



МИНОБРНАУКИ РОССИИ  
*Федеральное государственное бюджетное образовательное учреждение высшего  
образования*  
*«МИРЭА – Российский технологический университет»*

**РТУ МИРЭА**

---

Отчет по выполнению практического задания №7.2

**Тема:**

Графы: создание, алгоритмы обхода, важные задачи теории  
графов

Дисциплина: «Структуры и алгоритмы обработки данных»

Выполнил студент: Васильев Б.А.

Группа: ИКБО-20-23

Москва 2024

## СОДЕРЖАНИЕ

ЦЕЛЬ РАБОТЫ .....	3
ХОД РАБОТЫ .....	3
Формулировка задачи .....	3
Описание подхода к решению .....	4
Коды программ.....	5
Результаты тестирования .....	9
ВЫВОД.....	11
СПИСОК ИНФОРМАЦИОННЫХ ИСТОЧНИКОВ .....	11

## ЦЕЛЬ РАБОТЫ

Получить опыт работы с графами. Изучить и реализовать алгоритм Крускала по построению минимального остовного дерева.

## ХОД РАБОТЫ

### Формулировка задачи

Составить программу создания графа и реализовать процедуру для работы с графом, определенную индивидуальным вариантом задания.

Самостоятельно выбрать и реализовать способ представления графа в памяти.

В программе предусмотреть ввод с клавиатуры произвольного графа. В вариантах построения остовного дерева также разработать доступный способ (форму) вывода результирующего дерева на экран монитора.

Провести тестовый прогон программы на предложенном в индивидуальном варианте задания графе. Результаты тестирования в виде скриншотов экранов включить в отчет по выполненной работе.

Сделать выводы о проделанной работе, основанные на полученных результатах.

Оформить отчет с подробным описанием рассматриваемого графа, принципов программной реализации алгоритмов работы с графом, описанием текста исходного кода и проведенного тестирования программы.

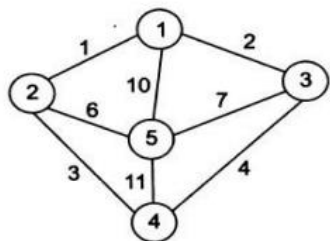
Вариант	Алгоритм	Предложенный граф
1	Построение остовного дерева алгоритмом Крускала	

Рисунок 1 – Индивидуальный вариант задачи

## Описание подхода к решению

Алгоритм Крускала находит минимальное остовное дерево для связного взвешенного графа. Его принцип работы заключается в следующем: сначала сортируются все рёбра графа по возрастанию веса, затем рёбра по одному добавляются к остовному дереву, начиная с самых лёгких. Чтобы избежать циклов, добавляется только то ребро, которое соединяет разные компоненты связности. Алгоритм завершается, когда количество ребёр в остовном графе будет равно  $V - 1$ , где  $V$  — количество вершин в исходном графе.

В алгоритме Крускала для поиска минимального остовного дерева используется Система непересекающихся множеств (disjoint-set или union-find). Эта структура данных помогает эффективно управлять компонентами связности графа. Каждый узел начинается как отдельное множество, и при добавлении ребра соединяются только множества, содержащие его вершины, если они ещё не соединены. Благодаря этому можно избежать циклов в остовном дереве, что критично для корректной работы алгоритма. В данной работе можно не проводить сжатие пути во время операций над множествами, поскольку деревья, которые будут рассматриваться состоят из небольшого количества вершин и ребёр.

