Министерство образования и науки Российской Федерации

Федеральное государственное бюджетное образовательное учреждение высшего образования

**«Пермский национальный исследовательский**

**политехнический университет»**

**(ПНИПУ)**

Электротехнический факультет

Кафедра «Информационные технологии и автоматизированные системы»

Направление подготовки: 09.03.01- «Информатика и вычислительная техника»

**Творческая работа №2**

**по дисциплине**

**«Основы алгоритмизации и программирования 2 семестр»**

Выполнил студент гр. ИВТ-21-1б

Кокшаров Кирилл Сергеевич

Проверил:

Яруллин Денис Владимирович

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(оценка) (подпись)

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(дата)

Пермь 2022

**Постановка задачи:**

Решить задачу коммивояжера метод ветвей и границ с выводом графа

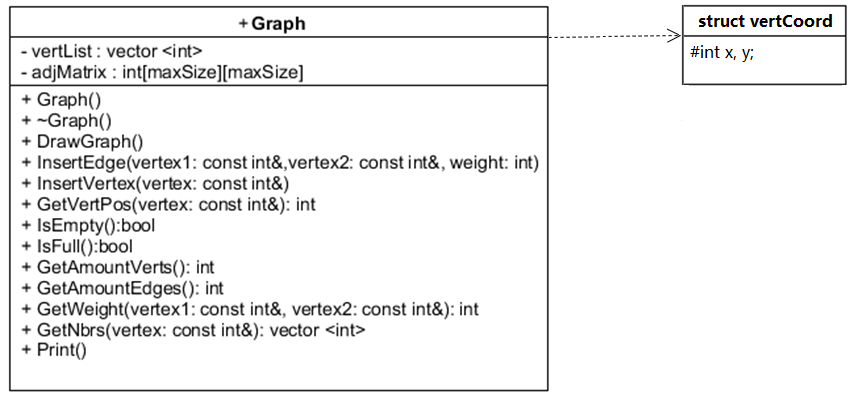
**Описание решения:**

Для решения задачи коммивояжера был использован метод ветвей и границ. Для визуализации графа было использовано OpenGL.При создание проекта встретились трудности при установке OpenGL но при помощи интернет ресурсов эта проблема была решена. Автор гордится ручным вводом графа ведь это увеличивает функциональность программы и проверкой корректности ввода графа. Дизайн был выбран простой вершины графов были сделаны выделяющимся желтым цветом.В программе есть три основных функции preparation,answer,tsalesman

функция preparation создает вспомогательные массивы и переменные.

после чего все эти данные передаются в функцию answer где происходит решение задачи коммивояжёра. И с помощью функции tsalesman результат выводится в консоль.

**Диаграмма классов:**

****

**Код:**

**CPP**

#include "graph.h"

#include <glut.h>

#include <stdio.h>

#include <iostream>

#include <vector>

#include <sstream>

int main()

{

setlocale(LC\_ALL, "rus");

graph = makeGraph();

graph.Print();

tsalesman(mat, n, help, result);

glutInitDisplayMode(GLUT\_DOUBLE | GLUT\_RGBA);

glutInitWindowSize(1350, 730);

glutCreateWindow("Graph");

glutDisplayFunc(display);

glutReshapeFunc(reshape);

glutMainLoop();

return 0;

}

**h**

#pragma once

#include <glut.h>

#include <stdio.h>

#include <iostream>

#include <vector>

#include <sstream>

using namespace std;

int n;

int\*\* help;

int\* result;

int\*\*\* mat;

int WinW;

int WinH;

const int maxSize = 20;

int amountVerts;

template<class T>

class Graph

{

vector<T> vertList;

vector<T> labelList;

bool\* visitedVerts = new bool[vertList.size()];

public:

int adjMatrix[maxSize][maxSize] = { 0 };

Graph();

~Graph();

void DrawGraph();

void InsertEdge(const T& vertex1, const T& vertex2, int weight);

void InsertVertex(const T& vertex);

int GetVertPos(const T& vertex);

bool IsEmpty();

bool IsFull();

int GetAmountVerts();

int GetAmountEdges();

int GetWeight(const T& vertex1, const T& vertex2);

vector<T> GetNbrs(const T& vertex);

void Print();

};

int R;

struct vertCoord

{

int x, y;

};

vertCoord vertC[20];

Graph<int> graph;

void preparation(int\*\*\*& mat, int& n, int\*\*& help, int\*& result)

{

n = amountVerts;

help = new int\* [n];

result = new int[n];

mat = new int\*\* [n];

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

{

help[i] = new int[n];

}

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

{

mat[i] = new int\* [n];

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

{

if (graph.adjMatrix[i][j] == 0) {

mat[i][j] = nullptr;

continue;

}

mat[i][j] = new int(graph.adjMatrix[i][j]);

}

}

}

void answer(int\*\*\* mat, int n, int\*\* help, int\* path)

{

for (int l = 0; l < n; l++)

{

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

{

int min = 1000000;

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

if (mat[i][j] && min > \*mat[i][j])

min = \*mat[i][j];

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

if (mat[i][j])

\*mat[i][j] -= min;

}

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

{

int min = 1000000;

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

if (mat[i][j] && min > \*mat[i][j])

min = \*mat[i][j];

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

if (mat[i][j])

\*mat[i][j] -= min;

}

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

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

help[i][j] = 0;

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

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

{

if (mat[i][j] && !\*mat[i][j])

{

int hmin = 1000000;

int vmin = 1000000;

for (int l = 0; l < n; l++)

if (l != i && mat[l][j] && hmin > \*mat[l][j])

hmin = \*mat[l][j];

for (int l = 0; l < n; l++)

if (l != j && mat[i][l] && vmin > \*mat[i][l])

vmin = \*mat[i][l];

help[i][j] = hmin + vmin;

}

}

int mcost = 0, mi = 0, mj = 0;

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

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

if (mat[i][j] && mcost < help[i][j])

{

mcost = help[i][j];

mi = i;

mj = j;

}

path[mi] = mj;

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

mat[i][mj] = nullptr;

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

mat[mi][i] = nullptr;

mat[mj][mi] = nullptr;

}

}

void tsalesman(int\*\*\* mat, int n, int\*\* help, int\* result)

{

preparation(mat, n, help, result);

int s = 0;

answer(mat, n, help, result);

cout << "\nОтрезки путей: ";

for (int i = 0, j = 0; i < n; i++)

{

j = result[i];

cout << i + 1 << " -> " << j + 1 << '\t';

s += graph.adjMatrix[i][j];

}

cout << endl;

cout << "\nКратчайший путь: ";

int temp = 0;

for (int l = 0; l < n;)

{

for (int i = 0, j = 0; i < n; i++)

{

if (temp == 0 || i + 1 == temp)

{

if (temp == 0) cout << i + 1;

j = result[i];

temp = j + 1;

if (temp > 0) cout << " -> " << temp;

l++;

}

}

}

cout << "\nРасстояние: " << s;

cout << endl;

}

template<class T>

void Graph<T>::InsertVertex(const T& vertex) {

if (!this->IsFull()) {

this->vertList.push\_back(vertex);

}

else {

cout << "Граф уже заполнен. Невозможно добавить новую вершину " << endl;

return;

}

}

template<class T>

int Graph<T>::GetAmountEdges() {

int amount = 0; // обнуляем счетчик

if (!this->IsEmpty()) { // проверяем, что граф не пуст

for (int i = 0, vertListSize = this->vertList.size();

i < vertListSize; ++i) {

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

if (this->adjMatrix[i][j] ==

this->adjMatrix[j][i] &&

this->adjMatrix[i][j] != 0) // находим рёбра

amount += 1; // считаем количество рёбер

}

}

return (amount / 2); // приводим счетчик к корректному результату и возвращаем его

}

else

return 0; // если граф пуст, возвращаем 0

}

template<class T>

int Graph<T>::GetWeight(const T& vertex1, const T& vertex2) {

if (!this->IsEmpty()) {

int vertPos1 = GetVertPos(vertex1);

int vertPos2 = GetVertPos(vertex2);

return adjMatrix[vertPos1][vertPos2];

}

return 0;

}

template<class T>

int Graph<T>::GetAmountVerts() {

return this->vertList.size();

}

template<class T>

bool Graph<T>::IsEmpty() {

if (this->vertList.size() != 0)

return false;

else

return true;

}

template<class T>

bool Graph<T>::IsFull() {

return (vertList.size() == maxSize);

}

template <class T>

int Graph<T>::GetVertPos(const T& vertex) {

for (int i = 0; i < this->vertList.size(); ++i) {

if (this->vertList[i] == vertex)

return i;

}

return -1;

}

template<class T>

Graph<T>::Graph() {

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

{

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

{

this->adjMatrix[i][j] = 0;

}

}

}

template<class T>

Graph<T>::~Graph() {

}

Graph<int> makeGraph()

{

Graph<int> graph; // создание графа имеюшего вершины с номерами

int amountEdges, sourceVertex, targetVertex, edgeWeight; // создание необходимых переменных

cout << "Видите количество вершин графа: "; cin >> amountVerts; cout << endl; // ввод количества рёбер графа в переменную amountVerts

cout << "Введите количество ребер графа: "; cin >> amountEdges; cout << endl; // ввод количества рёбер графа в переменную amountEdges

for (int i = 1; i <= amountVerts; ++i) {

int\* vertPtr = &i; // запоминаем адрес вершины с помощью указателя

graph.InsertVertex(\*vertPtr); //передаём ссылку на вершину в функцию InsertVertex; происходит вставка вершины в вектор вершин

}

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

cout << "Исходная вершина: "; cin >> sourceVertex; cout << endl; // ввод исходной вершины

int\* sourceVertPtr = &sourceVertex; // запоминаем адрес исходной вершины

cout << "Конечная вершина: "; cin >> targetVertex; cout << endl; // ввод вершины до которой будет идти ребро

cout << "Вес ребра: "; cin >> edgeWeight; cout << endl; // ввод числового значения веса ребра в переменную edgeWeight

graph.InsertEdge(\*sourceVertPtr, \*targetVertPtr, edgeWeight);// вставка ребра весом edgeWeight между исходной и конечной вершинами

}

cout << endl;

return graph;

}

template<class T>

void Graph<T>::InsertEdge(const T& vertex1, const T& vertex2, int weight) {

if (this->GetVertPos(vertex1) != (-1) && this->GetVertPos(vertex2) != (-1)) {

int vertPos1 = GetVertPos(vertex1);

int vertPos2 = GetVertPos(vertex2);

if (this->adjMatrix[vertPos1][vertPos2] != 0

&& this->adjMatrix[vertPos2][vertPos1] != 0) {

cout << "Ребро между вершинами уже есть" << endl;

return;

}

else {

this->adjMatrix[vertPos1][vertPos2] = weight;

this->adjMatrix[vertPos2][vertPos1] = weight;

}

}

else {

cout << "Обеих вершин (или одной из них) нет в графе " << endl;

return;

}

}

template<class T>

void Graph<T>::Print() {

if (!this->IsEmpty()) {

cout << "Матрица смежности графа: " << endl;

for (int i = 0, vertListSize = this->vertList.size(); i < vertListSize; ++i) {

cout << this->vertList[i] << " ";

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

cout << " " << this->adjMatrix[i][j] << " ";

}

cout << endl;

}

}

}

void setCoord(int i, int n)

{

int R\_;

int x0 = WinW / 2;

int y0 = WinH / 2;

if (WinW > WinH)

{

R = 5 \* (WinH / 13) / n;

R\_ = WinH / 2 - R - 10;

}

else {

R = 5 \* (WinW / 13) / n;

R\_ = WinW / 2 - R - 10;

}

float theta = 2.0f \* 3.1415926f \* float(i) / float(n);

float y1 = R\_ \* cos(theta) + y0;

float x1 = R\_ \* sin(theta) + x0;

vertC[i].x = x1;

vertC[i].y = y1;

}

void drawCircle(int x, int y, int R) //рисуем круг в заданных координатах

{

glColor3f(1.0, 1.0, 0.0);

float x1, y1;

glBegin(GL\_POLYGON);

for (int i = 0; i < 360; i++)

{

float theta = 2.0f \* 3.1415926f \* float(i) / float(360);

y1 = R \* cos(theta) + y;

x1 = R \* sin(theta) + x;;

glVertex2f(x1, y1);

}

glEnd();

glColor3f(0.0f, 0.0f, 0.0f);

float x2, y2;

glBegin(GL\_LINE\_LOOP);

for (int i = 0; i < 360; i++)

{

float theta = 2.0f \* 3.1415926f \* float(i) / float(360);

y2 = R \* cos(theta) + y;

x2 = R \* sin(theta) + x;

glVertex2f(x2, y2);

}

glEnd();

}

void drawText(int nom, int x1, int y1)

{

GLvoid\* font = GLUT\_BITMAP\_HELVETICA\_18;

string s = to\_string(nom);

glRasterPos2i(x1 - 5, y1 - 5);

for (int j = 0; j < s.length(); j++)

glutBitmapCharacter(font, s[j]);

}

void drawVertex(int n)

{

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

drawCircle(vertC[i].x, vertC[i].y, R);

drawText(i + 1, vertC[i].x, vertC[i].y);

}

}

void drawLine(int text, int x0, int y0, int x1, int y1) //ребро неориентированный взвешенный граф

{

glColor3f(0.0f, 0.0f, 0.0f);

glBegin(GL\_LINES);

glVertex2i(x0, y0);

glVertex2i(x1, y1);

glEnd();

drawText(text, (x0 + x1) / 2 + 10, (y0 + y1) / 2 + 10);

}

template<class T>

void Graph<T>::DrawGraph()

{

int n = vertList.size();

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

{

setCoord(i, n);

}

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

{

for (int j = i + 1; j < n; j++)

{

int a = adjMatrix[i][j];

if (a != 0)

{

drawLine(a, vertC[i].x, vertC[i].y, vertC[j].x, vertC[j].y);

}

}

}

drawVertex(n);

}

void reshape(int w, int h)

{

WinW = w;

WinH = h;

glViewport(0, 0, (GLsizei)WinW, (GLsizei)WinH);

glMatrixMode(GL\_PROJECTION);

glLoadIdentity();

gluOrtho2D(0, (GLdouble)WinW, 0, (GLdouble)WinH);

glutPostRedisplay();

}

void drawMenuText(string text, int x1, int y1)

{

GLvoid\* font = GLUT\_BITMAP\_HELVETICA\_18;

string s = text;

glRasterPos2i(x1 + 5, y1 - 20);

for (int j = 0; j < s.length(); j++)

glutBitmapCharacter(font, s[j]);

}

void display()

{

glShadeModel(GL\_SMOOTH);

glMatrixMode(GL\_PROJECTION);

glLoadIdentity();

gluOrtho2D(0, WinW, 0, WinH);

glViewport(0, 0, WinW, WinH);

glClearColor(1.0, 1.0, 1.0, 1.0);

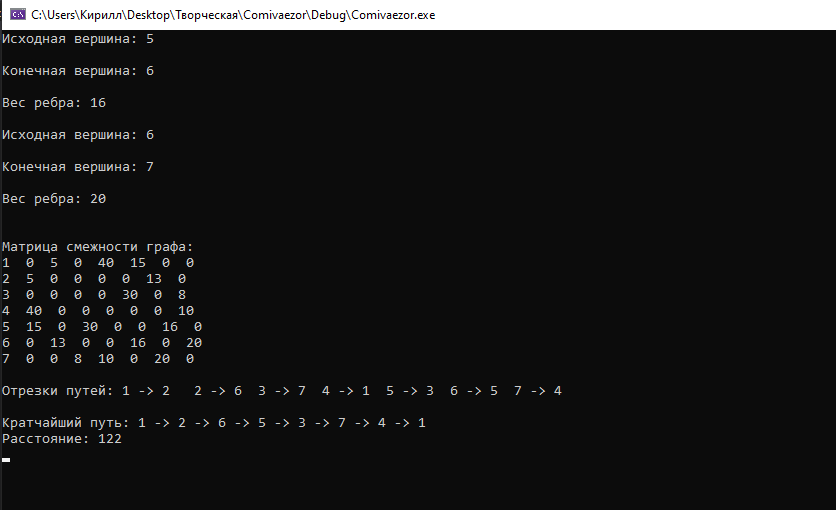
glClear(GL\_COLOR\_BUFFER\_BIT);

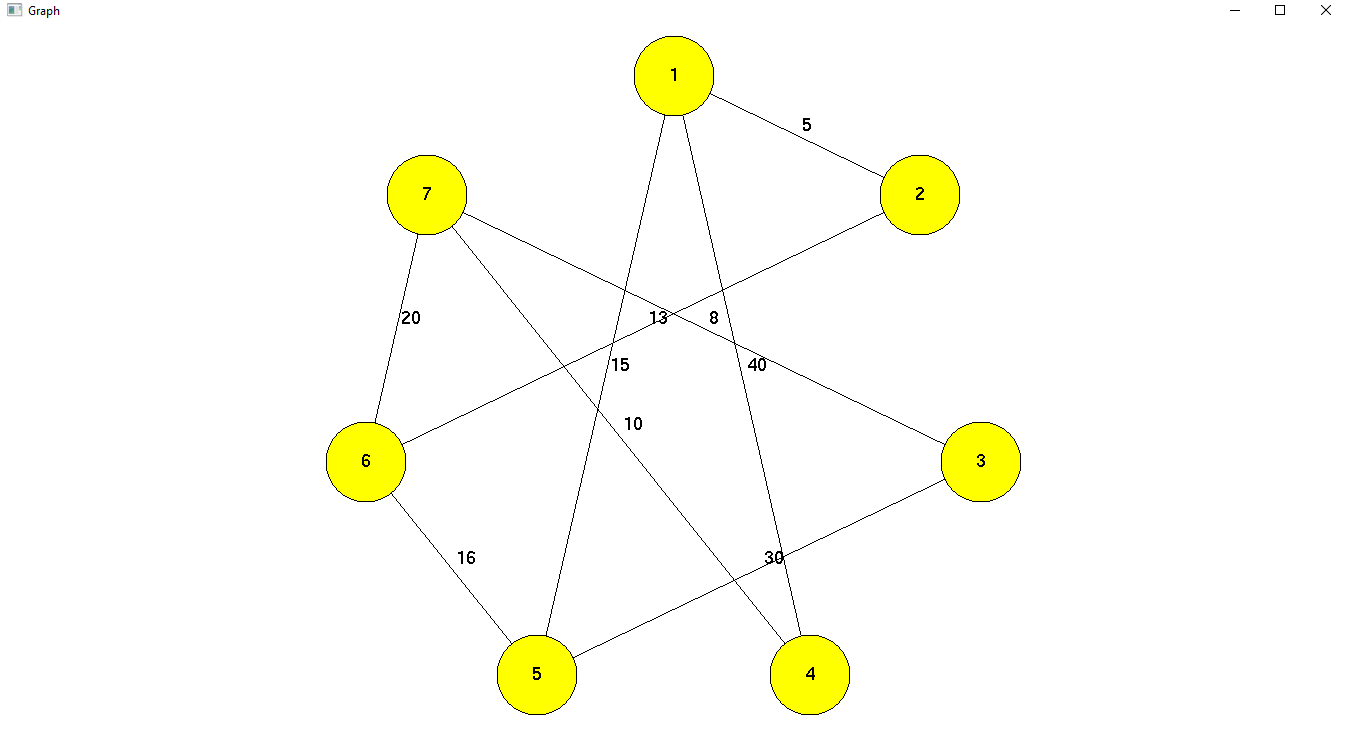
graph.DrawGraph();

glutSwapBuffers();

}

**Скриншоты:**

****



**Ссылки:**

<https://github.com/kirillX18>

**https://www.youtube.com/watch?v=1O2b8QoMfmU&t=29s**