



МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
Федеральное государственное бюджетное образовательное
учреждение высшего образования
"МИРЭА - Российский технологический университет"

РТУ МИРЭА

Институт информационных технологий (ИТ)
Кафедра математического обеспечения и стандартизации информационных
технологий (МОСИТ)

ОТЧЕТ ПО ПРАКТИЧЕСКОЙ РАБОТЕ № 7.2

по дисциплине

«Структуры и алгоритмы обработки данных»

Тема. Графы: создание, алгоритмы обхода, важные задачи теории графов.

Выполнил студент группы ИКБО-41-23

Гольд Д.В.

Принял старший преподаватель

Рысин М.Л.

Москва 2024

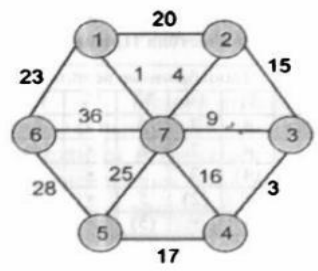
СОДЕРЖАНИЕ

Цель	3
Вариант.....	3
Код программы	3
Тестирование	6
Вывод.....	7

Цель

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

Вариант

Вариант	Алгоритм	Предложенный граф
9	Построение остоного дерева алгоритмом Крускала	

Код программы

```
1  #include <iostream>
2  #include <vector>
3  #include <algorithm>
4  #include <iomanip>
5
6  using namespace std;
7
8  struct Edge { //структура для хранения ребра
9      int u, v, weight;
10     Edge(int u, int v, int weight) : u(u), v(v), weight(weight) {}
11 };
12
13 bool compare_edges(const Edge& a, const Edge& b) { //функция для сравнения ребер для сортировки по весу
14     return a.weight < b.weight;
15 }
16
17 class UnionFind { //система непересек множеств
18 private:
19     vector<int> parent, rank;
20
21 public:
22     UnionFind(int n) {
23         parent.resize(n); //массив для род узлов
24         rank.resize(n, 0); //массив рангов с нулями
25         for (int i = 0; i < n; ++i) {
26             parent[i] = i; //каждый узел - себе сам родитель
27         }
28     }
29
30     int find(int x) {
31         if (parent[x] != x) { //рекурсия
32             parent[x] = find(parent[x]);
33         }
34         return parent[x]; //корень множества
35     }
36
37     void Union(int x, int y) {
38         int root_x = find(x);
```

```

37 void Union(int x, int y) {
38     int root_x = find(x);
39     int root_y = find(y);
40
41     if (root_x != root_y) {//объединяем множества
42         if (rank[root_x] > rank[root_y]) {
43             parent[root_y] = root_x;
44         } else if (rank[root_x] < rank[root_y]) {
45             parent[root_x] = root_y;
46         } else {
47             parent[root_y] = root_x;//увелич его ранг
48             rank[root_x]++;
49         }
50     }
51 }
52 };
53
54 void print_matrix(const vector<vector<int>>& matrix) {
55     int n = matrix.size();
56     cout << "original graph:\n";
57     for (int i = 0; i < n; ++i) {
58         for (int j = 0; j < n; ++j) {
59             cout << setw(3) << matrix[i][j] << " ";
60         }
61         cout << endl;
62     }
63 }
64
65 void kruskal(const vector<vector<int>>& matrix) {
66     int n = matrix.size();//колво узлов в графе
67     vector<Edge> edges;//вектор для хранения ребер
68
69     for (int i = 0; i < n; ++i) {// извлечение ребер из матрицы смежности
70         for (int j = i + 1; j < n; ++j) {//ток верхний треугол матрицы
71             if (matrix[i][j] != 0) {
72                 edges.emplace_back(i, j, matrix[i][j]);
73             }
74         }
75     }

```

```

77     sort(edges.begin(), edges.end(), compare_edges); //сортировка по веса
78
79     UnionFind uf(n);
80     vector<Edge> minimum_ostovnoe_tree;
81     int minimum_ostovnoe_tree_weight = 0;
82     vector<Edge> edges_not_included;
83
84     for (const auto& edge : edges) { //по сорту ребрам
85         if (uf.find(edge.u) != uf.find(edge.v)) { //если не цикл
86             minimum_ostovnoe_tree.push_back(edge); //добав ребро в мод
87             minimum_ostovnoe_tree_weight += edge.weight;
88             uf.Union(edge.u, edge.v); //объединяем два множества
89
90             if (minimum_ostovnoe_tree.size() == n - 1) {
91                 break;
92             }
93         } else {
94             edges_not_included.push_back(edge); //не вход в мод
95         }
96     }
97
98     print_matrix(matrix);
99
100    cout << "\n\ndedges minimum ostovnoe tree:\n";
101    for (const auto& edge : minimum_ostovnoe_tree) {
102        cout << edge.u + 1 << " - " << edge.v + 1 << " : " << edge.weight << endl;
103    }
104    cout << "\ntotal all weight: " << minimum_ostovnoe_tree_weight << endl;
105
106    cout << "\nnumber of edges in the minimum ostovnoe tree: " << minimum_ostovnoe_tree.size() << endl;
107
108    cout << "\ndedges are not included in minimum ostovnoe tree:\n";
109    for (const auto& edge : edges_not_included) {
110        cout << edge.u + 1 << " - " << edge.v + 1 << " : " << edge.weight << endl;
111    }

```

```

108     cout << "\nedges are not included in minimum ostovnoe tree:\n";
109     for (const auto& edge : edges_not_included) {
110         cout << edge.u + 1 << " - " << edge.v + 1 << " : " << edge.weight << endl;
111     }
112 }
113
114 int main() {
115     vector<vector<int>>> matrix = {
116         {0, 20, 0, 0, 0, 23, 1},
117         {20, 0, 15, 0, 0, 0, 4},
118         {0, 15, 0, 3, 0, 0, 9},
119         {0, 4, 3, 0, 17, 0, 16},
120         {0, 0, 0, 17, 0, 28, 25},
121         {23, 0, 0, 0, 28, 0, 36},
122         {1, 4, 9, 16, 25, 36, 0}
123     };
124
125     kruskal(matrix);
126
127     return 0;
128 }
129

```

Тестирование

original graph:

0	20	0	0	0	23	1
20	0	15	0	0	0	4
0	15	0	3	0	0	9
0	4	3	0	17	0	16
0	0	0	17	0	28	25
23	0	0	0	28	0	36
1	4	9	16	25	36	0

edges minimum ostovnoe tree:

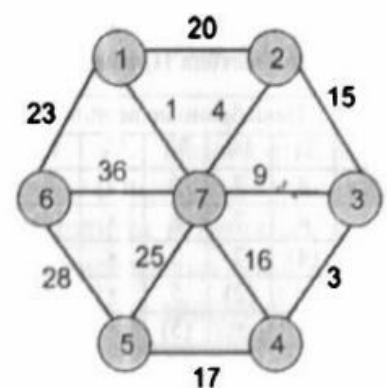
1 - 7 : 1
 3 - 4 : 3
 2 - 7 : 4
 3 - 7 : 9
 4 - 5 : 17
 1 - 6 : 23

total all weight: 57

number of edges in the minimum ostovnoe tree: 6

edges are not included in minimum ostovnoe tree:

2 - 3 : 15
 4 - 7 : 16
 1 - 2 : 20



Вывод

Составил программу создания графа и реализовал процедуру для работы с графом, определенную индивидуальным вариантом задания.