3. Best First Search

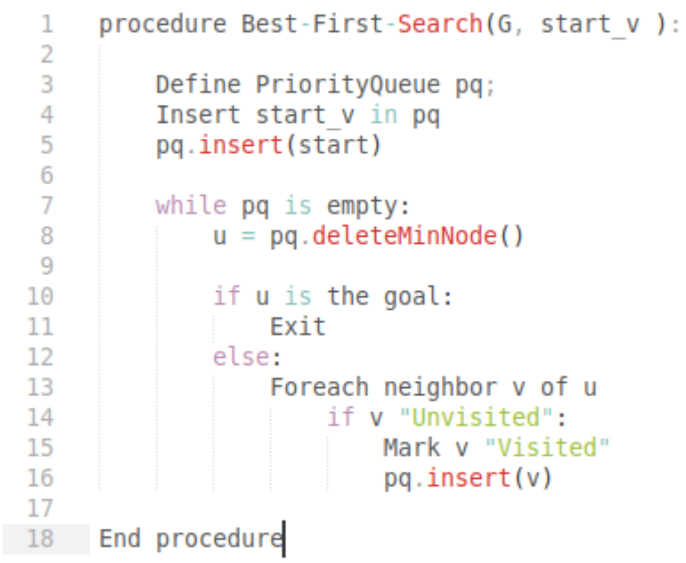
01/08/2019

INTRODUCTION

Best First Search is an informed search technique. Unlike BFS and DFS where we blindly explore paths without keeping a cost function in mind, here, we use a priority queue to store node costs. In Best first search a node is selected for expansion based on an evaluation function f(n). Traditionally, the node which is the lowest evaluation is selected for the expansion because the evaluation measures the distance to the goal. Best first search can be implemented within the general search framework via a priority queue, a data structure that will maintain the fringe in ascending order of function f(n) values.

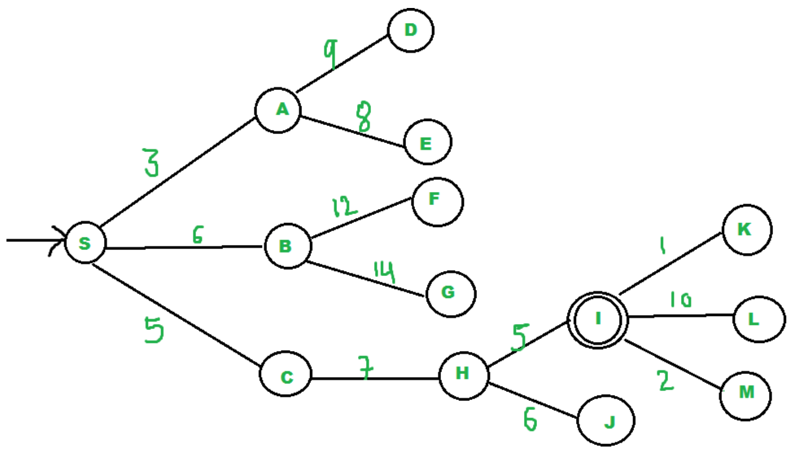
Let's understand BFS Heuristic Search through pseudocode.

ALGORITHM OF BEST FIRST SEARCH



```
procedure Best-First-Search(G, start_v ):

    Define PriorityQueue pq;
    Insert start_v in pq
    pq.insert(start)

    while pq is empty:
        u = pq.deleteMinNode()

        if u is the goal:
            Exit
        else:
            Foreach neighbor v of u
                if v "Unvisited":
                    Mark v "Visited"
                    pq.insert(v)

End procedure
```

IMPLEMENTATION USING DATA STRUCTURE



1. We start from source ‘S’ and search for goal ‘I’ using given costs and Best First search. We define the priority queue as pq.
2. pq initially contains ‘S’ We remove s from and process unvisited neighbours of ‘S’ to pq. pq now contains {A, C, B} (C is put before B because C has lesser cost).
3. We remove A from pq and process unvisited neighbours of ‘A’ to pq. pq now contains {C, B, E, D}.
4. We remove ‘C’ from pq and process unvisited neighbours of ‘C’ to pq. pq now contains {B, H, E, D}.
5. We remove B from pq and process unvisited neighbours of ‘B’ to pq. pq now contains {H, E, D, F, G}.
6. We remove H from pq. Since our goal ‘I’ is a neighbour of H, we return.

PYTHON IMPLEMENTATION OF BEST FIRST SEARCH

```
from queue import PriorityQueue

class Graph:

    def __init__(self, num):
        self.graph = [[] for i in range(num)]

    def addEdge(self, u, v, cost):
        self.graph[u].append((v, cost))
        self.graph[v].append((u, cost))

    def bestFirstSearch(self, start, end):
        visited = [False for i in self.graph]
        queue = PriorityQueue()
        queue.put((-1, start))
        visited[start] = True
        path = []

        while True:
            if not queue.empty():
                curr_node = queue.get()[1]
            else:
                break
            path.append(curr_node)
            if curr_node == end:
                print("Path found from %d to %d (%s)" % (
                    start, end, ' -> '.join(map(str, path))
                ))
                break
            for node, cost in self.graph[curr_node]:
                if not visited[node]:
                    visited[node] = True
                    queue.put((cost, node))

if __name__ == "__main__":
    graph = Graph(14)
    graph.addEdge(0, 1, 3)
    graph.addEdge(0, 2, 6)
    graph.addEdge(0, 3, 5)
    graph.addEdge(1, 4, 9)
    graph.addEdge(1, 5, 8)
    graph.addEdge(2, 6, 12)
    graph.addEdge(2, 7, 14)
    graph.addEdge(3, 8, 7)
    graph.addEdge(8, 9, 5)
    graph.addEdge(8, 10, 6)
    graph.addEdge(9, 11, 1)
    graph.addEdge(9, 12, 10)
    graph.addEdge(9, 13, 2)

    graph.bestFirstSearch(0, 9)
```

Output : Path found from 0 to 9 -> 0, 3, 8, 9

APPLICATIONS OF BEST FIRST SEARCH

1. **Web Crawler:** In a web crawler, each web page is treated as a node, and all the hyperlinks on the page are treated as unvisited successor nodes. A crawler that uses best-first search generally uses an evaluation function that assigns priority to links based on how closely the contents of their parent page resemble the search query (Menczer, Pant, Ruiz, and Srinivasan, 2001).
2. **Games:** In games, best-first search may be used as a path-finding algorithm for game characters. For example, it could be used by an enemy agent to find the location of the player in the game world. Some games divide up the terrain into "tiles" which can either be blocked or unblocked.
3. **Energy Demand control system:** The Best First Search algorithm with appropriate heuristic function is employed to determine the best combination of these household appliances.

ADVANTAGES AND DISADVANTAGES OF BEST FIRST SEARCH

Advantages:

1. It is more efficient than that of BFS and DFS.
2. Time complexity of Best first search is much less than Breadth first search.
3. The Best first search allows us to switch between paths by gaining the benefits of both breadth first and depth first search.

Disadvantage:

1. Sometimes, it covers more distance than our consideration.

COMPLEXITY OF BEST FIRST SEARCH

Best First Search's worst-case time complexity is O(n*log(n)), where n is number of nodes. In the worst scenario, before we achieve the objective, we may have to check all nodes. Note that using Min (or Max) heap, the priority queue is implemented and inserting and removing activities takes O(log(n)) moment.

CONCLUSION

Best first search serves as a combination of depth first and breadth first search algorithm. Best first search algorithm is often referred greedy algorithm this is because they quickly attack the most desirable path as soon as it's heuristic weight becomes the most desirable.