Министерство образования и науки Российской Федерации

Федеральное государственное бюджетное образовательное

учреждение

высшего образования

«**Пермский национальный исследовательский**

**политехнический университет»**

Кафедра «Информационные технологии и автоматизированные

системы»

**ОТЧЁТ**

**по лабораторной работе №21**

Дисциплина: «Основы алгоритмизации и программирования»

Тема: “Графы.”

Вариант 17

Выполнил:

Студент группы ИВТ-20-2б

Сафронов Владислав Владиславович

Проверила:

Доцент кафедры ИТАС

Полякова О.А.

**Пермь, 2021**

**Постановка задачи**

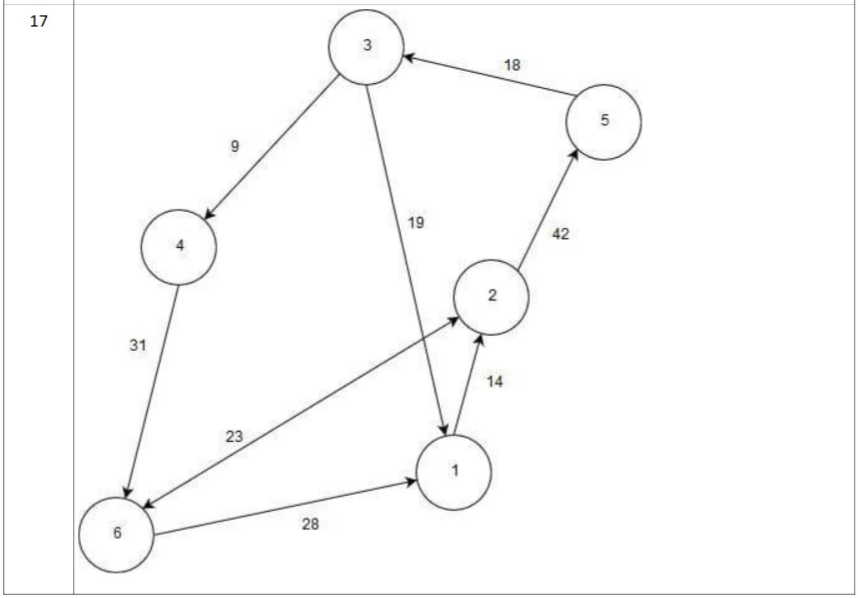
Реализовать алгоритм Дейкстры для графа, соответствующего выбранному варианту.

1. Без использования Qt.

2. Интерфейс на усмотрение разработчика.

3. Визуализация графа в OpenGL.

4. Реализуется только метод Дейкстры



**Анализ задачи**

1. Для решения задачи необходимо:
   1. Организовать класс Graph, который наследуется от класса QMainWindow, чтобы была возможность использовать интерфейс.
   2. Организовать функцию Dejkstra(), чтобы вывести кратчайшие пути до вершин в поле на окне программы.
2. В ходе работы были использованы следующие типы данных:
   1. Класс Graph с необходимыми методами для отрисовки линий, узлов и самого графа в форму. В качестве поля используется двумерный вектор matrix и указатель класса QGraphicsScene - \*scene.

class Graph : public QMainWindow

{

Q\_OBJECT

private:

Ui::Graph \*ui;

QGraphicsScene \*scene;

public:

int kol;

std::vector<std::vector<int>> matrix;

int result[6] = { INT\_MAX, INT\_MAX, INT\_MAX, INT\_MAX,INT\_MAX, INT\_MAX };

bool visited[6] = { 0, 0, 0, 0, 0, 0 };

Graph(QWidget \*parent = nullptr);

~Graph();

int GetWeight(const int vert1, const int vert2);

void DrawLines(QGraphicsScene\* scene);

void DrawArrows(QGraphicsScene\* scene);

void DrawNodes(QGraphicsScene\* scene);

void DrawGraph(QGraphicsScene\* scene);

QString Dejkstra();

void Draw();

private slots:

};

1. Для решения задачи данные были представлены в следующем виде:
   1. Глобальные переменные radiusA, radiusB отвечающие за радиус узлов и расстояние расположения этих узлов от центра.

int radiusA = 30,

radiusB = 230;

1. Для операций ввода и вывода использовались следующие операторы и функции:
   1. За отрисовку сцены отвечает указатель scene класса QGraphicsScene

scene = new QGraphicsScene();

scene->setSceneRect(0, 0, 700, 630);

ui->graphicsView->setScene(scene);

1. Поставленные задачи будут решены следующими действиями:
   1. Функция Dejkstra() используется для вывода кратчайших путей до вершин – реализация метода Дейкстры.

