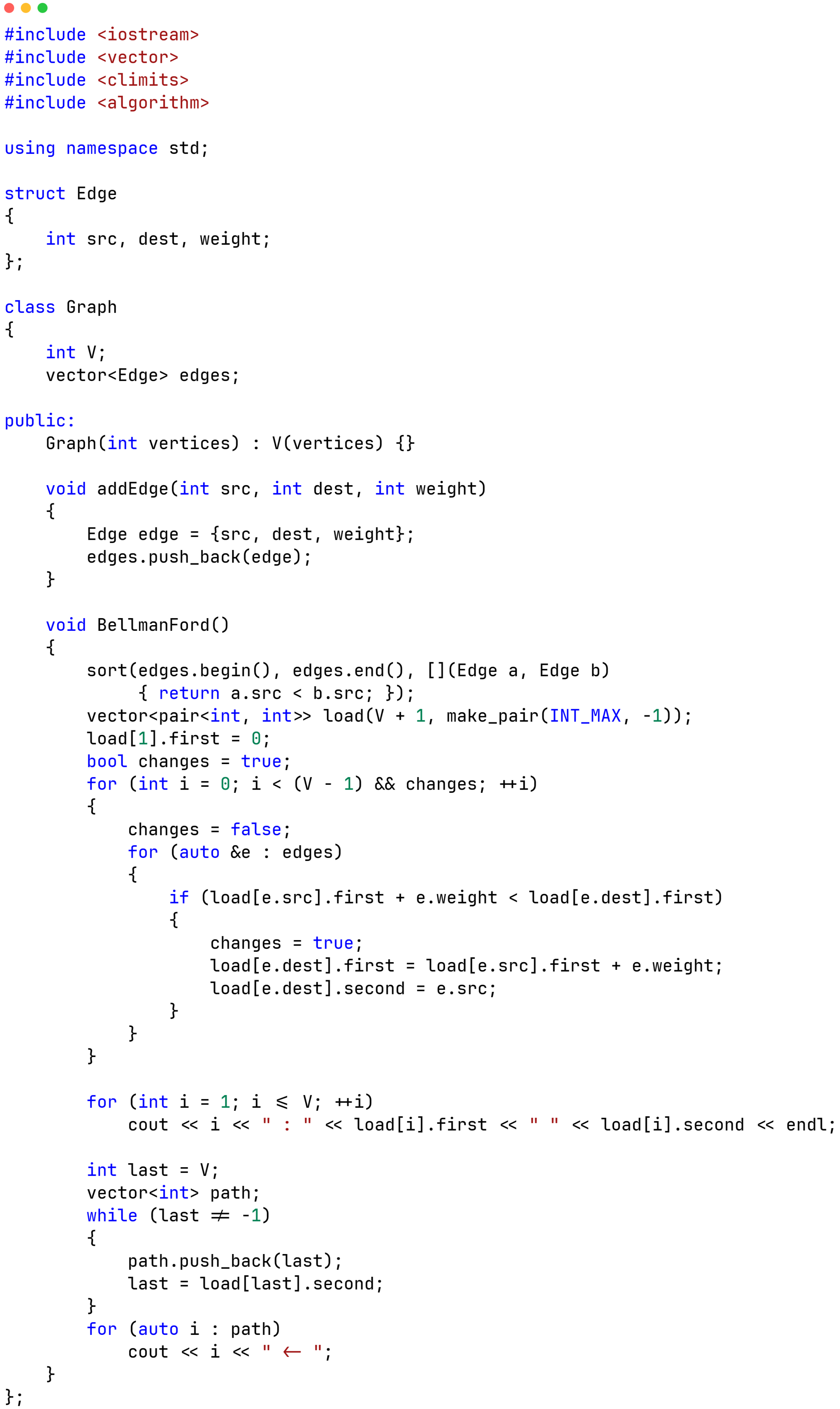
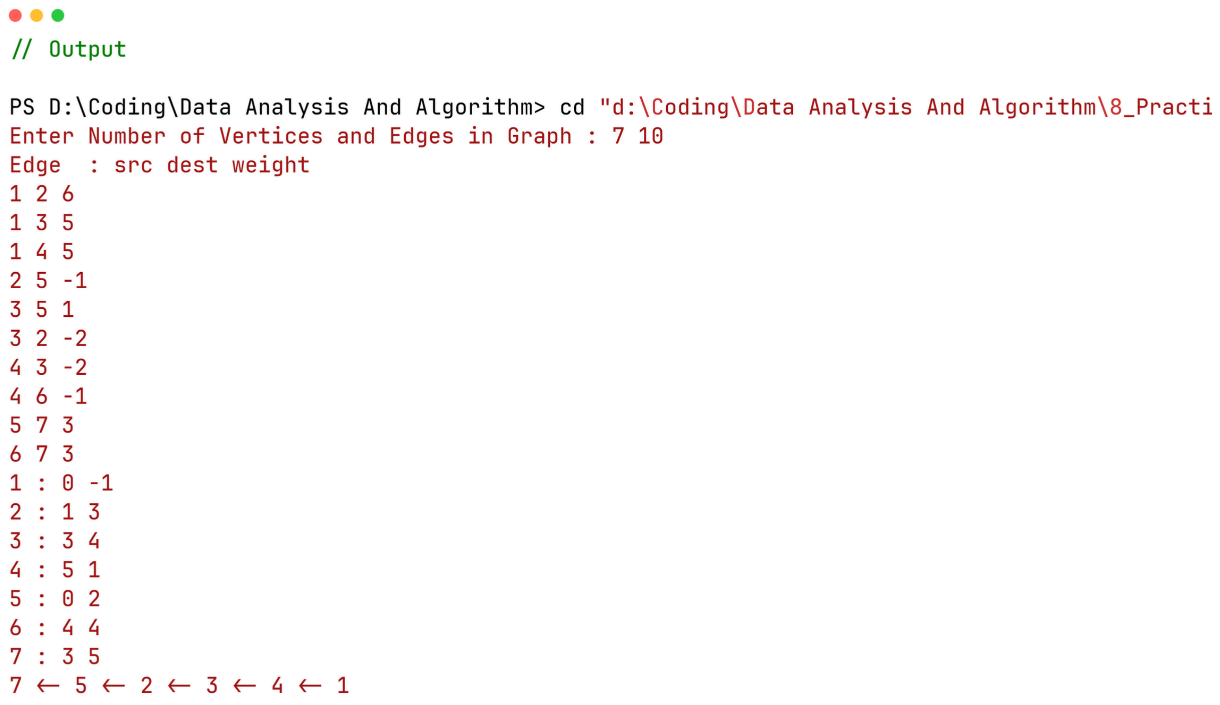
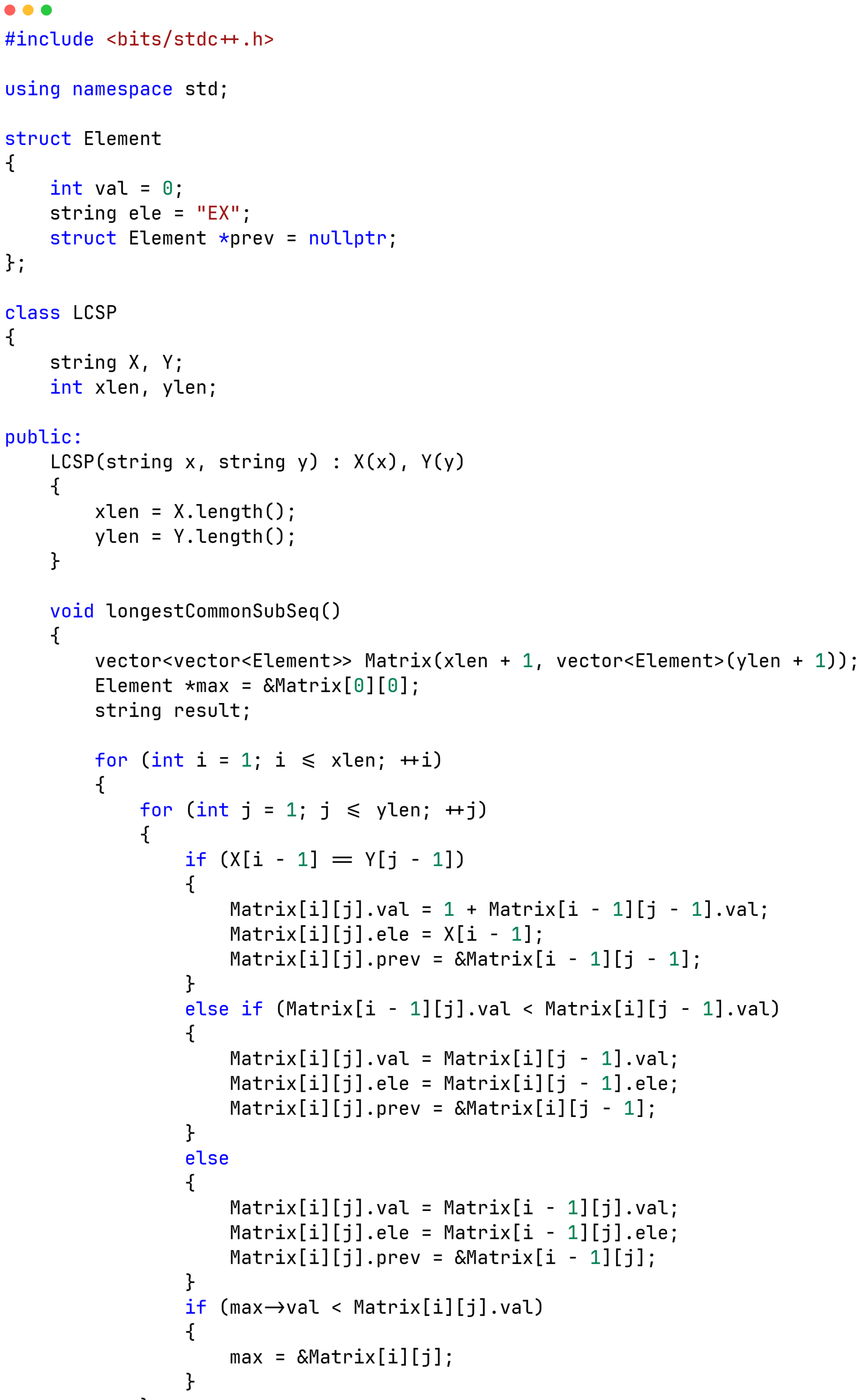


```cpp
#include <iostream>
#include <vector>
#include <climits>
#include <algorithm>

using namespace std;

struct Edge
{
    int src, dest, weight;
};

class Graph
{
    int V;
    vector<Edge> edges;

public:
    Graph(int vertices) : V(vertices) {}

    void addEdge(int src, int dest, int weight)
    {
        Edge edge = {src, dest, weight};
        edges.push_back(edge);
    }

    void BellmanFord()
    {
        sort(edges.begin(), edges.end(), [](Edge a, Edge b)
             { return a.src < b.src; });
        vector<pair<int, int>> load(V + 1, make_pair(INT_MAX, -1));
        load[1].first = 0;
        bool changes = true;
        for (int i = 0; i < (V - 1) && changes; ++i)
        {
            changes = false;
            for (auto &e : edges)
            {
                if (load[e.src].first + e.weight < load[e.dest].first)
                {
                    changes = true;
                    load[e.dest].first = load[e.src].first + e.weight;
                    load[e.dest].second = e.src;
                }
            }
        }

        for (int i = 1; i <= V; ++i)
            cout << i << " : " << load[i].first << " " << load[i].second << endl;

        int last = V;
        vector<int> path;
        while (last != -1)
        {
            path.push_back(last);
            last = load[last].second;
        }
        for (auto i : path)
            cout << i << " <- ";
    }
};
```

