Московский Авиационный Институт  
(национальный исследовательский университет)

Кафедра 304

Отчет по лабораторной работе №4

«Поиск кратчайших путей в графе. Построение остовного дерева графа.»

по дисциплине «Структуры и алгоритмы обработки данных»

(вариант №3)

Выполнил:

Студент группы 3О-210Б

Сомов Д. Н.

Проверил:

Силаев А. В.

Москва, 2015

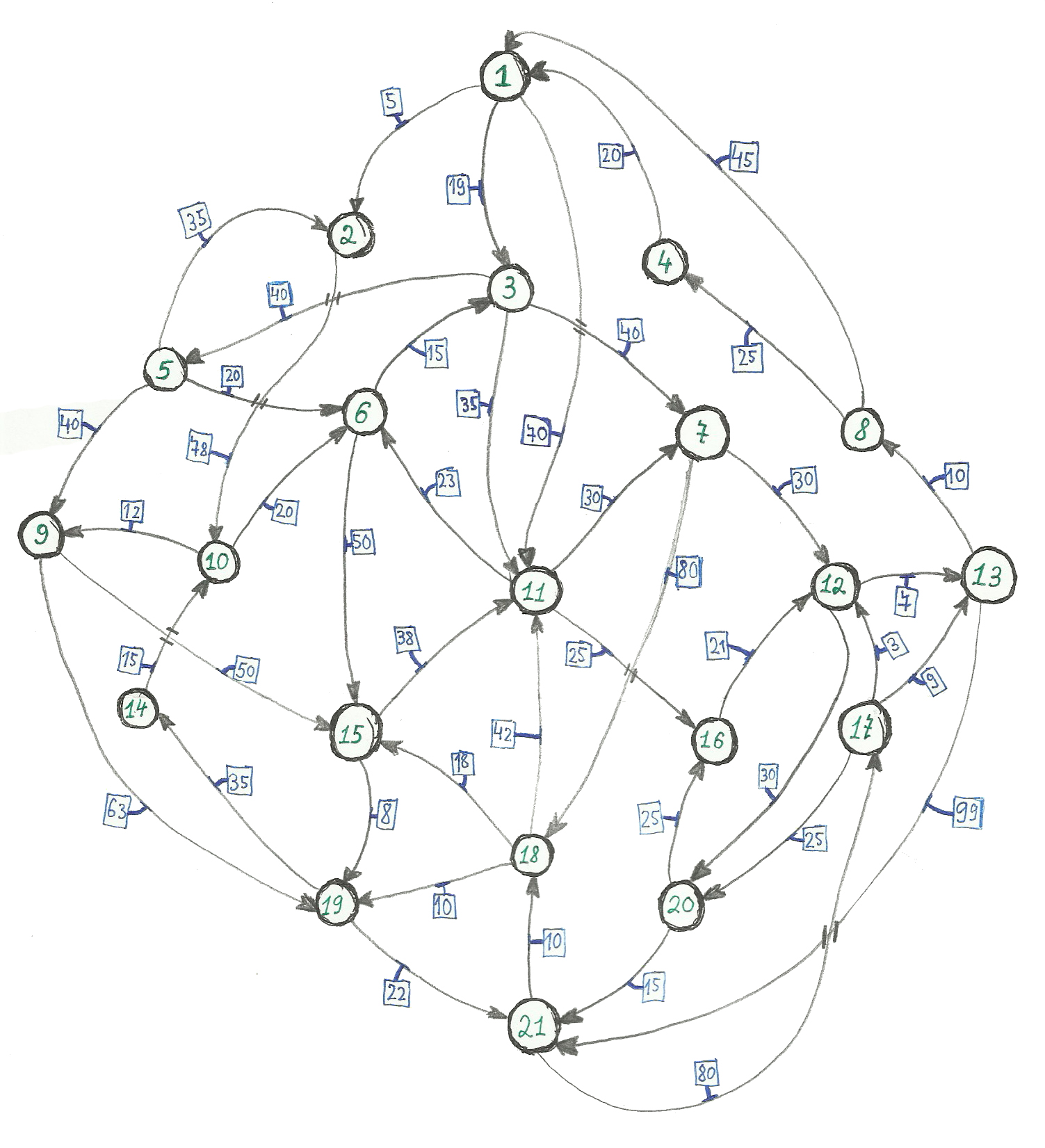
**Цель работы**: изучить алгоритмы поиска кратчайших путей в ориентированном графе; изучить алгоритмы построения остовных деревьев неориентированного графа минимального веса; научиться реализации этих алгоритмов в виде компьютерных программ.

