Artificial Intelligence

Week 4

Topic:

Implementation of DFS AND BFS

Aim:-

Implementation of BFS & DFS algorithms.

Problem Statemenent :

1) Depth First Search(DFS) :

Depth first Search is a recursive algorithm for searching all the vertices of a graph or tree data structure.It uses stack datastructure for visiting the graph

2) Breadth First Search(BFS) :

BFS stands for Breadth First Search is a vertex based technique for finding the shortest path in a graph. It uses a Queue data structure which follows first in first out. In BFS, one vertex is selected at a time when it is visited and marked, then its adjacent vertices are visited and stored in the queue.

Algorithm:

1)BFS

1. Take the empty queue and bool type array (visit) initialize with FALSE.
2. Push the starting node in the queue and set the value TRUE for this node in the visited array.
3. Pop out the front node of the queue and print the node.
4. Push the adjacent node of the pop node in the queue which is not visited. Set the value TRUE in the visited array of adding nodes.
5. Repeat step 3 and 4 until the queue becomes empty.

2)DFS

1. Take the empty stack and bool type array (visit) initialize with FALSE.
2. Push the starting node in the stack and set the value TRUE for this node in the visited array.
3. Pop the top node from the stack and print that node.
4. Push the adjacent node of the pop node in the stack which is not visited. Set the value TRUE in the visited array of adding nodes.
5. Repeat step 3 and 4 until the stack becomes empty.

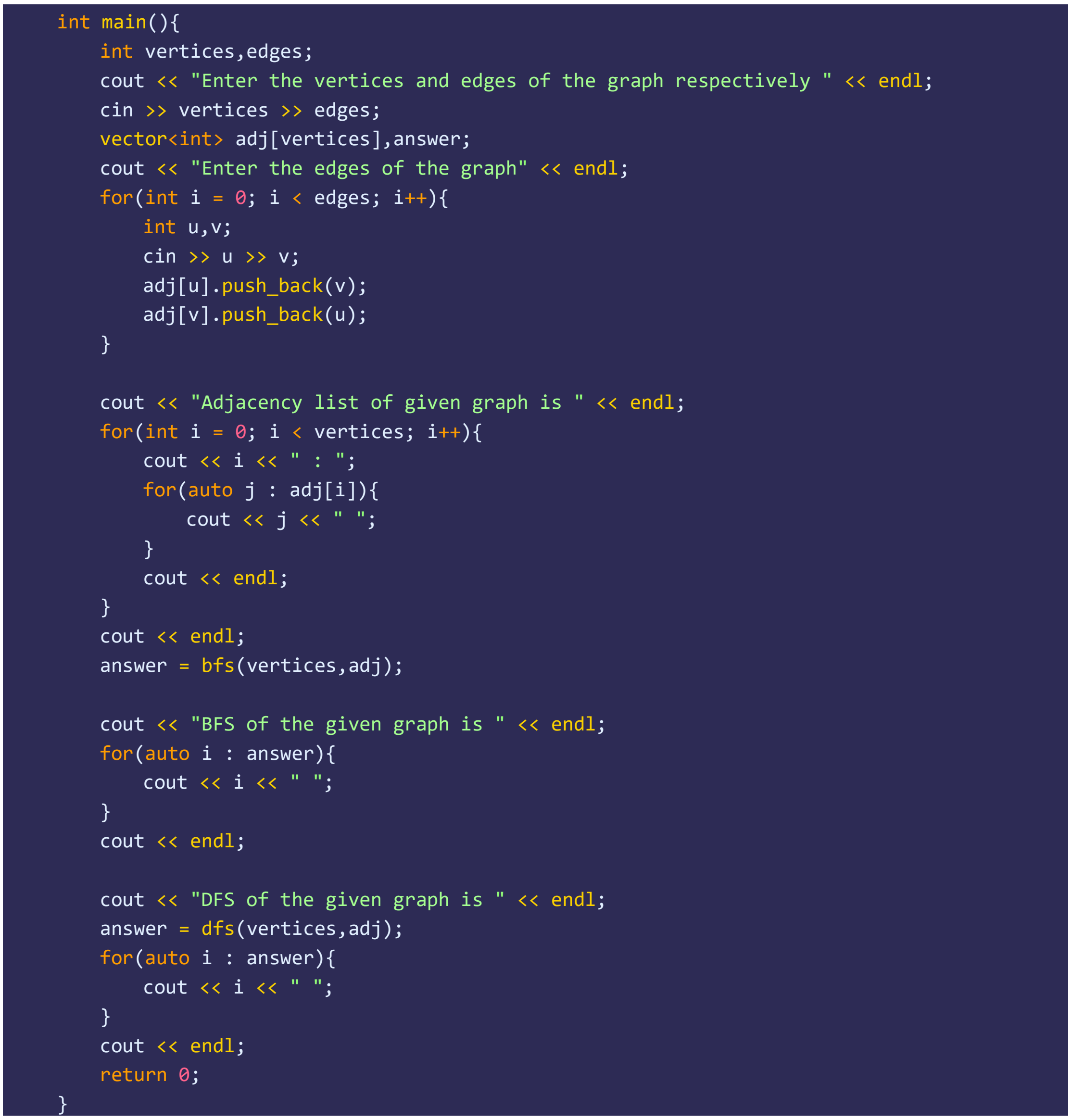
Code:

```
    #include<bits/stdc++.h>
    using namespace std;

    vector<int> bfs(int vertices,vector<int> adj[]){
        vector<int>answer;
        vector<bool> visited(vertices,false);
        queue<int> track;

        for(int i = 0; i < vertices; i++){
            if(!visited[i]){
                visited[i] = true;
                track.push(i);
                while(!track.empty()){
                    int temp = track.front();
                    track.pop();
                    answer.push_back(temp);
                    for(auto i : adj[temp]){
                        if(!visited[i]){
                            track.push(i);
                            visited[i] = true;
                        }
                    }
                }
            }
        }
        return answer;
    }
    vector<int> dfs(int vertices,vector<int> adj[]){
        vector<int>answer;
        vector<bool> visited(vertices,false);
        stack<int> track;

        for(int i = 1; i < vertices; i++){
            if(!visited[i]){
                visited[i] = true;
                track.push(i);
                while(!track.empty()){
                    int temp = track.top();
                    track.pop();
                    answer.push_back(temp);
                    for(auto i : adj[temp]){
                        if(!visited[i]){
                            track.push(i);
                            visited[i] = true;
                        }
                    }
                }
            }
        }
        return answer;
}
```



