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 поиска независимых множеств вершин

графа»

25.12.25
отлично

Выполнил:
студент группы 24ВВВ3
Агапов И.А.

Принял:
Юрова О.В.

Пенза 2025

ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
Факультет Вычислительной техники

Кафедра "Вычислительная техника"

"УТВЕРЖДАЮ"

Зав. кафедрой ВТ _____

«___»_____20__

ЗАДАНИЕ

на курсовое проектирование по курсу

"Логика и основы алгоритмизации в инженерных задачах"

Студенту Иванов Иван Андреевич Группа 18ВВ3

Тема_проекта Написание алгоритма поиска независимых множеств вершин графа

Исходные данные (технические требования) на проектирование

Разработка алгоритмов и программных средств в соответствии с рамками задания курсового проекта

Пояснительная записка должна содержать:

1) Постановку задачи;
2) Теоретические часть задачи;
3) Описание алгоритма поставленной задачи;
4) Пример ручного расчета задачи и выполнения
5) Описание самой программы
6) Тесты
7) Список литературы
8) Листинг программы
9) Результаты работы программы

Объем работы по курсу

1. Расчетная часть

Ручной расчет работы алгоритма

2. Графическая часть

Схема алгоритма в формате блок-схем

3. Экспериментальная часть

Тестирование программы
Результаты работы программы на тестовых данных

Срок выполнения проекта по разделам

1 Моделирование теоретической основы
2 Разработка алгоритмов программы
3 Разработка программ
4 Отладка и завершение разработки программ
5 Оформление пояснительной записки
6
7
8

Дата выдачи задания "12" октября 2025

Дата защиты проекта "__" _____

Руководитель Юрова Ольга Викторовна

Задание получил "12" октября 2025 г.

Студент Ананьев Иван Андреевич

Содержание

Реферат

Отчет 31 стр, 13 рисунков, 2 таблицы.

ПОИСК НЕЗАВИСИМЫХ МНОЖЕСТВ ВЕРШИН ГРАФА.

Цель исследования – разработка программы, способной выполнять алгоритм поиска независимых множеств вершин графа.

В работе рассмотрен способ рекурсивного поиска независимых вершин графа. Программа генерирует граф случайным образом и заносит его в файл, или, на выбор пользователя, использует заранее заполненный файл. Затем происходит поиск всех множеств независимых вершин используя рекурсию и записывает результат в указанный пользователем файл.

Введение

Данная курсовая работа посвящена разработке и реализации алгоритма поиска независимых множеств вершин в неориентированных графах. Графы используются для представления сетей коммуникаций, социальных связей, молекулярных структур, транспортных систем и многих других объектов, где важны отношения между компонентами.

Важным понятием в рамках теории графов является независимое множество - подмножество вершин, никакие две из которых не соединены ребром. Иными словами, это набор вершин, не связанных непосредственно друг с другом. Задачи, связанные с поиском независимых множеств, имеют существенное значение в оптимизации и применяются в различных предметных областях.

К таким областям относятся, например, задачи распределения ресурсов, частотного планирования, кодирования данных и биоинформатики, где требуется находить непересекающиеся или непротиворечивые наборы элементов. В частности, в задачах расписания независимое множество может соответствовать максимальному набору одновременно выполняемых работ без конфликтов.

В качестве среды разработки мною была выбрана среда Microsoft Visual Studio 2022, язык программирования – C#.

Целью данной курсовой работы является разработка программного обеспечения реализующего алгоритм поиска всех независимых множеств вершин в неориентированном графе, представленном в виде матрицы смежности. Программа будет обеспечивать два основных режима работы: чтение графа из файла и генерацию случайного графа. Программа выведет найденные независимые множества в консоль и сохранит их в выходном файле.

1. Постановка задачи

Исходный граф в программе должен задаваться матрицей смежности, причём при генерации данных должны быть предусмотрены ограниченные условия.

