1. BFS

|  |  |  |  |
| --- | --- | --- | --- |
| Bước | Đỉnh | Open | Close |
| 1 |  | S | O |
| 2 | S | A, B, C | S |
| 3 | A | B, C, D | S, A |
| 4 | B | C, D, E | S, A, B |
| 5 | C | D, E, F | S, A,B, C |
| 6 | D | E, F, G | S, A, B, C, D |
| 7 | E | F, G | S, A, B, C, D, E |
| 8 | F | G | S, A, B, C, D, E, F |
| 9 | G | O | S, A, B, C, D, E, F, G |

S→A→B→C→D→E→F→G

2)

|  |  |  |  |
| --- | --- | --- | --- |
| Bước | Đỉnh | Open | Close |
| 1 |  | S | O |
| 2 | S | C, B, A | S |
| 3 | A | C, B, D | S, A |
| 4 | D | C, B, E | S, A, D |
| 5 | E | C, B, G | S, A, D, E |
| 6 | G | C, B, F | S, A, D, E, G |
| 7 | F | C, B | S, A, D, E, G, F |
| 8 | B | C | S, A, D, E, G, F, B |
| 9 | C | O | S, A, D, E, G, F, B, C |

S → A → D → E → G → F → B → C

3)

#include <iostream>

#include <vector>

#include <queue>

#include <stack>

#include <map>

#include <algorithm>

using namespace std;

class Graph {

private:

    map<char, vector<char>> adjList; // Danh sách kề

public:

    void addEdge(char u, char v) {

        adjList[u].push\_back(v);

    }

    void BFS(char start) {

        queue<char> q;

        map<char, bool> visited;

        q.push(start);

        visited[start] = true;

        cout << "BFS: ";

        while (!q.empty()) {

            char node = q.front();

            q.pop();

            cout << node << " ";

            for (char neighbor : adjList[node]) {

                if (!visited[neighbor]) {

                    visited[neighbor] = true;

                    q.push(neighbor);

                }

            }

        }

        cout << endl;

    }

    void DFS(char start) {

        stack<char> s;

        map<char, bool> visited;

        s.push(start);

        cout << "DFS: ";

        while (!s.empty()) {

            char node = s.top();

            s.pop();

            if (!visited[node]) {

                visited[node] = true;

                cout << node << " ";

            }

            // Sắp xếp danh sách kề theo thứ tự A → Z (để khi push vào stack, nó ra đúng thứ tự mong muốn)

            sort(adjList[node].begin(), adjList[node].end());

            // Push ngược lại vào stack để đảm bảo thứ tự đúng

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

                if (!visited[\*it]) {

                    s.push(\*it);

                }

            }

        }

        cout << endl;

    }

};

int main() {

    Graph g;

    g.addEdge('S', 'A');

    g.addEdge('S', 'B');

    g.addEdge('S', 'C');

    g.addEdge('A', 'D');

    g.addEdge('B', 'E');

    g.addEdge('C', 'F');

    g.addEdge('D', 'G');

    g.addEdge('E', 'G');

    g.addEdge('F', 'G');

    g.BFS('S');

    g.DFS('S');

    return 0;

}