Практическая работа № 11

ПОСТРОЕНИЕ ОСТОВНОГО ДЕРЕВА ГРАФА

Вариант-11.1.3

Постановка задачи

Составить программу реализации алгоритма Крускала построения остовного дерева минимального веса.

Выбрать и реализовать способ представления графа в памяти.

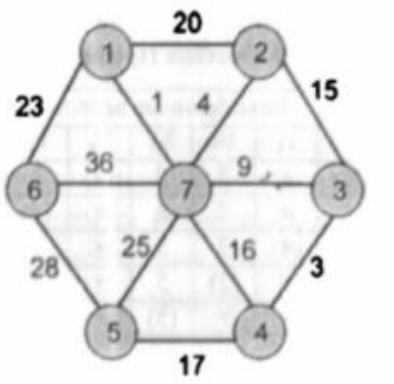
Предусмотреть ввод с клавиатуры произвольного графа.

Разработать доступный способ (форму) вывода результирующего дерева на экран монитора.

Провести тестовый прогон программы для заданного графа в соответствии с индивидуальным заданием

Индивидуальное задание:

11.1.3



1. Описание алгоритма

Алгоритм программы состоит из функции main и вызываемых в ней вспомогательных функций:

- **void add** – функция добавления связи в граф.
- **void show**– функция графа.
- **void sort** – функция сортировки графа по весам.
- **void make_min_ostav**– функция создания минимального оставного дерева.

Алгоритм Краскала — эффективный алгоритм построения минимального остовного дерева взвешенного связного неориентированного графа. Также алгоритм используется для нахождения некоторых приближений для задачи Штейнера. В начале текущее множество рёбер устанавливается пустым. Затем, пока это возможно, проводится следующая операция: из всех рёбер, добавление которых к уже имеющемуся множеству не вызовет появление в нём цикла, выбирается ребро минимального веса и добавляется к уже имеющемуся множеству. Когда таких рёбер больше нет, алгоритм завершён. Подграф данного графа, содержащий все его вершины и найденное множество рёбер, является его остовным деревом минимального веса. Подробное описание алгоритма можно найти в литературе.