Программа должна работать как со случайной генерацией графов, так и считывать графы из файла. Пользователь должен иметь выбор какой из этих вариантов использовать. При выборе случайной генерации пользователь должен указать размер графа. Также в случае случайной генерации результат генерации должен сохраняться в текстовый файл, указываемый пользователем. Результат работы программы также сохраняется в указанный пользователем файл

Устройство ввода – клавиатура и мышь.

2. Теоретическая часть задания

В неориентированном графе G = (V, E), где V – множество вершин, а E – множество ребер, независимое множество (также называемое стабильным множеством) – это подмножество вершин I ⊆ V такое, что никакие две вершины в I не соединены ребром. Другими словами, для любых двух вершин, v ∈ I, (u, v) ∉ E. На рисунке 1 представлен неориентированный граф.

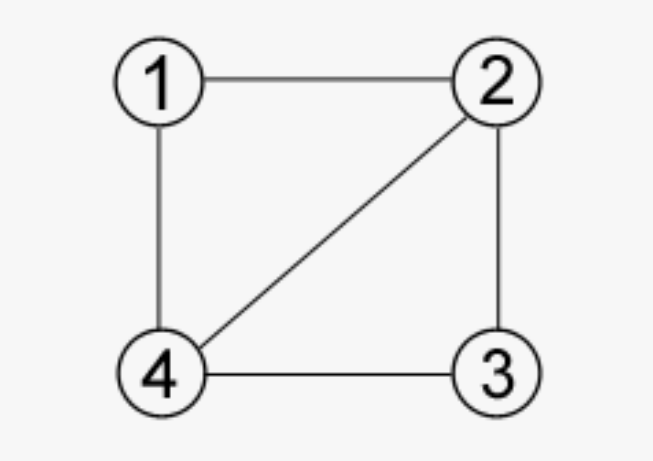

Рисунок 1– Неориентированный граф

Рассмотрим граф с вершинами {A, B, C, D, E} и ребрами {(A, B), (B, C), (C, D), (D, E), (E, A)}, в котором:

{B, D} – это независимое множество.

{A, C} – это независимое множество.

{A, B, C} - это не независимое множество, так как вершины A и B соединены ребром, как и вершины B и C.

{A, C, E} - это не независимое множество, так как вершины A и E соединены ребром.

3. Описание алгоритма программы

Для реализации алгоритма поиска независимых вершин графа необходим исходный граф. Его можно как задать с помощью случайной генерации, указав его размеры в программе, так и считать с текстового файла, указанного пользователем.

Главной функцией, выполняющей задание является findIndependentSets, которая, используя рекурсию, проверяет все возможные множества на независимость, считая их независимыми до того момента, пока не будет доказано обратное. После проверки множества, в случае если оно действительно независимо, множество заносится в указанный пользователем ранее текстовый файл, а также выводится в консоль.

Ниже представлен псевдокод функции findIndependentSets:

```
ЕСЛИ index равно numVertices ТО:
  independent = ИСТИНА
    ДЛЯ i от 0 ДО count – 1 ПОКА independent
        ДЛЯ j от i + 1 ДО count - 1:
            ЕСЛИ graph[subset[i]][subset[j]] равно 1 ТО:
                Independent = ЛОЖЬ
                ПРЕРВАТЬ
            КОНЕЦ ЕСЛИ
        КОНЕЦ ДЛЯ
    КОНЕЦ ДЛЯ
    ЕСЛИ independent И count > 0 ТО:
        ВЫВЕСТИ subset на консоль
        ПОПЫТАТЬСЯ
            ДЛЯ I от 0 ДО count-1
                ЗАПИСАТЬ subset в файл
            КОНЕЦ ДЛЯ
```

ИСКЛЮЧЕНИЕ

ВЫВЕСТИ “Ошибка при записи в файл”

КОНЕЦ ПОПЫТКИ

КОНЕЦ ЕСЛИ

ВЕРНУТЬ

КОНЕЦ ЕСЛИ

Рекурсивный вызов с включением текущей вершины в подмножество

Рекурсивный вызов без включения текущей вершины в подмножество
КОНЕЦ функции

Полный код программы можно увидеть в Приложении А.

