**МИНОБРНАУКИ РОССИИ**

**Санкт-Петербургский государственный**

**электротехнический университет**

**«ЛЭТИ» им. В.И. Ульянова (Ленина)**

**Кафедра ИС**

Курсовая РАБОТА

**по дисциплине «Алгоритмы и структуры данных»**

Тема: **Реализация алгоритма поиска минимального остова на основе алгоритма Краскала**

|  |  |  |
| --- | --- | --- |
| Студент гр. 3373 |  | Чистяков С.М. |
| Преподаватель |  | Молдовян Д.Н. |

Санкт-Петербург

2024

**ЗАДАНИЕ**

**на курсовую работу**

|  |  |  |
| --- | --- | --- |
| Студент Чистяков С.М. | | |
| Группа 3373 | | |
| Тема работы: Реализация алгоритма поиска минимального остова на основе алгоритма Краскала | | |
| Исходные данные:  Продемонстрировать знания следующих вопросов:   * сортировка * обход графов (в глубину и в ширину) * хранение графов (списки смежности, матрицы смежности, инцидентности) * построение системы непересекающихся множеств | | |
| Содержание пояснительной записки:  Введение, реализация структур данных, реализация алгоритма Краскала, чтение матрицы смежности из файла, результаты выполнения программы, заключение. | | |
|  | | |
| Дата выдачи задания: 01.09.2024 | | |
| Дата сдачи реферата: 13.12.2024 | | |
| Дата защиты реферата: 13.12.2024 | | |
| Студент |  | Чистяков С.М. |
| Преподаватель |  | Молдовян Д.Н. |

**Аннотация**

В курсовой работе по теме "Реализация алгоритма поиска минимального остова на основе алгоритма Краскала" мной были разработаны собственные структуры данных, включая динамический массив, ребро графа и методы для работы с ними. Также была реализована сортировка MergeSort и функция чтения матрицы смежности графа из текстового файла. Результатом выполнения программы является список ребер минимального остовного дерева и их суммарный вес.

Работоспособность программы была проверена на контрольном примере. Вся программа была создана с использованием среды разработки Visual Studio Community.

**содержание**

[**ВВЕДЕНИЕ** 6](#_Toc182932195)

[**1. РЕАЛИЗАЦИЯ СТРУКТУР ДАННЫХ** 7](#_Toc182932196)

[1.1. Ребро графа 9](#_Toc182932200)

[1.2. Методы ребра графа 10](#_Toc182932201)

[1.3. Программная реализация ребра графа 10](#_Toc182932202)

[**2. РЕАЛИЗАЦИЯ АЛГОРИТМА КРАСКАЛА** 11](#_Toc182932203)

[2.1. Алгоритм Краскала 11](#_Toc182932204)

[2.2. Программная реализация алгоритма Краскала 11](#_Toc182932205)

[**3. ЧТЕНИЕ МАТРИЦЫ СМЕЖНОСТИ ИЗ ФАЙЛА** 13](#_Toc182932206)

[3.1. Программная реализация чтения матрицы смежности из файла 13](#_Toc182932207)

[3.2. Вспомогательные функции для чтения матрицы смежности из файла 13](#_Toc182932208)

[**4. РЕЗУЛЬТАТЫ ВЫПОЛНЕНИЯ ПРОГРАММЫ** 14](#_Toc182932209)

[**ЗАКЛЮЧЕНИЕ** 16](#_Toc182932210)

# **ВВЕДЕНИЕ**

Задача:

Реализовать алгоритм поиска минимального остова на основе алгоритма Краскала (Крускала).

Цель:

Продемонстрировать знания следующих вопросов:

* сортировка
* обход графов (в глубину и в ширину)
* хранение графов (списки смежности, матрицы смежности, инцидентности)
* построение системы непересекающихся множеств

Входные данные:

Любой текстовый файл, содержащий матрицу смежности графа в виде:

A B C

0 3 1

3 0 2

1 2 0

где первая строка содержит через пробел список всех рёбер, за которым следует матрица смежности. В матрице значение 0 стоит, если ребра между вершинами нет, и положительное число, которое соответствует весу, когда ребро между вершинами существует.

