Министерство образования Российской Федерации

Пензенский государственный университет

Кафедра «Вычислительная техника»

**ПОЯСНИТЕЛЬНАЯ ЗАПИСКА**

К курсовому проектированию

по курсу «Логика и основы алгоритмизации в инженерных задачах»

на тему «Реализация алгоритма поиска независимых множеств ребер графа»

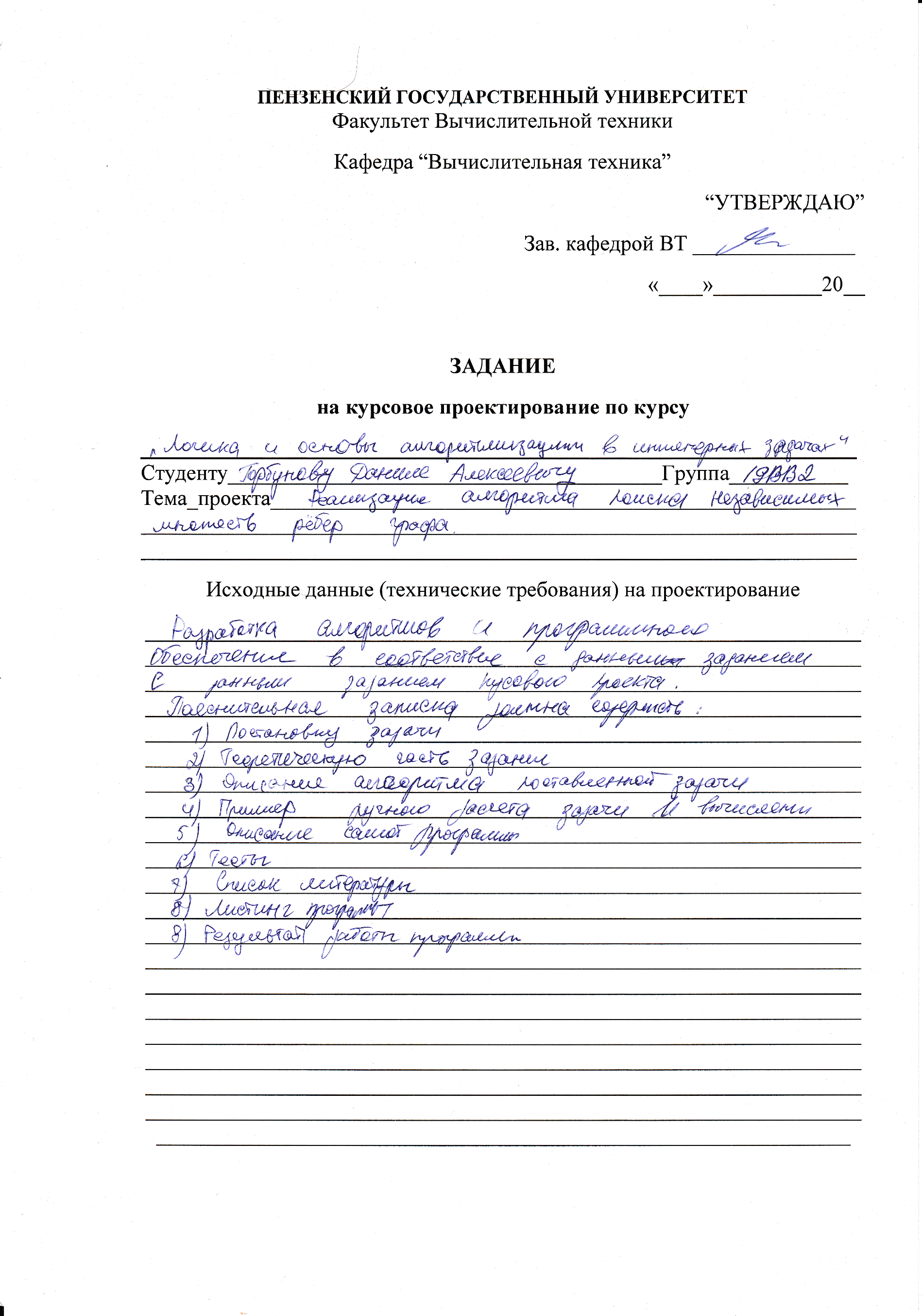
Выполнил:

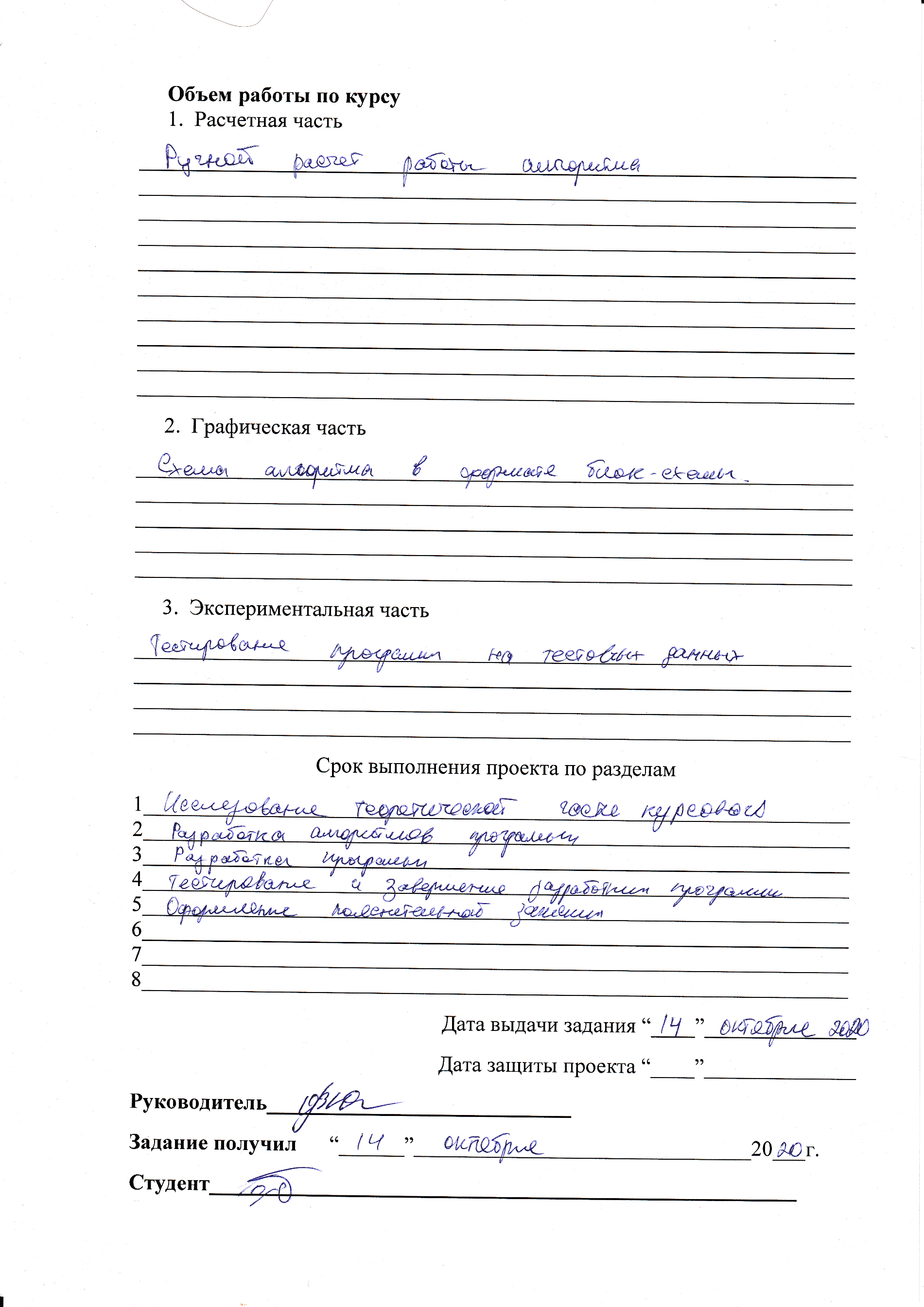
студент группы 19ВВ2

Горбунов Д. А.

Приняла:

Пенза 2020





Оглавление

[Реферат 5](#_Toc59311913)

[Введение 6](#_Toc59311914)

[Постановка задачи 7](#_Toc59311915)

[Теоретическая часть задания 8](#_Toc59311916)

[Описание алгоритма программы 9](#_Toc59311917)

[Описание программы 13](#_Toc59311918)

[Тестирование 16](#_Toc59311919)

[Ручной расчет задачи 21](#_Toc59311920)

[Заключение 23](#_Toc59311921)

[Список литературы 24](#_Toc59311922)

[Приложение А. 25](#_Toc59311923)

[Листинг программы. 25](#_Toc59311924)

# Реферат

Отчет 29 стр., 21 рисунок

ГРАФ, АЛГОРИТМ ПОИСКА НЕЗАВИСИМЫХ МНОЖЕСТВ РЕБЕР ГРАФА

Цель исследования – разработка программы, способная находить независимые множества ребер графа.

В работе рассмотрены граф, который выражен при помощи матрицы инцидентности, основанная на матрице смежности. Установлено, что при помощи матрицы инцидентности удобно и легко реализовать алгоритм поиска независимых ребер.

# Введение

Алгоритм поиска независимых множеств ребер графа позволяет найти все независимые множества неориентированного невзвешенного графа. Он построчно обрабатывает каждое ребро и на основании полученных данных, программа выводит на экран пользователю результат.

В качестве среды разработки мною была выбрана среда Microsoft Visual Studio 2019, язык программирования – Си.

Целью данной курсовой работы является разработка программы на языке Си, который является широко используемым на данный момент. Именно с его помощью в данном курсовом проекте реализуется алгоритм поиска независимых множества ребер графа на основе матрицы инцидентности.