**Задание (вариант №3):** для созданного Вами ориентированного взвешенного графа, состоящего как минимум из 20 вершин, реализовать:

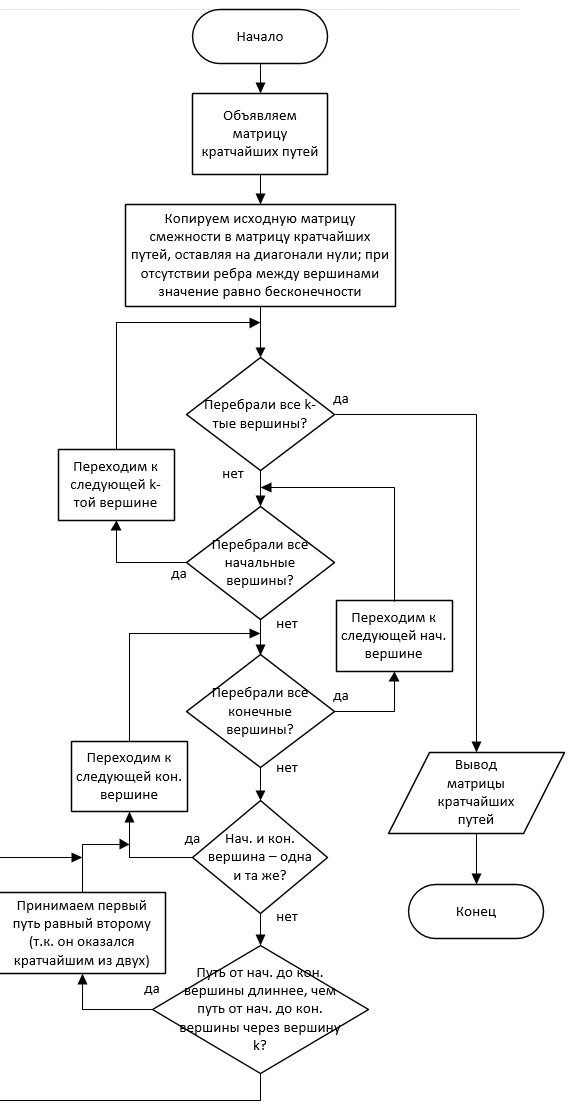
1. Алгоритм Флойда для поиска кратчайшего пути между всеми возможными парами вершин.
2. Сделав тот же самый граф неориентированным, построить его остовное дерево минимальной стоимости методом Прима.

В п.1 задания распечатать результаты на каждой итерации.