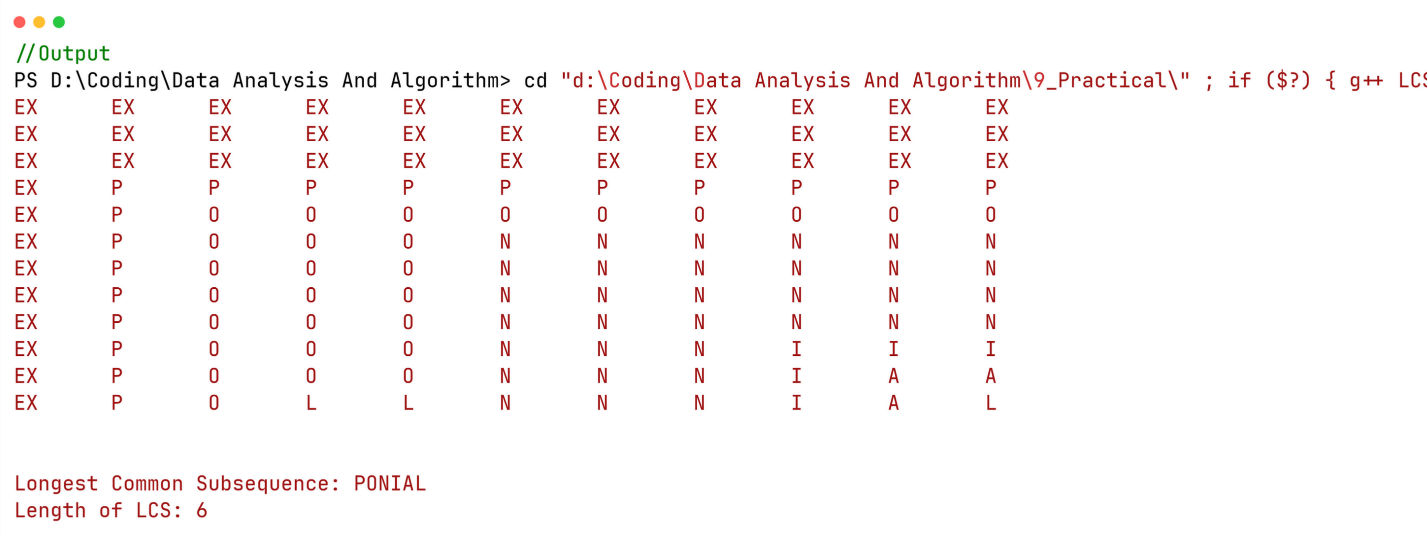
```cpp
int main()
{
    int V, E;
    cout << "Enter Number of Vertices and Edges in Graph : ";
    cin >> V >> E;
    Graph graph(V);

    cout << "Edge  : src dest weight" << endl;
    for (int i = 1; i <= E; ++i)
    {
        int s, d, w;
        // cout << "Edge " << i << " : ";
        cin >> s >> d >> w;
        graph.addEdge(s, d, w);
    }
    graph.BellmanFord();
    cout << endl;
    return 1;
}
```



