

Assignment No 02: Best-First search in Graph representation Of Problems

CSE-0408 Summer 2021

Sheikh Afrin

Department of Computer Science and Engineering
State University of Bangladesh (SUB)
Dhaka, Bangladesh
sheikhafrin2016@gmail.com

Abstract—Breadth-first search (BFS) is an algorithm for searching a tree data structure for a node that satisfies a given property. It starts at the tree root and explores all nodes at the present depth prior to moving on to the nodes at the next depth level. Extra memory, usually a queue, is needed to keep track of the child nodes that were encountered but not yet explored.

Index Terms—C++,Python

I. INTRODUCTION

Breadth First Search (BFS) is an algorithm for traversing or searching layerwise in tree or graph data structures. If we consider searching as a form of traversal in a graph, an uninformed search algorithm would blindly traverse to the next node in a given manner without considering the cost associated with that step. An informed search, like Best first search, on the other hand would use an evaluation function to decide which among the various available nodes is the most promising (or 'BEST') before traversing to that node. The Best first search uses the concept of a Priority queue and heuristic search. To search the graph space, the BFS method uses two lists for tracking the traversal.

II. LITERATURE REVIEW

Best First Search is a merger of Breadth First Search . Best First Search is implemented using the priority queue, while Breadth First Search arrives at a solution without search guaranteed that the procedure does not get caught. Best First Search, being a mixer of these two, licenses exchanging between paths. At each stage the nodes among the created ones, the best appropriate node is chosen for facilitating expansion, might be this node have a place to a similar level or different, hence can flip between Depth First and Breadth First Search [3]. It is also known as greedy search. Time complexity is $O(bd)$ and space complexity is $O(bd)$, where b is branching factor and d is solution depth [2].

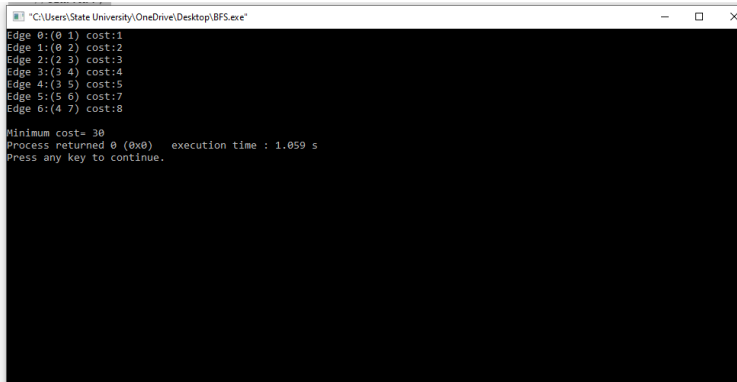
III. PROPOSED METHODOLOGY

1.Begin the search algorithm, by knowing the key which is to be searched. Once the key/element to be searched is decided the searching begins with the root (source) first.

IV. CODE

```
1 #include <iostream>
2 using namespace std;
3 int P,Q;
4 int parent[100];
5 int cost[1000][1000];
6 int find(int i)
7 {
8     while (parent[i] != -1)
9         i = parent[i];
10     return i;
11 }
12
13 void unioni(int i, int j)
14 {
15     int a = find(i);
16     int b = find(j);
17     parent[a] = b;
18 }
19
20 void BFSST()
21 {
22     int mincost = 0;
23     int edge_count = 0;
24     while (edge_count < P - 1)
25     {
26         int min = INT_MAX, a = -1, b = -1;
27         for (int i = 0; i < P; i++)
28             for (int j = 0; j < P; j++)
29             {
30                 if (find(i) != find(j) && cost[i][j] < min)
31                 {
32                     min = cost[i][j];
33                     a = i;
34                     b = j;
35                 }
36             }
37         unioni(a, b);
38         cout << "Edge " << edge_count << ": (" << a << " " << b << ") cost: " << min << endl;
39         mincost += min;
40         edge_count++;
41     }
42     cout << endl << "Minimum cost: " << mincost << endl;
43 }
44
45 int main()
46 {
47     // freopen("input.txt", "r", stdin);
48     // freopen("output.txt", "w", stdout);
49     int n;
50     cin >> n;
51     for (int i = 0; i < n; i++)
52     {
53         for (int j = 0; j < n; j++)
54             cost[i][j] = INT_MAX;
55     }
56     // for (int i = 0; i < n; i++)
57     // {
58     //     for (int j = 0; j < n; j++)
59     //         cost[i][j] = INT_MAX;
60     // }
61     // for (int i = 0; i < n; i++)
62     // {
63     //     for (int j = 0; j < n; j++)
64     //         cost[i][j] = INT_MAX;
65     // }
66     int p,q;
67     cin >> p;
68     cin >> q;
69     // cost[p][q] = 1;
70     // cost[q][p] = 1;
71     cost[p][p] = 0;
72     cost[q][q] = 0;
73     cost[p][q] = 1;
74     cost[q][p] = 1;
75     cost[p][p] = 0;
76     cost[q][q] = 0;
77     cost[p][q] = 1;
78     cost[q][p] = 1;
79     cost[p][p] = 0;
80     cost[q][q] = 0;
81     for (int i = 0; i < n; i++)
82         parent[i] = -1;
83     // Print the solution
84     BFSST();
85     return 0;
86 }
```

V. OUTPUT



```
"C:\Users\State University\OneDrive\Desktop\BFS.exe"
Edge 0:(0 1) cost:1
Edge 1:(0 2) cost:2
Edge 2:(2 3) cost:3
Edge 3:(3 4) cost:4
Edge 4:(3 5) cost:5
Edge 5:(5 6) cost:7
Edge 6:(4 7) cost:8

Minimum cost= 30
Process returned 0 (0x0)   execution time : 1.059 s
Press any key to continue.
```

2. Visit the contiguous unvisited vertex. Mark it as visited. Display it (if needed). If this is the required key, stop. Else, add it in a queue.

3. On the off chance that no neighboring vertex is discovered, expel the first vertex from the Queue.

4. Repeat step 2 and 3 until the queue is empty.

VI. CONCLUSION

The BFS algorithm is useful for analyzing the nodes in a graph and constructing the shortest path of traversing through these.

ACKNOWLEDGMENT

I would like to thank my honourable **Khan Md. Hasib Sir** for his time, generosity and critical insights into this project.

REFERENCES

- [1] Dellin, C., & Srinivasa, S. (2016, March). A unifying formalism for shortest path problems with expensive edge evaluations via lazy best-first search over paths with edge selectors. In Proceedings of the International Conference on Automated Planning and Scheduling (Vol. 26, No. 1).
- [2] Hifi, M., & Ouafi, R. (1997). Best-first search and dynamic programming methods for cutting problems: the cases of one or more stock plates. Computers & industrial engineering, 32(1), 187-205.
- [3] Korf, R. E., & Chickering, D. M. (1996). Best-first minimax search. Artificial intelligence, 84(1-2), 299-337.