4. Описание программы

Данная программа реализована на языке программирования C#. C# - современный объектно-ориентированный язык программирования, разработанный компанией Microsoft, который сочетает в себе простоту использования, мощь платформы .NET и широкие возможности для разработки различных типов приложений.

Программа создана в виде консольного приложения .NET, что обеспечивает её кроссплатформенность и возможность запуска на различных операционных системах, поддерживающих среду выполнения .NET.

Работа программы начинается с запроса действий пользователя. Если пользователь выбрал «Считать граф из файла», то у него попросят указать путь к файлу, в котором находится его граф. Затем его попросят указать файл, в который будет сохранён результат работы программы. В случае выбора пункта «Сгенерировать случайный граф», пользователя попросят ввести размер графа, путь к файлу для сохранения графа, а также путь к файлу для сохранения результата работы программы.

Часть кода, ответственная за выбор функций:

```
Console.WriteLine("Выберите режим работы:");
Console.WriteLine("1. Считать граф из файла");
Console.WriteLine("2. Сгенерировать случайный граф");
```

Эта часть кода просит пользователя выбрать режим работы:

1) Считывание графа из файла

2) Генерация случайного графа

Выбор пользователя сохраняется в переменную choice, используя которую с функциями if и else if используется необходимая часть кода.

Часть кода, ответственная за режим работы с чтением графа из файла:

```
if (choice == 1)
{
    Console.Write("Введите имя файла: ");
    filename = Console.ReadLine();

    if (!ReadGraphFromFileImproved(filename))
```

```
        {

            Console.WriteLine("\nНажмите любую клавишу для выхода...");
            Console.ReadKey();
            return;
        }
    }
```

Эта часть кода приводится в действие в том случае, если пользователь подал на ввод единицу, что соответствует режиму работы программы с чтением графа из файла. В ней пользователя просят ввести название файла, где в случае успеха происходит запись количества вершин в переменную numVertices и запись самого графа в двумерный массив graph[i][j].

Часть кода, ответственная за режим работы со случайно сгенерированным графом:

```
    else if (choice == 2)
    {
        Console.Write("Введите количество вершин: ");
        if (!int.TryParse(Console.ReadLine(), out numVertices))
        {
            Console.WriteLine("Неверное количество вершин.");
            return;
        }

        if (numVertices <= 0 || numVertices > MAX_VERTICES)
        {
            Console.WriteLine($"Ошибка: количество вершин должно быть от 1 до
{MAX_VERTICES}");
            return;
        }

        Console.Write("Введите имя файла для сохранения графа: ");
        graphFilename = Console.ReadLine();

        if (!GenerateRandomGraphImproved(numVertices, graphFilename))
        {

            Console.WriteLine("\nНажмите любую клавишу для выхода...");
            Console.ReadKey();
            return;
        }
    }
```

Эта часть кода приводится в действие в том случае, если пользователь подал на ввод двойку, что соответствует режиму работы программы со случайной генерацией графа. В ней пользователя просят ввести количество вершин генерируемого графа, сохраняемое в переменную numVertices, и название файла в который будет записан сгенерированный граф, сохраняемое в переменную graphFilename.

Часть кода, отвечающая за обработку неверно выбранного режима работы программы:

```
else
{
    Console.WriteLine("Неверный выбор режима работы.");
    return;
}
```

Эта часть кода приводится в действие в том случае, если пользователь подал на вход значение отличное от единицы или двойки. В этом случае выводится сообщение об ошибке и происходит закрытие программы.

Часть кода, ответственная за запись результата работы программы в файл:

```
Console.Write("\nВведите имя файла для вывода независимых множеств: ");
      outputFilename = Console.ReadLine();
            if (index == numVertices)
            {
                bool independent = true;

                for (int i = 0; i < count && independent; i++)
                {
                    for (int j = i + 1; j < count; j++)
                    {
                        if (graph[subset[i], subset[j]] == 1)
                        {
                            independent = false;
                            break;
                        }
                    }
                }

                if (independent && count > 0)
                {

                    Console.Write("{ ");
                    for (int i = 0; i < count; i++)
                    {
                        Console.Write(subset[i] + (i == count - 1 ? "" : ",
"));
                    }
                    Console.WriteLine(" }");


                    try
                    {
                        using (StreamWriter outputFile = new
StreamWriter(outputFilename, true))
                        {
                            for (int i = 0; i < count; i++)
                            {
                                outputFile.Write(subset[i] + (i == count - 1 ?
"" : " "));
                            }
                            outputFile.WriteLine();
                        }
                    }
```

```
                        catch (Exception ex)
                        {
                            Console.WriteLine($"Ошибка при записи в файл:
{ex.Message}");
                        }
                    }
                    return;
                }

                subset[count] = index;
                FindIndependentSets(subset, index + 1, count + 1, outputFilename);
                FindIndependentSets(subset, index + 1, count, outputFilename);
            }
        }
    }
```

Эта часть кода приводится в действие в том случае, если в частях ответственные за генерацию графа и его записи в файл или же считывании графа с пользовательского файла отсутствуют ошибки. В этом случае приводится в работы рекурсивный алгоритм поиска множеств независимых вершин графа. После работы алгоритма пользователя просят ввести название файла, сохраняемое в переменную outputFilename, в который будет сохранен результат работы. В случае корректного ввода происходит запись результата в файл и завершение программы, в ином случае выводится сообщение об ошибке и происходит завершение работы программы.

На рисунке 2 можно увидеть оформление начального запроса и дальнейшие действия с ним.

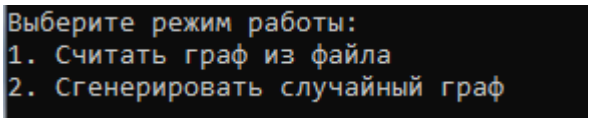

Рисунок 2 – Стартовое окно программы

Выбор пункта со случайной генерацией графа и запрос на ввод количества вершин представлены на рисунке 3.

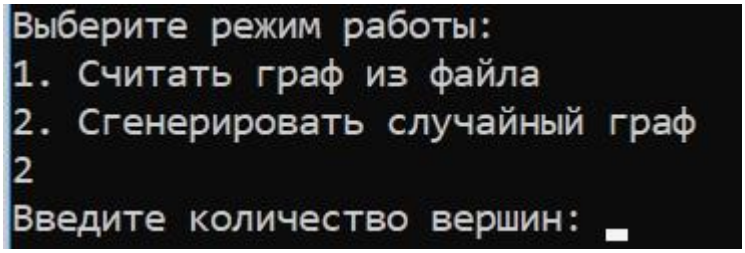

Рисунок 3 – Случайная генерация графа

Выбор пункта со считыванием графа из файла с последующим запросом файла от программы показан на рисунке 4.

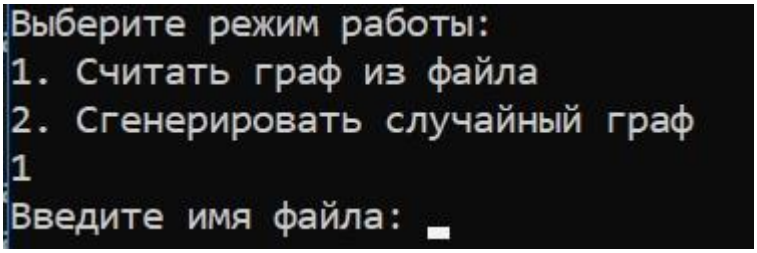

Рисунок 4 – Считывание из файла

Выбор файла для записи результата работы программы, а также вывод результата в консоль показаны на рисунке 5.

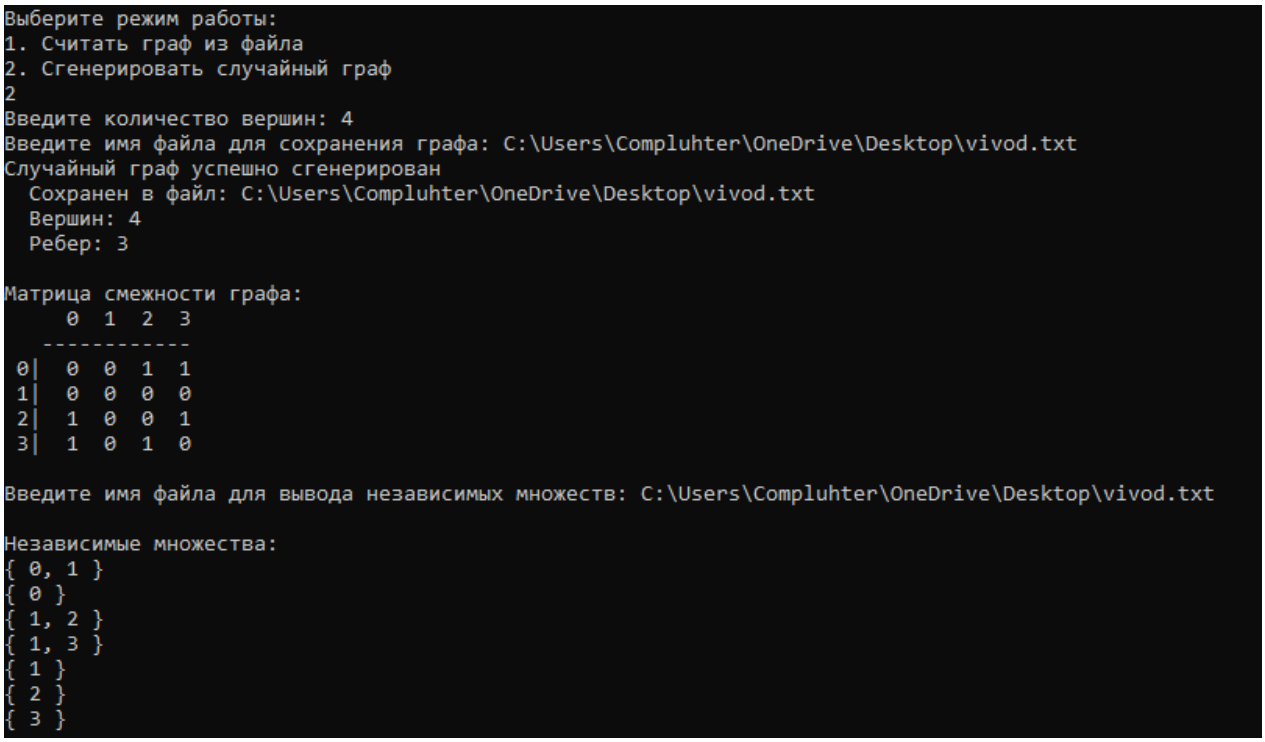

Рисунок 5 – Поиск независимых множеств

5. Тестирование

Среда разработки Microsoft Visual Studio 2022 предоставляет все средства, необходимые при разработке и отладке многомодульной программы.

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

Ниже продемонстрирован результат тестирования программы при вводе пользователем файла с исходным графом, ввода файла для сохранения результата алгоритма, ошибки при неправильном указании файла.

Таблица 1 – Описание ожидаемого поведения программы.

Описание теста	**Предусловие**	**Тестирование**	**Ожидаемый результат**
Запуск программы	нет	Запуск программы с помощью VisualStudio	Вывод пользователю списка с выбором режима работы программы
Выбор режима работы с пользовательским графом в файле	Программа запущена	Выбрать режим работы считывания графа из файла	Вывод пользователю сообщения с просьбой задать файл для чтения
Задание верного пути к файлу для чтения	Программа запущена, выведена просьба ввести файл для чтения	Ввести полный путь к файлу, содержащему граф	Вывод пользователю матрицы смежности графа и сообщения с просьбой ввода файла для записи результата программы

Продолжение таблицы 1:

Описание теста	**Предусловие**	**Тестирование**	**Ожидаемый результат**
Задание верного пути к файлу для записи	Выведена просьба ввести файл для записи	Ввести полный путь к файлу для записи результата программы	Вывод пользователю результат работы программы в консоль, запись результатов работы программы в указанный файл
Задание неверного пути к файлу для чтения	Выведена просьба ввести файл для чтения	Ввести неверный путь к файлу, содержащему граф	Вывод сообщение об ошибки чтения данных из файла
Задание неверного пути к файлу для записи	Выведена просьба ввести файл для записи	Ввести неверный путь к файлу для записи результата работы программы	Вывод сообщение об ошибки записи данных в файл
Выбор режима работы со случайной генерацией графа	Программа запущена, выведено меню пользователя	Выбрать режим работы случайной генерации графа	Вывод пользователю сообщения с просьбой ввести размер генерируемого графа
Ввод размера генерируемого графа	Выведена просьба ввести размер генерируемого графа	Ввести целое положительное число	Вывод сообщения с просьбой ввести файл для записи случайно сгенерированного графа
Ввод пути к файлу записи случайно сгенерированного графа	Выведена просьба ввести файл для записи	Ввести полный путь к файлу для записи случайно сгенерированного графа	Вывод матрицы смежности графа сообщения с просьбой ввести файл для записи результатов работы программы

Тестирование на ввод файла для чтения представлен на рисунке 6.

```
Выберите режим работы:
1. Считать граф из файла
2. Сгенерировать случайный граф
1
Введите имя файла: C:\Users\Compluhter\OneDrive\Desktop\vivod.txt

Попытка чтения файла: C:\Users\Compluhter\OneDrive\Desktop\vivod.txt
Количество вершин: 4
Граф успешно загружен из файла 'C:\Users\Compluhter\OneDrive\Desktop\vivod.txt'
  Вершин: 4

Матрица смежности графа:
     0  1  2  3
   ------------
 0|  0  0  0  1
 1|  0  0  0  1
 2|  0  0  0  0
 3|  1  1  0  0

Введите имя файла для вывода независимых множеств:
```

Рисунок 6 – Тестирование при вводе файла для чтения.

Тестирование на ввод файла для сохранения представлен на рисунке 7.

```
Выберите режим работы:
1. Считать граф из файла
2. Сгенерировать случайный граф
1
Введите имя файла: C:\Users\Compluhter\OneDrive\Desktop\vivod.txt

Попытка чтения файла: C:\Users\Compluhter\OneDrive\Desktop\vivod.txt
Количество вершин: 4
Граф успешно загружен из файла 'C:\Users\Compluhter\OneDrive\Desktop\vivod.txt'
  Вершин: 4

Матрица смежности графа:
     0  1  2  3
   ------------
 0|  0  0  0  1
 1|  0  0  0  1
 2|  0  0  0  0
 3|  1  1  0  0

Введите имя файла для вывода независимых множеств: C:\Users\Compluhter\OneDrive\Desktop\vivod.txt

Независимые множества:
{ 0, 1, 2 }
{ 0, 1 }
{ 0, 2 }
{ 0 }
{ 1, 2 }
{ 1 }
{ 2, 3 }
{ 2 }
{ 3 }
```

Рисунок 7 – Тестирование при вводе файла для сохранения.

Тестирование на ввод заведомо неверного пути к файлу представлено на рисунке 8.

```
Выберите режим работы:
1. Считать граф из файла
2. Сгенерировать случайный граф
1
Введите имя файла: C:\Users\Compluhter\OneDrive\Desktop\netfaila.txt

Попытка чтения файла: C:\Users\Compluhter\OneDrive\Desktop\netfaila.txt
Ошибка: файл 'C:\Users\Compluhter\OneDrive\Desktop\netfaila.txt' не найден!
```

Рисунок 8 – Тестирование при вводе заведомо неверного пути к файлу.

Проверка записи в файл случайной генерации графа, с указанием количества вершин представлен на рисунке 9.

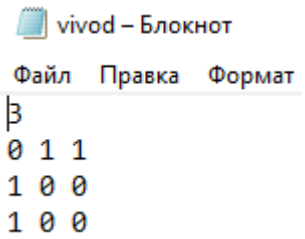

Рисунок 9 – Запись случайной генерации графа в файл.

