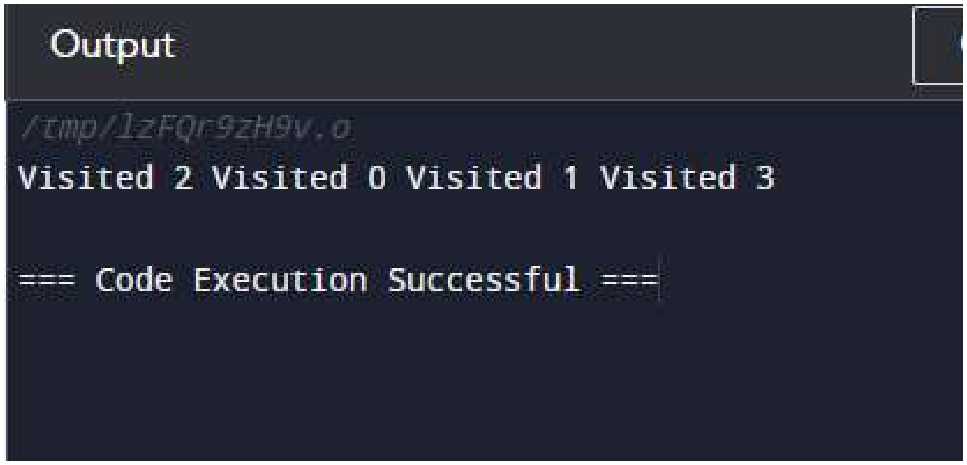
BFS TRAVERSAL IN A GRAPH

```
// BFS algorithm in C++
#include <iostream>
#include <list>
using namespace std;
class Graph {
  int numVertices;
  list<int>* adjLists;
  bool* visited;
   public:
  Graph(int vertices);
  void addEdge(int src, int dest);
  void BFS(int startVertex);
};

// Create a graph with given vertices,
// and maintain an adjacency list
Graph::Graph(int vertices) {
  numVertices = vertices;
  adjLists = new list<int>[vertices];
}

// Add edges to the graph
void Graph::addEdge(int src, int dest) {
  adjLists[src].push_back(dest);
  adjLists[dest].push_back(src);
}
```

```
// BFS algorithm
void Graph::BFS(int startVertex) {
  visited = new bool[numVertices];
  for (int i = 0; i < numVertices; i++)
    visited[i] = false;

  list<int> queue;

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

  list<int>::iterator i;

  while (!queue.empty()) {
    int currVertex = queue.front();
    cout << "Visited " << currVertex << " ";
    queue.pop_front();

    for (i = adjLists[currVertex].begin(); i != adjLists[currVertex].end(); ++i) {
      int adjVertex = *i;
      if (!visited[adjVertex]) {
        visited[adjVertex] = true;
        queue.push_back(adjVertex);
      }
    }
  }
}

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

  g.BFS(2);

  return 0;
}
```

OUTPUT:



```
/tmp/lzFQr9zH9v.o
Visited 2 Visited 0 Visited 1 Visited 3

=== Code Execution Successful ===
```