

Оглавление

[Реферат 5](#_Toc59606580)

[Введение 6](#_Toc59606581)

[1 Постановка задачи 7](#_Toc59606582)

[2 Теоретическая часть задания 8](#_Toc59606583)

[3 Описание алгоритма программы 10](#_Toc59606584)

[4 Описание программы 15](#_Toc59606585)

[5 Тестирование 18](#_Toc59606586)

[6 Ручной расчет задачи 21](#_Toc59606587)

[Заключение 23](#_Toc59606588)

[Список литературы 24](#_Toc59606589)

[Приложение А. 25](#_Toc59606590)

# Реферат

Отчет 27 стр., 13 рисунков

ГРАФ, АЛГОРИТМ ПОИСКА НЕЗАВИСИМЫХ МНОЖЕСТВ РЕБЕР ГРАФА

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

В работе рассмотрены граф, который выражен с помощью матрицы инцидентности, основанной 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, а также сохраняет его в отдельный файл.

В качестве среды разработки мною была выбрана среда CLion 2021.3, 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. Изначальные данные тоже вводит пользователь. В программе предусмотрен ввод размера графа, автоматическая генерация элементов, а также ручной ввод с защитой, которая не позволяет вводить некорректные данные. Затем программа, после работы алгоритма, выводит все множества независимых ребер графа на экран.

Содержимое окна вывода (результаты работы программы) сохраняются в отдельном файле, выбираемом пользователем в самом начале запуска программы. Необходимо предусмотреть различные исходы поиска, что бы программы не выдавала ошибок и работала правильно. Устройство ввода - клавиатура. Устройство вывода - монитор.

# 2 Теоретическая часть задания

Множество независимых ребер в графе это набор ребер, которые не содержат никаких общих вершин с другими ребрами. То есть, каждое ребро в множестве независимых ребер имеет уникальные вершины, которые не совпадают с вершинами других ребер в этом множестве.

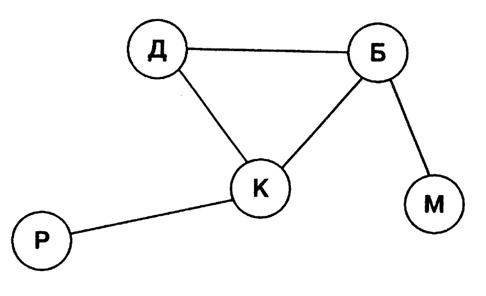
Неориентированный граф - это специальный тип графа, в котором ребра не имеют направления. То есть, ребро между двумя вершинами может быть пройдено в обоих направлениях. Например, если существует ребро между вершинами Б и М, то это ребро может быть пройдено от вершины Б к вершине М, а также от вершины М к вершине Б. Пример неориентированного графа показан на рисунке 1.

Рисунок 1 – Пример графа

Неориентированный граф может быть представлен в виде матрицы смежности. Одним из способов реализации алгоритма поиска независимых множеств ребер графа является использование матриц смежности и инцидентности. Матрица смежности – это матрица, которая используется для представления графа в виде матрицы размера N x N, где N – количество вершин в графе. Каждый элемент матрицы смежности a[i][j] равен 1, если между вершинами i и j существует ребро, и 0 в противном случае.

Матрица инцидентности – это матрица, которая используется для представления ребер графа в виде матрицы размера M x N, где M – количество ребер в графе, а N – количество вершин. Каждый элемент матрицы инцидентности b[i][j] равен 1, если ребро i инцидентно (то есть соединяет) вершину j, и 0 в противном случае. Пример независимых ребер продемонстрирован на рисунке 2.

