Artificial Intelligence (18CSC305J)

Faculty : Helen Victoria A

Ex- 5 : Team Tesla 2.0

Name	Regno.	Email Id
Abhighyan B	RA1911033010002	ab2134@srmist.edu.in
Sanjana N B	RA1911033010016	sn8740@srmist.edu.in
Roehit Ranganathan	RA1911033010017	rr9344@srmist.edu.in
Venkata Naga Sai Ram Nomula	RA1911033010021	vn4903@srmist.edu.in
K. Dushyant Reddy	RA1911033010029	kr7119@srmist.edu.in

Experiment 5

Implementation of Best First Search and A* Algorithm

Problem Statement:

BFS: Best First Search is a searching algorithm which works on a set of defined rules. It makes use of the concept of priority queues and heuristic search. The objective of this algorithm is to reach the goal state or final state from an initial state by the shortest route possible.

A * Algorithm : It is a searching algorithm that searches for the shortest path between the initial and the final state.

Algorithm:

1. The A* algorithm will first calculate the cost to travel to neighboring nodes and choose the lowest cost.

2. We have to use different heuristic values as per the problem.
3. Starting with node A, calculate the g(x) and we get the heuristic value.
4. Similarly compute for other nodes.

Code:

Best First Search

```
from queue import PriorityQueue
import networkx as nx

def best_first_search(source, target, n):
    visited = [0] * n
    visited[source] = True
    pq = PriorityQueue()
    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))

G = nx.Graph()
v = int(input("Enter the number of nodes: "))
graph = [[] for i in range(v)]
e = int(input("Enter the number of edges: "))
print("Enter the edges along with their weights:")
for i in range(e):
```

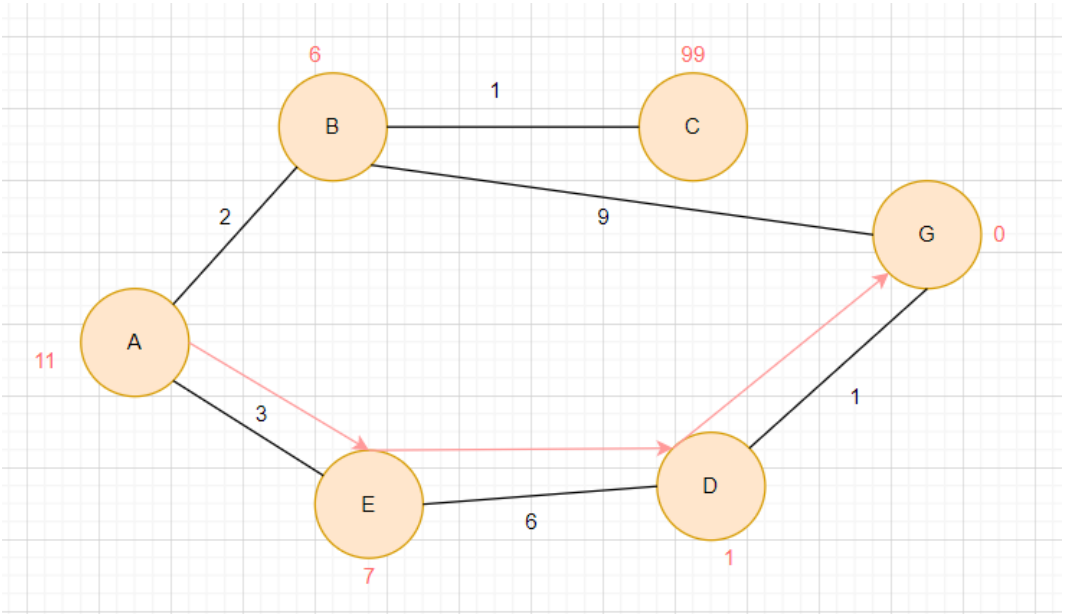
```
    x, y, z = list(map(int, input().split()))
    addedge(x, y, z)
    G.add_edge(x, y, weight = z)

source = int(input("Enter the Source Node: "))
target = int(input("Enter the Target/Destination Node: "))
print("\nPath: ", end = "")
best_first_search(source, target, v)
```

Time Complexity and Space Complexity :

Best First Search is simply Breadth First Search (BFS) but with the nodes re-ordered by their heuristic value. In the worst case, the time and space complexity for Best First Search are the same as that of BFS: O(b^(d+1)) for Time and O(b^d) for Space.