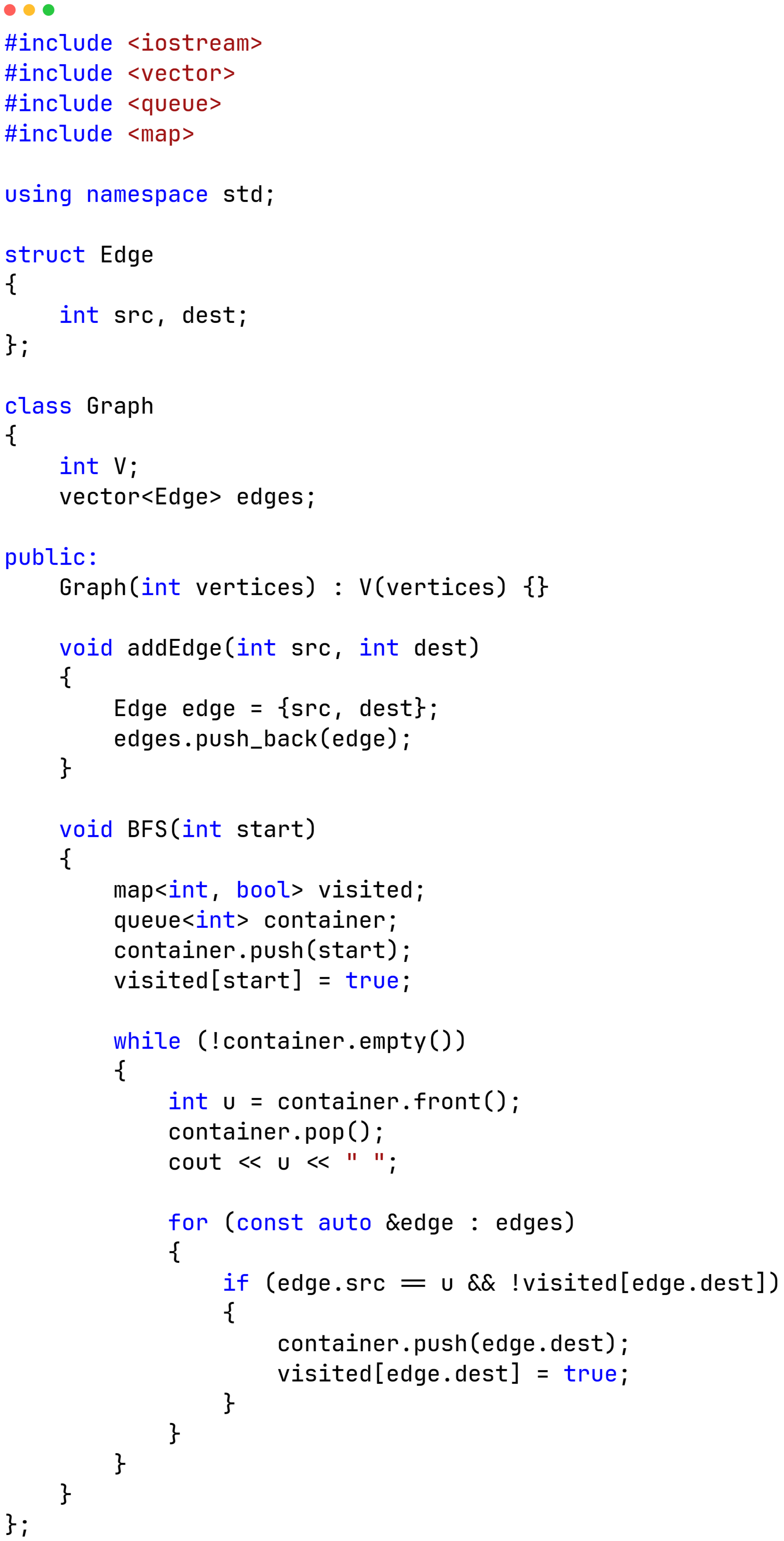
```
// Output

PS D:\Coding\Data Analysis And Algorithm> cd "d:\Coding\Data Analysis And Algorithm\8_Practi
Enter Number of Vertices and Edges in Graph : 7 10
Edge  : src dest weight
1 2 6
1 3 5
1 4 5
2 5 -1
3 5 1
3 2 -2
4 3 -2
4 6 -1
5 7 3
6 7 3
1 : 0 -1
2 : 1 3
3 : 3 4
4 : 5 1
5 : 0 2
6 : 4 4
7 : 3 5
7 <- 5 <- 2 <- 3 <- 4 <- 1
```