QString Graph::Dejkstra()

{

QString tmp = "";

int top = 0; //вершина с номером 1

queue<int> line;

line.push(top);

result[0] = 0;

while (!line.empty()) {

int vert = line.front();

line.pop();

for (int i = 0; i < matrix[vert].size(); i++) {

if (!visited[i] && matrix[vert][i] && (matrix[vert][i] + result[vert] < result[i])) {

result[i] = matrix[vert][i] + result[vert];

line.push(i);

}

}

}

for (int i = 0; i < 6; i++) {

cout << i + 1 << ": " << result[i] << endl;

tmp += QString::number(i + 1) + ": " + QString::number(result[i]) + "\n";

}

return tmp;

}

* 1. Метод GetWeight() для получения длины пути (веса) от одного узла до другого.

int Graph::GetWeight(const int vert1, const int vert2)

{

if (vert1 >= 0 && vert1 < matrix.size() && vert2 >=0 && vert2 < matrix.size())

return matrix[vert1][vert2];

else

return 0;

}

* 1. Метод DrawGraph() вызывает методы отрисовки вершин (DrawNodes), ребер (DrawLines) и стрелок (DrawArrows).
  2. Метод DrawLines отрисовывает ребра графа.

void Graph::DrawLines(QGraphicsScene\* scene) //отрисовка ребер графа

{

QPen pen(Qt::black);

pen.setWidth(2);

int centerX= scene->width() / 2,

centerY = scene->height() / 2;

double iter = 2 \* 3.1415 / matrix.size();

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

for (int j = 0; j < matrix.size(); j++){

if (GetWeight(i, j) > 0){

int vert\_1\_x = centerX + cos(iter \* i) \* radiusB,

vert\_1\_y = centerY - sin(iter \* i) \* radiusB,

vert\_2\_x = centerX + cos(iter \* j) \* radiusB,

vert\_2\_y = centerY - sin(iter \* j) \* radiusB;

scene->addLine(vert\_1\_x, vert\_1\_y, vert\_2\_x, vert\_2\_y, pen);

QGraphicsTextItem \*number = scene->addText(QString::number(GetWeight(i, j)), QFont("Times", 14));

number->setPos((vert\_2\_x + vert\_1\_x) / 2, (vert\_2\_y + vert\_1\_y) / 2);

}

}

}

}

* 1. Метод DrawArrows отрисовывает стрелки графа.

void Graph::DrawArrows(QGraphicsScene\* scene) //отрисовка стрелок

{

QPen pen(Qt::black);

pen.setWidth(2);

int centerX= scene->width() / 2,

centerY = scene->height() / 2;

int r1 = radiusA / 3;

double len = 15.0;

double ostr = 0.35;

double arr\_x, arr\_y;

double iter = 2 \* 3.1415 / matrix.size();

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

for (int j = 0; j < matrix.size(); j++){

if (GetWeight(i, j) > 0){

int vert\_1\_x = centerX + cos(iter \* i) \* radiusB,

vert\_1\_y = centerY - sin(iter \* i) \* radiusB,

vert\_2\_x = centerX + cos(iter \* j) \* radiusB,

vert\_2\_y = centerY - sin(iter \* j) \* radiusB;

double angle = atan2(((double)vert\_2\_y - vert\_1\_y), ((double)(vert\_2\_x - vert\_1\_x)));

//double t = radiusA / sqrt((vert\_2\_x - vert\_1\_x) \* (vert\_2\_x - vert\_1\_x) + (vert\_2\_y - vert\_1\_y) \* (vert\_2\_y - vert\_1\_y));

arr\_x = (vert\_2\_x - (radiusA \* cos(angle))) - (len \* cos(angle + ostr));

arr\_y = (vert\_2\_y - (radiusA \* sin(angle))) - (len \* sin(angle + ostr));

scene->addLine(vert\_2\_x - (radiusA \* cos(angle)), vert\_2\_y - (radiusA \* sin(angle)), arr\_x, arr\_y, pen);

arr\_x = (vert\_2\_x - (radiusA \* cos(angle))) - (len \* cos(angle - ostr));

arr\_y = (vert\_2\_y - (radiusA \* sin(angle))) - (len \* sin(angle - ostr));

scene->addLine(vert\_2\_x - (radiusA \* cos(angle)), vert\_2\_y - (radiusA \* sin(angle)), arr\_x, arr\_y, pen);

