Name: Niraj Milind Amrutkar

Roll No. 332002

PRN: 22010910

Batch B1

Assignment No. 2

Best First Search:

Aim:

Implementation of best first search (not always optimal)

Theory:

Best-first search is a class of search algorithms, which explore a graph by expanding the most promising node chosen according to a specified rule the best-first search as estimating the promise of *n* by a "heuristic evaluation function which, in general, may depend on the description of *n*, the description of the goal, the information gathered by the search up to that point, and most importantly, on any extra knowledge about the problem domain."

Some authors have used "best-first search" to refer specifically to a search with a heuristic that attempts to predict how close the end of a path is to a solution (or, goal), so that paths which are judged to be closer to a solution (or, goal) are extended first. This specific type of search is called *greedy best-first search* or *pure heuristic search*.

Efficient selection of the current best candidate for extension is typically implemented using a priority queue.

Steps

- Step 1: Place the starting node into the OPEN list.
- Step 2: If the OPEN list is empty, Stop and return failure.
- Step 3: Remove the node n, from the OPEN list which has the lowest value of h(n), and places it in the CLOSED list.
- Step 4: Expand the node n, and generate the successors of node n.
- Step 5: Check each successor of node n, and find whether any node is a goal node or not. If any successor node is goal node, then return success and terminate the search, else proceed to Step 6.

- Step 6: For each successor node, algorithm checks for evaluation function f(n), and then check if the node has been in either OPEN or CLOSED list. If the node has not been in both list, then add it to the OPEN list.
- Step 7: Return to Step 2.

Advantages:

- Best first search can switch between BFS and DFS by gaining the advantages of both the algorithms.
- This algorithm is more efficient than BFS and DFS algorithms.

Disadvantages:

- It can behave as an unguided depth-first search in the worst case scenario.
- It can get stuck in a loop as DFS.
- This algorithm is not optimal

Code:

```
from queue import PriorityQueue
v = 6
graph = [[] for i in range(v)]

def bestFirst(sr, t, n):
    vis = [False] * n
    pq = PriorityQueue()
    pq.put((0, sr))
    vis[sr] = True

    while pq.empty() != True:
        u = pq.get()[1]
        print(u, end=" ")
        if u == t:
            break

        for v, c in graph[u]:
            if vis[v] == False:
                vis[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, 10)
addedge(0, 2, 7)
addedge(0, 3, 3)
addedge(1, 2, 17)
addedge(3, 5, 26)
addedge(2, 4, 5)


source = 0
t = 9
print("Best First of Tree : ")
bestFirst(source, t, v)


# Niraj Amrutkar
```

Output:

```
C:\Users\ADITYA\Desktop\Python>python -u "c:\Users\ADITYA\Desktop\Python\BestFirst.py"
Best First of Tree :
0 3 2 4 1 5
```