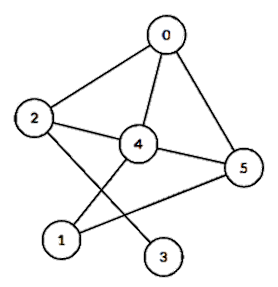




Рисунок 2 - Пример независимых ребер

# 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=(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:

ЕСЛИ t РАВНО 1:

ИНИЦИАЛИЗИРОВАТЬ ГЕНЕРАТОР СЛУЧАЙНЫХ ЧИСЕЛ С ИСПОЛЬЗОВАНИЕМ ТЕКУЩЕГО ВРЕМЕНИ

ДЛЯ I ОТ 0 ДО M ВКЛЮЧИТЕЛЬНО:

ДЛЯ J ОТ I ДО M ВКЛЮЧИТЕЛЬНО:

GRAPH[i][j] = GRAPH[j][i] = СЛУЧАЙНОЕ ЧИСЛО ИЗ ДИАПАЗОНА [0, 1]

GRAPH[i][i] = 0

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). Что бы не было петель, в графе предусмотрено обнуление главной диагонали.

Так происходит ручной ввод элементов:

ВЫВЕСТИ "Введите элементы матрицы 0 и 1:"

ДЛЯ i ОТ 0 ДО m ВКЛЮЧИТЕЛЬНО:

ДЛЯ j ОТ i ДО m ВКЛЮЧИТЕЛЬНО:

ЕСЛИ i НЕ РАВНО j:

ВЫВЕСТИ "a[i][j] = "

СЧИТАТЬ ВВОД ПОЛЬЗОВАТЕЛЯ В graph[i][j]

ЕСЛИ graph[i][j] БОЛЬШЕ 1 ИЛИ graph[i][j] МЕНЬШЕ 0:

ВЫВЕСТИ "\nОшибка!\n"

graph[i][i] = 0

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:

ДЛЯ i ОТ 0 ДО m ВКЛЮЧИТЕЛЬНО:

ДЛЯ j ОТ 0 ДО m ВКЛЮЧИТЕЛЬНО:

ВЫВЕСТИ " graph[i][j] "

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.

ПОКА ch НЕ РАВНО n:

ДЛЯ i ОТ 0 ДО n ВКЛЮЧИТЕЛЬНО:

ДЛЯ j ОТ ch ДО n ВКЛЮЧИТЕЛЬНО:

ЕСЛИ graph[i][j] НЕ РАВНО 0:

reb УВЕЛИЧИТЬ НА 1

ch УВЕЛИЧИТЬ НА 1

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

ДЛЯ i ОТ 0 ДО n ВКЛЮЧИТЕЛЬНО:

ДЛЯ j ОТ 0 ДО n ВКЛЮЧИТЕЛЬНО:

ЕСЛИ graph[i][j] НЕ РАВНО 0:

inc[i][reb] = 1

inc[j][reb] = 1

reb УВЕЛИЧИТЬ НА 1

А выводится вот так:

ДЛЯ i ОТ 0 ДО n ВКЛЮЧИТЕЛЬНО:

ВЫВЕСТИ "\n"

ДЛЯ j ОТ 0 ДО reb ВКЛЮЧИТЕЛЬНО:

ВЫВЕСТИ " inc[i][j] "

Приступим к алгоритму поиска независимых ребер. Два ребра независимы если единицы в матрице двух столбцов этих ребер не будут пересекаться, что показано на рисунке 3.

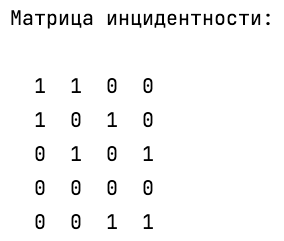
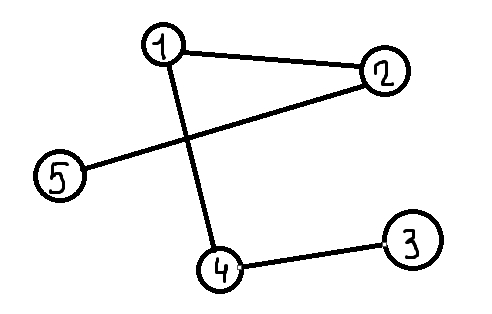




Рисунок 3 - Независимые вершины на матрице инцидентности

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



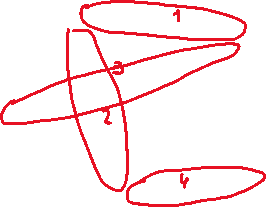


Рисунок 4 - Граф с независимыми вершинами

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.

ВЫВЕСТИ "Независимые рёбра:\n"

ДЛЯ i ОТ 0 ДО reb ВКЛЮЧИТЕЛЬНО:

ОПРЕДЕЛИТЬ sr КАК ЛОЖЬ

ДЛЯ j ОТ i+1 ДО reb ВКЛЮЧИТЕЛЬНО:

sr ОПРЕДЕЛИТЬ КАК ЛОЖЬ

ДЛЯ k ОТ 0 ДО n ВКЛЮЧИТЕЛЬНО:

ЕСЛИ inc[k][i] РАВНО 1 И inc[k][j] РАВНО 1:

sr ОПРЕДЕЛИТЬ КАК ИСТИНА

ЕСЛИ sr РАВНО ЛОЖЬ:

ВЫВЕСТИ i+1, " - ", j+1, "\n"

# 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.

Работа программы начинается с того, что пользователю необходимо ввести название файла (возможность сохранения результата работы программы была реализована с помощью функции «fopen» в режиме «w+»), в который в последующем будут сохранены результаты выполнения программы. Далее происходит запрос - 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.

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

## 

Рисунок 5 - Запрос данных от пользователя

Случайная генерация матрицы продемонстрирована на рисунке 6.

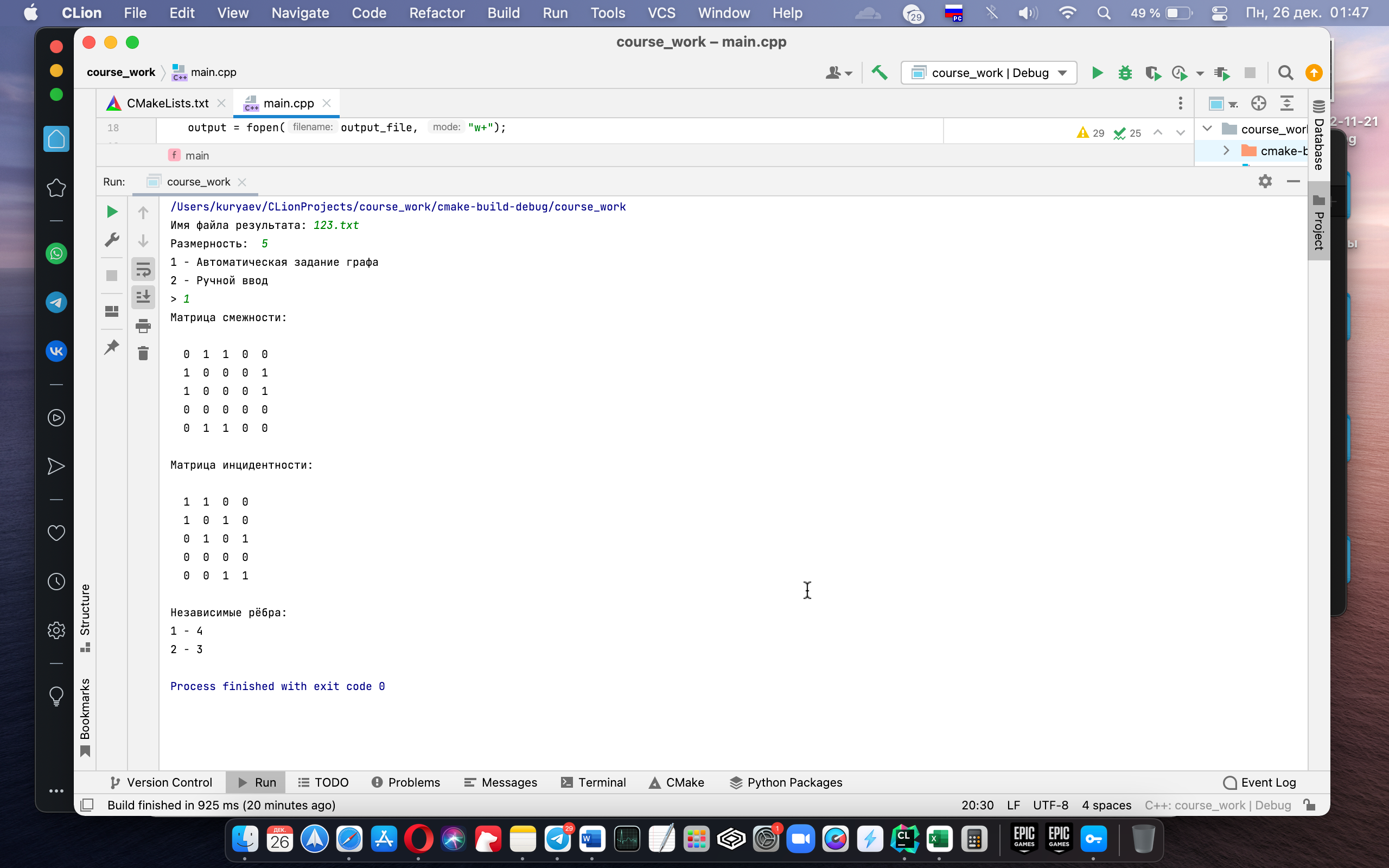


Рисунок 6 – Автоматический ввод

Ручной ввод матрицы изображен на рисунке 7.

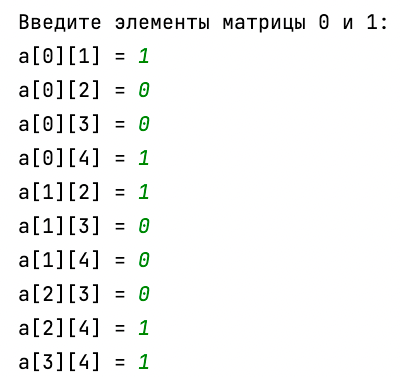


Рисунок 7 - Ручной ввод

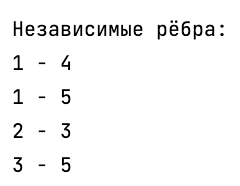
Далее после вывода всех матриц выводятся все независимые вершины (на примере результата работы программы на рисунке 7), что и показано на рисунке 8.

Рисунок 8 - Независимые вершины

# 5 Тестирование

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

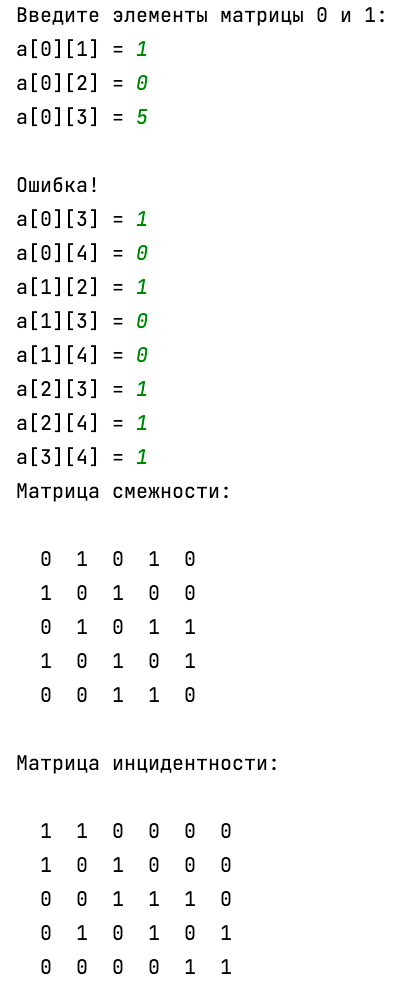


Рисунок 9 - Ручной ввод с примером некорректного ввода.

