МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ
УЧРЕЖДЕНИЕ ОБРАЗОВАНИЯ
“БРЕСТСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ”
КАФЕДРА ИИТ

ОТЧЁТ

по лабораторной работе №3

«Нахождение минимального остовного дерева связанного неориентированного графа»

Выполнил:

Студент 2 курса
группы ПО-9
Харитонович Захар Сергеевич

Проверила:

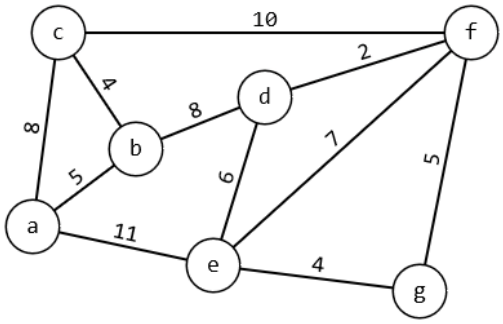
Глущенко Т. А.

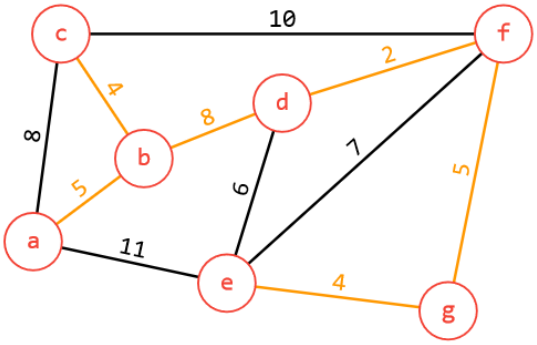
Брест 2022

Задание.

1. Найти минимальное остовное дерево для заданного графа G алгоритмом Прима и Крускаля. Варианты графов указаны в таблице 1. Граф задан списком ребер.
2. Ответить на поставленные вопросы.
3. Графически изобразить граф и его минимальное остовное дерево.

10.	7	11	(a,b),(a,c),(a,e) (b,c),(b,d), (c,f), (d,e),(d,f),(e,g),(e,f),(g,f)	5,8,11,4,8,10, 6,2,4,7,5





Листинг кода:

```
main.cpp
#include <iostream>
#include "prime.h"
#include "kruskal.h"

int main() {
    int n;
    std::cout << "Node amount: ";
    std::cin >> n;
    std::vector<std::pair<int, int>> graph[n];
    int m;
    std::cout << "Edge amount: ";
    std::cin >> m;
    std::vector<edge> edges;
    for (int i = 0; i < m; i++) {
        int a, b, w; //a node, b node, weight
        std::cin >> a >> b >> w;
        graph[a].push_back(std::make_pair(b, w));
        edge curEdge;
        curEdge.a = a;
        curEdge.b = b;
        curEdge.len = w;
        edges.push_back(curEdge);
    }
    std::cout << "Prime:" << std::endl;
    prime(graph, n);
    std::cout << std::endl << "Kruskal:" << std::endl;
    kruskal(edges, n);
    return 0;
}
prime.h
#ifndef LAB6_PRIME_H
#define LAB6_PRIME_H

#include <bits/stdc++.h>

void prime(std::vector <std::pair<int, int>> graph[], int size);
#endif //LAB6_PRIME_H
```

prime.cpp

```
#include "prime.h"

void prime(std::vector <std::pair<int, int>> graph[], int size) {
    bool used[size];
    int pr[size];
    for (int i = 0; i < size; i++) {
        pr[i] = -1;
        used[i] = 0;
    }
    int mst_weight = 0;

    std::priority_queue<std::pair<int,    int>,    std::vector<std::pair<int,
int>>, std::greater<std::pair<int, int>>> q;

    q.push({0, 0});
    int last;
    while (!q.empty()) {
        std::pair<int, int> c = q.top();
        q.pop();

        int dst = c.first, v = c.second;

        if (used[v]) {
            continue;
        }
        last = v;
        used[v] = true;
        mst_weight += dst;

        for (auto e: graph[v]) {
            int u = e.first, len_vu = e.second;

            if (!used[u]) {
                pr[u] = v;
                q.push({len_vu, u});
            }
        }
    }

    std::cout << "Minimum spanning tree weight is " << mst_weight << ":" <<
std::endl;
    std::vector<int> path;

    path.push_back(last);

    while (pr[last] != -1) {
        last = pr[last];
        path.push_back(last);
    }
    reverse(path.begin(), path.end());
    for (auto v: path) {
        std::cout << v + 1 << " ";
    }
    std::cout << std::endl;
}
```

kruskal.h

```
#ifndef LAB6_KRUSKAL_H
#define LAB6_KRUSKAL_H

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

```
struct edge {
    int a, b, len;

    bool operator<(const edge &other) {
        return len < other.len;
    }
};

void kruskal(std::vector<edge> edges, int size);
#endif //LAB6_KRUSKAL_H
```

kruskal.cpp

```
#include "kruskal.h"

int get_root(int *p, int v) {
    if (p[v] == v) {
        return v;
    } else {
        return p[v] = get_root(p, p[v]);
    }
}

bool merge(int *p, int *rk, int a, int b) {
    int ra = get_root(p, a), rb = get_root(p, b);

    if (ra == rb) {
        return false;
    } else {
        if (rk[ra] < rk[rb]) {
            p[ra] = rb;
        } else if (rk[rb] < rk[ra]) {
            p[rb] = ra;
        } else {
            p[ra] = rb;
            rk[rb]++;
        }

        return true;
    }
}

void kruskal(std::vector<edge> edges, int size) {
    int p[size];
    int rk[size];
    for (int i = 0; i < size; i++) {
        p[i] = i;
        rk[i] = 1;
    }
    std::sort(edges.begin(), edges.end());

    int mst_weight = 0;
    std::vector<edge> res;
    for (edge e: edges) {
        if (merge(p, rk, e.a, e.b)) {
            mst_weight += e.len;
            res.push_back(e);
        }
    }
    std::cout << "Minimum spanning tree weight is " << mst_weight << ":" <<
std::endl;
    for (auto e : res) {
```

```
        std::cout << e.a << " - " << e.b << std::endl;
    }
}
```

Результат работы:

```
$ ./lab6.exe
Node amount: 7
Edge amount: 11
0 1 5
0 2 8
0 4 11
1 2 4
1 3 8
2 5 10
3 4 6
3 5 2
4 6 4
4 5 7
5 6 5
Prime:
Minimum spanning tree weight is 28

Kruskal:
Minimum spanning tree weight is 28:
3 - 5
1 - 2
4 - 6
0 - 1
5 - 6
1 - 3
```

Контрольные вопросы

Минимальное остовное дерево

Вопросы.

1. Для какого графа определяет число остовных деревьев формула Кэли?

Теорема Кэли о числе деревьев – теорема, утверждающая, что число деревьев с n пронумерованными вершинами равно n^{n-2}. Используется для неориентированных графов.

2. Подсчитать по формуле Кэли число остовных деревьев для n = 3.

$3^{3-2} = 3$

3. * пункт 2 выполнить для n = 4.

$4^{4-2} = 16$

4. Какое остовное дерево находится алгоритмом Дейкстры?

С наименьшим весом.

5. Может ли быть несколько минимальных остовных деревьев?

Возможно выделить в графе несколько остовных деревьев, каждое из которых будет являться минимальным, при этом величина минимального остовного дерева для всех этих деревьев будет равной.