# Постановка задачи

Требуется разработать программу, которая выведет на экран все независимые ребра графа.

Исходный граф в программе должен задаваться матрицей смежности, причем граф должен быть невзвешенным для правильной работы программы. Затем эта матрица внутри программы преобразуется в матрицу инцидентности, с которой намного проще работать, если речь идет про ребра графа. Обе матрицы должны выводиться на экран пользователя. Изначальные данные тоже вводит пользователь. В программе предусмотрен ввод размера графа, автоматическая генерация элементов, а также ручной ввод с «защитой от дурака», которая не позволяет вводить некорректные данные. Затем программа, после работы алгоритма, выводит все независимые ребра графа на экран. Необходимо предусмотреть различные исходы поиска, что бы программы не выдавала ошибок и работала правильно. Устройство ввода - клавиатура. Устройство вывода - монитор.

Задания выполняются в соответствии с вариантом №20.

# Теоретическая часть задания

Граф *G* (рисунок 1) задается множеством вершин X1, X2, X3 и так далее и множеством ребер, которые соединяют между собой определенные вершины. Ребра из множества A не ориентированы. Такой граф называется неориентированным.

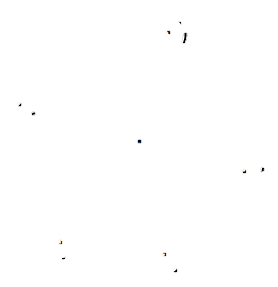


Рисунок 1 – Пример графа

При представлении графа матрицей смежности информация о ребрах хранится в квадратной матрице, где присутствие пути от одной вершиной обозначается единицей, иначе нулем. Матрица инцидентностей представляет матрицу, с количеством строк равному количеству вершин, а столбцов количеством ребер. Ненулевое значение в ячейке матрицы указывает связь между вершиной и ребром (их инцидентность).

Независимые ребра представляют из себя такие ребра, которые не связаны между собой общей вершиной (рисунок 2).

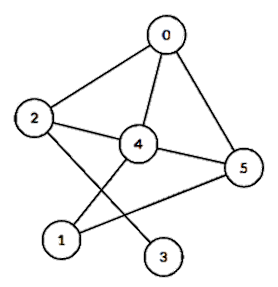




Рисунок 2 - Пример независимых ребер

# Описание алгоритма программы

Для программной реализации понадобится два динамических массива: graph(int) – для матрицы смежности и inc(int) – для матрицы инцидентности. Имеется граф graph=(V, E). После ввода информации по размеру графа и их элементам, на экран выводится матрица смежности. Матрица смежности представляет собой квадратную матрицу, состоящую из 1 и 0. Выводится она на экран при помощи двух циклов *for:* один отвечает за строки матрицы, а второй на столбцы. Так происходит ввод автоматический ввод элементов:

srand(time(NULL));

for (i = 0; i < m; ++i) {

for (j = i; j < m; ++j) {

graph[i][j] = graph[j][i] = rand() % 2;

}

graph[i][i] = 0; //обнуление вершины

}

В этом случае используется функция *rand*, которая позволяет случайно генерировать элемент для последующей записи. Что бы при перезапуске генерация происходила снова, есть строка кода srand(time(NULL));, которая работает на основе текущего времени, которая выступает в роли сида в генерации. Что бы не было петель, в графе предусмотрено обнуление главной диагонали.

Так происходит ручной ввод элементов:

printf("Вводите элементы матрицы (0 и 1):\n");

for (i = 0; i < m; ++i) {

for (j = i; j < m; ++j) {

if (i != j) {

ret:

printf("a[%d][%d] = ", i, j);

scanf\_s("%d", &graph[i][j]);

if (graph[i][j] > 1 || graph[i][j] < 0) {

printf("\nНекорректный ввод!\n");

goto ret;

}

}

}

graph[i][i] = 0;

}

for (i = 0; i < n; i++) //отзеркаливание матрицы

{

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

{

graph[j][i] = graph[i][j];

}

}

Ручной ввода реализован максимально просто для пользователя. Он дает возможность ввода элементов до главной диагонали, а также, минуя саму диагональ. Для заполнения остальных элементов предусмотрена функция отзеркаливания матрицы. Так же реализована функция запрета на ввод чисел, отличных от 0 и 1 для корректной работы алгоритма.

Так происходит вывод матрицы смежности на экран:

for (i = 0; i < m; ++i)

{

printf("[%d] ", i + 1);

for (j = 0; j < m; ++j)

printf(" %d ", graph[i][j]);

printf("\n");

}

Реализация вывода матрицы на экран основана на построчном выводе элементов.