## Код программы

Реализуем программу на языке C++, код представлен ниже (рис. 2-5).

```
1  #include <iostream>
2  #include <vector>
3  #include <algorithm>
4  #include <fstream>
5  #include <cstdlib>
6  #include <limits>
7  using namespace std;
8
9  // Data structure to represent an edge in the graph
10 struct Edge {
11     int src, dest, weight;
12
13     // Constructor to adjust user input (1-based index) to internal representation (0-based index)
14     Edge(int s, int d, int w) : src(s - 1), dest(d - 1), weight(w) {}
15 };
16
17 // Comparator function to sort edges based on their weights
18 bool compareEdges(const Edge& e1, const Edge& e2) {
19     return e1.weight < e2.weight;
20 }
21
22 // Utility function for finding id of DSU (Disjoint Set Union) member
23 int getRoot(int i, int id[]) {
24     while (i != id[i]) {
25         i = id[i];
26     }
27     return i;
28 }
29
30 // Connected operation of DSU (Disjoint Set Union) data structure
31 bool connected(int p, int q, int id[]) {
32     return (getRoot(p, id) == getRoot(q, id));
33 }
34
35 // Union operation of DSU
36 void unionSet(int p, int q, int *id) {
37     int i = getRoot(p, id);
38     int j = getRoot(q, id);
39     id[i] = j;
40 }
41
42 // Kruskal's algorithm for finding MST
43 vector<Edge> kruskalMST(vector<Edge>& edges, int V) {
44     vector<Edge> MST; // to store the MST
45     int *id = new int[V];
46     for (int i = 0; i < V; i++) {
47         id[i] = i;
48     }
```

Рисунок 2 – код программы (часть 1)

```

50 // Sort the edges based on their weights
51 sort(edges.begin(), edges.end(), compareEdges);
52
53 int count = 0;
54 int i = 0;
55 while (count < V - 1 && i < edges.size()) {
56     int src = edges[i].src;
57     int dest = edges[i].dest;
58
59     // Check if adding the current edge forms a cycle
60     if (!connected(src, dest, id)) {
61         MST.push_back(edges[i]);
62         unionSet(src, dest, id);
63         count++;
64     }
65     i++;
66 }
67
68 delete[] id;
69 return MST;
70 }
71
72
73 // Function to generate a DOT file for the graph and create an image
74 void generateGraphPNG(const vector<Edge>& edges, const string& filename) {
75     string dotFileName = filename + ".dot";
76     string outputImageFileName = filename + ".png";
77     ofstream file(dotFileName);
78
79     // Write the header for the DOT file
80     file << "graph G {" << endl;
81
82     // Set default node attributes (e.g., make all nodes circles)
83     file << "    node [shape=circle];" << endl;
84
85     for (const auto& edge : edges) {
86         // Adding 1 back to src and dest for DOT output
87         file << "    " << edge.src + 1 << " -- " << edge.dest + 1
88             << " [label=" << edge.weight << "];" << endl;
89     }
90
91     file << "}" << endl;
92     file.close();
93
94     // Automatically generate an image from the DOT file
95     string command = "dot -Tpng " + dotFileName + " -o " + outputImageFileName;
96     system(command.c_str());
97     cout << "Graph image generated: " << outputImageFileName << endl;
98
99     string removeCommand = "del " + dotFileName;
100     system(removeCommand.c_str());
101 }
102
103
104 // Utility function for correctly clearing input stream
105 void clearCin()
106 {
107     cin.clear();
108     cin.ignore(numeric_limits<streamsize>::max(), '\n');
109 }

```

Рисунок 3 – код программы (часть 2)

```

111 int getNumberOfVertices()
112 {
113     int numV;
114     while(true)
115     {
116         cout << "Enter the number of vertices: ";
117
118         if (cin >> numV && numV > 0) {
119             clearCin();
120             return numV;
121         }
122         clearCin();
123         cout << "Invalid Input. Number of vertices should be a positive integer\n";
124     }
125 }
126
127 int getNumberOfEdges(int numV)
128 {
129     int numE;
130     while(true)
131     {
132         cout << "Enter the number of edges: ";
133
134         if (cin >> numE && numE >= numV - 1)
135         {
136             clearCin();
137             return numE;
138         }
139         clearCin();
140         cout << "Invalid Input. Number of edges should be greater or equal to the number of vertices - 1\n";
141     }
142 }
143
144 bool askIfUseExampleGraph()
145 {
146     char choice;
147     while (true)
148     {
149         cout << "Do you want to perform operations on an example graph [Y] or create your own [N]? ";
150         cin >> choice;
151
152         // Convert choice to lower case for easier comparison
153         choice = tolower(choice);
154
155         if (choice == 'y')
156         {
157             return true;
158         }
159         else if (choice == 'n')
160         {
161             return false;
162         }
163         else
164         {
165             clearCin(); // Clear the input buffer
166             cout << "Invalid input. Please enter 'Y' or 'N'. \n";
167         }
168     }
169 }
170 }