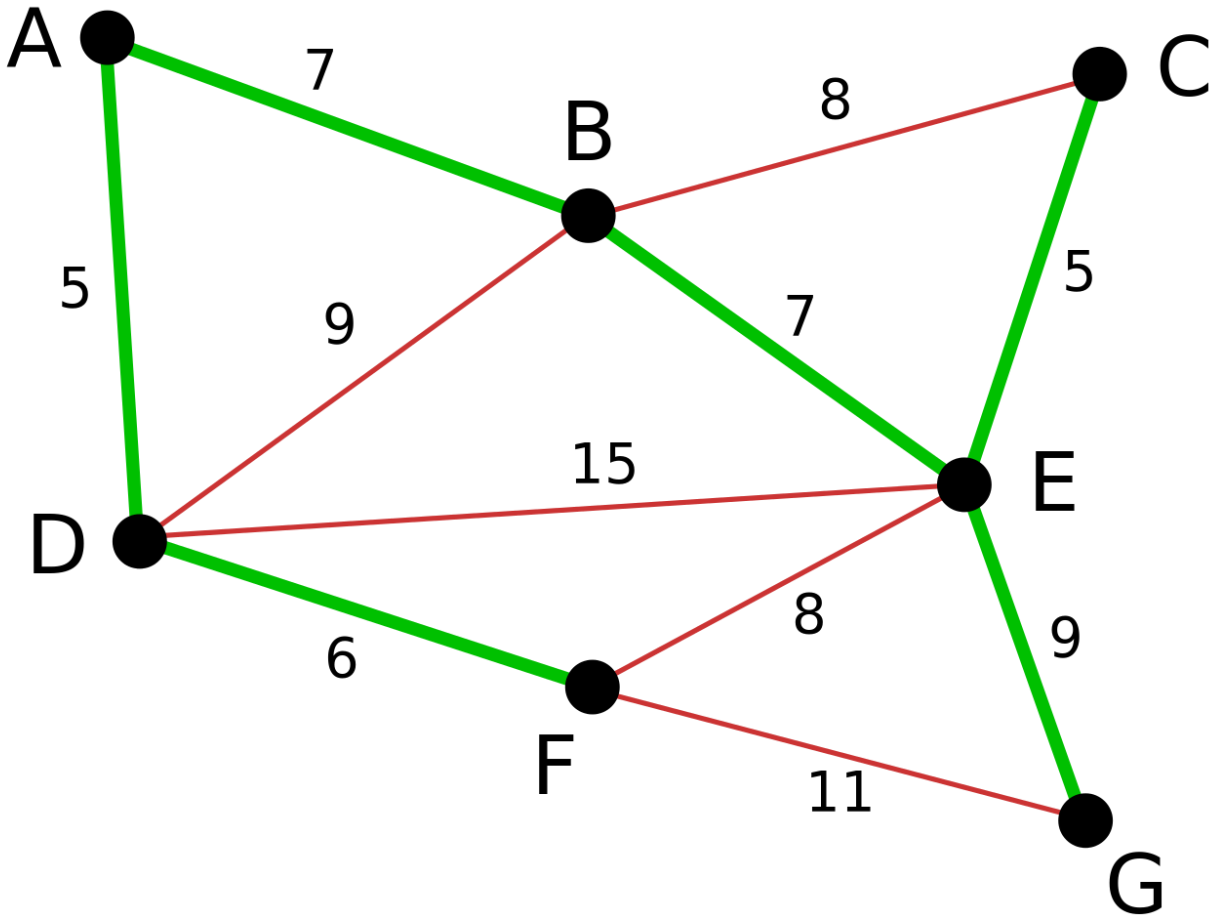


Рис.1 Схема минимального остовного дерева

Функция main создает объект класса Graph и вызывает меню.

