Министерство науки и высшего 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 22ВВС1

Разин Д.С

Принял:

Акифьев И.В

Пенза 2023

Оглавление

[Реферат 5](#_Toc59606580)

[Введение 6](#_Toc59606581)

[1 Постановка задачи 7](#_Toc59606582)

[2 Теоретическая часть задания 8](#_Toc59606583)

[3 Описание алгоритма программы 9](#_Toc59606584)

[4 Описание программы 13](#_Toc59606585)

[5 Тестирование 17](#_Toc59606586)

[6 Ручной расчет задачи 22](#_Toc59606587)

[Заключение 24](#_Toc59606588)

[Список литературы 25](#_Toc59606589)

[Приложение А. 26](#_Toc59606590)

# Реферат

Отчет 29 стр., 21 рисунок

ГРАФ, АЛГОРИТМ ПОИСКА НЕЗАВИСИМЫХ МНОЖЕСТВ РЕБЕР ГРАФА

Цель исследования – разработка программы, способная с помощью алгоритма поиска находить независимые множества ребер графа.

В работе рассмотрен 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

Роль теории графов в математическом моделировании трудно переоценить. Модели на графах применяются в самых различных областях науки и техники: от социологии до компьютерных технологий. Это объясняется тем, что такие модели обладают высокой степенью наглядности и потому понятны и удобны в использовании.

Алгоритм поиска независимых множеств ребер графа позволяет найти все независимые множества неориентированного невзвешенного графа. Его функция заключается в том, чтобы он построчно обрабатывал каждое ребро и на основании полученных данных, программа выводила бы на экран пользователю результат.

В качестве среды разработки мною была выбрана среда Microsoft Visual Studio 2022, 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.

# 1 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. В программе предусмотрен ввод размера графа автоматической генераций элементов. Затем программа, после работы алгоритма, выводит все независимые ребра графа на экран, если она таких не найдет она также выведет на экран пользователю, что не нашла ни одного независимого ребра. Необходимо предусмотреть различные исходы поиска, что бы программы не выдавала ошибок и работала правильно. Устройство ввода - клавиатура. Устройство вывода - монитор.

# 2 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.

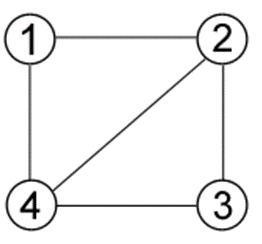


Рисунок 1 – Пример графа

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

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

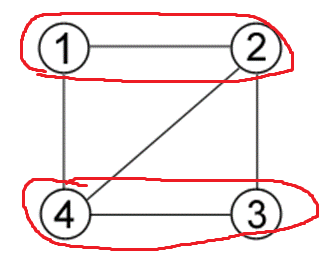


Рисунок 2 - Пример независимых ребер

# 3 Описание алгоритма программы

Для программной реализации понадобится два динамических массива: graph(int) – для матрицы смежности и inc(int) – для матрицы инцидентности. Имеется граф graph. 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 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

}

Здесь 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 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 инцидентности. 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).

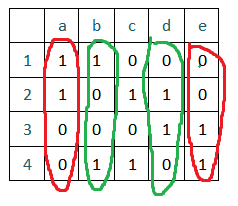




Рисунок 3 - Независимые вершины на матрице инцидентности

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

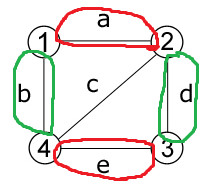


Рисунок 4 - Граф с независимыми вершинами

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:

counter = 1;

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

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) {

counter = 0;

printf("\n%d - %d\n", i + 1, j + 1);

}

}

}

if (counter == 1) {

printf("таких рёбер не встретилось");

}

# 4 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. После получения необходимой информации, программа выводит на экран матрицу смежности, инцидентности, а затем и информацию по независимым ребрам.

setlocale(LC\_ALL, "Rus");

srand(time(NULL));

printf("Создание матрицы смежности\n");

printf("Введите с клавиатуры желаемое количество вершин в матрице: ");

scanf("%d", &m);

while (m == 0 || m < 0) {

printf("\nОшибка!!!\n");

printf("Введите с клавиатуры количество вершин в матрице (больше нуля): ");

scanf("%d", &m);

}

n = m;

graph = new int\* [m];

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

graph[i] = new int[m];

}

//генерация матрицыы

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

}

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

{

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

{

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

}

}

Затем выводится матрица смежности:

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");

}

И матрица инцидентности:

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");

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

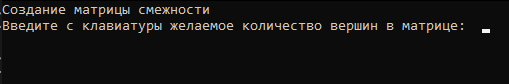


Рисунок 5 - Запрос данных от пользователя

После ввода количества вершин выводится матрица смежности и инцидентности (рисунок 6).

