DFS.cpp

```
#include <iostream>
#include <list>
using namespace std;
//graph class for DFS travesal
class DFSGraph
{
int V;
list<int> *adjList;
void DFS_util(int v, bool visited[]);
public:
DFSGraph(int V)
{
this->V = V;
adjList = new list<int>[V];
}
void addEdge(int v, int w){
adjList[v].push_back(w);
}
void DFS();
};
void DFSGraph::DFS_util(int v, bool visited[])
{
visited[v] = true;
cout << v << " ";
list<int>::iterator i;
for(i = adjList[v].begin(); i != adjList[v].end(); ++i)
if(!visited[*i])
DFS_util(*i, visited);
}
void DFSGraph::DFS()
{
bool *visited = new bool[V];
for (int i = 0; i < V; i++)
visited[i] = false;
for (int i = 0; i < V; i++)

if (visited[i] == false)
DFS_util(i, visited);
}
int main()
{
int n;
```

```
int starting,ending;
cout<<"Enter the number of nodes : "<<endl;
cin>>n;
DFSGraph gdfs(n);
for(int i = 0 ; i < n ; i ++)
{
cout<<"Enter the starting vertex: "<<endl;
cin>>starting;
cout<<"Enter the ending vertex: "<<endl;
cin>>ending;
gdfs.addEdge(starting,ending);
}
cout << "Depth-first traversal is:"<<endl;
gdfs.DFS();
return 0;
}
```

BFS.cpp

```
// 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() {
          int n;
          cout<<"Enter the no of vertices : "<<endl; cin>>n;
          int f,s;
  Graph g(n);
          for (int i = 0; i < n; i++)
          {
          cout<<"Enter first edge : "<<endl;  cin>>f;
```

```
            cout<<"Enter second edge : "<<endl;  cin>>s;
            g.addEdge(f,s);
            }

//   g.addEdge(0, 1);
//   g.addEdge(0, 2);
//   g.addEdge(1, 2);
//   g.addEdge(2, 0);
//   g.addEdge(2, 3);
//   g.addEdge(3, 3);
int start;
            cout<<"Enter start vertec : "<<endl; cin>>start;
  g.BFS(start);

  return 0;
}
```