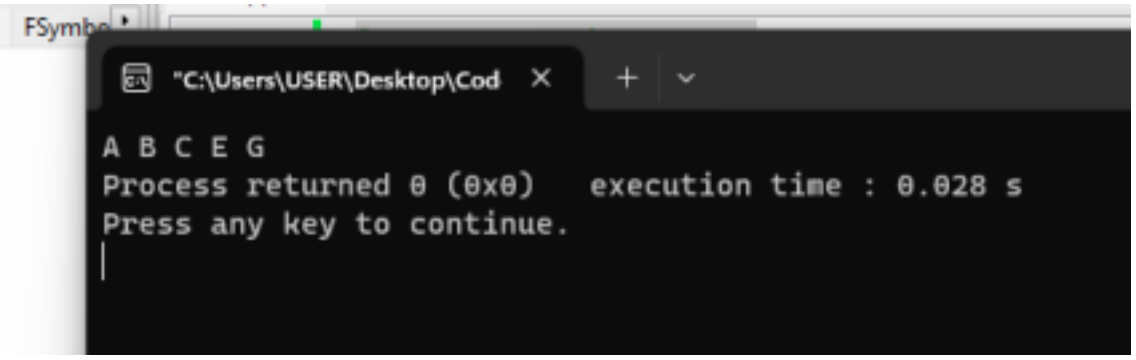
1. First Problem :

```
#include <bits/stdc++.h>
using namespace std;

void add_edge(vector<vector<char>>& adj, char src, char dest) {
    adj[src - 'A'].push_back(dest);
    adj[dest - 'A'].push_back(src);
}

bool BFS(const vector<vector<char>>& adj, char src, char dest, int
        v, vector<char>& pred, vector<int>& dist) {
    vector<bool> visited(v, false);
    visited[src - 'A'] = true;
    dist[src - 'A'] = 0;
    queue<char> q;
    q.push(src);

    while (!q.empty()) {
        char u = q.front();
        q.pop();
        for (char neighbor : adj[u - 'A']) {
            if (!visited[neighbor - 'A']) {
                visited[neighbor - 'A'] = true;
                dist[neighbor - 'A'] = dist[u - 'A'] + 1;
                pred[neighbor - 'A'] = u;
                q.push(neighbor);
                if (neighbor == dest)
                    return true;
            }
        }
    }

    return false;
}

void PrintPath(const stdvector<char>& pred, char s, char d) {
    if (s == d) {
        cout << s << ' ';
    } else if (pred[d - 'A'] == '\0') {
        cout << "No path from " << s << " to " << d << '\n';
    } else {
        PrintPath(pred, s, pred[d - 'A']);
        cout << d << ' ';
    }
}
```

```
int main() {
    int v = 7;
    vector<vector<char>> adj(7, vector<char>());
    add_edge(adj, 'A', 'B');
    add_edge(adj, 'B', 'C');
    add_edge(adj, 'B', 'D');
    add_edge(adj, 'C', 'E');
    add_edge(adj, 'D', 'F');
    add_edge(adj, 'E', 'G');
    add_edge(adj, 'F', 'G');

    char source = 'A', dest = 'G';
    vector<char> pred(v, '\0');
    vector<int> dist(v, -1);

    if (BFS(adj, source, dest, v, pred, dist)) {
        PrintPath(pred, source, dest);
    } else {
        cout << "No path from " << source << " to " << dest << '\n';
    }

    return 0;
}
```



2. Second Problem

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

using namespace std;

void dfs(int node, unordered_map<int, vector<int>>& adjacencyList, vector<int>&
component, unordered_set<int>& visited) {
```

```
    visited.insert(node);
    component.push_back(node);

    for (int neighbor : adjacencyList.at(node)) {
        if (visited.find(neighbor) == visited.end()) {
            dfs(neighbor, adjacencyList, component, visited);
        }
    }
}

vector<vector<int>> findConnectedComponents( unordered_map<int, vector<int>>&
adjacencyList) {
    unordered_set<int> visited;
    vector<vector<int>> connectedComponents;

    for (const auto& entry : adjacencyList) {
        int node = entry.first;
        if (visited.find(node) == visited.end()) {
            vector<int> component;
            dfs(node, adjacencyList, component, visited);
            connectedComponents.push_back(component);
        }
    }

    return connectedComponents;
}

int main() {
    unordered_map<int, vector<int>> adjacencyList = {
        {1, {2, 3}},
        {2, {1, 4}},
        {3, {1}},
        {4, {2, 5}},
        {5, {4}},
        {6, {7}},
        {7, {6}}
    };

    vector<vector<int>> connectedComponents = findConnectedComponents(adjacencyList);

    // Print the result
    int componentNumber = 1;
    for (const vector<int>& component : connectedComponents) {
        cout << "Component " << componentNumber << ": [";
        for (int i = 0; i < component.size(); ++i) {
            cout << component[i];
```

```
            if (i < component.size() - 1) {
                cout << ", ";
            }
        }
        cout << "]" << endl;
        componentNumber++;
    }

    return 0;
}
```