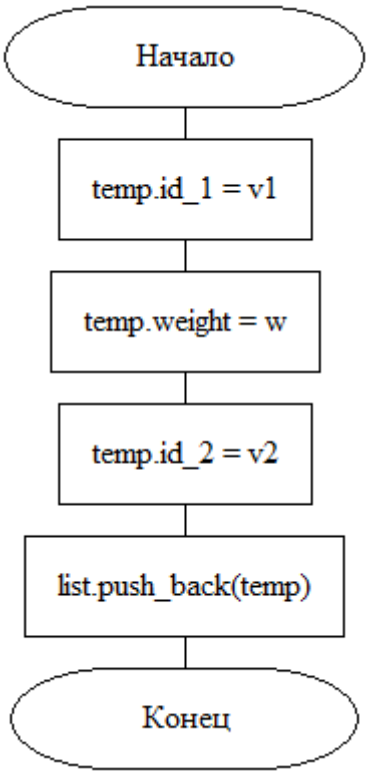


Рис.2 Схема алгоритма функции add

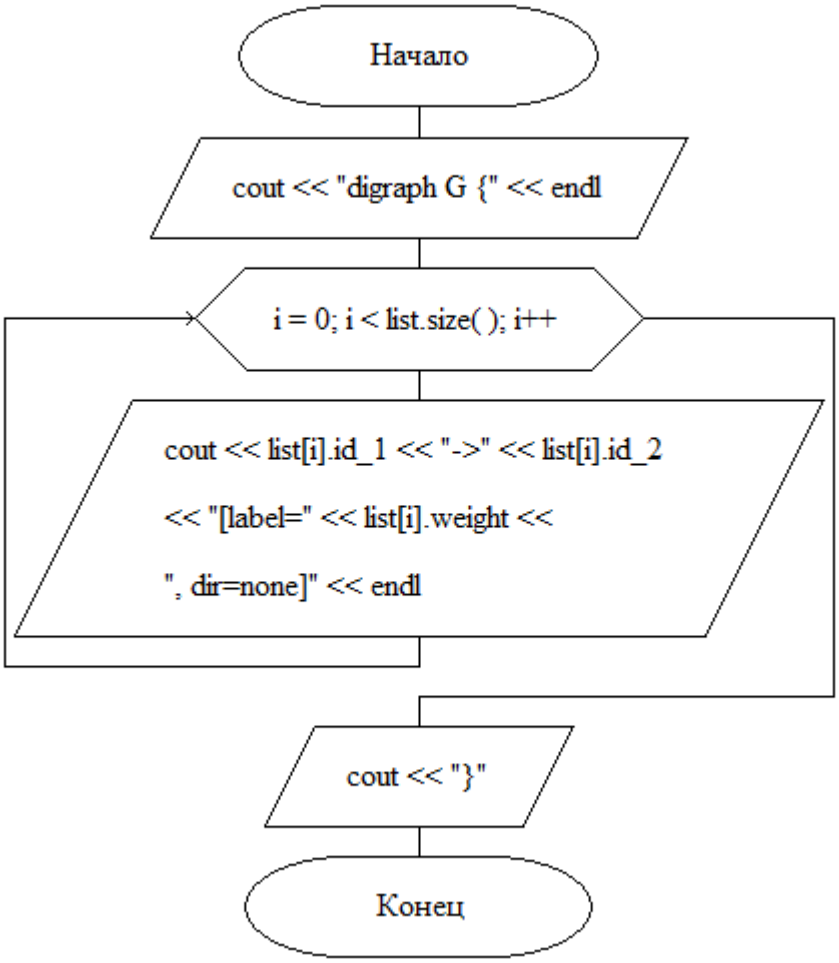


Рис.3 Схема алгоритма функции show

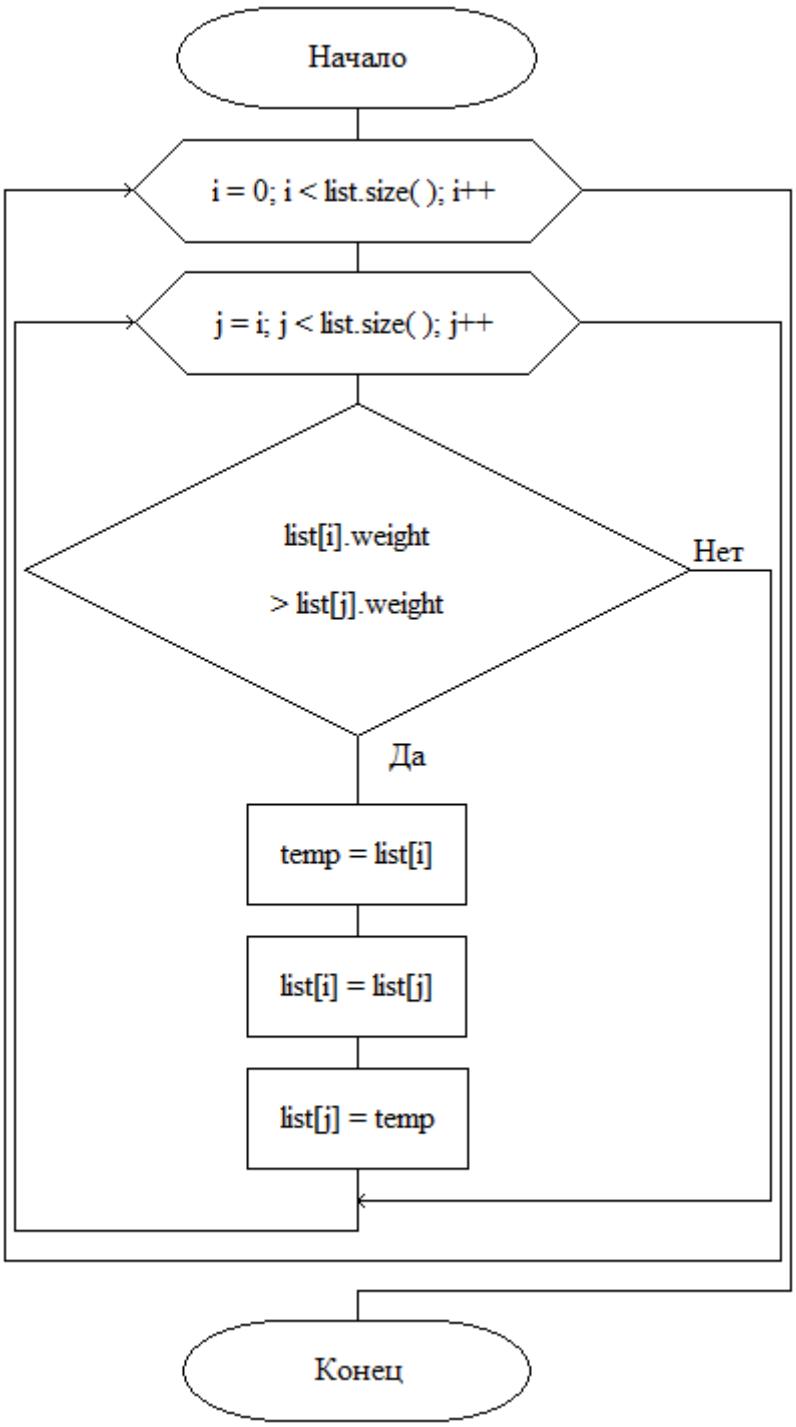


Рис.4 Схема алгоритма функции sort

Рис.5 Схема алгоритма функции вставки первой записи

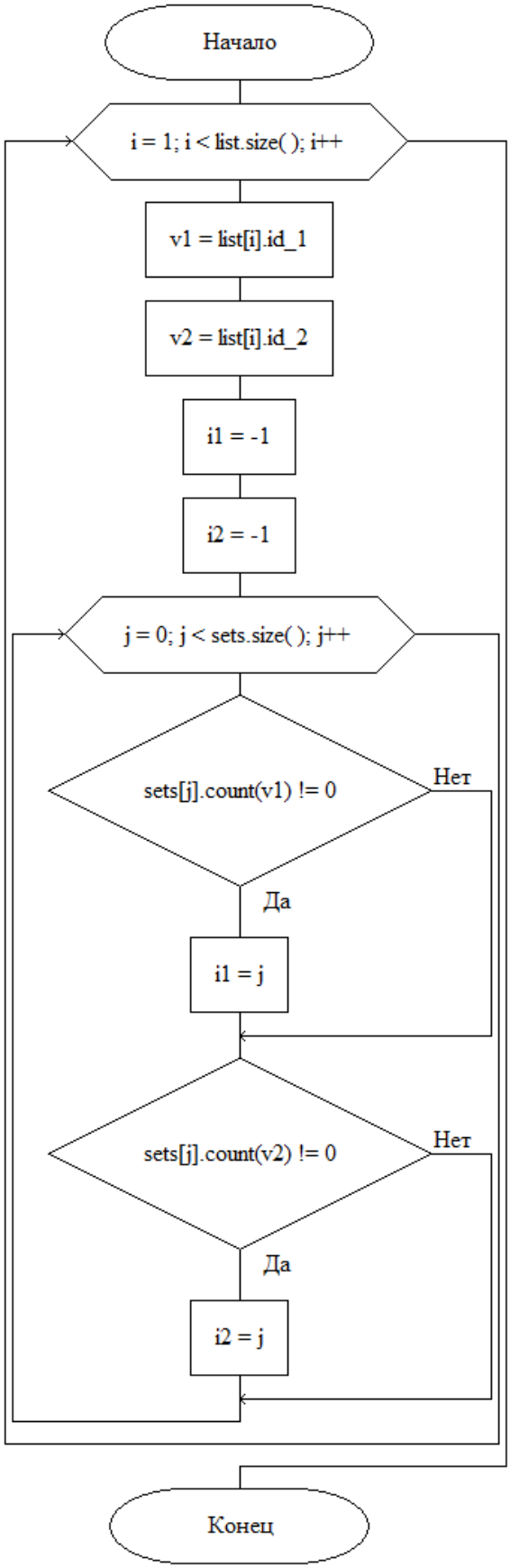


Рис.6 Схема алгоритма функции поиска индексов во множествах

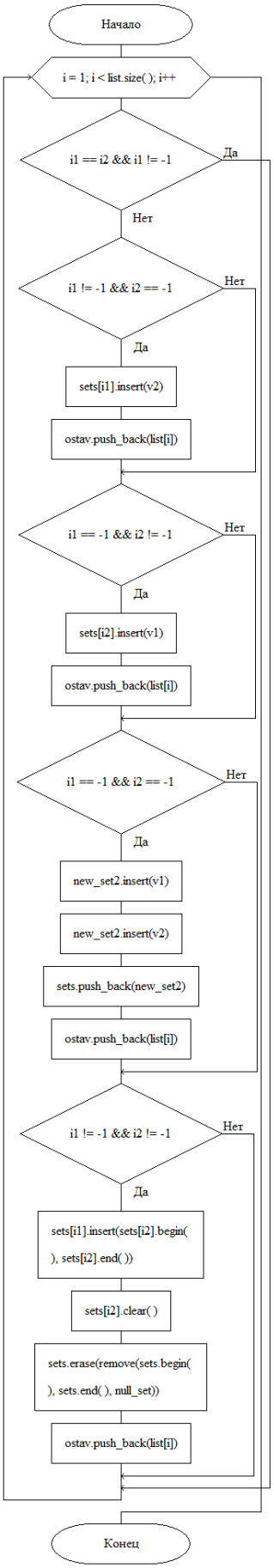


Рис.5 Схема алгоритма функции make_min_ostav

Реализация алгоритма

Текст исходного кода программы

main.cpp

```
#include "Graph.h"
void menu(Graph g) {
	cout << "Выберите команду:" << endl;
	cout << "[1] - Добавить связь." << endl;
	cout << "[2] - Вывести граф (в DOT-нотации)." << endl;
	cout << "[3] - Вывести минимальное оставное дерево (в DOT-нотации)." << endl;
	cout << "[4] - Завершить программу." << endl;
	cout << "---->";
	int ch = 0;
	cin >> ch;
	if (ch == 1) {
		cout << "Введите номер 1-й вершины, вес ребра, номер 2-й вершины (через
пробелы)" << endl;
		int x1, x2, x3;
		cin >> x1 >> x2 >> x3;
		g.add(x1, x2, x3);
		menu(g);
	}
	if (ch == 2) {
		g.show();
		menu(g);
	}
	if (ch == 3) {
		g.make_min_ostav();
		menu(g);
	}
	if (ch == 4) {