Создание матрицы инцидентности происходит на основе ранее созданной матрице смежности. Для начала алгоритм ведет подсчет количества самих ребер графа. Происходит это при помощи двух дополнительных переменных *reb* и *ch*. Первая нужна как раз для подсчета количества. Вторая, что бы в алгоритме не вылезти за рамки главной диагонали. Все происходит, пока переменная *ch* не сравнится с количеством строк. Для подсчета создается два цикла, первый для строк, второй для столбцов. Сам поиск реализован во втором. Цикл начинается с j = 0, с динамическим шагом *ch*. Если все элементы строки графа не равны нулю, то добавляется ребро и переменная *ch* увеличивается на один. Говоря просто, алгоритм ищет количество единиц до главной диагонали – это количество и есть количество ребер графа.

while (ch != n)

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

{

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

if (graph[i][j] != 0)

reb++;

ch++;

}

Затем создается динамический массив *inc*, который в будущем будет матрицей смежности. Матрица создается так:

for (int i = 0; i < n; i++) //создание матрицы инцидентности

{

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

if (graph[i][j] != 0)

{

inc[i][reb] = 1; //начало

inc[j][reb] = 1; //конец

reb++;

}

}

А выводится вот так:

printf("\n ");

for (int i = 0; i < ch1; i++) {

printf("[%d] ", i + 1);

}

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

{

printf("\n[%d]", i + 1);

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

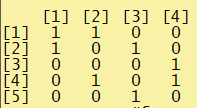
{

printf(" %d ", inc[i][j]);

}

}

Приступим к алгоритму поиска независимых ребер. Исходя из многочисленных попыток найти зависимость, при которой в матрице инцидентности два ребра будут независимые, я пришел к выводу, что два ребра независимы если единицы в матрице двух столбцов этих ребер не будут пересекаться (рисунок 3).



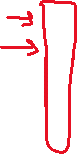
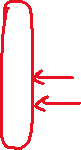
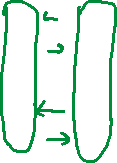
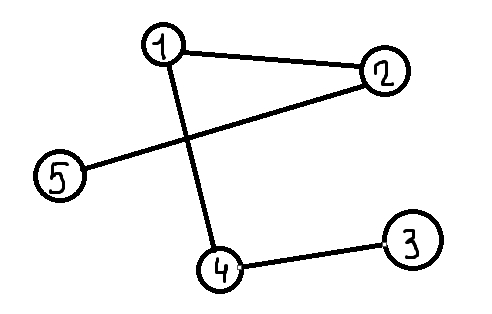


Рисунок 3 - Независимые вершины на матрице инцидентности

Если визуализировать эту матрицу в виде графа (рисунок 4), то получится что эти ребра действительно независимые.



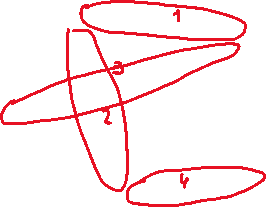


Рисунок 4 - Граф с независимыми вершинами

Так же в этом графе есть независимые 3 и 4 ребра.

Этот алгоритм основан на сравнении двух столбцов (ребер). Сначала берется самое первое ребро для сравнения с остальными. В алгоритме используется специальная переменная *sr*, которая выступает в роли флага, который активируется если в двух столбцах одной строки будет найдено две единицы. В случае активации флага, сравниваемое ребро будет отброшено, и к сравнению приступает другое. Если флаг был не активирован, то выводится первое ребро и ребро, с которым мы его сравнивали. И так дальше, пока не кончатся остальные ребра. Затем берется второе ребро и происходят те же действия, описанные выше. Код алгоритма:

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

bool sr = false;

for (int j = i + 1; j < reb; j++) {

sr = false;

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

if (inc[k][i] == 1 && inc[k][j] == 1) {

sr = true;

}

}

if (sr == false) { printf("%d - %d\n", i + 1, j + 1); }

}

}

# Описание программы

Для написания данной программы использован язык программирования Си. Язык программирования Си - универсальный язык программирования, который завоевал особую популярность у программистов, благодаря сочетанию возможностей языков программирования высокого и низкого уровней.

Проект был создан в виде консольного приложения Win32 (Visual C++).

Работа программы начинается с запроса – пользователю дается возможность ввести количество вершин в графе. Затем дается два способа ее заполнения: ручной и автоматический. После получения необходимой информации, программа выводит на экран матрицу смежности, инцидентности, а затем и сами независимые ребра.

setlocale(LC\_ALL, "Rus");

printf("Размерность: ");

scanf\_s("%d", &m);

n = m;

graph = new int\* [m];

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

graph[i] = new int[m];

}

//генерация матрицы

int t;

printf("1 - Автоматическая генерация\n2 - Ручной ввод\n> ");

scanf\_s("%d", &t);

if (t == 1) {

srand(time(NULL));

for (i = 0; i < m; ++i) {

for (j = i; j < m; ++j) {

graph[i][j] = graph[j][i] = rand() % 2;