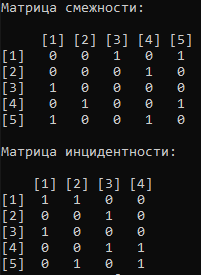


Рисунок 6 – Вывод матриц

Далее после вывода всех матриц выводятся все независимые вершины (рисунок 7).

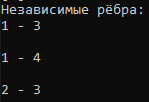


Рисунок 7 - Независимые вершины

После окончания работы программы пользователю будет предложено начать программу заново или закончить её (рисунок 8).



Рисунок 8 – Запрос на продолжение программы

Если пользовать нажмет на клавиатуре 1, то тогда все старые значение сотрутся и программа начнётся заново. Если пользователь нажмет на клавиатуре 0, то тогда программа остановится.

# 5 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.

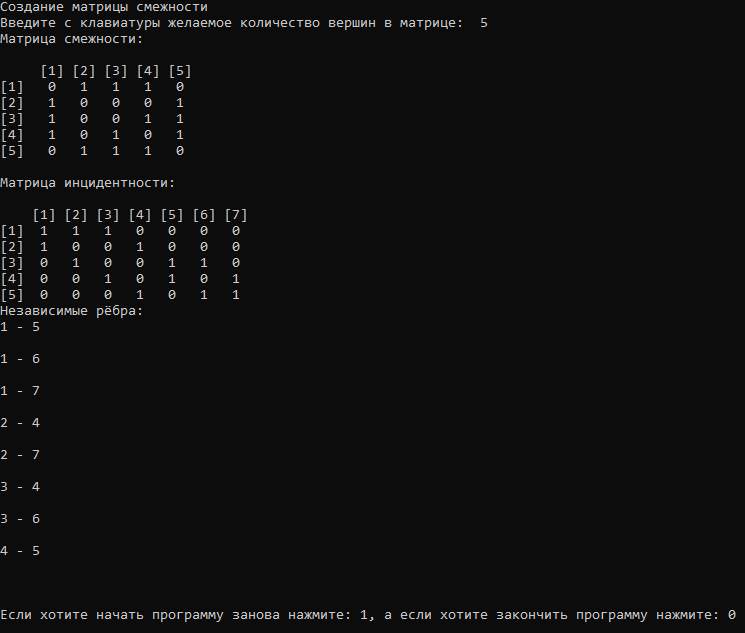


Рисунок 9 – Показывает когда есть независимые рёбра(5 вершин)

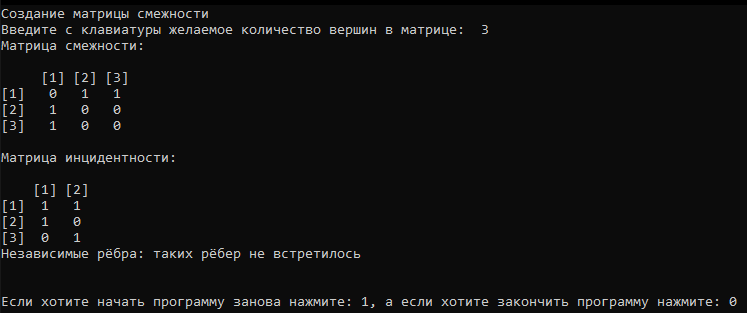


Рисунок 10 – Показывает, когда нет независимые рёбра

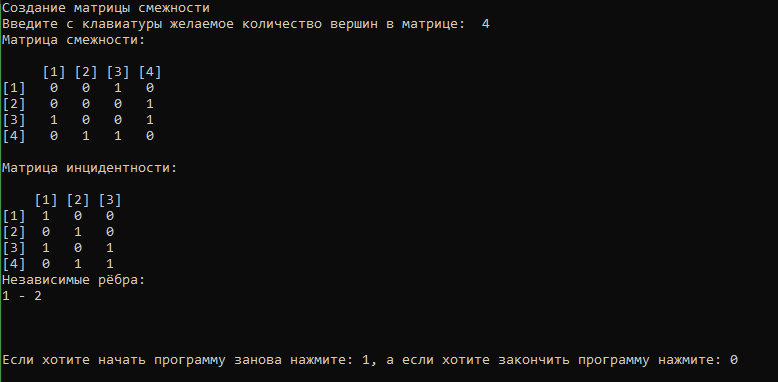


Рисунок 11 – Правильность работы программы (4 вершины)

