EXPERIMENT NO.5 **A N V SREEVISHNU**

DATE: 5/03/2021 **RA1811003010333**

<u>IMPLEMENTING BFS and A* SEARCH</u>

AIM: To implement BFS and A* Search using python.

ALGORITHM:

Create a graph of your choice.

Input the values accordingly into the slots of graph

Run the function so the minimum path is found

Output is given with the shortest path and distance

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()

    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))
```

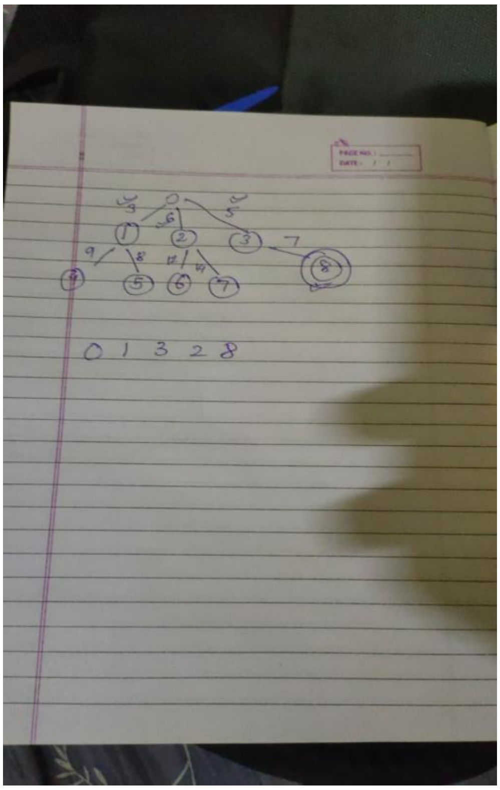
```
    print()
def addedge(x, y, cost):
    graph[x].append((y, cost))
    graph[y].append((x, cost))
addedge(0, 1, 3)
addedge(0, 2, 6)
addedge(0, 3, 5)
addedge(1, 4, 9)
addedge(1, 5, 8)
addedge(2, 6, 12)
addedge(2, 7, 14)
addedge(3, 8, 7)
source = 0
target = 8
best_first_search(source, target, v)
```

OUTPUT:

main.py

```
        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)

source = 0
```

```
0 1 3 2 8
```



A* Search

CODE:

```
def aStarAlgo(start_node, stop_node):
    open_set = set(start_node)
    closed_set = set()
    g = {} #store distance from starting node
    parents = {}# parents contains an adjacency map of all nodes
    #ditance of starting node from itself is zero
    g[start_node] = 0
    #start_node is root node i.e it has no parent nodes
    #so start_node is set to its own parent node
    parents[start_node] = start_node
    while len(open_set) > 0:
      n = None
```

```
#node with lowest f() is found
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):
        #nodes 'm' not in first and last set are added to first
        #n is set its parent
        if m not in open_set and m not in closed_set:
            open_set.add(m)
            parents[m] = n
            g[m] = g[n] + weight



        #for each node m,compare its distance from start i.e g(m) to the
        #from start through n node
        else:
            if g[m] > g[n] + weight:
                #update g(m)
                g[m] = g[n] + weight
                #change parent of m to n
                parents[m] = n

                #if m in closed set,remove and add to open
                if m in closed_set:
                    closed_set.remove(m)
                    open_set.add(m)

if n == None:
```

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

        # if the current node is the stop_node
        # then we begin reconstructin the path from it to the start_node
        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

        # remove n from the open_list, and add it to closed_list
        # because all of his neighbors were inspected
        open_set.remove(n)
        closed_set.add(n)

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

#define fuction to return neighbor and its distance
#from the passed node
def get_neighbors(v):
  if v in Graph_nodes:
    return Graph_nodes[v]
  else:
    return None
#for simplicity we ll consider heuristic distances given
#and this function returns heuristic distance for all nodes
def heuristic(n):
```

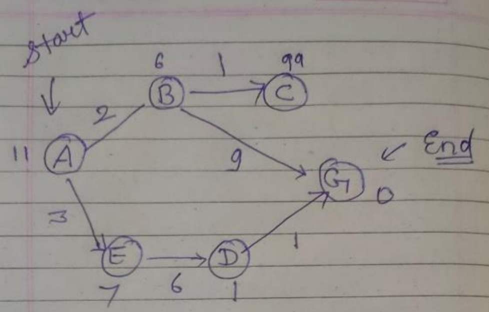
```
        H_dist = {
            'A': 11,
            'B': 6,
            'C': 99,
            'D': 1,
            'E': 7,
            'G': 0,

        }

        return H_dist[n]
#Describe your graph here
Graph_nodes = {
    'A': [('B', 2), ('E', 3)],
    'B': [('C', 1),('G', 9)],
    'C': None,
    'E': [('D', 6)],
    'D': [('G', 1)],

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

OUTPUT:

```
    else:
        return None
#for simplicity we ll consider heuristic distances given
#and this function returns heuristic distance for all nodes
def heuristic(n):
        H_dist = {
            'A': 11,
            'B': 6,
            'C': 99,
            'D': 1,
            'E': 7,
            'G': 0,

        }

        return H_dist[n]

#Describe your graph here
Graph_nodes = {
    'A': [('B', 2), ('E', 3)],
    'B': [('C', 1),('G', 9)],
    'C': None,
    'E': [('D', 6)],
    'D': [('G', 1)],

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

input

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



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

$A = 0+11 = 11$

$A \rightarrow B = 2+6 = 8$ ✓

$A \rightarrow E = 3+6 \; 9$

$A \rightarrow B \rightarrow C = (2+1)+99 = 102$

$A \rightarrow B \rightarrow G = (2+9)+0 = 11$ ✓

Here A→B→G has least count but it is still more than cost A→E.

$A \rightarrow E \rightarrow D = (3+6)+1 = 10$

✓ $A \rightarrow E \rightarrow D \rightarrow G = (3+6+1)+0 = 10$

RESULT: Thus, implementation of BFS and A* Search has been done successfully.