}

graph[i][i] = 0; //обнуление вершины

}

}

else {

printf("Вводите элементы матрицы (0 и 1):\n");

for (i = 0; i < m; ++i) {

for (j = i; j < m; ++j) {

if (i != j) {

ret:

printf("a[%d][%d] = ", i, j);

scanf\_s("%d", &graph[i][j]);

if (graph[i][j] > 1 || graph[i][j] < 0) {

printf("\nНекорректный ввод!\n");

goto ret;

}

}

}

graph[i][i] = 0;

}

Затем выводится матрица смежности:

for (i = 0; i < m; ++i)

{

printf("[%d] ", i + 1);

for (j = 0; j < m; ++j)

printf(" %d ", graph[i][j]);

printf("\n");

}

И матрица инцидентности:

printf("\n ");

for (int i = 0; i < ch1; i++) {

printf("[%d] ", i + 1);

}

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

{

printf("\n[%d]", i + 1);

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

{

printf(" %d ", inc[i][j]);

}

}

На рисунке 5 можно увидеть оформление начального запроса и дальнейшие действия.

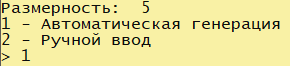


Рисунок 5 - Запрос данных от пользователя

Случайная генерация матрицы (рисунок 6).

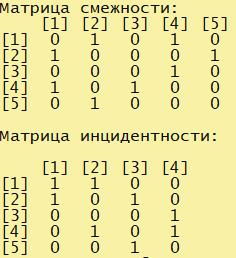


Рисунок 6 – Автоматический ввод

Ручной ввод матрицы (рисунок 7).

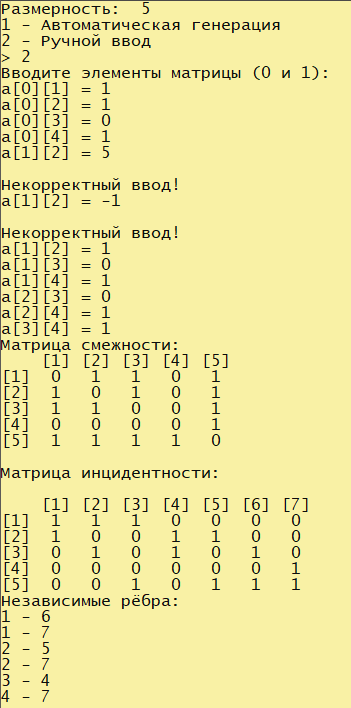


Рисунок 7 - Ручной ввод

Так же здесь продемонстрирована функция некорректного ввода.

Далее после вывода всех матриц выводятся все независимые вершины (на примере результата работы программы на рисунке 7) (рисунок 8).

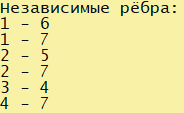


Рисунок 8 - Независимые вершины

# Тестирование

Среда разработки Microsoft Visual Studio 2019 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.

Ниже продемонстрирован результат тестирования программы при вводе пользователем различных количеств вершин и варианта ввода элементов.



Рисунок 9 - Ручной ввод с примером некорректного ввода (5 вершин)