Результат в виде отсортированных по имени пар и суммарный вес:

A C

B C

3

Максимальный размер входных данных: 50 вершин. Вершины могут быть заданы любой текстовой последовательностью без пробелов. Вес ребра ограничен интервалом от 1 до 1023 включительно.

# **1. РЕАЛИЗАЦИЯ СТРУКТУР ДАННЫХ**

## 

## 1.1. Ребро графа

Графы — это абстрактный способ представления типов отношений, например дорог, соединяющих города, и других видов сетей. Графы состоят из рёбер и вершин.

Ребро графа — это линия, соединяющая пару смежных вершин графа.

Вершина графа — это точка на графе.

## 1.2. Методы ребра графа

ostream& operator <<(ostream& os, Edge e) – переопределение оператора << для структуры Edge.

bool operator >(Edge edge1, Edge edge2) – переопределение оператора сравнения для структуры Edge.

bool operator <(Edge edge1, Edge edge2) – переопределение оператора сравнения для структуры Edge.

bool operator ==(Edge edge1, Edge edge2) – переопределение оператора сравнения для структуры Edge.

## 1.3. Программная реализация ребра графа

struct Edge

{

int src, dest, weight;

string srcStr, destStr;

};

ostream& operator <<(ostream& os, Edge e)

{

os << e.srcStr << " " << e.destStr << "\n";

return os;

}

bool operator >(Edge edge1, Edge edge2)

{

return edge1.weight > edge2.weight;

}

bool operator <(Edge edge1, Edge edge2)

{

return edge1.weight < edge2.weight;

}

bool operator ==(Edge edge1, Edge edge2)

{

return edge1.weight == edge2.weight;

}

# **2. РЕАЛИЗАЦИЯ АЛГОРИТМА КРАСКАЛА**

## 2.1. Алгоритм Краскала

**Алгоритм Краскала** — эффективный алгоритм построения минимального остовного дерева взвешенного связного неориентированного графа.

Процесс алгоритма Краскала можно описать следующим образом:

1. Сортировка всех ребер графа в порядке возрастания их весов.

2. Создание пустого остовного дерева.

3. Последовательный выбор ребер с наименьшими весами.

4. Добавление выбранного ребра к остовному дереву, при условии, что оно не создаст цикл.

5. Повторение шага 4 до тех пор, пока остовное дерево не будет содержать все вершины графа.

6. Вывод результатов на экран.

## 2.2. Программная реализация алгоритма Краскала

void findMinSpanningTree(myVector<Edge> edges, int numVertices)

{

edges.quickSort(0, edges.size() - 1);

myVector<int> parents(numVertices, -1);

myVector<Edge> mst;

int numEdgesAdded = 0;

for (int i = 0; i < edges.size(); i++)

{

int srcParent = -1;

int destParent = -1;

int src = edges.get(i).src;

int dest = edges.get(i).dest;

while (parents.get(src) != -1)

{

src = parents.get(src);

}

while (parents.get(dest) != -1)

{

dest = parents.get(dest);

}

srcParent = src;

destParent = dest;

if (srcParent != destParent)

{

mst.push\_back(edges.get(i));

numEdgesAdded++;

parents.changeElem(destParent, srcParent);

}

if (numEdgesAdded == numVertices - 1)

{

break;

}

}

int sumWeight = 0;

for (int i = 0; i < mst.size(); i++)

{

sumWeight += mst.get(i).weight;

}

mst.print();

cout << "Weight: " << sumWeight;

}

# **3. ЧТЕНИЕ МАТРИЦЫ СМЕЖНОСТИ ИЗ ФАЙЛА**

## 3.1. Программная реализация чтения матрицы смежности из файла

int main()

