Aim-implementation using adjancy list

#include <iostream>

#include <unordered\_map>

#include <vector>

#include <queue>

#include <stack>

#include <set>

using namespace std;

class Graph {

public:

unordered\_map<int, vector<int>> adjList;

void addVertex(int vertex) {

adjList[vertex]; // Automatically creates an empty list for the vertex if it doesn't exist

}

void addEdge(int vertex1, int vertex2) {

adjList[vertex1].push\_back(vertex2);

adjList[vertex2].push\_back(vertex1); // For undirected graph

}

void display() {

for (const auto& pair : adjList) {

cout << pair.first << ": ";

for (int neighbor : pair.second) {

cout << neighbor << " ";

}

cout << endl;

}

}

vector<int> bfs(int start) {

set<int> visited;

queue<int> q;

vector<int> result;

q.push(start);

visited.insert(start);

while (!q.empty()) {

int vertex = q.front();

q.pop();

result.push\_back(vertex);

for (int neighbor : adjList[vertex]) {

if (visited.find(neighbor) == visited.end()) {

visited.insert(neighbor);

q.push(neighbor);

}

}

}

return result;

}

vector<int> dfs(int start) {

set<int> visited;

stack<int> s;

vector<int> result;

s.push(start);

while (!s.empty()) {

int vertex = s.top();

s.pop();

if (visited.find(vertex) == visited.end()) {

visited.insert(vertex);

result.push\_back(vertex);

// Add neighbors in reverse order to visit them in the correct order

for (auto it = adjList[vertex].rbegin(); it != adjList[vertex].rend(); ++it) {

if (visited.find(\*it) == visited.end()) {

s.push(\*it);

}

}

}

}

return result;

}

};

int main() {

Graph g;

g.addVertex(1);

g.addVertex(2);

g.addVertex(3);

g.addVertex(4);

g.addEdge(1, 2);

g.addEdge(1, 3);

g.addEdge(2, 4);

g.display();

vector<int> bfsResult = g.bfs(1);

cout << "BFS: ";

for (int v : bfsResult) cout << v << " ";

cout << endl;

vector<int> dfsResult = g.dfs(1);

cout << "DFS: ";

for (int v : dfsResult) cout << v << " ";

cout << endl;

return 0;

}