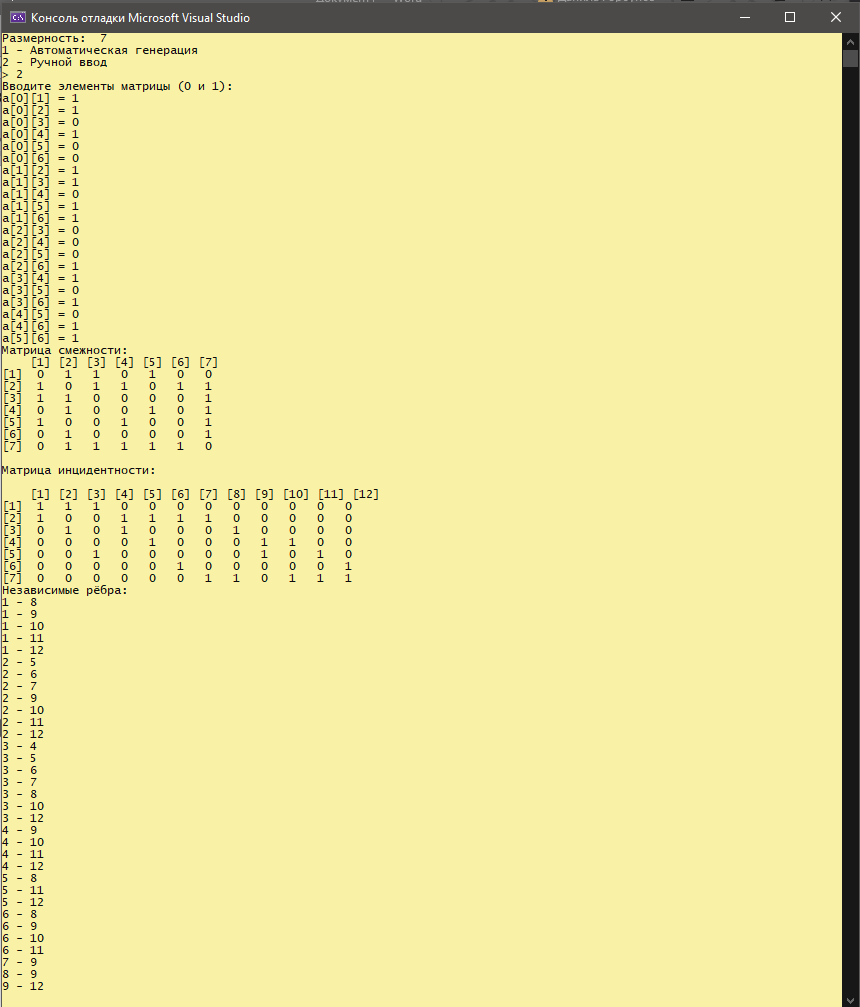


Рисунок 10 - Ручной ввод (7 вершин)



Рисунок 11 - Автоматический ввод (6 вершин)

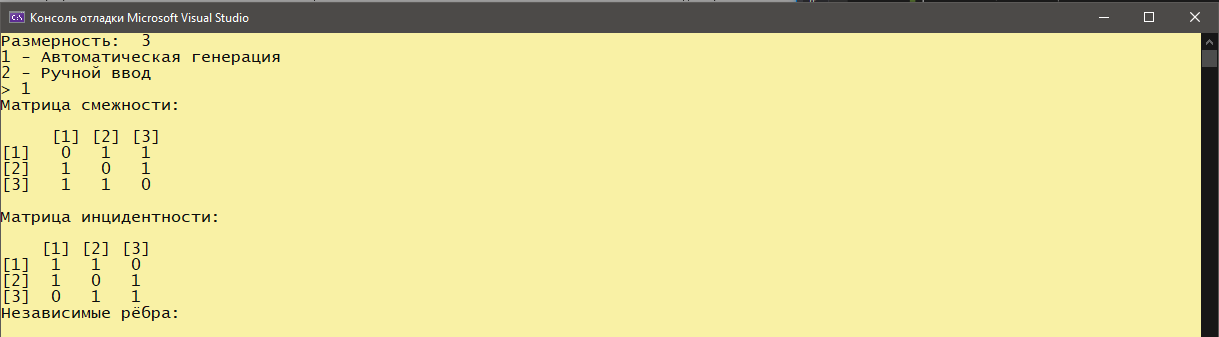


Рисунок 12 - Автоматический ввод, где не найдено независимых ребер (3 вершины)

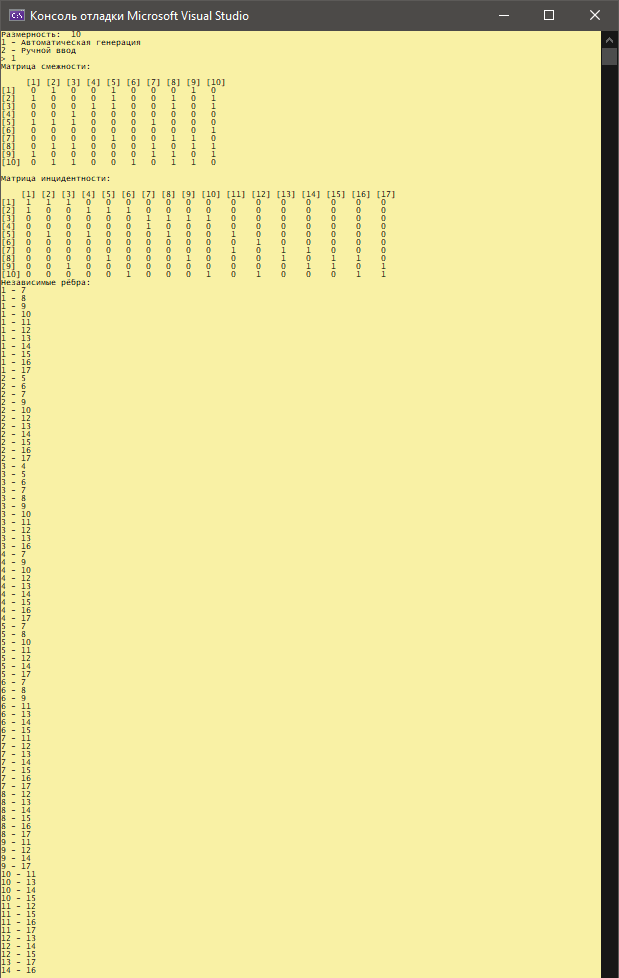


Рисунок 13 - Автоматический ввод (10 вершин)

Таблица 1 - Описание поведения программы при тестировании

|  |  |  |
| --- | --- | --- |
| Описание теста | Ожидаемый результат | Полученный результат |
| Запуск программы | Вывод сообщение о вводе количества вершин в графе | Верно |
| Выбор способа заполнения графа | Вывод сообщения о выборе: сгенерировать матрицу или ввести вручную | Верно |
| Ручной ввод | Вывод индексов элементов для ввода матрицы и возможность ввода, а после ее вывода | Верно |
| Ошибка при вводе | Вывод сообщение о некорректном вводе элемента матрицы и возможность повторного ввода | Верно |
| Автоматическая генерация матрицы | Моментальный вывод матрицы на экран | Верно |
| Генерация матриц различных размеров | Матрицы корректно выведется на экран, элементы не «уползут» куда не надо | Верно |

В результате тестирования было выявлено, что программа успешно проверяет данные на соответствие необходимым требованиям.

