AI LAB EXP – 5

BFS AND A* ALGORITHM FOR REAL WORLD PROBLEMS

Tejas Ashok

RA1911030010090

AIM

To implement Best First Algorithm and A* Algorithm using python.

BEST FIRST SEARCH

ALGORITHM

- Define a list, OPEN, consisting solely of a single node, the start node, s.
- IF the list is empty, return failure.
- Remove from the list the node n with the best score (the node where f is the minimum), and move it to a list, CLOSED.
- Expand node n.
- IF any successor to n is the goal node, return success and the solution (by tracing the path from the goal node to s).
- FOR each successor node:
    1. Apply the evaluation function, f, to the node.
    2. IF the node has not been in either list, add it to OPEN.
- Looping structure by sending the algorithm back to the second step.

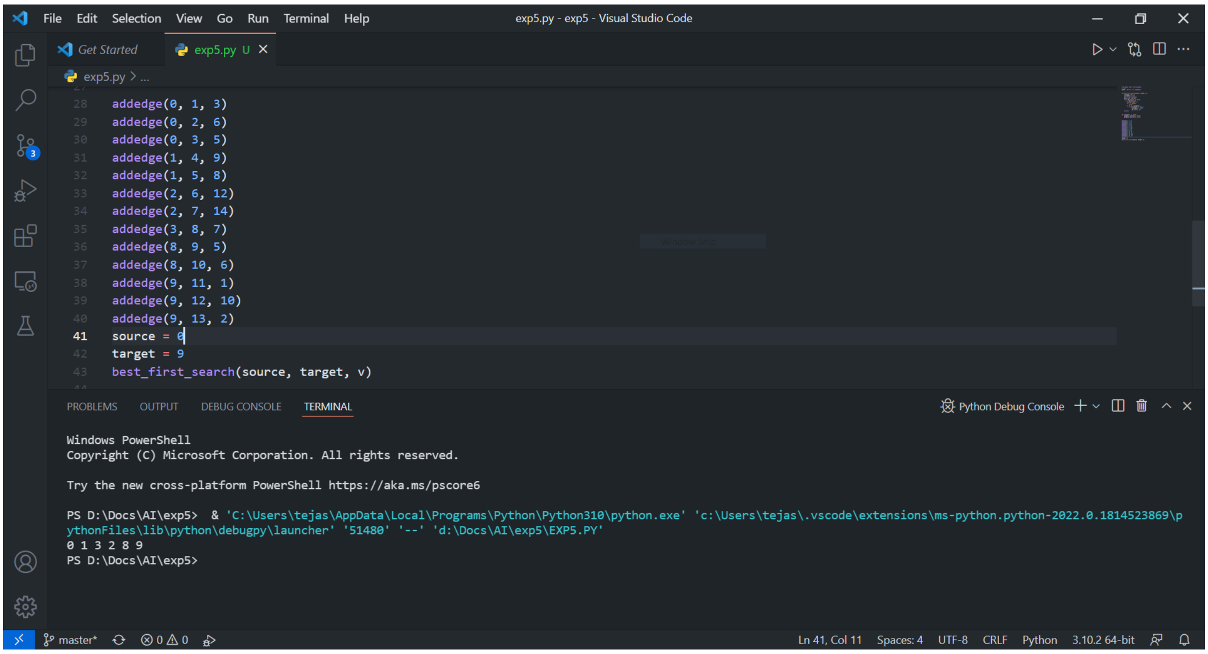
CODE

```
from queue import PriorityQueue

v = 14

graph = [[] for i in range(v)]



def best_first_search(source, target, n):

    visited = [0] * n

    visited[0] = True

    pq = PriorityQueue()
```

```python
    pq.put((0, source))
    while pq.empty() == False:
        u = pq.get()[1]
        print(u, end=" ")
        if u == target:
            break
        for v, c in graph[u]:
            if visited[v] == False:
                visited[v] = True
                pq.put((c, v))
    print()


def addedge(x, y, cost):
    graph[x].append((y, cost))
    graph[y].append((x, cost))



addedge(0, 1, 3)
addedge(0, 2, 6)
addedge(0, 3, 5)
addedge(1, 4, 9)
addedge(1, 5, 8)
addedge(2, 6, 12)
addedge(2, 7, 14)
addedge(3, 8, 7)
addedge(8, 9, 5)
addedge(8, 10, 6)
addedge(9, 11, 1)
addedge(9, 12, 10)
addedge(9, 13, 2)
source = 0
```

```
target = 9

best_first_search(source, target, v)
```