{

setlocale(0, "");

myVector<Edge> edges;

string s;

ifstream database("input.txt");

if (!database.is\_open())

{

cout << "Ошибка открытия файла!\n";

return 0;

}

if (database.eof())

{

cout << "Файл пуст!\n";

return 0;

}

getline(database, s);

myVector<string> verticesNames = splitToMyVectorStr(s);

int numVertices = verticesNames.size();

int counterStr = 0;

while (!database.eof())

{

string s1;

getline(database, s1);

myVector<int> weights = splitToMyVectorInt(s1);

for (int i = counterStr + 1; i < weights.size(); i++)

{

if (weights.get(i) > 0)

{

edges.push\_back({ counterStr, i, weights.get(i), verticesNames.get(counterStr), verticesNames.get(i) });

}

}

counterStr++;

}

database.close();

findMinSpanningTree(edges, numVertices);

}

## 3.2. Вспомогательные функции для чтения матрицы смежности из файла

myVector<string> splitToMyVectorStr(string s)

{

myVector<string> res;

for (int i = 0; i < s.size(); i++)

{

string str = "";

while (s[i] != ' ' && i < s.size())

{

str += s[i];

i++;

}

res.push\_back(str);

}

return res;

}

myVector<int> splitToMyVectorInt(string s)

{

myVector<int> res;

for (int i = 0; i < s.size(); i++)

{

int num = 0;

while (s[i] != ' ' && i < s.size())

{

num = num \* 10 + (s[i] - '0');

i++;

}

res.push\_back(num);

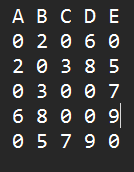
}

return res;

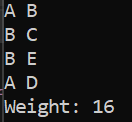
}

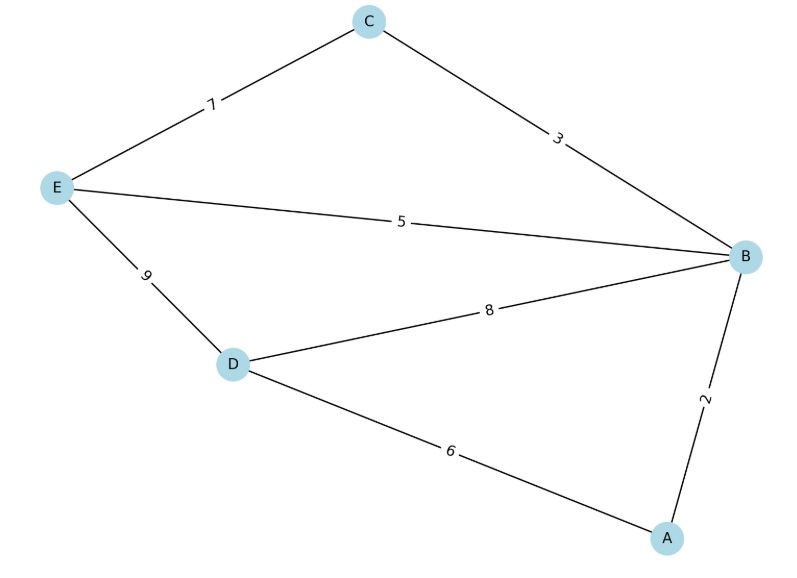
# **4. РЕЗУЛЬТАТЫ ВЫПОЛНЕНИЯ ПРОГРАММЫ**

Входные данные:



На выходе:





# **ЗАКЛЮЧЕНИЕ**

В ходе выполнения данной курсовой работы были разработаны несколько структур данных, включая динамический массив и ребро графа, реализованы методы для работы с ними. Мы изучили различные способы представления и хранения графов, такие как матрица смежности. Кроме того, был реализован алгоритм быстрой сортировки quickSort. Все эти компоненты в совокупности были использованы для разработки алгоритма Краскала, позволяющего найти минимальное остовное дерево.