		cout << "Программа завершена";
	}
}

int main()
{
	setlocale(LC_ALL, "Russian");
    Graph a;
    menu(a);
}
```

Graph.h

```
#pragma once
#include <set>
#include <iostream>
#include <vector>
#include <algorithm>
using namespace std;
struct Connection
{
	int id_1 = 0;
	int weight = 0;
	int id_2 = 0;
};
class Graph
{
```

```
private:
	vector<Connection> list;

public:
	void add(int v1, int w, int v2);
	void show();
	void sort();
	void make_min_ostav();
};
```

Graph.cpp

```
#include "Graph.h"

void Graph::add(int v1, int w, int v2) {
	Connection temp;
	temp.id_1 = v1;
	temp.weight = w;
	temp.id_2 = v2;

	list.push_back(temp);

}

void Graph::show() {
	cout << "digraph G {" << endl;
	for (int i = 0; i < list.size(); i++) {

		cout << list[i].id_1 << "->" << list[i].id_2 << "[label=" << list[i].weight
<< ", dir=none]" << endl;

	}
	cout << "}";
}

void Graph::sort() {
	for (int i = 0; i < list.size(); i++) {
		for (int j = i; j < list.size(); j++) {
			if (list[i].weight > list[j].weight) {
				Connection temp = list[i];
				list[i] = list[j];
				list[j] = temp;
			}
		}
	}
}

void Graph::make_min_ostav() {
	sort();
	set<int> null_set;
	vector<set<int>> sets;