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

using namespace std;

struct Element
{
    int val = 0;
    string ele = "EX";
    struct Element *prev = nullptr;
};

class LCSP
{
    string X, Y;
    int xlen, ylen;

public:
    LCSP(string x, string y) : X(x), Y(y)
    {
        xlen = X.length();
        ylen = Y.length();
    }

    void longestCommonSubSeq()
    {
        vector<vector<Element>> Matrix(xlen + 1, vector<Element>(ylen + 1));
        Element *max = &Matrix[0][0];
        string result;

        for (int i = 1; i <= xlen; ++i)
        {
            for (int j = 1; j <= ylen; ++j)
            {
                if (X[i - 1] == Y[j - 1])
                {
                    Matrix[i][j].val = 1 + Matrix[i - 1][j - 1].val;
                    Matrix[i][j].ele = X[i - 1];
                    Matrix[i][j].prev = &Matrix[i - 1][j - 1];
                }
                else if (Matrix[i - 1][j].val < Matrix[i][j - 1].val)
                {
                    Matrix[i][j].val = Matrix[i][j - 1].val;
                    Matrix[i][j].ele = Matrix[i][j - 1].ele;
                    Matrix[i][j].prev = &Matrix[i][j - 1];
                }
                else
                {
                    Matrix[i][j].val = Matrix[i - 1][j].val;
                    Matrix[i][j].ele = Matrix[i - 1][j].ele;
                    Matrix[i][j].prev = &Matrix[i - 1][j];
                }
                if (max->val < Matrix[i][j].val)
                {
                    max = &Matrix[i][j];
                }
            }
        }

        for (int i = 0; i <= xlen; ++i)
        {
            for (int j = 0; j <= ylen; ++j)
            {
                cout << Matrix[i][j].ele << "\t";
            }
            cout << endl;
        }
        cout << endl
             << endl;

        while (max->ele != "EX")
        {
            if (result.empty() || result.back() != max->ele[0])
            {
                result += max->ele;
            }
            max = max->prev;
        }

        reverse(result.begin(), result.end());
        int length = result.size();
        cout << "Longest Common Subsequence: " << result << endl;
        cout << "Length of LCS: " << length << endl;
    }
};

int main()
{
    LCSP saharsh("EXPONENTIAL", "POLYNOMIAL");
    saharsh.longestCommonSubSeq();
    return 0;
}
```

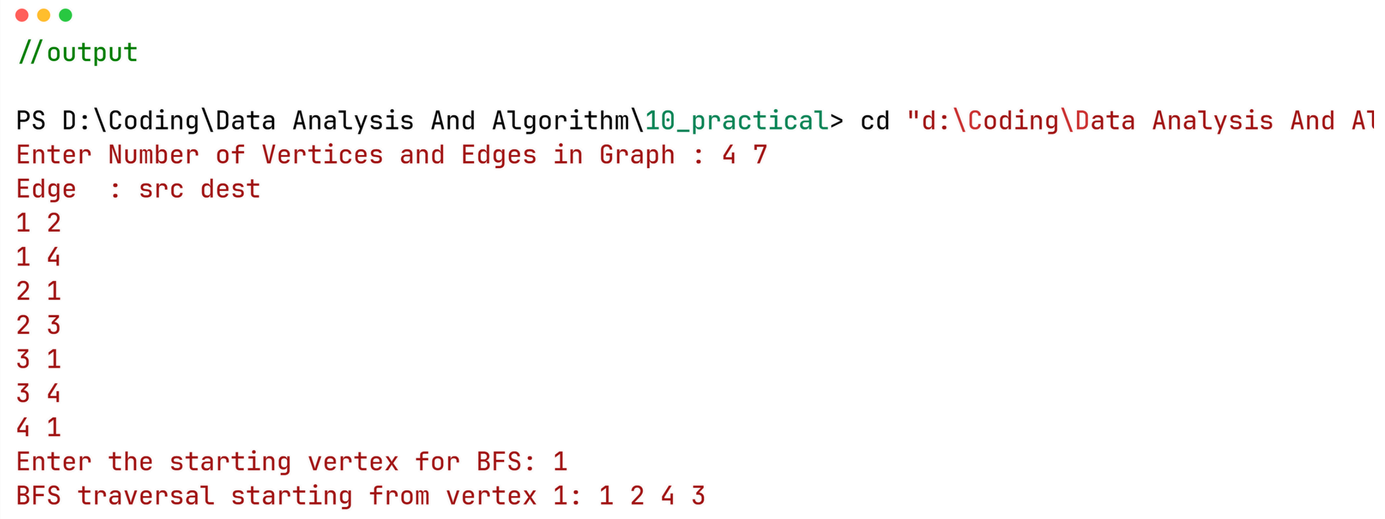


