रा.इ.सू.प्रौ.सं NIELIT

NIELIT GORAKHPUR

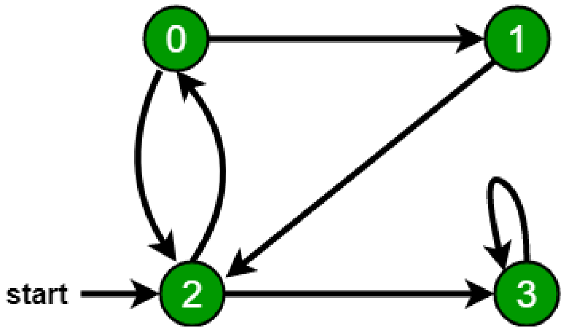
Course Name: **A Level (2nd Sem)** **Subject:** **Data Structure using C++**
Topic: **Breadth First Search without using Queue** **Date:** **29-04-2020**

Breadth First Search without using Queue

Breadth-first search is a graph traversal algorithm which traverse a graph or tree level by level. In this article, BFS for a Graph is implemented using Adjacency list without using a Queue.

Examples:

Input:



Output: BFS traversal = 2, 0, 3, 1

Explanation:
In the following graph, we start traversal from vertex 2. When we come to vertex 0, we look for all adjacent vertices of it. 2 is also an adjacent vertex of 0. If we don't mark visited vertices, then 2 will be processed again and it will become a non-terminating process. Therefore, a Breadth-First Traversal of the following graph is 2, 0, 3, 1.

Approach:
This problem can be solved using simple breadth-first traversal from a given source. The implementation uses **adjacency list representation of graphs**.

STL Vector container is used to store lists of adjacent nodes and queue of nodes needed for BFS traversal.
A **DP array** is used to store the distance of the nodes from the source. Every time we move from a node to another node, the distance increases by 1. If the distance to reach the nodes becomes smaller than the previous distance, we update the value stored in the DP[node].

Implementation of the above approach:

```
#include <bits/stdc++.h>
using namespace std;
void BFS(int curr, int N, vector<bool>& vis, vector<int>& dp, vector<int>& v, vector<vector<int> >&
adj)
{
    while (curr <= N)
{
       int node = v[curr - 1];
      cout << node << ", ";
       for (int i = 0; i < adj[node].size(); i++)
       {
          int next = adj[node][i];

        if ((!vis[next])
                && (dp[next] < dp[node] + 1)) {

                // Stores the adjacent node
                v.push_back(next);

                // Increases the distance
                dp[next] = dp[node] + 1;

                // Mark it as visited
                vis[next] = true;
            }
        }
        curr += 1;
    }
}

void bfsTraversal(  vector<vector<int> >& adj,   int N, int source)
{
    // Initially mark all nodes as false
    vector<bool> vis(N + 1, false);

    // Initialize distance array with 0
    vector<int> dp(N + 1, 0), v;
    v.push_back(source);
    dp = 0;
    vis = true;
      // Call the BFS function
    BFS(1, N, vis, dp, v, adj);
}

// Driver code
int main()
{
    // No. of nodes in graph
    int N = 4;
     // Creating adjacency list
    // for representing graph
    vector<vector<int> > adj(N + 1);
    adj[0].push_back(1);
    adj[0].push_back(2);
    adj[1].push_back(2);
    adj[2].push_back(0);
    adj[2].push_back(3);
    adj[3].push_back(3);

    // Following is BFS Traversal
    // starting from vertex 2
    bfsTraversal(adj, N, 2);
    return 0;
}
```

Output:

```
2, 0, 3, 1,
```