	vector<Connection> ostav;
	set<int> new_set;
	new_set.insert(list[0].id_1);
	new_set.insert(list[0].id_2);
	sets.push_back(new_set);
	ostav.push_back(list[0]);
	for (int i = 1; i < list.size(); i++) {
		int v1 = list[i].id_1;
		int v2 = list[i].id_2;
		int i1 = -1;
		int i2 = -1;
```

```
		for (int j = 0; j < sets.size(); j++) {
			if (sets[j].count(v1) != 0)i1 = j;
			if (sets[j].count(v2) != 0)i2 = j;
		}
		if (i1 == i2 && i1!=-1) {
			continue;
		}
		if (i1 != -1 && i2 == -1) {
			sets[i1].insert(v2);
			ostav.push_back(list[i]);
		}
		if (i1 == -1 && i2 != -1) {
			sets[i2].insert(v1);
			ostav.push_back(list[i]);
		}
		if (i1 == -1 && i2 == -1) {
			set<int> new_set2;
			new_set2.insert(v1);
			new_set2.insert(v2);
			sets.push_back(new_set2);
			ostav.push_back(list[i]);
		}
		if (i1 != -1 && i2 != -1) {
			sets[i1].insert(sets[i2].begin(), sets[i2].end());
			sets[i2].clear();
			sets.erase(remove(sets.begin(), sets.end(), null_set));
			ostav.push_back(list[i]);
		}

	}
	cout << "digraph G {" << endl;
	for (int i = 0; i < ostav.size(); i++) {

		cout << ostav[i].id_1 << "->" << ostav[i].id_2
<<"[label="<<ostav[i].weight<<", dir=none]"<<endl;

	}
	cout << "}";
	}
```