```
//Output
PS D:\Coding\Data Analysis And Algorithm> cd "d:\Coding\Data Analysis And Algorithm\9_Practical\" ; if ($?) { g++ LCS
EX      EX      EX      EX      EX      EX      EX      EX      EX      EX      EX
EX      EX      EX      EX      EX      EX      EX      EX      EX      EX      EX
EX      EX      EX      EX      EX      EX      EX      EX      EX      EX      EX
EX      P       P       P       P       P       P       P       P       P       P
EX      P       O       O       O       O       O       O       O       O       O
EX      P       O       O       O       N       N       N       N       N       N
EX      P       O       O       O       N       N       N       N       N       N
EX      P       O       O       O       N       N       N       N       N       N
EX      P       O       O       O       N       N       N       N       N       N
EX      P       O       O       O       N       N       N       I       I       I
EX      P       O       O       O       N       N       N       I       A       A
EX      P       O       L       L       N       N       N       I       A       L


Longest Common Subsequence: PONIAL
Length of LCS: 6
```



```cpp
#include <iostream>
#include <vector>
#include <queue>
#include <map>

using namespace std;

struct Edge
{
    int src, dest;
};

class Graph
{
    int V;
    vector<Edge> edges;

public:
    Graph(int vertices) : V(vertices) {}

    void addEdge(int src, int dest)
    {
        Edge edge = {src, dest};
        edges.push_back(edge);
    }

    void BFS(int start)
    {
        map<int, bool> visited;
        queue<int> container;
        container.push(start);
        visited[start] = true;

        while (!container.empty())
        {
            int u = container.front();
            container.pop();
            cout << u << " ";

            for (const auto &edge : edges)
            {
                if (edge.src == u && !visited[edge.dest])
                {
                    container.push(edge.dest);
                    visited[edge.dest] = true;
                }
            }
        }
    }
};
```

```cpp
int main()
{
    int V, E;
    cout << "Enter Number of Vertices and Edges in Graph : ";
    cin >> V >> E;
    Graph graph(V);

    cout << "Edge  : src dest" << endl;
    for (int i = 1; i <= E; ++i)
    {
        int s, d;
        cin >> s >> d;
        graph.addEdge(s, d);
    }

    int startVertex;
    cout << "Enter the starting vertex for BFS: ";
    cin >> startVertex;

    cout << "BFS traversal starting from vertex " << startVertex << ": ";
    graph.BFS(startVertex);
    cout << endl;

    return 0;
}
```



```
//output

PS D:\Coding\Data Analysis And Algorithm\10_practical> cd "d:\Coding\Data Analysis And Al
Enter Number of Vertices and Edges in Graph : 4 7
Edge  : src dest
1 2
1 4
2 1
2 3
3 1
3 4
4 1
Enter the starting vertex for BFS: 1
BFS traversal starting from vertex 1: 1 2 4 3
```