Assignment No:02 Breath First Search

CSE-0408 Summer 2021

Rukaiya Khan Mithila (UG02-43-16-019)
Department of Computer Science and Engineering
State University of Bangladesh (SUB)
Dhaka, Bangladesh
khanrukaiya78@gmail.com

***Abstract*—This paper introduced for Breadth-First Search(BFS) problem solved using C++ language.**

***Index Terms*—Mostly used C++ language and Code-block code editor.**

I. Introduction

BFS is a traversing algorithm where we should start traversing from a selected node (source or starting node) and traverse the graph layer-wise thus exploring the neighbour nodes (nodes which are directly connected to source node). We must then move towards the next-level neighbour nodes. Breadth-first search starts at a given vertex s, which is at level 0. In the first stage, we visit all the vertices that are at the distance of one edge away. When we visit there, we paint as "visited," the vertices adjacent to the start vertex s - these vertices are placed into level 1. In the second stage, we visit all the new vertices we can reach at the distance of two edges away from the source vertex s. These new vertices, which are adjacent to level 1 vertices and not previously assigned to a level, are placed into level 2, and so on. The BFS traversal terminates when every vertex has been visited.

II. Literature Review

BFS and its application in finding connected components of graphs were invented in 1945 by Konrad Zuse, in his (rejected) Ph.D. thesis on the Plankalk¨ul programming language, but this was not published until 1972. It was reinvented in 1959 by Edward F. Moore, who used it to find the shortest path out of a maze, and later developed by C. Y. Lee into a wire routing algorithm (published 1961). In 2012 Farhad S. et. al. proposed new resolution for solving N-queens by using combination of DFS (Depth First Search) and BFS (Breadth First Search) techniques.

III. Proposed Methodology

1. for each u in V s
2. do color[u] ← W HIT E
3. d[u] ← inf inity
4. [u] ← NIL
5. color[s] ← GRAY
6. d[s] ← 0
7. [s] ← NIL
8. Q ←
9. ENQUEUE(Q, s)
10. while Q is not empty
11. do u ← DEQUEUE(Q)
12. for each v adjacent to u
13. do if color[v] ← WHIT E
14. then color[v] ← GRAY
15. d[v] ← d[u] + 1
16. [v] ← u
17. ENQUEUE(Q, v)
18. DEQUEUE(Q)
19. color[u] ← BLACK

Acknowledgment

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

References

[1]] Borges, Paulo HR, et al. "Carbonation of CH and C–S–H in composite cement pastes containing high amounts of BFS." Cement and concrete research 40.2 (2010): 284-292.

[2] Borges, P. H., Costa, J. O., Milestone, N. B., Lynsdale, C. J., Streatfield, R. E. (2010). Carbonation of CH and C–S–H in composite cement pastes containing high amounts of BFS. Cement and concrete research, 40(2), 284-292.

[3] Borges, Paulo HR, Juliana O. Costa, Neil B. Milestone, Cyril J. Lynsdale, and Roger E. Streatfield. "Carbonation of CH and C–S–H in composite cement pastes containing high amounts of BFS." Cement and concrete research 40, no. 2 (2010): 284-292

[4] Borges, P.H., Costa, J.O., Milestone, N.B., Lynsdale, C.J. and Streatfield, R.E., 2010. Carbonation of CH and C–S–H in composite cement pastes containing high amounts of BFS. Cement and concrete research, 40(2), pp.284-292.

[5] Borges PH, Costa JO, Milestone NB, Lynsdale CJ, Streatfield RE. Carbonation of CH and C–S–H in composite cement pastes containing high amounts of BFS. Cement and concrete research. 2010 Feb 1;40(2):284-92.

[6] M. Young, The Technical Writer's Handbook. Mill Valley, CA: University Science, 1989.

Output:

```
8 7
1 5
1 2
2 6
3 6
3 4
6 7
7 8
2
From node 2
Distance of 1 is : 1
Distance of 2 is : 0
Distance of 3 is : 2
Distance of 4 is : 3
Distance of 5 is : 2
Distance of 6 is : 1
Distance of 7 is : 2
Distance of 8 is : 3

Path from 1 to source: 2 1
Path from 2 to source: 2
Path from 3 to source: 2 6 3
Path from 4 to source: 2 6 3 4
Path from 5 to source: 2 1 5
Path from 6 to source: 2 6
Path from 7 to source: 2 6 7
Path from 8 to source: 2 6 7 8

Path from 1 to source: 2 1
Path from 2 to source: 2
Path from 3 to source: 2 6 3
Path from 4 to source: 2 6 3 4
Path from 5 to source: 2 1 5
Path from 6 to source: 2 6
Path from 7 to source: 2 6 7
Path from 8 to source: 2 6 7 8
```

```
#include
using namespace std;

#define MX 110

vector < int > graph[MX];
bool vis[MX];
int dist[MX];
int parent[MX];

void bfs(int source){
    queue < int > Q;
    // initialization
    vis[source] = 1;
    dist[source] = 0;
    Q.push(source);

    while(!Q.empty()){
        int node = Q.front();
        Q.pop();

        for (int i = 0; i < graph[node].size(); i++){
            int next = graph[node][i];
            if (vis[next] == 0){
                vis[next] = 1; // visit
                dist[next] = dist[node] + 1; // update
                Q.push(next); // push to queue

                // set parent
                parent[next] = node;
            }
        }
    }
}


// recursive function
void printPathRecursive(int source, int node){
    if (node == source){
        cout << node << " "; // print from source
        return;
    }
    printPathRecursive(source, parent[node]);
    cout << node << " ";
}

// iterative function
void printPathIterative(int source, int node){
    vector path_vector;

    while(node != source){
        path_vector.push_back(node);
        node = parent[node];
    }
    path_vector.push_back(source); // inserting source

    for (int i = path_vector.size() - 1; i >= 0; i--){
        cout << path_vector[i] << " ";
    }
}

int main()
{
    int nodes, edges;
    cin >> nodes >> edges;

    for (int i = 1; i <= edges; i++){
        int u, v;
        cin >> u >> v;
        graph[u].push_back(v);
        graph[v].push_back(u);
    }
```

```
int source;
    cin >> source;

    bfs(source);

    cout << "From node " << source << endl;
    for (int i = 1; i <= nodes; i++){
        cout << "Distance of " << i << " is : " << dist[i] << endl;
    }
    cout << endl;

    // path printing example

    // recursive version
    for (int i = 1; i <= nodes; i++){
        cout << "Path from " << i << " to source: ";
        printPathRecursive(source, i);
        cout << endl;
    }

    cout << endl;

    // iterative version
    for (int i = 1; i <= nodes; i++){
        cout << "Path from " << i << " to source: ";
        printPathIterative(source, i);
        cout << endl;
    }

    return 0;
}
```