OUTPUT



A* SEARCH ALGORITHM

ALGORITHM

- We create two lists – Open List and Closed List (just like Dijkstra Algorithm)
- Initialize the open list
- Initialize the closed list put the starting node on the open list (you can leave its f at zero)
- While the open list is not empty
    1. Find the node with the least f on the open list, call it "q"
    2. Pop q off the open list
    3. Generate q's 8 successors and set their parents to q
    4. For each successor
        i. If successor is the goal, stop search
        ii. Else, compute both g and h for successor
            - successor.g = q.g + distance between successor and q
            - successor.h = distance from goal to successor(This can be done using many ways, we will discuss three heuristics- Manhattan, Diagonal and Euclidean Heuristics)
            - successor.f = successor.g + successor.h
        iii. If a node with the same position as successor is in the OPEN list which has a lower f than successor, skip this successor

iv. If a node with the same position as successor is in the CLOSED list which has a lower f than successor, skip this successor otherwise, add the node to the open list
end (for loop)

v. Push q on the closed list
end (while loop)

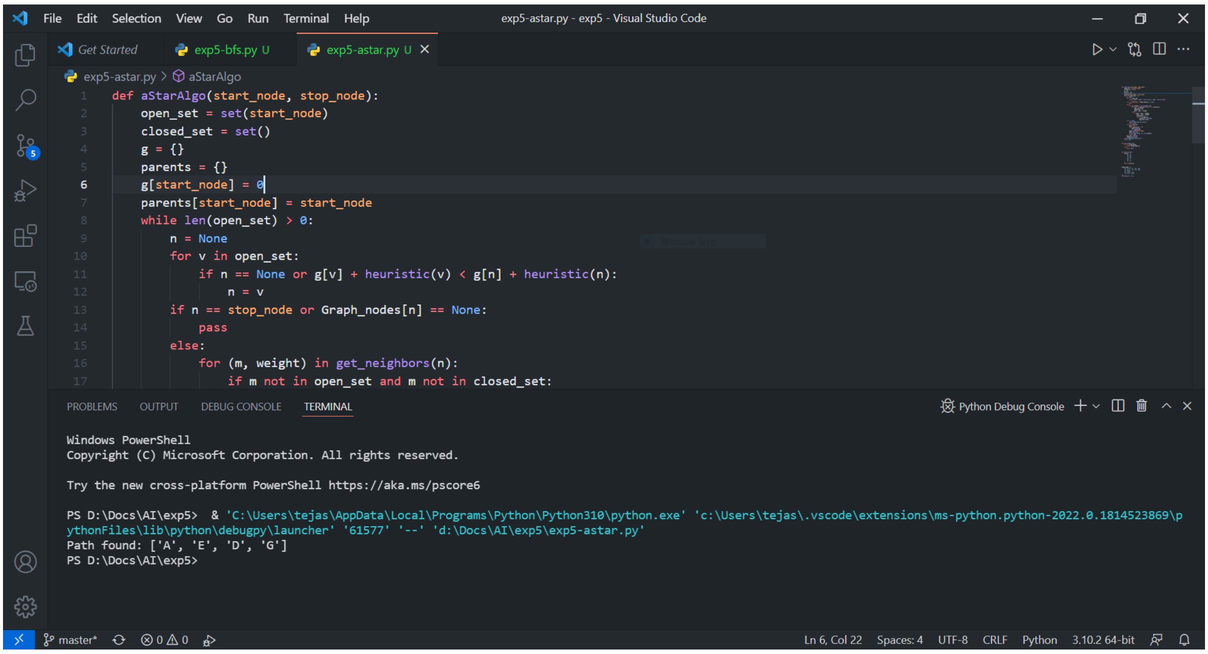
CODE

```
def aStarAlgo(start_node, stop_node):

    open_set = set(start_node)

    closed_set = set()

    g = {}

    parents = {}

    g[start_node] = 0

    parents[start_node] = start_node

    while len(open_set) > 0:

        n = None

        for v in open_set:

            if n == None or g[v] + heuristic(v) < g[n] + heuristic(n):

                n = v

        if n == stop_node or Graph_nodes[n] == None:

            pass

        else:

            for (m, weight) in get_neighbors(n):

                if m not in open_set and m not in closed_set:

                    open_set.add(m)

                    parents[m] = n

                    g[m] = g[n] + weight

                else:

                    if g[m] > g[n] + weight:

                        g[m] = g[n] + weight

                        parents[m] = n

                        if m in closed_set:
```

```
                            closed_set.remove(m)
                            open_set.add(m)
        if n == None:
            print('Path does not exist!')
            return None
        if n == stop_node:
            path = []
            while parents[n] != n:
                path.append(n)
                n = parents[n]
            path.append(start_node)
            path.reverse()
            print('Path found: {}'.format(path))
            return path
        open_set.remove(n)
        closed_set.add(n)
    print('Path does not exist!')
    return None


def get_neighbors(v):
    if v in Graph_nodes:
        return Graph_nodes[v]
    else:
        return None


def heuristic(n):
    H_dist = {
        'A': 11,
        'B': 6,
        'C': 99,
```

```
        'D': 1,
        'E': 7,
        'G': 0,
    }
    return H_dist[n]



Graph_nodes = {
    'A': [('B', 2), ('E', 3)],
    'B': [('C', 1), ('G', 9)],
    'C': None,
    'E': [('D', 6)],
    'D': [('G', 1)],
}
aStarAlgo('A', 'G')
```

OUTPUT



RESULT

Best first search and A* search algorithm were successfully executed in python.