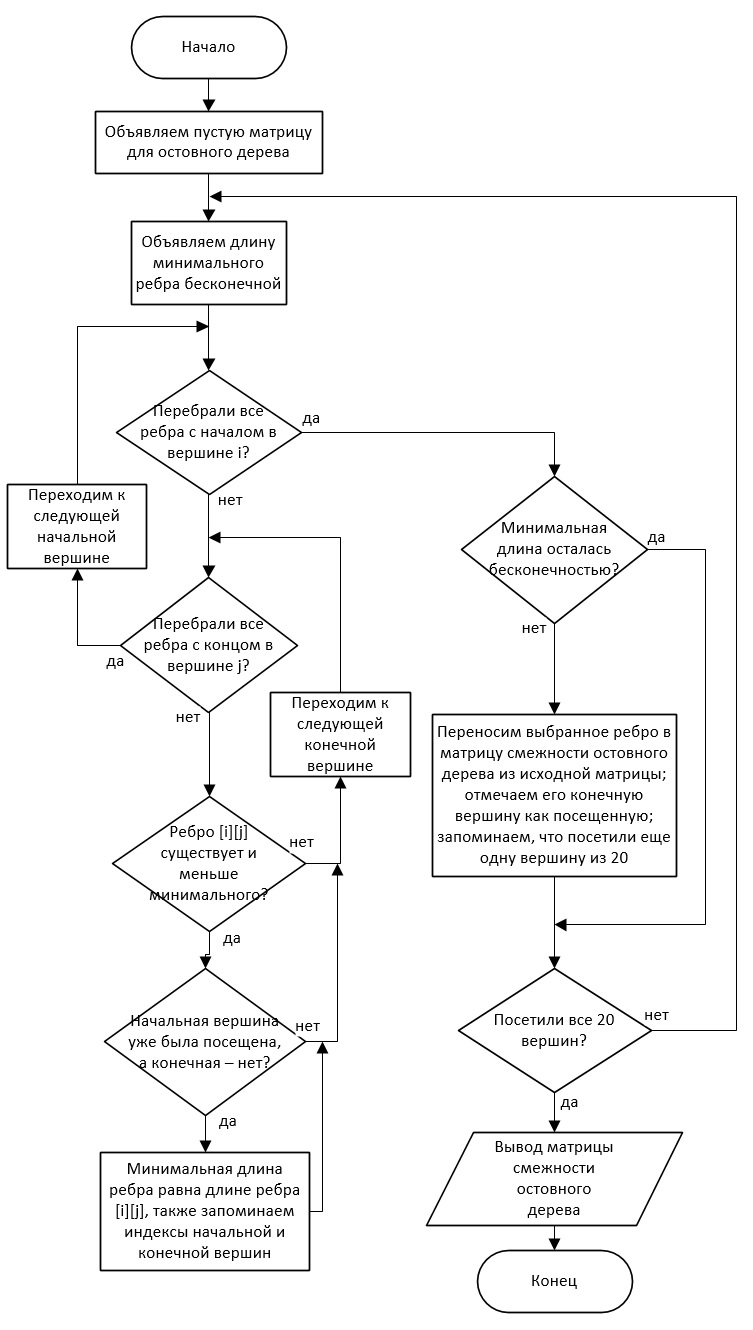
**Граф, составленный для выполнения заданий:**

****

**Структурная схема алгоритма Флойда:**

****

**Структурная схема алгоритма Прима:**

****

**Текст программы:**

*//Автор – Сомов Дмитрий, группа 3О-210Б*

*//Используется язык С++*

#include <iostream.h>

#include <fstream.h>

#include <stdlib.h>

int A[21][21] = { *//матрица смежности орграфа*

//1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21

0, 5, 19, 0, 0, 0, 0, 0, 0, 0, 70, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, //1

0, 0, 0, 0, 0, 0, 0, 0, 0, 78, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, //2

0, 0, 0, 0, 40, 0, 40, 0, 0, 0, 35, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, //3

20, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, //4

0, 35, 0, 0, 0, 20, 0, 0, 40, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, //5

0, 0, 15, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, //6

0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 30, 0, 0, 0, 0, 0, 80, 0, 0, 0, //7

45, 0, 0, 25, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, //8

0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 50, 0, 0, 0, 63, 0, 0, //9

0, 0, 0, 0, 0, 20, 0, 0, 12, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, //10

0, 0, 0, 0, 0, 23, 30, 0, 0, 0, 0, 0, 0, 0, 0, 25, 0, 0, 0, 0, 0, //11

0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 7, 0, 0, 0, 0, 0, 0, 30, 0, //12

0, 0, 0, 0, 0, 0, 0, 10, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 99, //13

0, 0, 0, 0, 0, 0, 0, 0, 0, 15, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, //14

0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 38, 0, 0, 0, 0, 0, 0, 0, 8, 0, 0, //15

0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 21, 0, 0, 0, 0, 0, 0, 0, 0, 0, //16

0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 3, 9, 0, 0, 0, 0, 0, 0, 25, 0, //17

0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 42, 0, 0, 0, 18, 0, 0, 0, 10, 0, 0, //18

0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 35, 0, 0, 0, 0, 0, 0, 22, //19

0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 25, 0, 0, 0, 0, 15, //20

0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 80, 10, 0, 0, 0 //21

};

int B[21][21] = {0}, inf = 100000;

int A1[21][21] = { *//матрица смежности неориентир. графа*

//1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21

0, 5, 19, 20, 0, 0, 0, 45, 0, 0, 70, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, //1

5, 0, 0, 0, 35, 0, 0, 0, 0, 78, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, //2

19, 0, 0, 0, 40, 15, 40, 0, 0, 0, 35, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, //3

20, 0, 0, 0, 0, 0, 0, 25, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, //4