}

}

}

}

* 1. Метод DrawNodes отрисовывает вершины графа.

void Graph::DrawNodes(QGraphicsScene\* scene) //отрисовка вершин графа

{

QPen pen(Qt::black);

pen.setWidth(3);

int centerX= scene->width() / 2,

centerY = scene->height() / 2;

double iter = 2 \* 3.1415 / matrix.size();

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

scene->addEllipse(centerX + cos(iter \* i) \* radiusB - radiusA,

centerY - sin(iter \* i) \* radiusB - radiusA, 2\*radiusA, 2\*radiusA,

pen, QBrush(QColor(193,251,227)));

QGraphicsTextItem \*number = scene->addText(QString::number(i+1), QFont("Times", 14));

number->setPos(centerX + cos(iter \* i) \* radiusB - radiusA + 18,

centerY - sin(iter \* i) \* radiusB - radiusA + 10);

}

}

**Блок-схема программы**

**Код программы**

**Файл graph.h:**

1. #ifndef GRAPH\_H
2. #define GRAPH\_H
4. #include <QMainWindow>
5. #include <QGraphicsScene>
6. #include <QGraphicsItem>
7. #include <QPen>
8. #include <QBrush>
9. #include <QFont>
10. #include <QString>
11. #include <QGraphicsTextItem>
12. #include <vector>

15. QT\_BEGIN\_NAMESPACE
16. namespace Ui { class Graph; }
17. QT\_END\_NAMESPACE
19. //int MAX\_SIZE = 20;
21. class Graph : public QMainWindow
22. {
23. Q\_OBJECT
25. private:
26. Ui::Graph \*ui;
27. QGraphicsScene \*scene;
29. public:
30. int kol;
31. std::vector<std::vector<int>> matrix;
32. int result[6] = { INT\_MAX, INT\_MAX, INT\_MAX, INT\_MAX,INT\_MAX, INT\_MAX };
33. bool visited[6] = { 0, 0, 0, 0, 0, 0 };
35. Graph(QWidget \*parent = nullptr);
36. ~Graph();
37. int GetWeight(const int vert1, const int vert2);
38. void DrawLines(QGraphicsScene\* scene);
39. void DrawArrows(QGraphicsScene\* scene);
40. void DrawNodes(QGraphicsScene\* scene);
41. void DrawGraph(QGraphicsScene\* scene);
42. QString Dejkstra();
43. void Draw();
44. private slots:
45. };
46. #endif // GRAPH\_H

