neighbor in graph[currentNode]:
           for key in neighbor.keys():
               newCost = costSoFar[currentNode] + costToReachNeighbor
               if key not in costSoFar or newCost < costSoFar[key]:
                   costSoFar[key] = [newCost]
                   priority = newCost + heuristics[key]
                   q.put((key,priority))
                   cameFrom[key] = [currentNode]
    return reconstructPath(cameFrom, startNode, goal)


reconstructPath(cameFrom, start, goal):
    current = goal
    initialize ‘path’ list
    while current is not start:
        append ‘current’ in path
        update ‘current’ to cameFrom[current]

    append ‘start’ in ‘path’

    reverse the ‘path’ list

    return path
```

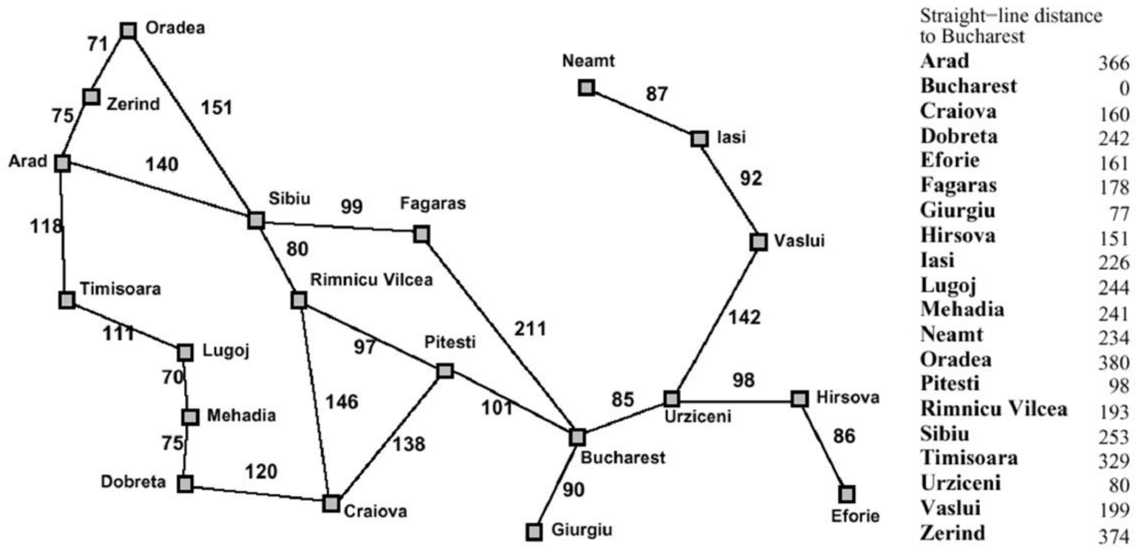
- CAMEFROM: Used to store path followed to access a certain node.
- COSTSOFAR: This dictionary will tell us the value of total costs from goal to the current node.

Lab Tasks:

1. Implement Priority Queue.

2. Implement A* algorithm in python for following graph:

Start Node: Arad, Goal: Bucharest



Straight−line distance
to Bucharest

Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	178
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	98
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

Graph 1

3. Write a script to decompose the given image into an undirected graph where the pixel represents the vertices and adjacent vertices are connected to each other via 4-connectivity and the cost on edges between adjacent nodes is their absolute intensity differences. The heuristic values should be calculated by taking the Manhattan distance for each pixel coordinates. The Manhattan distance between two points (x_0, y_0) and (x_1, y_1) can be calculated as:

$$D = |y_1 - y_0| + |x_1 - x_0|$$

Use A* algorithm to traverse decomposed image starting from pixel 150 to pixel 165.

150	2	5
80	145	45
74	102	165