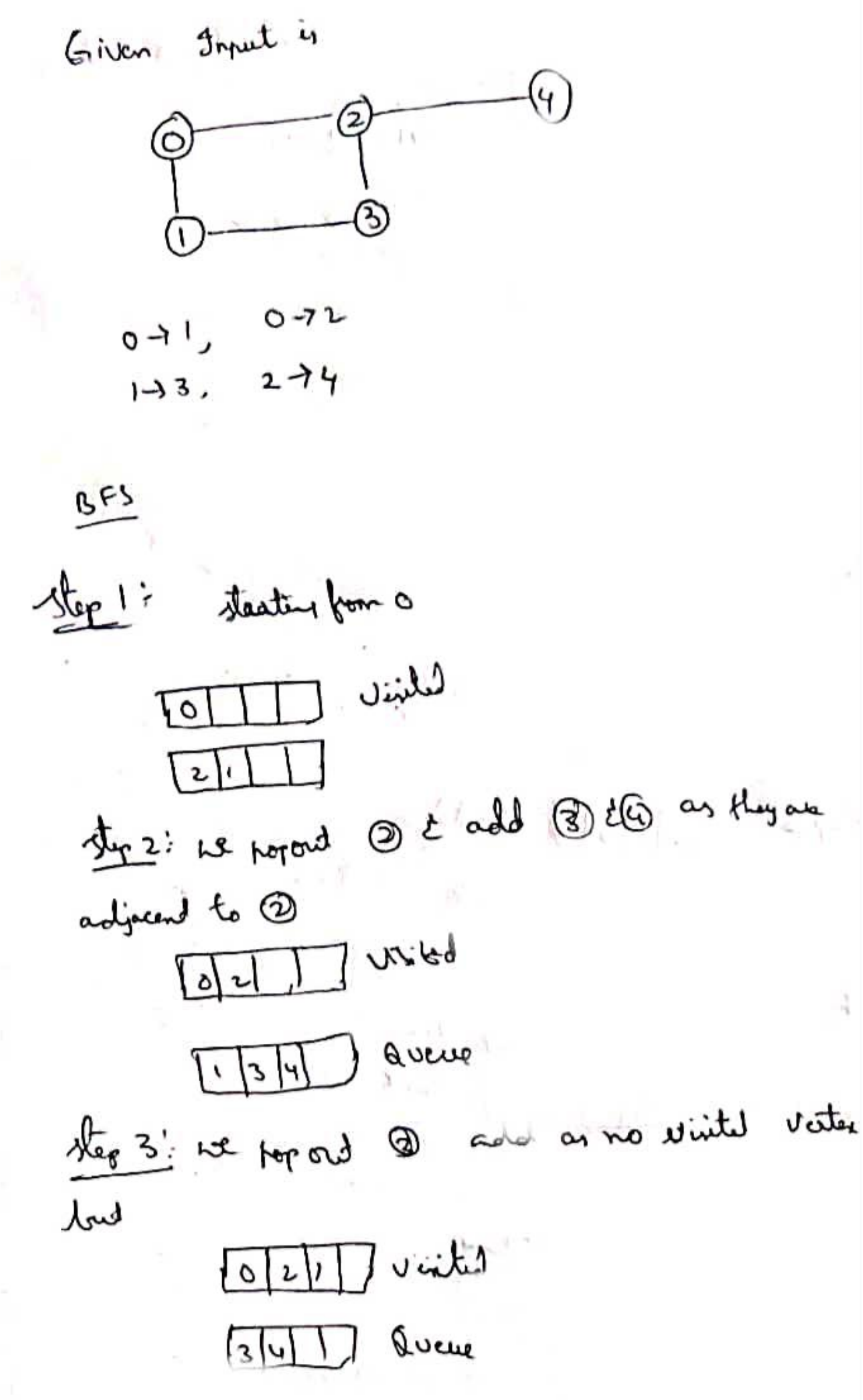
```
int main(){
    int vertices,edges;
    cout << "Enter the vertices and edges of the graph respectively " << endl;
    cin >> vertices >> edges;
    vector<int> adj[vertices],answer;
    cout << "Enter the edges of the graph" << endl;
    for(int i = 0; i < edges; i++){
        int u,v;
        cin >> u >> v;
        adj[u].push_back(v);
        adj[v].push_back(u);
    }

    cout << "Adjacency list of given graph is " << endl;
    for(int i = 0; i < vertices; i++){
        cout << i << " : ";
        for(auto j : adj[i]){
            cout << j << " ";
        }
        cout << endl;
    }
    cout << endl;
    answer = bfs(vertices,adj);

    cout << "BFS of the given graph is " << endl;
    for(auto i : answer){
        cout << i << " ";
    }
    cout << endl;

    cout << "DFS of the given graph is " << endl;
    answer = dfs(vertices,adj);
    for(auto i : answer){
        cout << i << " ";
    }
    cout << endl;
    return 0;
}
```