A*

Graph



```
def Aalgo(snode, enode):

        openst=set(snode)
        closest=set()
        g= {}
        prnt= {}

        g[snode]= 0
        prnt[snode]=snode


        while len(openst) > 0:
            n=None

            for vst in openst:
                if n == None or g[vst] + heuristic(vst) < g[n] +
heuristic(n):
                    n = vst


            if n == enode or Gnodes[n] == None:
                pass
            else:
                for (m, weight) in get_neighbors(n):
                    if m not in openst and m not in closest:
                        openst.add(m)
                        prnt[m] = n
                        g[m] = g[n] + weight


                    else:
                        if g[m] > g[n] + weight:
                            g[m] = g[n] + weight
                            prnt[m] = n

                            if m in closest:
                                closest.remove(m)
                                openst.add(m)

            if n == None:
```

```
                print('Path does not exist!')
                return None

            if n == enode:
                path = []

                while prnt[n] != n:
                    path.append(n)
                    n = prnt[n]

                path.append(snode)

                path.reverse()

                print('Path found: {}'.format(path))
                return path



            openst.remove(n)
            closest.add(n)

        print('Path does not exist!')
        return None


def get_neighbors(vst):
    if vst in Gnodes:
        return Gnodes[vst]
    else:
        return None

def heuristic(n):
        H_dist = {
            'A': 11,
            'B': 6,
            'C': 99,
            'D': 1,
            'E': 7,
            'G': 0,
```

```
        }

        return H_dist[n]

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

}
Aalgo('A', 'G')
```

Time Complexity:

The time complexity of A* Search Algorithm depends on the heuristic. In the worst case of an unbounded search space, the number of nodes expanded is exponential in the depth of the solution (the shortest path) d: O(b^d), where b is the branching factor (the average number of successors per state). This assumes that a goal state exists at all, and is reachable from the start state; if it is not, and the state space is infinite, the algorithm will not terminate.

Space Complexity

The space complexity of A* Search Algorithm is roughly the same as that of all other graph search algorithms i.e. O(b^d), as it keeps all generated nodes in memory.

Output:

Best First Search

```
Enter the number of nodes: 5
Enter the number of edges: 4
Enter the edges along with their weights:
0 1 3
0 2 6
0 3 5
1 4 9
Enter the Source Node: 0
Enter the Target/Destination Node: 9

Path: 0 1 3 2 4
```

A*

```
Path found: ['A', 'E', 'D', 'G']

['A', 'E', 'D', 'G']
```

Real World Solution:

- Its more advanced variants have been used in such applications as web crawlers
- Video games
- Traffic Navigational System

1. Understand its algorithmic properties

A* is one of the most successful search algorithms to find the shortest path between nodes or graphs.

2. Compare both the algorithm in terms of advantages and disadvantages

In best first search algorithm system moves to the next state based on heuristics function , the lowest heuristic value is chosen , however in A* algorithm the next state depends on the heuristic as well as the g component which is the path from initial to particular state.

The best first algorithm does not consider the cost of the path to a particular state. All it cares about is that which next state from current state has the lowest heuristics.The A* algorithm but considers the cost of going to that state along with the heuristic**.**A* also allows going back to a previous state however in best first search the decision is final.

3. State the real life applications of A* Algorithm

It finds applications in diverse problems, including the problem of parsing using stochastic grammars in NLP. Pathfinding problem in applications such as video games

4. Also Compare Dijkstra's and A*

A* is just like Dijkstra, the only difference is that A* tries to look for a better path by using a heuristic function which gives priority to nodes that are supposed to be better than others while Dijkstra's just explores all possible paths.

Result: BFS and A* algorithms were successfully implemented.