**Файл graph.cpp:**

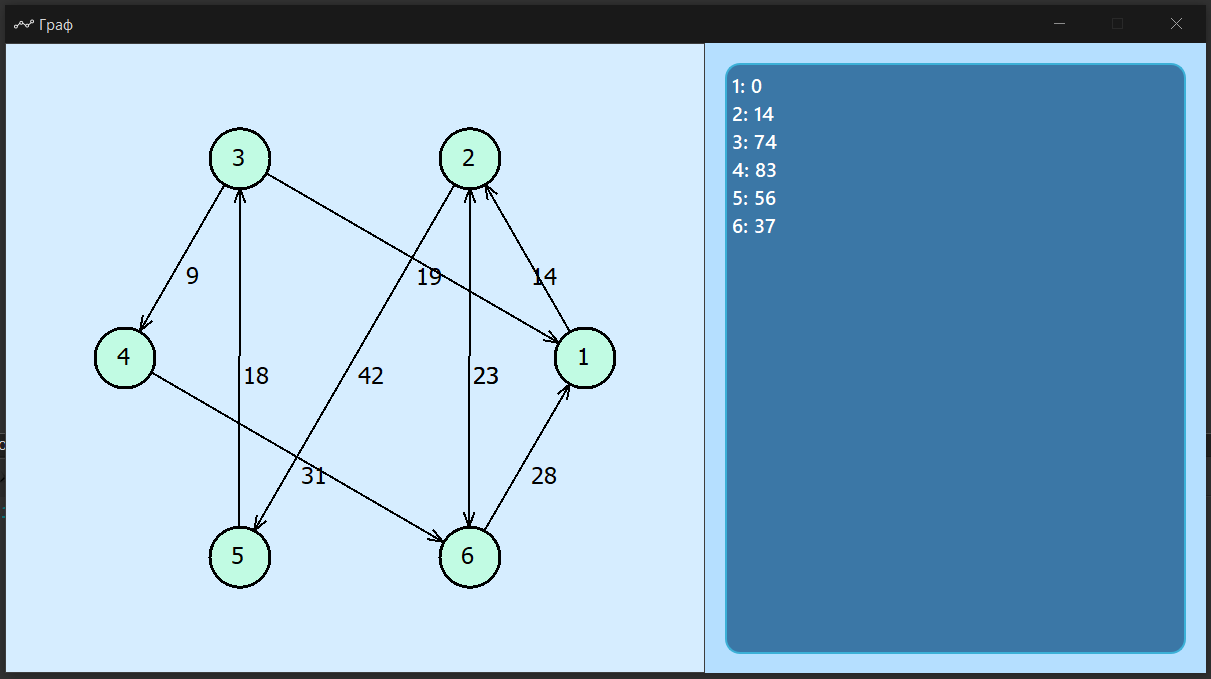
1. #include "graph.h"
2. #include "ui\_graph.h"
3. #include <cmath>
4. #include <queue>
5. #include <QTextStream>
6. using namespace std;
8. QTextStream cout(stdout);
9. QTextStream cin(stdin);
11. int radiusA = 30,
12. radiusB = 230;

15. QString Graph::Dejkstra()
16. {
17. QString tmp = "";
18. int top = 0; //вершина с номером 1
19. queue<int> line;
20. line.push(top);
21. result[0] = 0;
22. while (!line.empty()) {
23. int vert = line.front();
24. line.pop();
25. for (int i = 0; i < matrix[vert].size(); i++) {
26. if (!visited[i] && matrix[vert][i] && (matrix[vert][i] + result[vert] < result[i])) {
27. result[i] = matrix[vert][i] + result[vert];
28. line.push(i);
29. }
30. }
31. }
32. for (int i = 0; i < 6; i++) {
33. cout << i + 1 << ": " << result[i] << endl;
34. tmp += QString::number(i + 1) + ": " + QString::number(result[i]) + "\n";
35. }
36. return tmp;
37. }