# Ручной расчет задачи

Проведем расчет на примере матрицы размерности 5 (рисунок 9).

Для начала зарисуем сгенерируемую в этом случае матрицу для дальнейшего анализа (рисунок 14).

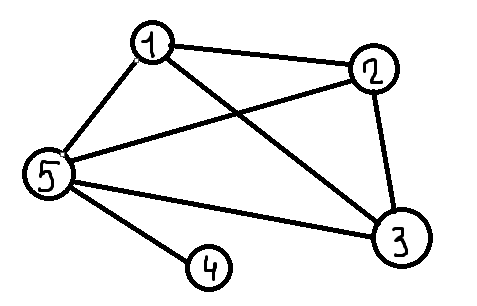
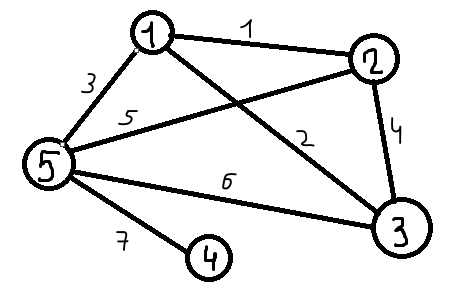
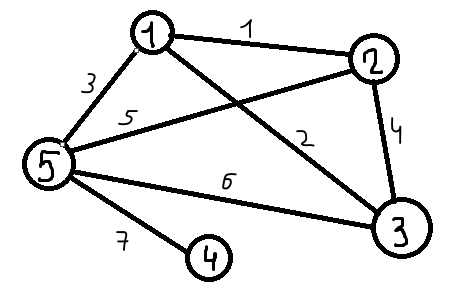
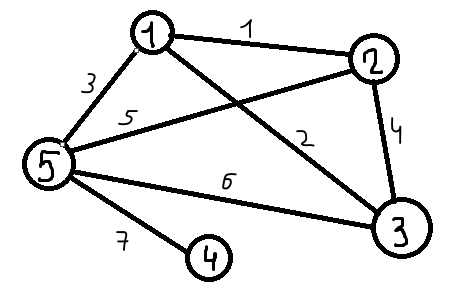
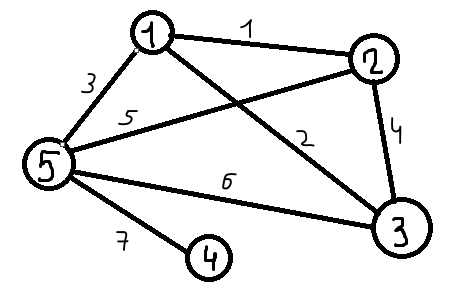
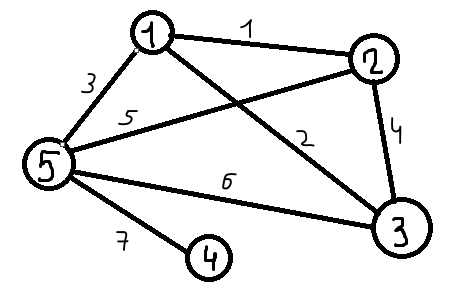
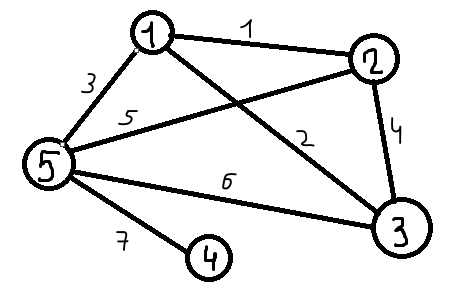




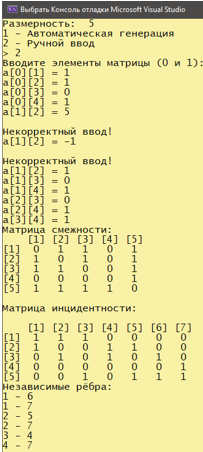
Рисунок 14 - Граф на основе рис. 9

На данном графе можно найти целых 6 независимых вершин:





Проверим с результатом работы программы на рисунке:





Таким образом, можно сделать вывод, что программа работает верно!

# Заключение

Таким образом, в процессе создания данного проекта разработана программа, реализующая алгоритм поиска независимых множеств ребер графа в Microsoft Visual Studio 2019.

При выполнении данной курсовой работы были получены навыки разработки программ и освоены приемы создания матриц смежностей, основанных на теории орграфов. Приобретены навыки по осуществлению алгоритма поиска в глубину. Углублены знания языка программирования Cи.