2. Тестирование программы

```
Выберите команду:
[1] - Добавить связь.
[2] - Вывести граф (в DOT-нотации).
[3] - Вывести минимальное оставное дерево (в DOT-нотации).
[4] - Завершить программу.
---->1
Введите номер 1-й вершины, вес ребра, номер 2-й вершины (через пробелы)
1 20 2
Выберите команду:
[1] - Добавить связь.
[2] - Вывести граф (в DOT-нотации).
[3] - Вывести минимальное оставное дерево (в DOT-нотации).
[4] - Завершить программу.
---->1
Введите номер 1-й вершины, вес ребра, номер 2-й вершины (через пробелы)
2 15 3
Выберите команду:
[1] - Добавить связь.
[2] - Вывести граф (в DOT-нотации).
[3] - Вывести минимальное оставное дерево (в DOT-нотации).
[4] - Завершить программу.
---->1
Введите номер 1-й вершины, вес ребра, номер 2-й вершины (через пробелы)
3 3 4
Выберите команду:
[1] - Добавить связь.
[2] - Вывести граф (в DOT-нотации).
[3] - Вывести минимальное оставное дерево (в DOT-нотации).
[4] - Завершить программу.
---->1
Введите номер 1-й вершины, вес ребра, номер 2-й вершины (через пробелы)
4 17 5
Выберите команду:
[1] - Добавить связь.
[2] - Вывести граф (в DOT-нотации).
[3] - Вывести минимальное оставное дерево (в DOT-нотации).
[4] - Завершить программу.
---->1
Введите номер 1-й вершины, вес ребра, номер 2-й вершины (через пробелы)
5 28 6
Выберите команду:
[1] - Добавить связь.
[2] - Вывести граф (в DOT-нотации).
[3] - Вывести минимальное оставное дерево (в DOT-нотации).
[4] - Завершить программу.
---->1
Введите номер 1-й вершины, вес ребра, номер 2-й вершины (через пробелы)
6 23 1
Выберите команду:
[1] - Добавить связь.
[2] - Вывести граф (в DOT-нотации).
[3] - Вывести минимальное оставное дерево (в DOT-нотации).
[4] - Завершить программу.
---->1
Введите номер 1-й вершины, вес ребра, номер 2-й вершины (через пробелы)
1 1 7
Выберите команду:
[1] - Добавить связь.
[2] - Вывести граф (в DOT-нотации).
[3] - Вывести минимальное оставное дерево (в DOT-нотации).
[4] - Завершить программу.
---->1
Введите номер 1-й вершины, вес ребра, номер 2-й вершины (через пробелы)
2 4 7
Выберите команду:
[1] - Добавить связь.
[2] - Вывести граф (в DOT-нотации).
[3] - Вывести минимальное оставное дерево (в DOT-нотации).
[4] - Завершить программу.
---->1
Введите номер 1-й вершины, вес ребра, номер 2-й вершины (через пробелы)
3 9 7
Выберите команду:
[1] - Добавить связь.
[2] - Вывести граф (в DOT-нотации).
[3] - Вывести минимальное оставное дерево (в DOT-нотации).
[4] - Завершить программу.
---->1
Введите номер 1-й вершины, вес ребра, номер 2-й вершины (через пробелы)
4 16 7
Выберите команду:
[1] - Добавить связь.
[2] - Вывести граф (в DOT-нотации).
[3] - Вывести минимальное оставное дерево (в DOT-нотации).
[4] - Завершить программу.
---->1
Введите номер 1-й вершины, вес ребра, номер 2-й вершины (через пробелы)
5 25 7
Выберите команду:
[1] - Добавить связь.
[2] - Вывести граф (в DOT-нотации).
[3] - Вывести минимальное оставное дерево (в DOT-нотации).
[4] - Завершить программу.
---->1
Введите номер 1-й вершины, вес ребра, номер 2-й вершины (через пробелы)
6 36 7
```

Рис.6 Скриншот добавления связей в граф

```
Выберите команду:
[1] - Добавить связь.
[2] - Вывести граф (в DOT-нотации).
[3] - Вывести минимальное оставное дерево (в DOT-нотации).
[4] - Завершить программу.
---->
2
digraph G {
1->2[label=20, dir=none]
2->3[label=15, dir=none]
3->4[label=3, dir=none]
4->5[label=17, dir=none]
5->6[label=28, dir=none]
6->1[label=23, dir=none]
1->7[label=1, dir=none]
2->7[label=4, dir=none]
3->7[label=9, dir=none]
4->7[label=16, dir=none]
5->7[label=25, dir=none]
6->7[label=36, dir=none]
```

Рис.7 Скриншот вывода исходного графа в DOT-нотации

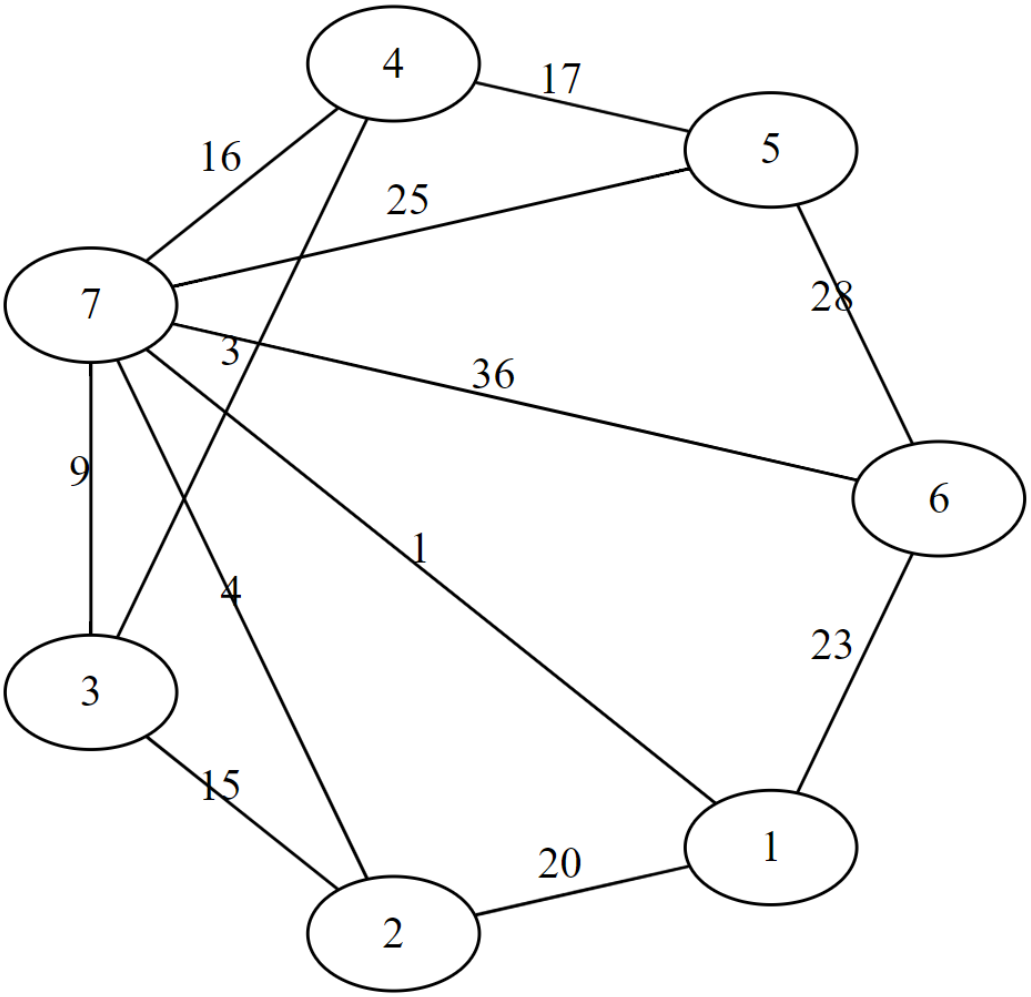


Рис.8 Скриншот построения графа по полученной DOT-нотации

```
}Выберите команду:
[1] - Добавить связь.
[2] - Вывести граф (в DOT-нотации).
[3] - Вывести минимальное оставное дерево (в DOT-нотации).
[4] - Завершить программу.
---->3
digraph G {
1->7[label=1, dir=none]
3->4[label=3, dir=none]
2->7[label=4, dir=none]
3->7[label=9, dir=none]
4->5[label=17, dir=none]
6->1[label=23, dir=none]
```

Рис.9 Скриншот вывода минимального остовного дерева в DOT-нотации

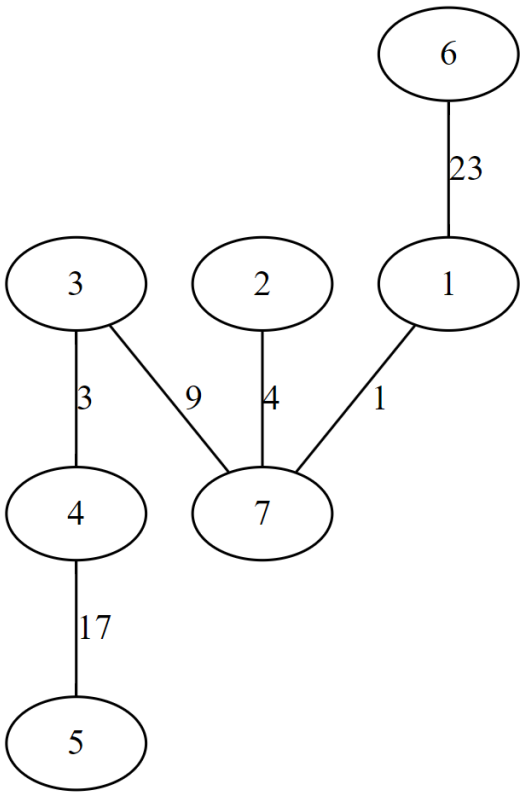


Рис.10 Скриншот построения минимального остовного дерева по полученной DOT-нотации

Выводы

1. В ходе работы была создана программа для работы с графами.
2. Также были реализованы функции добавления, вывода исходного графа и остовного дерева.
3. Были изучены особенности алгоритма Крускала:
4. Преимущества: Алгоритм Крускала позволяет эфективно строить минимальное остовное дерево благодаря своей линейной сложности. Для построения достаточно одного прохода по всем ребрам.
5. Недостатки: Алгоритм Крускала требует сортировки ребер графа по весу по убыванию, что в больших графах может некоторого времени.
6. Таким образом, была изучена работа Алгоритма Крускала и принцип представления графов в памяти компьютера .

Список используемых информационных источников

1. Сыромятников В.П. Структуры и алгоритмы обработки данных, лекции, РТУ МИРЭА, Москва, 2020/2021 уч./год.
2. Документация по языку программирования C++, интернет-ресурс: https://en.cppreference.com/w/ (Дата обращения – 30.11.2020)
3. Интегрированная среда разработки для языков программирования C и C++, разработанная компанией JetBrains - CLion / Copyright © 2000-2020 JetBrains s.r.o., интернет-ресурс: https://www.jetbrains.com/clion/learning-center/ (Дата обращения – 30.11.2020).
4. ГОСТ 19.701-90 ЕСПД. Схемы алгоритмов, программ, данных и систем. Обозначения условные и правила выполнения. Интернет-ресурс: http://docs.cntd.ru/document/gost-19-701-90-espd (Дата обращения – 030.11.2020).
5. Описание алгоритма Крускала. интернет-ресурс: https://ru.wikipedia.org/wiki/Алгоритм_Крускала (Дата обращения – 30.11.2020).
6. Построение графов по DOT-нотации. интернет-ресурс: https://dreampuf.github.io/GraphvizOnline (Дата обращения – 30.11.2020).