40. int Graph::GetWeight(const int vert1, const int vert2) //получение веса ребра
41. {
42. if (vert1 >= 0 && vert1 < matrix.size() && vert2 >=0 && vert2 < matrix.size())
43. return matrix[vert1][vert2];
44. else
45. return 0;
46. }
48. void Graph::DrawLines(QGraphicsScene\* scene) //отрисовка ребер графа
49. {
50. QPen pen(Qt::black);
51. pen.setWidth(2);
52. int centerX= scene->width() / 2,
53. centerY = scene->height() / 2;
54. double iter = 2 \* 3.1415 / matrix.size();
55. for (int i = 0; i < matrix.size(); i++){
56. for (int j = 0; j < matrix.size(); j++){
57. if (GetWeight(i, j) > 0){
58. int vert\_1\_x = centerX + cos(iter \* i) \* radiusB,
59. vert\_1\_y = centerY - sin(iter \* i) \* radiusB,
60. vert\_2\_x = centerX + cos(iter \* j) \* radiusB,
61. vert\_2\_y = centerY - sin(iter \* j) \* radiusB;
62. scene->addLine(vert\_1\_x, vert\_1\_y, vert\_2\_x, vert\_2\_y, pen);
63. QGraphicsTextItem \*number = scene->addText(QString::number(GetWeight(i, j)), QFont("Times", 14));
64. number->setPos((vert\_2\_x + vert\_1\_x) / 2, (vert\_2\_y + vert\_1\_y) / 2);
65. }
66. }
67. }
68. }
70. void Graph::DrawArrows(QGraphicsScene\* scene) //отрисовка стрелок
71. {
72. QPen pen(Qt::black);
73. pen.setWidth(2);
74. int centerX= scene->width() / 2,
75. centerY = scene->height() / 2;
76. int r1 = radiusA / 3;
77. double len = 15.0;
78. double ostr = 0.35;
79. double arr\_x, arr\_y;
80. double iter = 2 \* 3.1415 / matrix.size();
81. for (int i = 0; i < matrix.size(); i++){
82. for (int j = 0; j < matrix.size(); j++){
83. if (GetWeight(i, j) > 0){
84. int vert\_1\_x = centerX + cos(iter \* i) \* radiusB,
85. vert\_1\_y = centerY - sin(iter \* i) \* radiusB,
86. vert\_2\_x = centerX + cos(iter \* j) \* radiusB,
87. vert\_2\_y = centerY - sin(iter \* j) \* radiusB;
88. double angle = atan2(((double)vert\_2\_y - vert\_1\_y), ((double)(vert\_2\_x - vert\_1\_x)));
89. //double t = radiusA / sqrt((vert\_2\_x - vert\_1\_x) \* (vert\_2\_x - vert\_1\_x) + (vert\_2\_y - vert\_1\_y) \* (vert\_2\_y - vert\_1\_y));
90. arr\_x = (vert\_2\_x - (radiusA \* cos(angle))) - (len \* cos(angle + ostr));
91. arr\_y = (vert\_2\_y - (radiusA \* sin(angle))) - (len \* sin(angle + ostr));
92. scene->addLine(vert\_2\_x - (radiusA \* cos(angle)), vert\_2\_y - (radiusA \* sin(angle)), arr\_x, arr\_y, pen);
93. arr\_x = (vert\_2\_x - (radiusA \* cos(angle))) - (len \* cos(angle - ostr));
94. arr\_y = (vert\_2\_y - (radiusA \* sin(angle))) - (len \* sin(angle - ostr));
95. scene->addLine(vert\_2\_x - (radiusA \* cos(angle)), vert\_2\_y - (radiusA \* sin(angle)), arr\_x, arr\_y, pen);
96. }
97. }
98. }
99. }
101. void Graph::DrawNodes(QGraphicsScene\* scene) //отрисовка вершин графа
102. {
103. QPen pen(Qt::black);
104. pen.setWidth(3);
105. int centerX= scene->width() / 2,
106. centerY = scene->height() / 2;
107. double iter = 2 \* 3.1415 / matrix.size();
108. for (int i = 0; i < matrix.size(); i++){
109. scene->addEllipse(centerX + cos(iter \* i) \* radiusB - radiusA,
110. centerY - sin(iter \* i) \* radiusB - radiusA, 2\*radiusA, 2\*radiusA,
111. pen, QBrush(QColor(193,251,227)));
113. QGraphicsTextItem \*number = scene->addText(QString::number(i+1), QFont("Times", 14));
114. number->setPos(centerX + cos(iter \* i) \* radiusB - radiusA + 18,
115. centerY - sin(iter \* i) \* radiusB - radiusA + 10);
116. }
117. }

120. void Graph::DrawGraph(QGraphicsScene\* scene) //отрисовка графа
121. {
122. DrawLines(scene);
123. DrawArrows(scene);
124. DrawNodes(scene);
125. }
127. void Graph::Draw() //отрисовка сцены с графом
128. {
129. scene = new QGraphicsScene();
130. scene->setSceneRect(0, 0, 700, 630);
132. ui->graphicsView->setScene(scene);
133. scene->clear();
134. scene->setBackgroundBrush(QBrush(QColor(214,237,255), Qt::SolidPattern));
135. ui->label\_komi->setText(Dejkstra());
136. DrawGraph(scene);
137. }
139. Graph::Graph(QWidget \*parent)
140. : QMainWindow(parent)
141. , ui(new Ui::Graph)
142. {
143. ui->setupUi(this);
144. kol = 6;
145. matrix = { {0, 14, 0, 0, 0, 0 },
146. {0, 0, 0, 0, 42, 23 },
147. {19, 0, 0, 9, 0, 0 },
148. {0, 0, 0, 0, 0, 31 },
149. {0, 0, 18, 0, 0, 0 },
150. {28, 23, 0, 0, 0, 0 } };
151. Draw();
152. }
154. Graph::~Graph()
155. {
156. delete ui;
157. }

**Скриншоты результатов работы программы**

****