LAB ASSIGNMENT 2

NORTHERN UNIVERSITY
BANGLADESH
Knowledge for Innovation and Change

SUBMITTED BY

NAME: *Sadia Islam*

ID: *41220300363*

DEPARTMENT: *CSE*

SECTION: *4B*

SUBJECT: *Algorithm design and analysis lab work*

SEMESTER: *Fall 2023*

SUBMITTED TO

Md. Mahadi Hasan

(Lecturer of CSE in NUB)

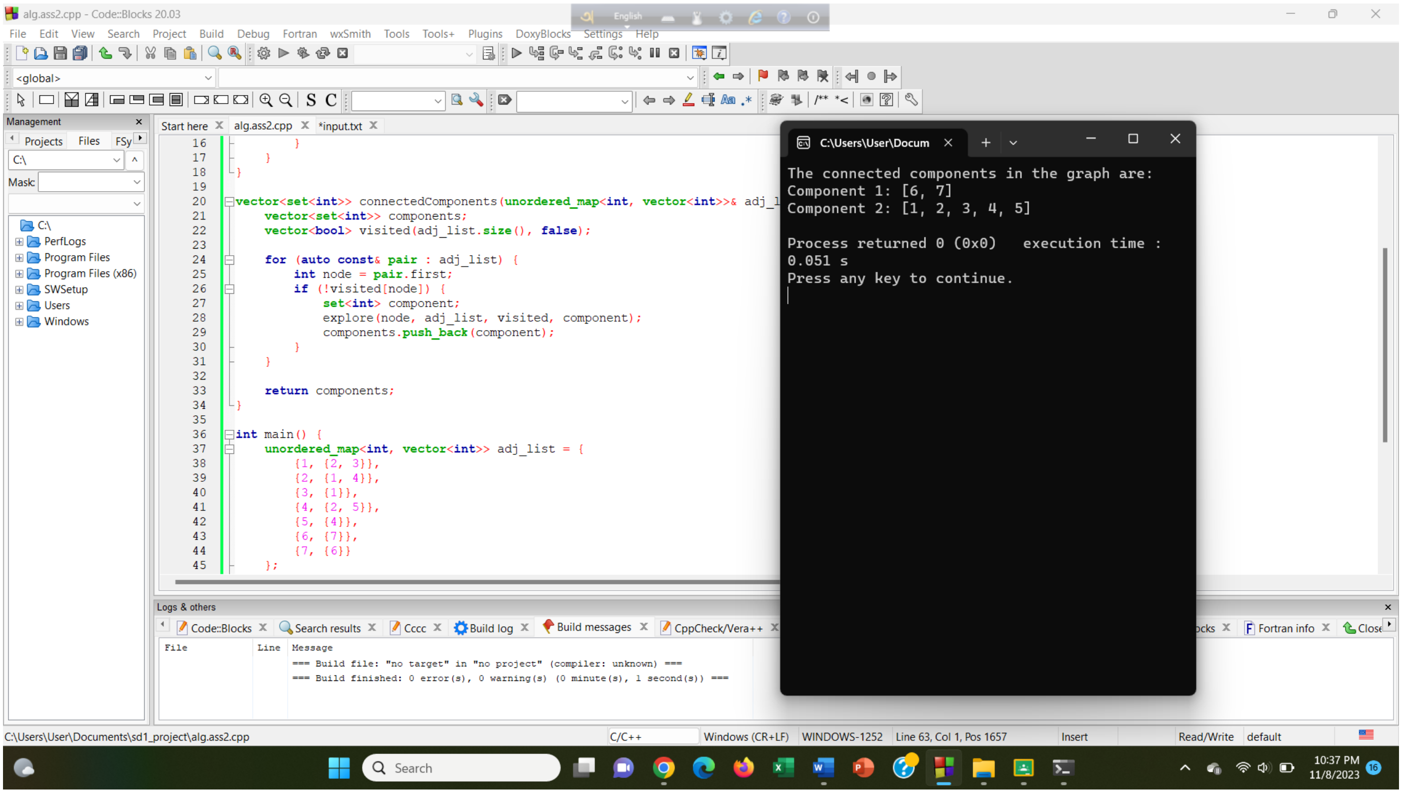
Submitted in 08 November 2023

Problem 1:

C++ code:

```
#include <iostream>

#include <vector>

#include <queue>

#include <unordered_map>

#include <string>


using namespace std;


vector<string> bfsShortestPath(vector<vector<int>>& graph, int start, int destination) {

    int n = graph.size();

    vector<bool> visited(n, false);

    vector<int> parent(n, -1);

    queue<int> q;


    q.push(start);

    visited[start] = true;


    while (!q.empty()) {

        int current = q.front();

        q.pop();


        if (current == destination) {

            break;

        }


        for (int i = 0; i < n; ++i) {

            if (graph[current][i] == 1 && !visited[i]) {

                q.push(i);

                visited[i] = true;

                parent[i] = current;

            }
        }
    }

    // Reconstructing the path
    vector<string> shortestPath;
    int current = destination;
    while (current != -1) {
        shortestPath.insert(shortestPath.begin(), to_string(current + 'A')); // Assuming nodes are represented as 'A', 'B', 'C', ...
        current = parent[current];
    }

    return shortestPath;
}

int main() {
    // Example input representing the adjacency matrix
    vector<vector<int>> graph = {
        {0, 1, 0, 0, 0, 0, 0},
        {0, 0, 1, 1, 0, 0, 0},
        {0, 0, 0, 0, 1, 0, 0},
        {0, 0, 0, 0, 0, 1, 0},
        {0, 0, 0, 0, 0, 0, 1},
        {0, 0, 0, 0, 0, 0, 1},
        {0, 0, 0, 0, 0, 0, 0}
    };

    int start = 0; // Index of starting location (A)
    int destination = 6; // Index of destination location (G)

    vector<string> shortestPath = bfsShortestPath(graph, start, destination);

    if (shortestPath.empty()) {
        cout << "No path found from the starting location to the destination." << endl;
    } else {
        cout << "The shortest path from A to G is: ";
        for (size_t i = 0; i < shortestPath.size(); ++i) {
            cout << shortestPath[i];
            if (i != shortestPath.size() - 1) {
                cout << " -> ";
            }
        }
        cout << endl;
    }

    return 0;
}
```

Problem 2:



```
        }
    }
}

vector<set<int>> connectedComponents(unordered_map<int, vector<int>>& adj_l
    vector<set<int>> components;
    vector<bool> visited(adj_list.size(), false);

    for (auto const& pair : adj_list) {
        int node = pair.first;
        if (!visited[node]) {
            set<int> component;
            explore(node, adj_list, visited, component);
            components.push_back(component);
        }
    }

    return components;
}

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

