

Assignment No : 7

```
#include <iostream>
#include <vector>
#include <cstdint>
#define MAX_NUM_CITIES 10

using namespace std;

struct edge
{
    int start;
    int end;
    int wt;
};

class graph
{
    int adj_mat[MAX_NUM_CITIES]
    [MAX_NUM_CITIES] = {{0}};
    string city_names[MAX_NUM_CITIES];
    int city_count;
    edge mst[MAX_NUM_CITIES - 1];
    void add_to_list(vector<edge> &, edge);
    int cost;

public:
    graph();

    void prims_algo(int);
    void display_mst();
};

void graph::add_to_list(vector<edge> &list, edge e)
{
    list.push_back(e);
    for (int i = list.size() - 1; i > 0; i--)
    {
        if (list[i].wt < list[i - 1].wt)
        {
            swap(list[i], list[i - 1]);
        } else {
            break;
        }
    }
}

graph::graph()
{
    cost = 0;
    cout << "Number of cities are (1-" << MAX_NUM_CITIES << "):\t";
    cin >> city_count;
    city_count = (city_count > MAX_NUM_CITIES) ? MAX_NUM_CITIES : city_count;
```

```

for (int i = 0; i < city_count; i++)
{
    cout << "Enter city:\n" << i + 1 << ":\t";
    cin >> city_names[i];
}

```

```

for (int i = 0; i < city_count; i++)
    for (int j = 0; j < city_count; j++) adj_mat[i][j] = INT32_MAX;

```

```

int num_pairs;
cout << "Number of city pairs are:\t";
cin >> num_pairs;
cout << "City codes are:\t" << endl;
for (int i = 0; i < city_count; i++) {
    cout << i << " - " << city_names[i] << endl;
}
int x, y, wt;
for (int i = 0; i < num_pairs; i++)
{
    cout << "Enter pair:\n" << i + 1 << ":\t";
    cin >> x >> y;
    cout << "Enter cost between city " << city_names[x] << " & city "
        << city_names[y] << ":\t";
    cin >> wt;
    adj_mat[x][y] = wt;
    adj_mat[y][x] = wt;
}
}

```

```

void graph::prims_algo(int start)
{
    bool visited[MAX_NUM_CITIES] = {0};
    int visited_count = 1;
    visited[start] = 1;
    vector<edge> adj;
    for (int i = 0; i < city_count; i++)
    {
        if (adj_mat[start][i] != INT32_MAX)
        {
            edge e;
            e.start = start;
            e.end = i;
            e.wt = adj_mat[start][i];
            add_to_list(adj, e);
        }
    }
}

```

```

while (visited_count != city_count)
{
    edge m = adj.front();

```

```

adj.erase(adj.begin());
if (!visited[m.end]) {
    mst[visited_count - 1] = m;
    cost += m.wt;
    for (int i = 0; i < city_count; i++)
    {
        if (adj_mat[m.end][i] != INT32_MAX)
        {
            edge e;
            e.start = m.end;
            e.end = i;
            e.wt = adj_mat[e.start][i];
            add_to_list(adj, e);
        }
    }
    visited[m.end] = 1;
    visited_count++;
}
}
}

void graph::display_mst()
{
    cout << "Most efficient network is:\t" << endl;

    for (int i = 0; i < city_count - 1; i++)
    {
        cout << city_names[mst[i].start] << " to " << city_names[mst[i].end]
            << " of weight " << mst[i].wt << endl;
    }
    cout << endl << "The cost of network is:\t" << cost << endl;
}

int main()
{
    graph g;
    int start;
    cout << "Enter beginning city:\t";
    cin >> start;
    start = (start > MAX_NUM_CITIES - 1) ? 0 : start;
    g.prim's_algo(start);
    g.display_mst();
    return 0;
}

```

```
#include <iostream>
#include <vector>
#include <cstdlib>
#define MAX_NUM_CITIES 10
```

```
using namespace std;
```

```
struct edge
```

```
{
    int start;
    int end;
    int wt;
};
```

```
class graph
```

```
{
    int adj_mat[MAX_NUM_CITIES]
    [MAX_NUM_CITIES] = {{0}};
    string city_names[MAX_NUM_CITIES];
    int city_count;
    edge mst[MAX_NUM_CITIES - 1];
    void add_to_list(vector<edge> &, edge);
    int cost;

```

```
public:
```

```
graph();
```

```
void prims_algo(int);
```

```
void display_mst();
```

```
};
```

```
void graph::add_to_list(vector<edge> &list, edge e)
```

```
{
    list.push_back(e);
    for (int i = list.size() - 1; i > 0; i--)
    {
        if (list[i].wt < list[i - 1].wt)
        {
            swap(list[i], list[i - 1]);
        } else {
            break;
        }
    }
}
```

```
student@student-OptiPlex-3010: ~/Desktop/Nikita
```

```
student@student-OptiPlex-3010:~$ cd Desktop
```

```
student@student-OptiPlex-3010:~/Desktop$ cd Nikita
```

```
student@student-OptiPlex-3010:~/Desktop/Nikita$ g++ -std=c++11 Ass7.cpp -o as7
```

```
student@student-OptiPlex-3010:~/Desktop/Nikita$ ./as7
```

```
Number of cities are (1-10): 3
```

```
Enter city:
```

```
1: pune
```

```
Enter city:
```

```
2: latur
```

```
Enter city:
```

```
3: satara
```

```
Number of city pairs are: 2
```

```
City codes are:
```

```
0 - pune
```

```
1 - latur
```

```
2 - satara
```

```
Enter pair:
```

```
1: 1
```

```
2: 0
```

```
Enter cost between city latur & city satara: 50
```

```
Enter pair:
```

```
2: 0
```

```
1: 1
```

```
Enter cost between city pune & city latur: 25
```

```
Enter beginning city: latur
```

```
Most efficient network is:
```

```
pune to latur of weight 25
```

```
latur to satara of weight 50
```

```
The cost of network is: 75
```

```
student@student-OptiPlex-3010:~/Desktop/Nikita$
```