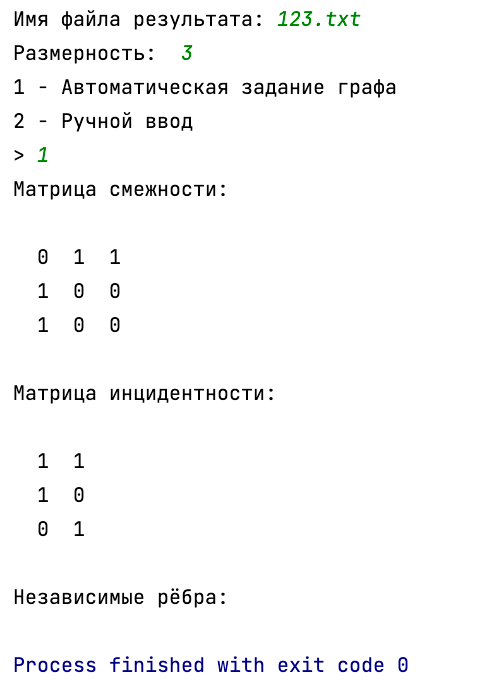


Рисунок 10 - Автоматический ввод, где не найдено независимых ребер

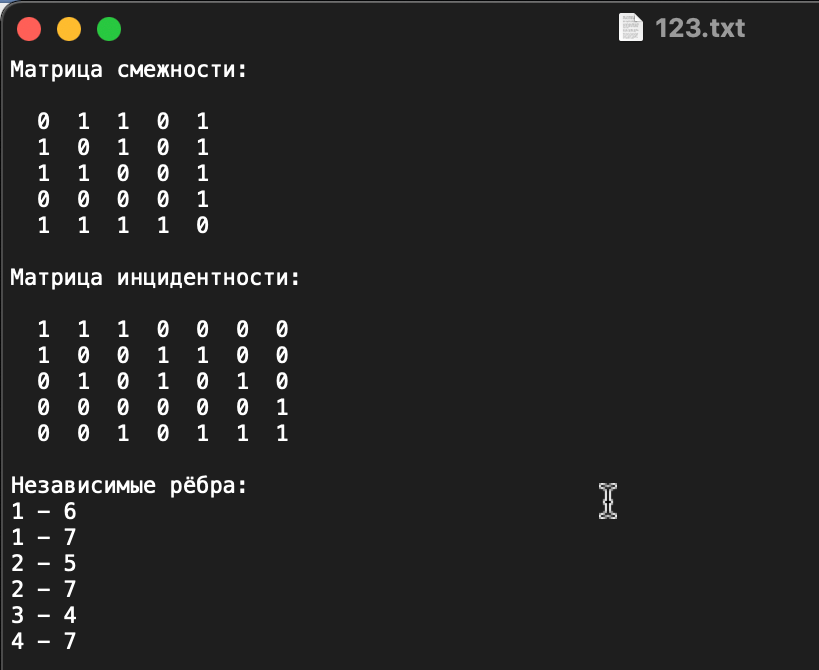


Рисунок 11 – файл, указанный пользователем для сохранения результатов

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 |
| Сохранение результатов в файл | Результаты сохранены в указанный пользователем файл | 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 5 (рисунок 12).

Для начала зарисуем сгенерируемую в этом случае матрицу для дальнейшего анализа.

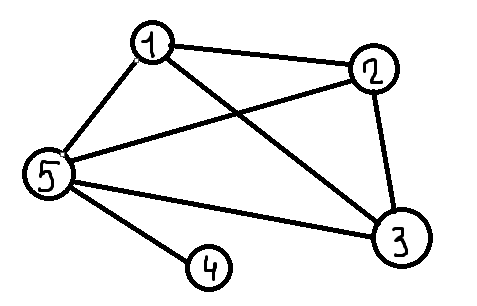
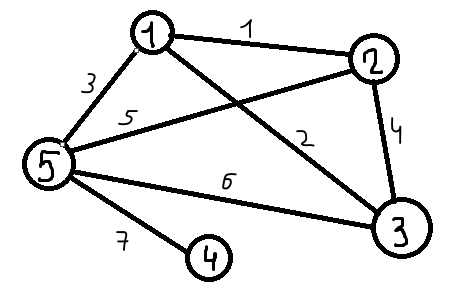
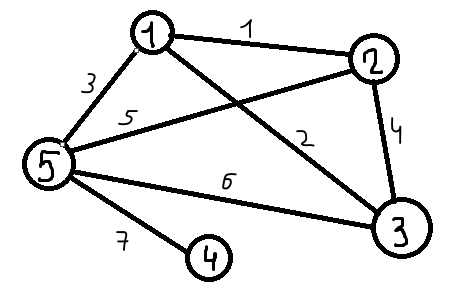
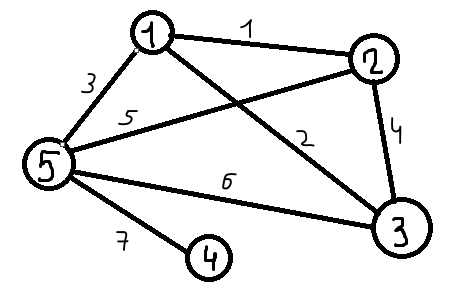
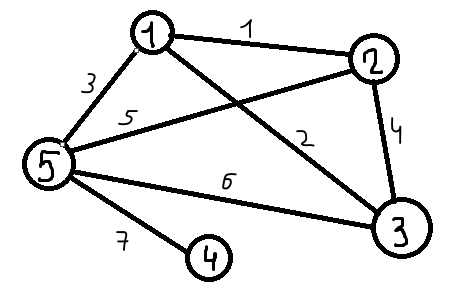
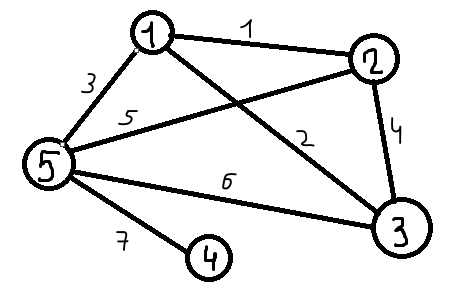
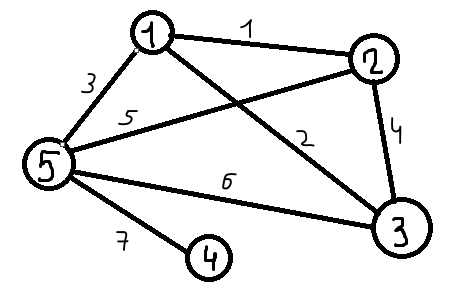
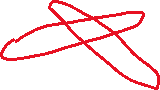




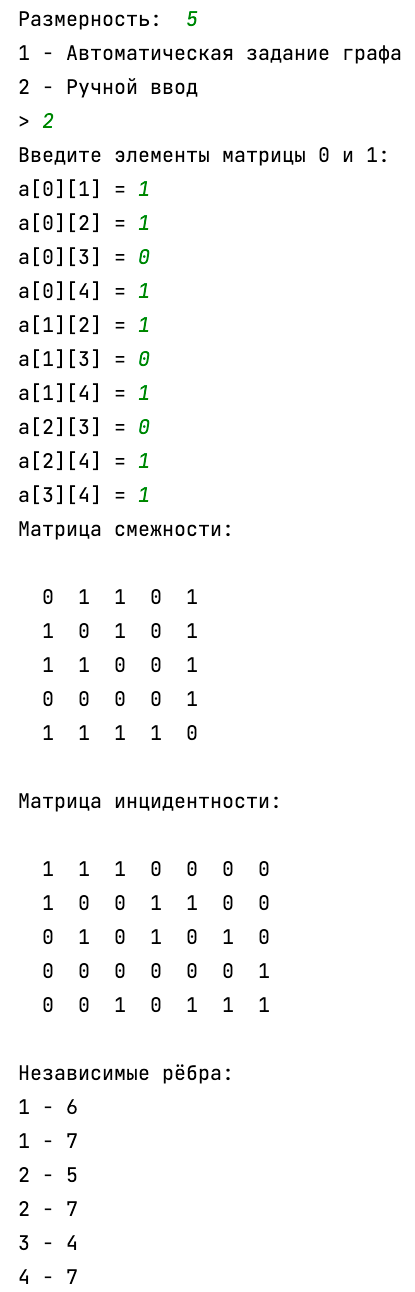
Рисунок 12 – Сгенерированный граф

На данном графе можно найти целых 6 независимых множеств ребер:





Проверим с результатом работы программы на рисунке 13:



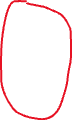


Рисунок 13 – Сгенерированный граф

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

# Заключение

Таким образом, в процессе создания данного проекта . Была разработана программа, которая позволяет генерировать случайный граф и находить независимые множества ребер в CLion 2021.3.

В данной курсовой работе мы рассмотерли вопрос реализации поиска независимых множеств ребер графа. Была представлена идея построения инцидентной матрицы, которая позволяет эффективно хранить информацию о связях между ребрами и вершинами. Была разработана программа, которая позволяет генерировать случайный граф и находить независимые множества ребер.

# 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 <locale.h>  
#include <time.h>  
#include <iostream>  
using namespace std;  
  
int i, j, m, v, n;  
  
int\*\* graph;  
  
int main()  
{  
 char output\_file[10000];  
 FILE \*output;  
 printf("Имя файла результата: ");  
 scanf("%s", output\_file);  
  
 output = fopen(output\_file, "w+");  
  
 printf("Размерность: ");  
 scanf("%d", &m);  
 n = m;  
 graph = new int\* [m];  
 for (int i = 0; i < m; i++) {  
 graph[i] = new int[m];  
 }  
  
 int t;  
 printf("1 - Автоматическая задание графа\n2 - Ручной ввод\n> ");  
 scanf("%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;  
 }  
 }  
 else {  
 printf("Введите 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("%d", &graph[i][j]);  
 if (graph[i][j] > 1 || graph[i][j] < 0) {  
 printf("\nОшибка!\n");

goto ret;  
 }  
 }  
 }  
 graph[i][i] = 0;  
 }  
 }  
 for (i = 0; i < n; i++)  
 {  
 for (j = 0; j < n; j++)  
 {  
 graph[j][i] = graph[i][j];  
 }  
 }  
  
  
 fprintf(output, "Матрица смежности: \n");  
 printf("Матрица смежности: \n");  
  
  
 fprintf(output, "\n");  
 printf("\n");  
  
 for (i = 0; i < m; ++i)  
 {  
 for (j = 0; j < m; ++j) {  
 fprintf(output, "%3d", graph[i][j]);  
 printf("%3d", graph[i][j]);  
 }  
 fprintf(output, "\n");  
 printf("\n");  
 }  
  
 fprintf(output, "\nМатрица инцидентности: \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;  
  
  
 reb = 0;  
 for (int i = 0; i < n; i++)  
 {  
 for (int j = 0; j < n; j++)  
 if (graph[i][j] != 0)  
 {  
 inc[i][reb] = 1;  
 inc[j][reb] = 1;  
 reb++;  
 }  
 }  
  
  
 for (int i = 0; i < n; i++)  
 {  
 printf("\n", i+1);  
 fprintf(output, "\n", i+1);  
 for (int j = 0; j < reb; j++)  
 {  
 printf("%3d", inc[i][j]);  
 fprintf(output, "%3d", inc[i][j]);  
 }  
 }  
  
 fprintf(output, "\n\n");  
 printf("\n\n");  
 fprintf(output, "Независимые рёбра:\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) { fprintf(output, "%d - %d\n", i + 1, j + 1);  
 printf("%d - %d\n", i + 1, j + 1);}  
 }  
 }  
 return 0;  
 fclose(output);  
}