0, 35, 40, 0, 0, 20, 0, 0, 40, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, //5

0, 0, 15, 0, 20, 0, 0, 0, 0, 20, 23, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, //6

0, 0, 40, 0, 0, 0, 0, 0, 0, 0, 30, 30, 0, 0, 0, 0, 0, 80, 0, 0, 0, //7

45, 0, 0, 25, 0, 0, 0, 0, 0, 0, 0, 0, 10, 0, 0, 0, 0, 0, 0, 0, 0, //8

0, 0, 0, 0, 40, 0, 0, 0, 0, 12, 0, 0, 0, 0, 50, 0, 0, 0, 63, 0, 0, //9

0, 78, 0, 0, 0, 20, 0, 0, 12, 0, 0, 0, 0, 15, 0, 0, 0, 0, 0, 0, 0, //10

70, 0, 35, 0, 0, 23, 30, 0, 0, 0, 0, 0, 0, 0, 38, 25, 0, 42, 0, 0, 0, //11

0, 0, 0, 0, 0, 0, 30, 0, 0, 0, 0, 0, 7, 0, 0, 21, 3, 0, 0, 30, 0, //12

0, 0, 0, 0, 0, 0, 0, 10, 0, 0, 0, 7, 0, 0, 0, 0, 9, 0, 0, 0, 99, //13

0, 0, 0, 0, 0, 0, 0, 0, 0, 15, 0, 0, 0, 0, 0, 0, 0, 0, 35, 0, 0, //14

0, 0, 0, 0, 0, 50, 0, 0, 50, 0, 38, 0, 0, 0, 0, 0, 0, 18, 8, 0, 0, //15

0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 25, 21, 0, 0, 0, 0, 0, 0, 0, 25, 0, //16

0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 3, 9, 0, 0, 0, 0, 0, 0, 25, 80, //17

0, 0, 0, 0, 0, 0, 80, 0, 0, 0, 42, 0, 0, 0, 18, 0, 0, 0, 10, 0, 10, //18

0, 0, 0, 0, 0, 0, 0, 0, 63, 0, 0, 0, 0, 35, 8, 0, 0, 0, 10, 0, 22, //19

0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 30, 0, 0, 0, 25, 25, 0, 0, 0, 15, //20

0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 99, 0, 0, 0, 80, 10, 22, 15, 0 //21

};

void Output(int iter) { *//вывод матрицы кратчайших расстояний*

ofstream f; *//после каждой итерации*

f.open("outFloyd.txt", ios::app);

f << "\n\n\n";

f << "Iteration #" << iter << endl;

for(int i = 0; i < 21; i++) {

for(int j = 0; j < 21; j++) {

f << B[i][j] << "\t";

}

f << endl;

}

f.close();

}

void Floyd() { *//реализация алгоритма Флойда*

for(int i = 0; i < 21; i++) { *//копируем м-цу смежности в м-цу путей* for(int j = 0; j < 21; j++) {

if(i == j) *//элементы диагонали равны нулю*

B[i][j] = 0;

else if(A[i][j] == 0) *//если пути между вершинами нет,*

B[i][j] = inf; *//то длина равна условно большой величине*

else *//в любом другом случае копировать длину ребра*

B[i][j] = A[i][j];

}

}

ofstream t;

t.open("logFloyd.txt", ios::out);

for(int k = 0; k < 21; k++) { *//итерация по промежуточным вершинам*

for(int i = 0; i < 21; i++) { *//итерация по начальным вершинам*

for(int j = 0; j < 21; j++) { *//итерация по конечным вершинам*

if(i != j) { *//если рассматриваем не петлю…*

t << "\nB[" << i+1 << "][" << j+1 << "] > B[" << i+1 << "][" << k+1 << "] + B[" << k+1 << "][" << j+1 << "]?";

t << "\n" << B[i][j] << " > " << B[i][k] << " + " << B[k][j] << "?";

if(B[i][j] > (B[i][k] + B[k][j])) { *//если прямой путь длиннее, чем путь в обход через промежуточную k-тую вершину…*

t << "\nYes, new number is ";

B[i][j] = B[i][k] + B[k][j]; *//записать путь в обход как кратчайший*

t << B[i][j];