Недостатком разработанной программы является примитивный пользовательский интерфейс. Потому что программа работает в консольном режиме, не добавляющем к сложности языка сложность программного оконного интерфейса.

Программа имеет небольшой, но достаточный для использования функционал возможностей.

# Список литературы

1. Алексеев В.В., Гаврилов Г.П., Сапоженко А.А. (ред.) Теория графов. Покрытия, укладки, турниры. Сборник переводов - М. : Мир, 1974.— 224 с.
2. Асельдеров З.М., Донец Г.А. Представление и восстановление графов - К.:Наукова Думка, 1991, 96 стр.
3. Березина Л. Ю. Графы и их применение: Пособие для учителей. — М.: Просвещение, 1979. — 143 с. с ил.
4. Алгоритмы и программы. Язык С++. Е. Конова, Г. Поллак
5. Демидович Е.М. Основы алгоритмизации и программирования. Язык СИ : учебн. Пособие. – СПб.: БХВ-Петербург, 2006.
6. Дэвис С. С++ для «чайников». – К. : Диалектика, 2005.

# Приложение А.

# Листинг программы.

#include <stdio.h>

#include <Windows.h>

#include <conio.h>

#include <locale.h>

#include <time.h>

using namespace std;

int i, j, m, n, v;

int\*\* graph;

int main()

{

setlocale(LC\_ALL, "Rus");

printf("Размерность: ");

scanf\_s("%d", &m);

n = m;

graph = new int\* [m];

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

graph[i] = new int[m];

}

//генерация матрицы

int t;

printf("1 - Автоматическая генерация\n2 - Ручной ввод\n> ");

scanf\_s("%d", &t);

if (t == 1) {

srand(time(NULL));

for (i = 0; i < m; ++i) {

for (j = i; j < m; ++j) {

graph[i][j] = graph[j][i] = rand() % 2;

}

graph[i][i] = 0; //обнуление вершины

}

}

else {

printf("Вводите элементы матрицы (0 и 1):\n");

for (i = 0; i < m; ++i) {

for (j = i; j < m; ++j) {

if (i != j) {

ret:

printf("a[%d][%d] = ", i, j);

scanf\_s("%d", &graph[i][j]);

if (graph[i][j] > 1 || graph[i][j] < 0) {

printf("\nНекорректный ввод!\n");

goto ret;

}

}

}

graph[i][i] = 0;

}

}

for (i = 0; i < n; i++) //отзеркаливание матрицы

{

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

{

graph[j][i] = graph[i][j];

}

}

printf("Матрица смежности: \n");

//нумерование столбцов

printf("\n ");

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

printf("[%d] ", i + 1);

}

printf("\n");

//нумерование строк

for (i = 0; i < m; ++i)

{

printf("[%d] ", i + 1);

for (j = 0; j < m; ++j) {

if ((i + 1) < 10)

{

if (j < 10)

printf("%3d ", graph[i][j]);

else

printf(" %3d ", graph[i][j]);

}

else if ((i + 1) < 100)

{

if (j == 0)

printf(" %d", graph[i][j]);

else if (j < 10)

printf(" %d", graph[i][j]);

else

printf(" %d", graph[i][j]);

}

}

printf("\n");

}

printf("\nМатрица инцидентности: \n");

int reb = 0;

int ch = 0;

while (ch != n)

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

{

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

if (graph[i][j] != 0)

reb++;

ch++;

}

/////

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

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

if (graph[i][j] != 0)

graph[j][i] = 0;

int\*\* inc;

inc = (int\*\*)malloc(n \* sizeof(int\*));

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

{

inc[i] = (int\*)malloc(reb \* sizeof(int));

}

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

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

inc[i][j] = 0;

int ch1 = reb;

reb = 0;

for (int i = 0; i < n; i++) //создание матрицы инцидентности

{

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

if (graph[i][j] != 0)

{

inc[i][reb] = 1; //начало

inc[j][reb] = 1; //конец

reb++;

}

}

printf("\n ");

for (int i = 0; i < ch1; i++) {

printf("[%d] ", i + 1);

}

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

{

printf("\n[%d]", i + 1);

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

{

if ((i + 1) < 10)

{

if (j < 10)

printf("%3d ", inc[i][j]);

else

printf(" %3d ", inc[i][j]);

}

else if ((i + 1) < 100)

{

if (j == 0)

printf(" %d", inc[i][j]);

else if (j < 10)

printf(" %d", inc[i][j]);

else

printf(" %d", inc[i][j]);

}

}

}

printf("\n");

printf("Независимые рёбра:\n");

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

bool sr = false;

for (int j = i + 1; j < reb; j++) {

sr = false;

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

if (inc[k][i] == 1 && inc[k][j] == 1) {

sr = true;

}

}

if (sr == false) { printf("%d - %d\n", i + 1, j + 1); }

}

}

return 0;

}