Manual Calculation(Dry Run)



Given Input is

0→1, 0→2
1→3, 2→4

BFS

Step 1: starting from 0

0			

Visited

2	1			

Step 2: we pop out ② & add ③ &④ as they are adjacent to ②

0	2	,1	

Visited

1	3	4	

Queue

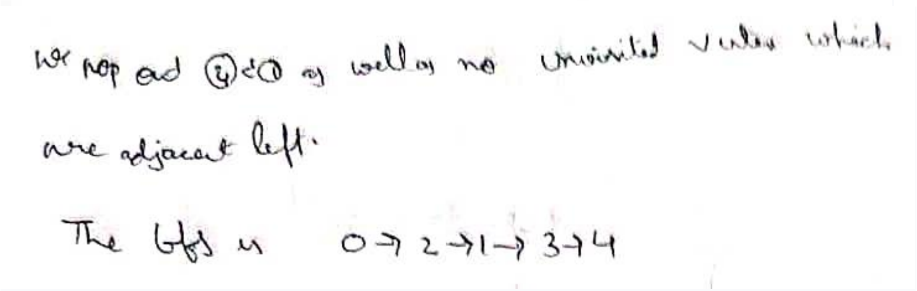
Step 3: we pop out ① add as no visited vertex left

0	2	1	

Visited

3	4	

Queue



We pop out ④&③ as well as no unvisited vertex which are adjacent left.