```
File        Line  Message
                  === Build file: "no target" in "no project" (compiler: unknown) ===
                  === Build finished: 0 error(s), 0 warning(s) (0 minute(s), 1 second(s)) ===
```

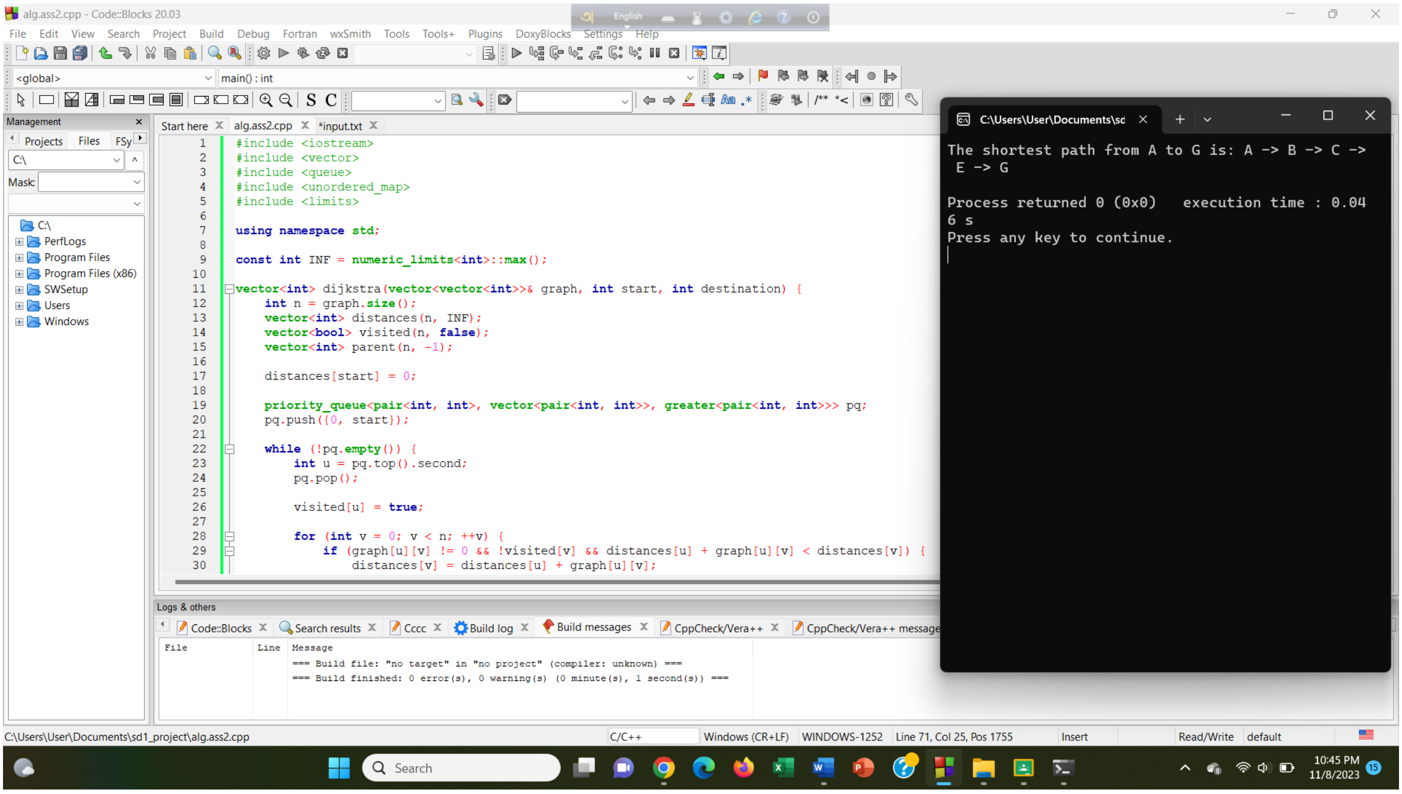
```
The connected components in the graph are:
Component 1: [6, 7]
Component 2: [1, 2, 3, 4, 5]

Process returned 0 (0x0)   execution time :
0.051 s
Press any key to continue.
```

C++ Code:

```
#include <iostream>

#include <vector>

#include <unordered_map>

#include <set>
```

```
using namespace std;



void explore(int node, unordered_map<int, vector<int>>& adj_list, vector<bool>& visited, set<int>& component) {
    visited[node] = true;
    component.insert(node);

    for (int neighbor : adj_list[node]) {
        if (!visited[neighbor]) {
            explore(neighbor, adj_list, visited, component);
        }
    }
}

vector<set<int>> connectedComponents(unordered_map<int, vector<int>>& adj_list) {
    vector<set<int>> components;
    vector<bool> visited(adj_list.size(), false);

    for (auto const& pair : adj_list) {
        int node = pair.first;
        if (!visited[node]) {
            set<int> component;
            explore(node, adj_list, visited, component);
            components.push_back(component);
        }
    }

    return components;
}

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

    vector<set<int>> components = connectedComponents(adj_list);

    cout << "The connected components in the graph are:" << endl;
    for (int i = 0; i < components.size(); ++i) {
        cout << "Component " << i + 1 << ": [";
        for (int node : components[i]) {
            cout << node;
            if (node != *prev(components[i].end())) {
                cout << ", ";
            }
        }
        cout << "]" << endl;
    }

    return 0;
}
```

Problem 3:



```
#include <iostream>
#include <vector>
#include <queue>
#include <unordered_map>
#include <limits>

using namespace std;

const int INF = numeric_limits<int>::max();

vector<int> dijkstra(vector<vector<int>>& graph, int start, int destination) {
    int n = graph.size();
    vector<int> distances(n, INF);
    vector<bool> visited(n, false);
    vector<int> parent(n, -1);

    distances[start] = 0;

    priority_queue<pair<int, int>, vector<pair<int, int>>, greater<pair<int, int>>> pq;
    pq.push({0, start});

    while (!pq.empty()) {
        int u = pq.top().second;
        pq.pop();

        visited[u] = true;

        for (int v = 0; v < n; ++v) {
            if (graph[u][v] != 0 && !visited[v] && distances[u] + graph[u][v] < distances[v]) {
                distances[v] = distances[u] + graph[u][v];
```

```
The shortest path from A to G is: A -> B -> C ->
 E -> G

Process returned 0 (0x0)   execution time : 0.04
6 s
Press any key to continue.
```

```
File          Line  Message
                    === Build file: "no target" in "no project" (compiler: unknown) ===
                    === Build finished: 0 error(s), 0 warning(s) (0 minute(s), 1 second(s)) ===
```

C++ Code:

```
#include <iostream>

#include <vector>

#include <queue>

#include <unordered_map>

#include <limits>


using namespace std;


const int INF = numeric_limits<int>::max();


vector<int> dijkstra(vector<vector<int>>& graph, int start, int destination) {

    int n = graph.size();

    vector<int> distances(n, INF);

    vector<bool> visited(n, false);

    vector<int> parent(n, -1);


    distances[start] = 0;
    priority_queue<pair<int, int>, vector<pair<int, int>>, greater<pair<int, int>>> pq;
    pq.push({0, start});

    while (!pq.empty()) {
        int u = pq.top().second;
        pq.pop();

        visited[u] = true;

        for (int v = 0; v < n; ++v) {
            if (graph[u][v] != 0 && !visited[v] && distances[u] + graph[u][v] < distances[v]) {
                distances[v] = distances[u] + graph[u][v];
                pq.push({distances[v], v});
                parent[v] = u;
            }
        }
    }



    vector<int> shortest_path;
    int current = destination;
    while (current != -1) {
        shortest_path.insert(shortest_path.begin(), current);
        current = parent[current];
    }

    return shortest_path;
}

void printPath(const vector<int>& path) {
    for (int i = 0; i < path.size(); ++i) {
        cout << (char)('A' + path[i]);
        if (i != path.size() - 1) {
            cout << " -> ";
        }
    }
    cout << endl;
}

int main() {

    vector<vector<int>> graph = {
        {0, 5, 0, 0, 0, 0, 0},
        {0, 0, 3, 7, 0, 0, 0},
        {0, 0, 0, 0, 4, 0, 0},
        {0, 0, 0, 0, 0, 8, 0},
        {0, 0, 0, 0, 0, 0, 6},
        {0, 0, 0, 0, 0, 0, 5},
        {0, 0, 0, 0, 0, 0, 0}
    };

    int start = 0;
    int destination = 6;

    vector<int> shortestPath = dijkstra(graph, start, destination);

    if (shortestPath.empty()) {
        cout << "No path found from the starting location to the destination." << endl;
    } else {
        cout << "The shortest path from A to G is: ";
        printPath(shortestPath);
    }

    return 0;
}
```