```

Рисунок 4 – код программы (часть 3)

```

172 int main() {
173     bool isOnExampleGraph = askIfUseExampleGraph();
174     int numV;
175     int numE;
176     vector<Edge> edges;
177     if (isOnExampleGraph)
178     {
179         // Example Graph
180         numV = 5;
181         numE = 8;
182         edges.emplace_back(1, 2, 1);
183         edges.emplace_back(1, 5, 10);
184         edges.emplace_back(1, 3, 2);
185         edges.emplace_back(2, 5, 6);
186         edges.emplace_back(2, 4, 3);
187         edges.emplace_back(5, 4, 11);
188         edges.emplace_back(5, 3, 7);
189         edges.emplace_back(4, 3, 4);
190     }
191     else
192     {
193         numV = getNumberOfVertices();
194         numE = getNumberOfEdges(numV);
195
196         for (int i = 0; i < numE; i++)
197         {
198             int src, dest, weight;
199             cout << "Enter source, destination, and weight for edge " << i + 1 << ": ";
200             cin >> src >> dest >> weight;
201             edges.push_back({src, dest, weight});
202         }
203     }
204
205     // Generate DOT file and image
206     generateGraphPNG(edges, "graph");
207
208     // Find MST using Kruskal's algorithm
209     vector<Edge> MST = kruskaIMST(edges, numV);
210
211     generateGraphPNG(MST, "MST");
212
213     return 0;
214 }

```

Рисунок 5 – код программы (часть 4)



## Результаты тестирования

Выполним тестирование программы (рис. 6-9).

```
Do you want to perform operations on an example graph [Y] or create your own [N]? Y
Graph image generated: graph.png
Graph image generated: MST.png
```

