Міністерство освіти і науки України
Центральноукраїнський національний технічний університет
Механіко-технологічний факультет
Кафедра кібербезпеки та програмного забезпечення
Дисципліна: Алгоритми та структури даних

Лабораторна робота №1

Тема: «АЛГОРИТМИ РОБОТИ З ГРАФАМИ. АЛГОРИТМ ПРИМА-КРУСКАЛА»

Виконав: ст. гр. КН-24
Куріщенко П. В.
Перевірив: викладач
Константинова Л. В.

Кропивницький 2025

Мета: Навчитися реалізовувати алгоритм Прима-Крускала для побудови каркасного дерева графу.

Варіант - 14

Завдання: Реалізувати програмно абстрактний тип даних неорієнтований граф. Створити його екземпляр, який заповнити даними, вказаними у Вашому варіанті. Розробити програму, що повинна побудувати для даного графу мінімальне каркасне дерево, використовуючи алгоритм Прима-Крускала, визначити сумарну вагу побудованого дерева, вивести одержані дані на екран.

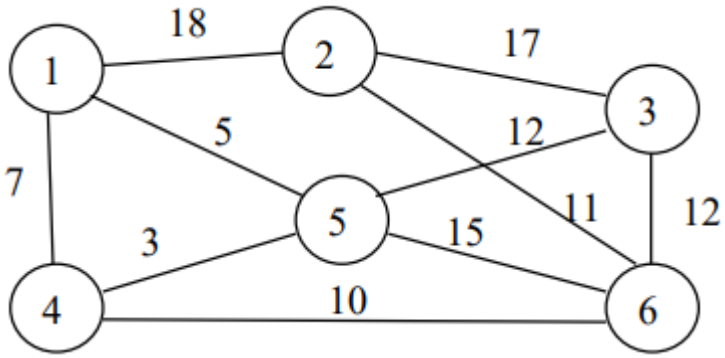

Рис. 1 – приклад неорієнтованого графу

Лістинг *graph.h*:

```
#ifndef GRAPH_H
#define GRAPH_H

#include <vector>
#include <iostream>

using namespace std;

struct Edge {
    int startVertex;
    int endVertex;
    int weight;
};

class Graph {
public:
    Graph(int vertices);

    void addEdge(int start, int end, int weight);
```

```
    void kruskalMST();

private:
    int vertexAmount;
    vector<Edge> edges;

    //helpers
    int find(int vertices, vector<int>& parent);
    void unite(int first, int second, vector<int>& parent);
    static bool compareEdges(const Edge&, const Edge&);
};

#endif // GRAPH_H
```

Лістинг *graph.cpp*:

```
#include "graph.h"

Graph::Graph(int vertices) : vertexAmount(vertices) {}

void Graph::addEdge(int start, int end, int weight) {
    edges.push_back({start, end, weight});
}

int Graph::find(int vertices, std::vector<int>& parent) {
    if (parent[vertices] == vertices) return vertices;
    return parent[vertices] = find(parent[vertices], parent);
}

void Graph::unite(int first, int second, std::vector<int>& parent) {
    first = find(first, parent);
    second = find(second, parent);
    if (first != second) parent[second] = first;
}

bool Graph::compareEdges(const Edge& first, const Edge& second) { return
first.weight < second.weight; }

void Graph::kruskalMST() {
    sort(edges.begin(), edges.end(), Graph::compareEdges);

    vector<int> parent(vertexAmount + 1);
    for (int index = 1; index <= vertexAmount; index++)
        parent[index] = index;

    int totalWeight = 0;
    int edgeCount = 0;
    cout << "Мінімальне каркасне дерево:\n";

    for (Edge edge : edges) {
        if (find(edge.startVertex, parent) != find(edge.endVertex, parent)) {
            unite(edge.startVertex, edge.endVertex, parent);
            totalWeight += edge.weight;
            cout << edge.startVertex << " <-(" << edge.weight << ")-> " <<
edge.endVertex << endl;

            edgeCount++;
            if (edgeCount == vertexAmount - 1) break;
```

```
        }
    }

    cout << "Сумарна вага: " << totalWeight << endl;
}
```

Лістинг *main.cpp*:

```
#include "graph.h"

int main() {
    Graph graph(6);

    vector<Edge> edges = {
        {1, 2, 18},
        {2, 3, 17},
        {3, 5, 12},
        {3, 6, 12},
        {5, 1, 5},
        {5, 6, 15},
        {5, 4, 3},
        {4, 6, 10},
        {6, 2, 11},
        {1, 4, 7}
    };

    for (const Edge& edge : edges)
        graph.addEdge(edge.startVertex, edge.endVertex, edge.weight);

    graph.kruskalMST();
}
```

Результат виконання:

```
Мінімальне каркасне дерево:
5 <-(3)-> 4
5 <-(5)-> 1
4 <-(10)-> 6
6 <-(11)-> 2
3 <-(12)-> 5
Сумарна вага: 41
```

Висновок: У ході виконання роботи було реалізовано алгоритм Прима-Крускала для побудови мінімального каркасного дерева графа. Отримано практичні навички роботи з графами, сортуванням ребер та використанням структури даних для об’єднання множин (DSU) з метою уникнення циклів. У результаті програма коректно будує мінімальне каркасне дерево та обчислює його сумарну вагу.