The bfs is 0→2→1→3→4

DFS : n=4

step 1 :-

0			

visited

1		

stack

step 2 :

0	1		

visited

3			

stack

step 3 :

0	1	3	

visited

2			

stack

step 4 :

0	1	3	2

visited

4			

stack

step 5 :

0	1	3	2	4

visited

The DFS is 0 → 1 → 3 → 2 → 4

Observation :

Time Complexity :

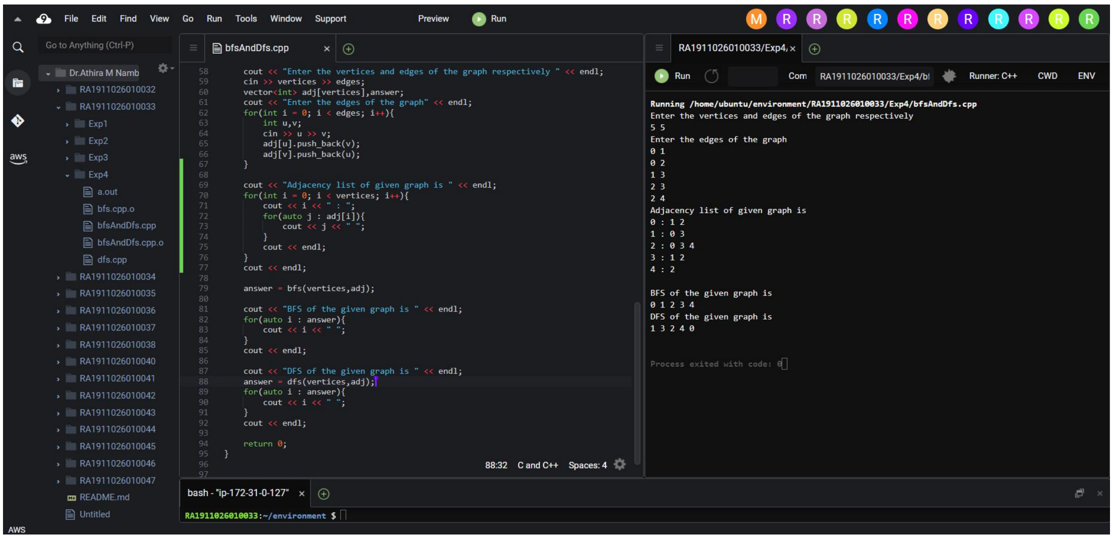
```
Time complexity of BFS and Dfs will be O(n + E)
 Time for visiting n nodes and e for visiting adjacent nodes in Adjacency list
```

Space Complexity :

```
Space Complexity will be O(N + E)       + O(N)      +  O(N)
                       Adjacency list    visited     Stack          - For DFS
                       Adjacency list    visited     queue          - For BFS
```

For minumum distance problem even though dfs and bfs take the space complexity.BFS is faster than DFS.

Output:



```cpp
    cout << "Enter the vertices and edges of the graph respectively " << endl;
    cin >> vertices >> edges;
    vector<int> adj[vertices],answer;
    cout << "Enter the edges of the graph" << endl;
    for(int i = 0; i < edges; i++){
        int u,v;
        cin >> u >> v;
        adj[u].push_back(v);
        adj[v].push_back(u);
    }

    cout << "Adjacency list of given graph is " << endl;
    for(int i = 0; i < vertices; i++){
        cout << i << " : ";
        for(auto j : adj[i]){
            cout << j << " ";
        }
        cout << endl;
    }
    cout << endl;

    answer = bfs(vertices,adj);

    cout << "BFS of the given graph is " << endl;
    for(auto i : answer){
        cout << i << " ";
    }
    cout << endl;

    cout << "DFS of the given graph is " << endl;
    answer = dfs(vertices,adj);
    for(auto i : answer){
        cout << i << " ";
    }
    cout << endl;

    return 0;
}
```

```
Running /home/ubuntu/environment/RA1911026010033/Exp4/bfsAndDfs.cpp
Enter the vertices and edges of the graph respectively
5 5
Enter the edges of the graph
0 1
0 2
1 3
2 3
2 4
Adjacency list of given graph is
0 : 1 2
1 : 0 3
2 : 0 3 4
3 : 1 2
4 : 2

BFS of the given graph is
0 1 2 3 4
DFS of the given graph is
1 3 2 4 0


Process exited with code: 0
```

```
RA1911026010033:~/environment $
```

Result:-

Bfs and Dfs algorithms are successfully implemented