Рисунок 6 – тестирование программы на графе из индивидуального варианта

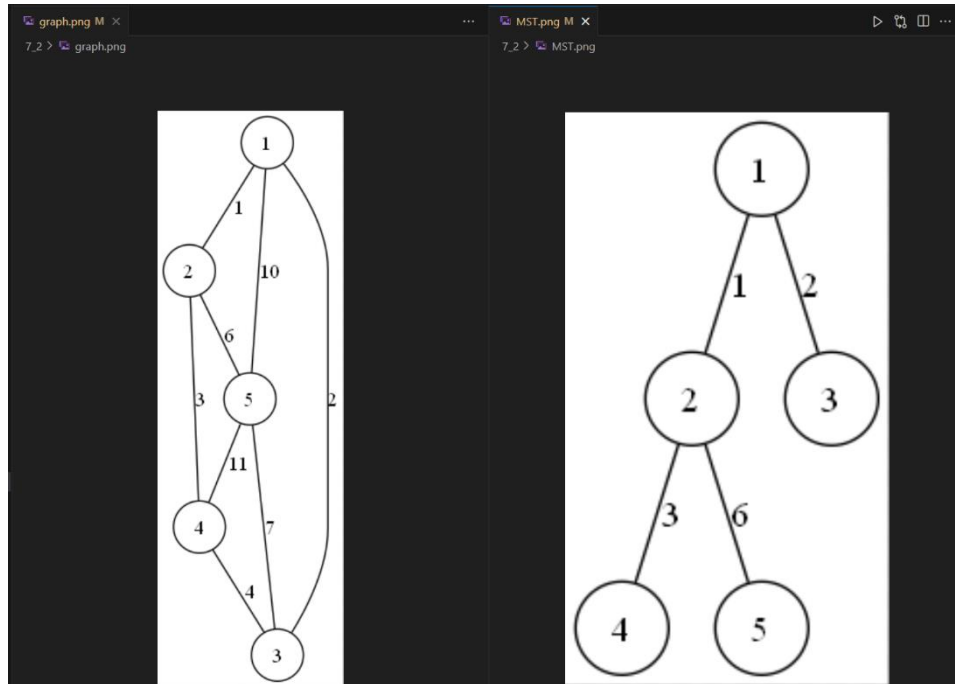


Рисунок 7 – файлы с визуализацией изначального и минимального остовного дерева (на примере дерева из индивидуального варианта).

```
Do you want to perform operations on an example graph [Y] or create your own [N]? n
Enter the number of vertices: 4
Enter the number of edges: 6
Enter source, destination, and weight for edge 1: 1 2 1
Enter source, destination, and weight for edge 2: 1 3 5
Enter source, destination, and weight for edge 3: 1 4 5
Enter source, destination, and weight for edge 4: 2 3 1
Enter source, destination, and weight for edge 5: 2 4 6
Enter source, destination, and weight for edge 6: 3 4 2
Graph image generated: graph.png
Graph image generated: MST.png
```

Рисунок 8 – тестирование программы на графе, введёном вручную

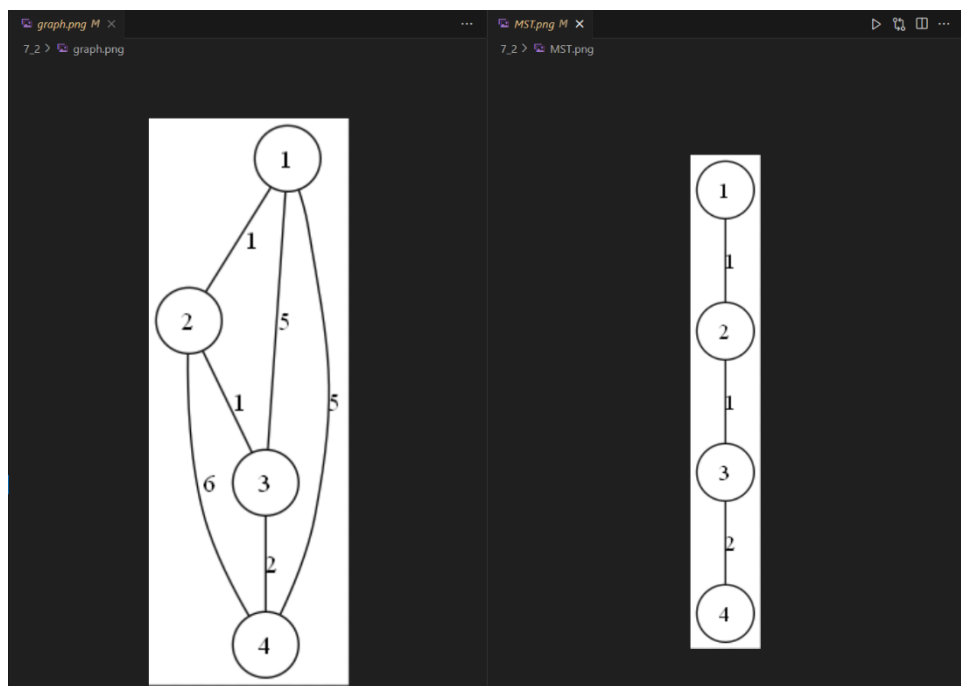


Рисунок 9 – файлы с визуализацией изначального и минимального остовного дерева (на примере дерева введённого вручную).

Тестирование показало, что программа работает корректно.

## **ВЫВОД**

В результате выполнения работы был получен опыт работы с графами, изучен и реализован алгоритм Крускала по построению минимального остовного дерева.

## **СПИСОК ИНФОРМАЦИОННЫХ ИСТОЧНИКОВ**

1. Рысин, М. Л. Введение в структуры и алгоритмы обработки данных : учебное пособие / М. Л. Рысин, М. В. Сартаков, М. Б. Туманова. — Москва : РТУ МИРЭА, 2022 — Часть 2 : Поиск в тексте. Нелинейные структуры данных. Кодирование информации. Алгоритмические стратегии — 2022. — 111 с. — Текст : электронный // Лань : электронно-библиотечная система. — URL: <https://e.lanbook.com/book/310826> (дата обращения: 30.10.2024).
2. Документация по языку C++ [Электронный ресурс]. URL: <https://docs.microsoft.com/ruru/cpp/cpp/> (дата обращения 30.10.2024).