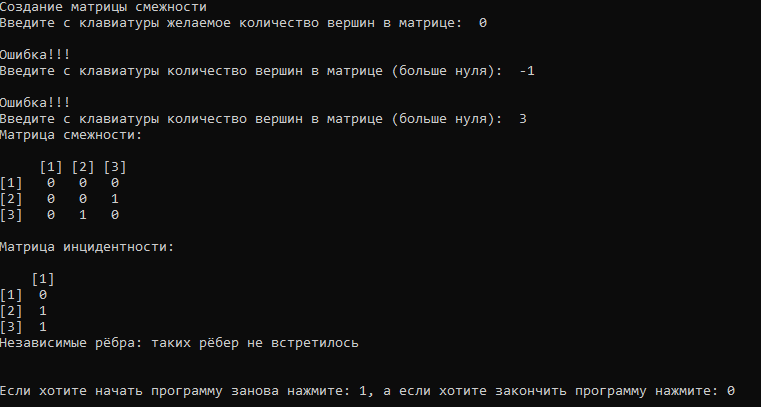


Рисунок 12 – Ошибка при создании матрицы, если ввести значения равные нулю или отрицательные

Таблица 1 - Описание поведения программы при тестировании

|  |  |  |
| --- | --- | --- |
| Описание теста | Ожидаемый результат | Полученный результат |
| Запуск программы | Вывод сообщение о вводе количества вершин в графе | Верно |
| Ошибка при вводе | Вывод сообщение о некорректном вводе вершины и возможность повторного ввода | Верно |
| Генерация матрицы | Моментальный вывод матрицы на экран | Верно |
| Генерация матриц различных размеров | Матрицы корректно выведется на экран, элементы не «уползут» куда не надо | Верно |
| Запуск и остановка программы по желанию пользователя | При запуске программы заново, программа начинается с самого начала. При остановке программы, программа останавливается моментально | Верно |

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

# 6 Ручной расчет задачи

Проведем расчет на примере матрицы размерности 4 (рисунок 11).

Для начала зарисуем сгенерируемую в этом случае матрицу для дальнейшего анализа (рисунок 13).

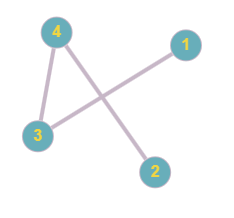
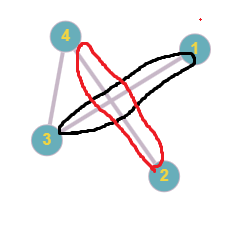


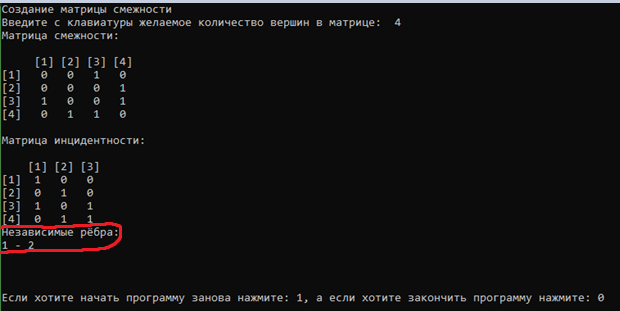
Рисунок 13 - Граф на основе рис.

На данном графе можно найти независимые рёбра:





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 2022.

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. Основные понятия графов сайт <https://skysmart.ru/articles/mathematic/osnovnye-ponyatiya-teorii-grafov>
3. Алексеев В.Е., Захарова Д.В. Теория графов. Учебное пособие 2017 г.
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.

#define \_CRT\_SECURE\_NO\_WARNINGS

#include <stdio.h>

#include <Windows.h>

#include <conio.h>

#include <locale.h>

#include <time.h>

#include <clocale>

#include <stdlib.h>

#include <limits.h>

using namespace std;

int i, j, m, n, v;

int\*\* graph;

int counter;

bool sr = false;

int nachalo;

int k, p;

int main(void)

{

k = 1;

while (k == 1) {

setlocale(LC\_ALL, "Rus");

srand(time(NULL));

printf("Создание матрицы смежности\n");

printf("Введите с клавиатуры желаемое количество вершин в матрице: ");

scanf("%d", &m);

while (m == 0 || m < 0) {

printf("\nОшибка!!!\n");

printf("Введите с клавиатуры количество вершин в матрице (больше нуля): ");

scanf("%d", &m);

}

n = m;

graph = new int\* [m];

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

graph[i] = new int[m];

}

//генерация матрицыы

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

}

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: ");

counter = 1;

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

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) {

counter = 0;

printf("\n%d - %d\n", i + 1, j + 1);

}

}

}

if (counter == 1) {

printf("таких рёбер не встретилось");

}

printf("\n \n \n");

printf("Если хотите начать программу занова нажмите: 1, а если хотите закончить программу нажмите: 0 \n");

if (scanf("%d", &nachalo) != 1 || nachalo == 0) return 0;

nachalo = k;

if (k == 1) {

k = true;

system("cls");

}

}

return 0;

}