Name: Achyut Shukla Batch: AA – A1 PRN: 20070122005

BREADTH FIRST SEARCH

BFS stands for "Breadth-First Search," which is a graph traversal algorithm used in computer science and graph theory. It is a graph searching algorithm that explores all the vertices of a graph or all the nodes of a tree in breadth-first order, i.e., it visits all the vertices/nodes at the same level before moving on to the next level.

In BFS, the algorithm starts at the root vertex (or any arbitrary vertex) and explores the graph by visiting all the adjacent vertices/nodes first, before moving on to the vertices/nodes that are at a greater depth (i.e., at the next level) from the starting vertex. BFS uses a queue data structure to keep track of the vertices/nodes to be visited, and it ensures that vertices/nodes at the same level are visited before moving to the next level.

Code

```
#include <bits/stdc++.h>

using namespace std;


class Graph

{

   int V;

   vector<list<int>> adj;


public:

   Graph(int V);


   void addEdge(int v, int w);

   void BFS(int s);

};


Graph::Graph(int V)

{

   this->V = V;

   adj.resize(V);

}
```

```
void Graph::addEdge(int v, int w)
{
   adj[v].push_back(w);
}

void Graph::BFS(int s)
{
   vector<bool> visited;
   visited.resize(V, false);

   list<int> queue;

   visited[s] = true;
   queue.push_back(s);

   while (!queue.empty())
   {

      s = queue.front();
      cout << s << " ";
      queue.pop_front();

      for (auto adjacent : adj[s])
      {
         if (!visited[adjacent])
         {
            visited[adjacent] = true;
            queue.push_back(adjacent);
         }
      }
   }
}
```

```
int main()
{
   Graph g(4);
   g.addEdge(0, 1);
   g.addEdge(0, 3);
   g.addEdge(1, 0);
   g.addEdge(1, 2);
   g.addEdge(2, 1);
   g.addEdge(2, 3);
   g.addEdge(3, 3);

   cout << "Following is Breadth First Traversal (starting from vertex 1) \n";
   g.BFS(1);

   return 0;
}
```

Complexity analysis of DFS

Time Complexity - O(V + E)

Space – O(V)

Advantages of Breadth-First Search (BFS):

Shortest Path: BFS can be used to find the shortest path between two vertices in an unweighted graph. Since BFS explores the graph in breadth-first order, it guarantees that the shortest path will be found when the destination vertex is reached.

Completeness: BFS is a complete algorithm, meaning it will always find a solution if one exists. If there is a path between the source and destination vertices, BFS will find it.

Level-by-Level Exploration: BFS visits vertices at the same level before moving to the next level. This property makes BFS suitable for certain problems where level-by-level exploration is desired, such as finding all the connected components of a graph.

Disadvantages of Breadth-First Search (BFS):

Space Complexity: BFS uses a queue data structure to keep track of vertices to be visited, which can require a large amount of memory, especially for large graphs with many vertices. The space complexity of BFS is O(V), where V is the number of vertices in the graph.

Not Suitable for Weighted Graphs: BFS is primarily designed for unweighted graphs, where all the edges have the same weight. It is not directly applicable to finding the shortest path in weighted graphs without modifications, as it does not take edge weights into account.

Redundant Exploration: BFS may visit some vertices multiple times, leading to redundant exploration and increased computation, especially in graphs with cycles or disconnected components.