}

}

}

}

Output(k+1); *//распечатка матрицы кратчайших путей*

}

}

void Prima() { *//реализация алгоритма Прима*

int ost[21][21] = {0}, count = 20; *//м-ца остовного дерева и кол-во вершин минус 1*

bool vis[21] = {false}; *//массив посещенных вершин*

vis[0] = true; *//начинаем с первой вершины*

do {

int min = INT\_MAX, im, jm; *//мин. ребро, его начало и конец*

for (int i = 0; i < 21; i++)

for (int j = 0; j < 21; j++) *//если ребро существует, меньше минимального, найденного ранее, и соединяет ранее посещенную и непосещенную вершины…*

if ((A1[i][j] != 0) && (A1[i][j] < min) && (vis[i]) && (!vis[j])) {

min = A1[i][j]; *//запоминаем вес этого ребра*

im = i; *//и его начало*

jm = j; *//и конец*

}

if (min != INT\_MAX) {

ost[im][jm] = A1[im][jm]; *//переносим найденное ребро в матрицу смежности*

A1[im][jm] = 0; *//остовного дерева*

vis[jm] = true; *//отмечаем, что посетили еще одну вершину*

count--; *//и уменьшаем их количество*

}

} while (count > 0); *//продолжаем, пока еще остаются непосещенные вершины*

ofstream f;

f.open("outPrima.txt", ios::out);

f << "Min tree matrix: \n";

for (int i = 0; i < 21; i++) {

f << "\n";

for (int j = 0; j < 21; j++)

f << ost[i][j] << "\t";

}

f << "\n\n";

}

void main() {

Floyd();

Prima();

system("pause");

}

**Результаты работы** можно найти на последней странице.

**Вывод**: в данной лабораторной работе были изучены алогритмы Флойда и Прима, сравнительно легкие в реализации в виде программного кода. Эти алгоритмы позволяют без проблем находить кратчайшие пути в графах, а также строить остовные деревья этих графов (в том числе минимальной стоимости), что бесспорно имеет большое практическое значение.

**Результаты работы:**

1. Матрица кратчайших расстояний после последней итерации:

Iteration #21

0 5 19 131 59 77 59 106 95 83 54 89 96 184 145 79 214 139 149 119 134

245 0 113 225 153 98 153 200 90 78 148 183 190 183 140 173 250 180 148 213 170

132 75 0 112 40 58 40 87 80 153 35 70 77 165 130 60 195 120 130 100 115

20 25 39 0 79 97 79 126 115 103 74 109 116 204 165 99 234 159 169 139 154

167 35 35 147 0 20 75 122 40 113 70 105 112 133 90 95 200 130 98 135 120

147 90 15 127 55 0 55 102 95 168 50 85 92 180 145 75 210 135 145 115 130

92 97 111 72 151 145 0 47 152 140 122 30 37 125 98 85 155 80 90 60 75

45 50 64 25 104 122 104 0 140 128 99 134 141 229 190 124 259 184 194 164 179

196 201 126 176 166 111 118 151 0 108 88 134 141 93 50 113 160 90 58 164 80

167 110 35 147 75 20 75 122 12 0 70 105 112 105 62 95 172 102 70 135 92

108 113 38 88 78 23 30 63 118 161 0 46 53 146 119 25 171 101 111 76 91

62 67 81 42 121 120 121 17 127 115 97 0 7 100 73 55 125 55 65 30 45

55 60 74 35 114 132 114 10 150 138 109 144 0 154 127 134 179 109 119 174 99

182 125 50 162 90 35 90 137 27 15 85 120 127 0 77 110 187 117 85 150 107

146 151 76 126 116 61 68 101 70 58 38 84 91 43 0 63 110 40 8 114 30

83 88 102 63 142 141 142 38 148 136 118 21 28 121 94 0 146 76 86 51 66

64 69 83 44 123 115 122 19 122 110 92 3 9 95 68 50 0 50 60 25 40

150 155 80 130 120 65 72 105 72 60 42 88 95 45 18 67 112 0 10 118 32

166 160 85 146 125 70 104 121 62 50 74 105 111 35 50 99 102 32 0 127 22

108 113 105 88 145 90 97 63 97 85 67 46 53 70 43 25 95 25 35 0 15

144 149 90 124 130 75 82 99 82 70 52 83 89 55 28 77 80 10 20 105 0

1. Граф, построенный согласно матрице остовного дерева, выведенной программой:

