Министерство образования Российской Федерации

Пензенский государственный университет

Кафедра «Вычислительная техника»

**ПОЯСНИТЕЛЬНАЯ ЗАПИСКА**

К курсовой работе

по курсу «Логика и основы алгоритмизации в инженерных задачах»

на тему «Реализация алгоритма Прима»

Выполнил:

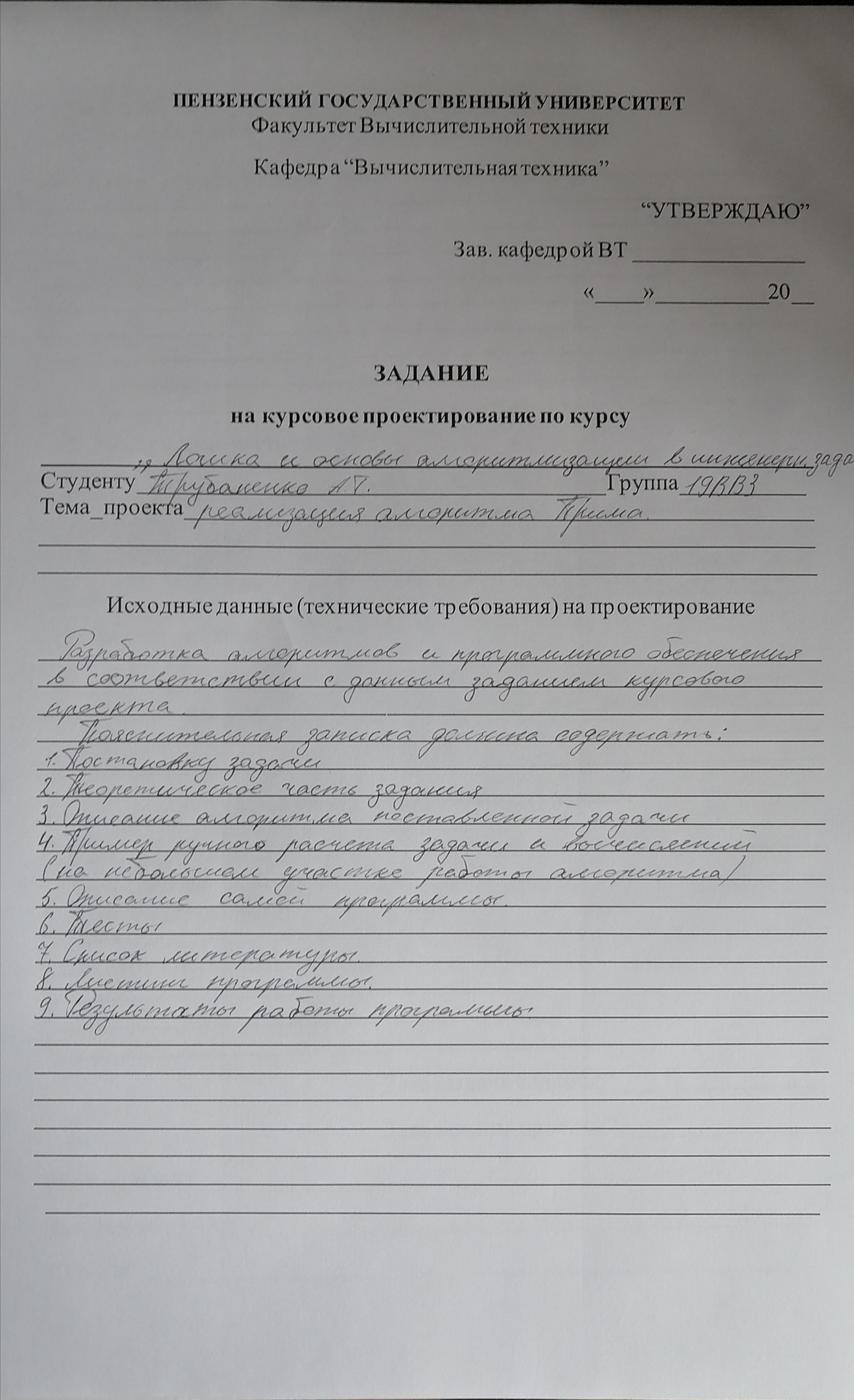
Студент группы 19ВВ3

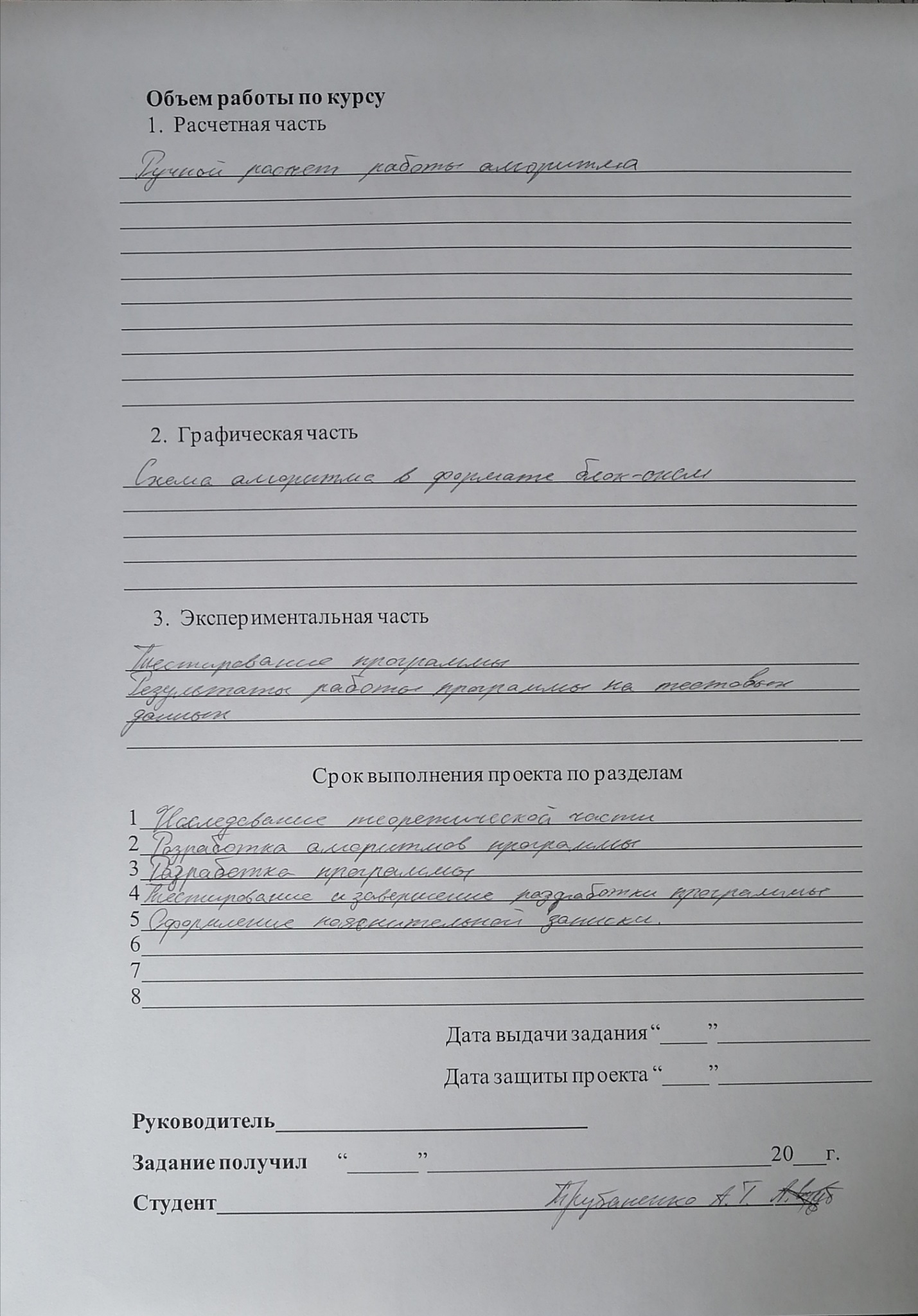
Трубаненко А.Г.

Принял:

Митрохин М.А.

Пенза 2020





**Оглавление**

[Реферат 6](#_Toc59991075)

[Введение 7](#_Toc59991076)

[Постановка задачи 8](#_Toc59991077)

[Теоретическая часть 9](#_Toc59991078)

[Описание алгоритма программы 10](#_Toc59991079)

[Пример расчет задачи 11](#_Toc59991080)

[Описание программы 13](#_Toc59991081)

[Тестирование 20](#_Toc59991082)

[Заключение 25](#_Toc59991083)

[Список литературы 26](#_Toc59991084)

[Приложение А 27](#_Toc59991085)

[Листинг 27](#_Toc59991086)

# **Реферат**

**Отчет 34 стр, 16 рисунков.**

Реализация алгоритма Прима.

Цель исследования – разработка программы, выполняющей алгоритм Прима.

В работе рассмотрен способ минимального остовного дерева .

# **Введение**

Алгоритм Прима - это алгоритм минимального остовного дерева, что принимает граф в качестве входных данных и находит подмножество ребер этого графа, который формирует дерево, включающее в себя каждую вершину, а также имеет минимальную сумму весов среди всех деревьев, которые могут быть сформированы из графа.

Временная сложность данной реализации алгоритма Прима составляет O().

Вес ребра – числовое значение, поставленное в соответствие данному ребру графа. Вес ребра можно интерпретировать как длину ребра.

Остовное дерево – это связный подграф без циклов данного связного неориентированного графа, в который входят все его вершины.

Минимальное остовное дерево – это остовное дерево данного графа имеющее минимальный возможный вес. Под весом остовного дерева понимается сумма весов всех ребер, входящих в него.

Постановка задачи

Требуется разработать программу, которая реализует алгоритм прима.

Условный граф должен задаваться матрицей смежности. Программа должна работать так, чтобы у пользователя был выбор самому вводить данные или случайно сгенерировать, после обработки данных на экран должна выводиться матрица смежности и результат работы алгоритма Прима.

Устройство ввода- клавиатура.

# **1 Теоретическая часть**

Граф G (рисунок 1) задается множеством вершин X1, X2, ..., Xn. и

множеством ребер , соединяющих между собой определенные вершины.

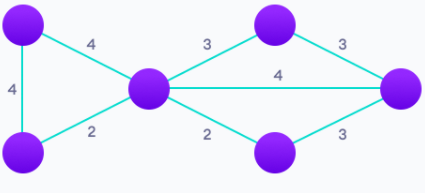


Рисунок 1-Пример графа

Для взвешенных графов существует понятие *веса* остовного дерева, которое определено как сумма весов всех рёбер, входящих в остовное дерево. Из него натурально вытекает понятие **минимального остовного дерева** - остовного дерева с минимальным возможным весом. При представлении графа матрицей смежности информация о ребрах графа хранится в квадратной матрице, где присутствие пути из одной вершины в другую обозначается весом ребра, иначе нулем.

# 

# **2 Описание алгоритма программы**

Псевдокод алгоритма прима показывает, как мы создаем два набора вершин U и VU. U содержит список вершин, которые были посещены, а VU - список вершин, которые не были посещены. Одна за другой мы перемещаем вершины из множества VU в множество U, соединяя ребро наименьшего веса.

**Вход**: Матрица смежности Matrix графа G,

**Выход:** Минимальные пути x,y,

**Алгоритм ПОШ**

1. Пока (no\_edge < graph->size- 1) выполняем
2. int min = INF;
3. x = 0;
4. y = 0;
5. Для (int i = 0; i < graph->size; i++)
6. {Если (selected[i]) {
   1. Для (int j = 0; j < graph->size; j++) {
      1. Если (!selected[j] && G[i][j]) {
      2. Если (min > G[i][j]) {
      3. min = G[i][j];
      4. x = i;
      5. y = j;

# **3 Пример ручного расчета задачи**

Начинаем со взвешенного графа.

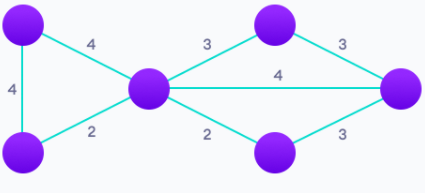


Рисунок 2 - Граф для ручного расчета

Для начала выберем вершину, затем выберем самое короткое ребро из этой вершины и добавим его.

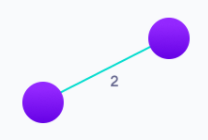


Рисунок 3 - Выбираем самое короткое ребро

Выбираем ближайшую вершину, еще не входившее в решение (Рисунок 4)

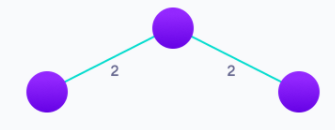


Рисунок 4- Выбор вершины, еще не входящую в решение

Далее выбираем ближайшее ребро, которого еще нет в решении, если есть несколько вариантов, то выбираем одно наугад (Рисунок 5)

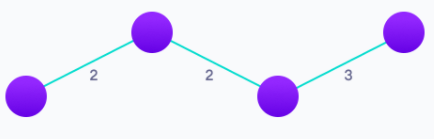


Рисунок 5- Выбор ближайшей вершины

Далее повторяем, пока не получиться остовное дерево

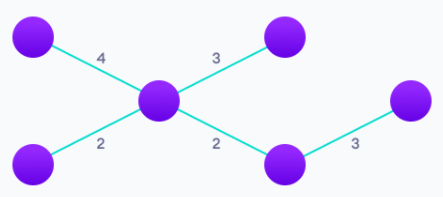


Рисунок 6- Достройка до остовного дерева

# **4 Описание программы**

Для написания данной программы были использованы языки программирование СИ и СИ++.

Проект был создан в виде консольного приложения в программе VS 2019.

Даная программа является многомодульной. Состоит из нескольких файлов courses.cpp, Matrix\_graph.cpp и заголовочного файла Matrix\_graph.h.

Заголовочный файл Matrix\_graph.h содержит библиотеки, структуру и объявления некоторых функций.

Matrix\_graph.h

#pragma once

#include<iostream>

#include <stdio.h>

#include <string.h>

#include <stdlib.h>

#include <time.h>

#include <stdint.h>

#include <conio.h>

#include <windows.h>

#include <iostream>

#include <queue>

#include <ctime>

#pragma warning(disable : 4996)

#define INF 9999999

using namespace std;

typedef struct Graph {

int\*\* matrix;

int size;

}Graph;

int\*\* matrix(int size);

void generate\_matrix(Graph\* graph);

Graph\* init\_graph(int size);

void free\_graph(Graph\* graph);

void print\_graph(Graph\* graph, FILE\* stream);

void prim\_auto(Graph\* graph, FILE\* stream);

Graph\* init\_manually();

Graph\* read\_graph(const char\* fname);

В файле Matrix\_graph.cpp содержатся функции генерации матрицы, чтения из файла, освобождения матрицы, инициализации матрицы и сам алгоритм Прима.

Функция генерация матрицы.

void generate\_matrix(Graph\* graph)

{

for (int i = 0; i < graph->size; i++) {

for (int j = 0; j < graph->size; j++) {

if (j != i) {

graph->matrix[i][j] = rand() % 100;

graph->matrix[j][i] = graph->matrix[i][j];

}

else if (i == j) graph->matrix[i][j] = 0;

}

}

}

Функция чтения из файла.

Graph\* read\_graph(const char\* fname) {

errno\_t err;

FILE\* input;

err = fopen\_s(&input, fname, "r");

if (err == 0) {

int size = 0;

char next\_char = ' ';

int position = ftell(input);

do {

if (next\_char != ' ') {

size++;

}

next\_char = fgetc(input);

} while (next\_char != '\n');

fseek(input, position, SEEK\_SET);

Graph\* graph = init\_graph(size);

for (int i = 0; i < graph->size; i++) {

for (int j = 0; j < graph->size; j++) {

fscanf(input, "%d", &graph->matrix[i][j]);

}

}

return graph;

}

else {

printf("The file '%s' was not opened\n", fname);

return NULL;

}

}

Функция вывода матрицы

void print\_graph(Graph\* graph,FILE\* stream)

{

for (int i = 0; i < graph->size; i++) {

fprintf(stream,"%3d|", i);

for (int j = 0; j < graph->size; j++) {

fprintf(stream,"%3d ",graph->matrix[i][j]);

}

fprintf(stream,"\n");

}

}

Работы программы начинается с запроса генерация матрицы. Если пользователь выбрал пункт меню под номером один, то ему предложат ввести размер матрицы, затем сгенерируют матрицу со случайными значениями заданного размера. Если выбрать пункт два, то пользователь загрузит матрицу из файла, где он сам задаёт размерность и значения матрицы.

Courses.cpp

int main() {

int d;

printf("\n ---> 1.Generate matrix");

printf("\n ---> 2.Generate a matrix from file");

printf("\n ---> 0.Exit");

d = \_getch();

if (!d) { return 0; }

Graph\* graph;

FILE\* output = fopen("output.txt", "w");

if(d == '1') {

graph = init\_manually();

}

else{

graph = read\_graph("input.txt");

}

После вывода результатов и матрицы в консоль, пользователю предлагается выбор сохранить результаты или нет.

printf("\n ---> 1.Save result");

printf("\n ---> 2.Without saving");

d = \_getch();

if (d == '1') {

prim\_auto(graph, output);

}

if (d == '2') {

prim\_auto(graph, NULL);

}

return 0;

}

Последний пункт “Выход”, завершает работу программы.

Ниже можно увидеть оформление меню и его последующие изменения, после выбора соответствующих пунктов

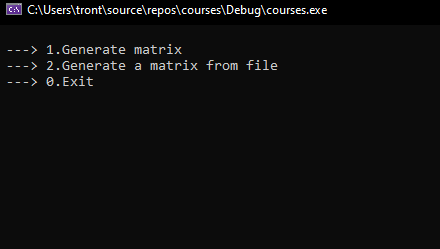


Рисунок 7-Меню

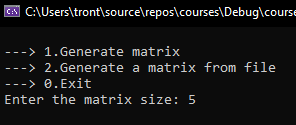


Рисунок 8-Ввод размера матрицы при выборе пункта 1

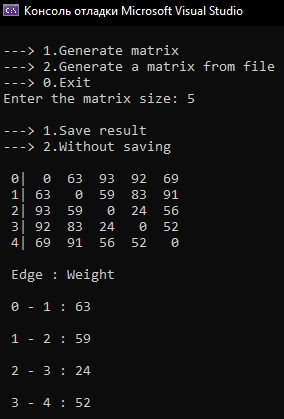


Рисунок 9- Результат программы без сохранения

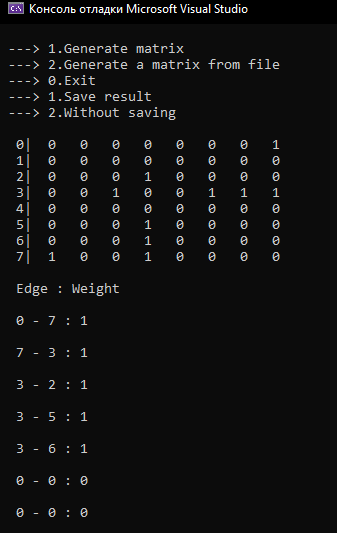


Рисунок 10 -Результат программы при выборе пункта 2 с сохранением в файл

“Save result” сохраняет результат работы программы в файл.

Graph\* graph;

FILE\* output = fopen("output.txt", "w");

if(d == '1') {

graph = init\_manually();

}

else{

graph = read\_graph("input.txt");

}

printf("\n ---> 1.Save result");

printf("\n ---> 2.Without saving");

d = \_getch();

if (d == '1') {

prim\_auto(graph, output);

}

if (d == '2') {

prim\_auto(graph, NULL);

}

return 0;

}



Рисунок 11- Результат работы программы сохраненный в файл

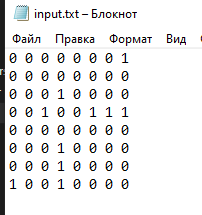


Рисунок 12- Ввод матрицы из файла

# **5 Тестирование**

Среда разработки MicrosoftVisualStudio 2010 предоставляет все средства, необходимые при разработке и отладке многомодульной программы.

Тестирование проводилось в рабочем порядке, в процессе разработки, после завершения написания программы. В ходе тестирования было выявлено и исправлено множество проблем, связанных с вводом данных, изменением дизайна выводимых данных, алгоритмом программы, взаимодействием функций.

Таблица 1

|  |  |  |
| --- | --- | --- |
| Описание теста | Ожидаемый результат | Полученный результат |
| Запуск программы | Вывод меню в консоль | Верно |
| Генерация матрицы случайным образом и ручным ввод размера | Вывод сообщения о вводе размера матрицы  И выводе результатов программы в консоль | Верно |
| Генерация матрицы из файла | Вывод сообщения о выборе генерации и выводе результатов программы | Верно |
| Сохранение в файл | Создание файла, содержащего результаты программы | Верно |
| Выход из программы | Завершение работы программы | Верно |
| Проверка на наличие изолированных вершин | Нет минимального остовного дерева | Верно |

Начало программы, в консоль выводиться меню.

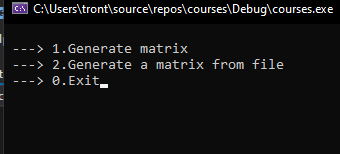


Рисунок 13- Вывод меню

Если выбрать в меню пункт 1, то выведется сообщение, что нужно ввести размер матрицы. Далее программы предлагает выбор сохранить результаты в файл или нет, выбираем пункт 2 и выводиться матрица с результатами без сохранения в файл

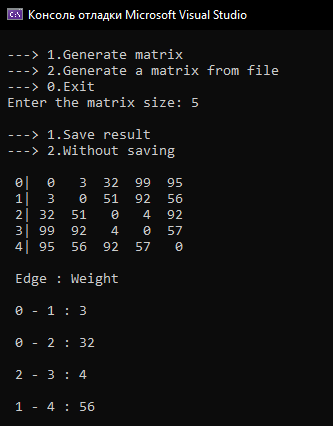


Рисунок 14- Генерация матрицы со случайными значениями и заданным размером матрицы с клавиатуры, вывод без сохранения в файл)

Выбрав пункт 2 матрица будет загружена из файла, далее программа спрашивает сохранить в файл или нет, выбираем пункт 2, выводиться матрица с результатами с сохранением в файл

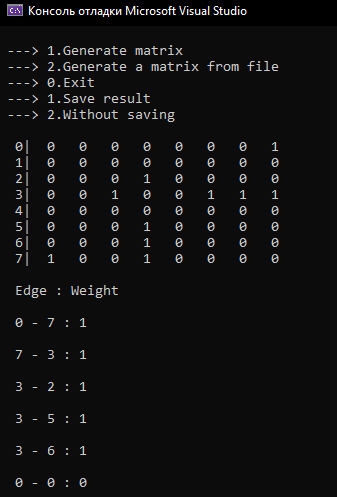


Рисунок 15- Генерация матрицы из файла с сохранением результатов в файл

Если в меню выбрать пункт 0, то выполниться предварительный выход из программы.

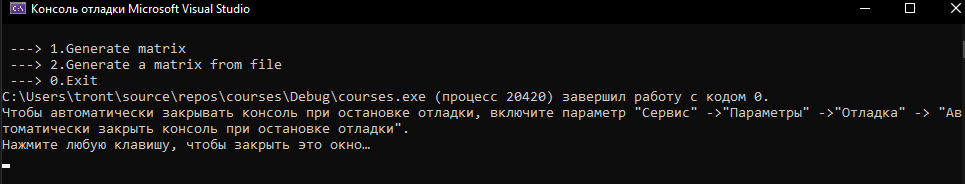


Рисунок 16 -Предварительный выход из программы

Результаты программы сохраненные в файл.



Рисунок 17 -Результат работы сохраненный в файл

Если матрица была сгенерирована с изолированными вершинами, то программа выдаёт, что минимального остовного дерева не существует и завершает работу

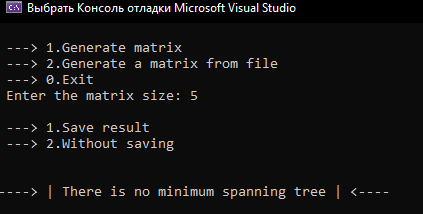


Рисунок 18 -Проверка сязность

# **Заключение**

Таким образом, в процессе создания данного проекта разработана программа, реализующая алгоритм Прима для поиска минимального пути MicrosoftVisualStudio 2019.

При выполнении данной курсовой работы были получены навыки разработки программ и освоены приемы создания матриц смежностей. Приобретены навыки по осуществлению алгоритма Прима. Углублены знания языка программирования Cи

Прима подпадает под класс алгоритмов, называемых [жадными алгоритмами,](https://www.programiz.com/dsa/greedy-algorithm) которые находят локальный оптимум в надежде найти глобальный оптимум.

# **Список литературы**

1. Керниган Б. Ритчи Д. Язык программирования С. 1985 г.
2. К. Джамса. Учимся программировать на языке C++. 1997 г
3. В. Г. Давыдов. Программирование и основы алгоритмизации. 2003
4. В.В. Подбельский, С.С.Фомин. Программирование на языке Си. М .: ФиС, 1999, 600 с.

# **Приложение А**

# **Листинг**

**Matrix\_graph.h**

#pragma once

#include<iostream>

#include <stdio.h>

#include <string.h>

#include <stdlib.h>

#include <time.h>

#include <stdint.h>

#include <conio.h>

#include <windows.h>

#include <iostream>

#include <queue>

#include <ctime>

#pragma warning(disable : 4996)

#define INF 9999999

using namespace std;

typedef struct Graph {

int\*\* matrix;

int size;

}Graph;

int izo\_matrix(Graph\* graph);

int\*\* matrix(int size);

void generate\_matrix(Graph\* graph);

Graph\* init\_graph(int size);

void free\_graph(Graph\* graph);

void print\_graph(Graph\* graph, FILE\* stream);

void prim\_auto(Graph\* graph, FILE\* stream);

Graph\* init\_manually();

Graph\* read\_graph(const char\* fname);

**Matrix\_graph.cpp**

#include "Matrix\_graph.h"

Graph\* init\_manually() {

int V;

printf("\n Enter the matrix size: ");

scanf("%d", &V);

Graph\* graph = init\_graph(V);

graph->size = V;

return graph;

}

void prim\_auto(Graph\* graph, FILE\* stream) {

int no\_edge;

int\* selected = (int\*)calloc(graph->size, sizeof(int));

no\_edge = 0;

selected[0] = true;

int x; // Номер Строки

int y; // Номер Столбца

printf(" \n\n");

print\_graph(graph,stdout);

if (stream){ print\_graph(graph, stream); }

printf("\n Edge : Weight\n");

if (stream){ fprintf(stream,"\n Edge : Weight\n"); }

while (no\_edge < graph->size- 1) {

int min = INF;

x = 0;

y = 0;

for (int i = 0; i < graph->size; i++) {

if (selected[i]) {

for (int j = 0; j < graph->size; j++) {

if (!selected[j] && graph->matrix[i][j]) {

if (min > graph->matrix[i][j]) {

min = graph->matrix[i][j];

x = i;

y = j;

}

}

}

}

}

printf("\n %d - %d : %d\n", x, y, graph->matrix[x][y]);

if (stream) { fprintf(stream, "\n %d - %d : %d\n", x, y, graph->matrix[x][y]); }

selected[y] = true;

no\_edge++;

}

}

void generate\_matrix(Graph\* graph)

{

for (int i = 0; i < graph->size; i++) {

for (int j = 0; j < graph->size; j++) {

if (j != i) {

graph->matrix[i][j] = rand() % 100;

graph->matrix[j][i] = graph->matrix[i][j];

}

else if (i == j) graph->matrix[i][j] = 0;

}

}

}

int izo\_matrix(Graph\* graph) {

bool k = false;

int\* matri = (int\*)malloc(graph->size \* sizeof(int));

for (int i = 0; i < graph->size; i++) {

matri[i] = 0;

}

for (int i = 0; i < graph->size; i++) {

for (int j = 0; j < graph->size; j++) {

matri[i] = matri[i] + graph->matrix[i][j];

}

}

for (int i = 0; i < graph->size; i++) {

if (matri[i] == 0) {

k = true;

}

}

return k;

}

int\*\* matrix(int V )

{

int\*\* matrix = (int\*\*)malloc(V \* sizeof(int\*));

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

{

matrix[i] = (int\*)malloc(V \* sizeof(int));

}

return matrix;

}

Graph\* init\_graph(int V)

{

Graph\* graph = (Graph\*)malloc(sizeof(Graph));

graph->matrix = matrix(V);

graph->size = V;

srand(time(NULL));

generate\_matrix(graph);

return graph;

};

void free\_graph(Graph\* graph)

{

for (int i = 0; i < graph->size; i++)

{

free(graph->matrix[i]);

}

free(graph->matrix);

free(graph);

}

void print\_graph(Graph\* graph,FILE\* stream)

{

for (int i = 0; i < graph->size; i++) {

fprintf(stream,"%3d|", i);

for (int j = 0; j < graph->size; j++) {

fprintf(stream,"%3d ",graph->matrix[i][j]);

}

fprintf(stream,"\n");

}

}

Graph\* read\_graph(const char\* fname) {

errno\_t err;

FILE\* input;

err = fopen\_s(&input, fname, "r");

if (err == 0) {

int size = 0;

char next\_char = ' ';

int position = ftell(input);

do {

if (next\_char != ' ') {

size++;

}

next\_char = fgetc(input);

} while (next\_char != '\n');

fseek(input, position, SEEK\_SET);

Graph\* graph = init\_graph(size);

for (int i = 0; i < graph->size; i++) {

for (int j = 0; j < graph->size; j++) {

fscanf(input, "%d", &graph->matrix[i][j]);

}

}

return graph;

}

else {

printf("The file '%s' was not opened\n", fname);

return NULL;

}

}

**courses.cpp**

#include "Matrix\_graph.h"

int main() {

int d;

printf("\n ---> 1.Generate matrix");

printf("\n ---> 2.Generate a matrix from file");

printf("\n ---> 0.Exit");

d = \_getch();

if (!d) { return 0; }

Graph\* graph;

FILE\* output = fopen("output.txt", "w");

if(d == '1') {

graph = init\_manually();

}

else{

graph = read\_graph("input.txt");

}

if (d == '0') {

exit(0);

}

k = izo\_matrix(graph);

if (d == '1') {

if (k) {

print\_graph(graph, stdout);

}

else

prim\_auto(graph, output);

}

if (d == '2') {

printf("\n");

if (k) {

print\_graph(graph, stdout);

printf("\n,There is no minimum spanning tree ");

}

else

prim\_auto(graph, NULL);

}

return 0;

}