Проверка записи в файл результатов работы программы

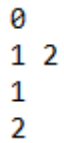

Рисунок 10 – Запись результатов работы программы в файл.

Таблица 2 – Поведение программы при тестировании.

Описание теста	**Ожидаемый результат**	**Полученный результат**
Выбор чтения графа с файла	Вывод пользователю предложения указать путь к файлу с графом	Верно
Выбор случайной генерации графа	Вывод пользователю предложение указать количество вершин графа	Верно
Ввод пути к файлу с пользовательским графом	Успешное открытие файла и дальнейшая обработка данных	Верно
Ввод заведомо ложного пути к файлу	Вывод сообщение об ошибке чтения файла	Верно
Ввод пути к файлу для записи результата работы программы	Успешное открытие файла и дальнейшая запись результатов работы	Верно

6. Ручной расчет задачи

Проведем проверку программы посредством ручных вычислений на примере графа с 3 вершинами (рисунок 11), сгенерированного автоматически в программе (рисунок 12).

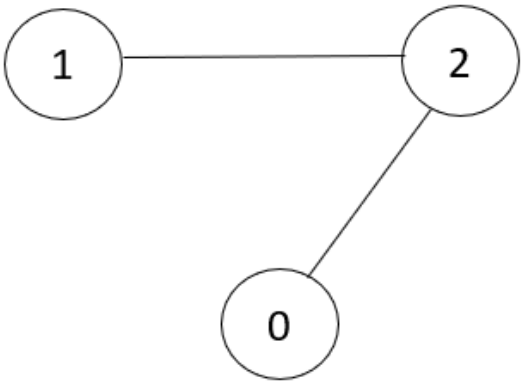

Рисунок 11 – Граф.

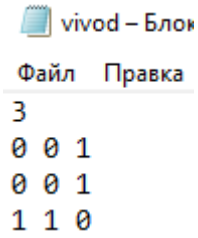

Рисунок 12 –Результат генерации.

Рассмотрим все множества и проверим, являются ли они независимыми: Множество {0; 1} не имеет общих ребер, оно может считаться независимым Множество {0; 2} имеет общее ребро, оно не может считаться независимым

Множество {1; 2} имеет общее ребро, оно не может считаться независимым

Множество {0; 1; 2} имеет общие ребра, оно не может считаться независимым

Каждая вершина по отдельности может считаться независимым множеством в том случае, если у нее нет петли.

Следовательно, результат ручных расчетов следующий:

{0}

{1}

{2}

{0; 1}

Ручной расчёт полностью совпадает с работой программы (рисунок 13).

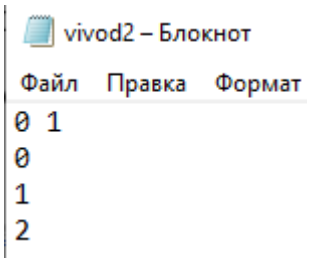

Рисунок 13 –Результат работы программы.

Заключение

Таким образом, в процессе создания данного проекта разработана программа, реализующая поиск всех независимых множеств графа в Microsoft Visual Studio 2022.

При выполнении данной курсовой работы были получены навыки разработки программ и освоены приемы создания двудольных графов, и динамических структур данных. Углублены знания языка программирования C#.

Недостатком разработанной программы является примитивный пользовательский интерфейс и высокая сложность алгоритма, что влечет за собой долгий просчёт для графов с большим количеством вершин.

Код написан максимально компактно и интерфейс удобен для использования. Полученные знания можно применять в дальнейшей деятельности.

Список литературы

1. Троелсен, Э. Язык программирования C# 7 и платформы .NET и .NET Core / Э. Троелсен. – Москва : Вильямс, 2018. – 1040 с. – ISBN 978-5-8459-2089-9.В. Л. Дольников О. П. Якимова. Основные алгоритмы на графах. 2011 г.

2. C# Basics: [сайт]. – URL: https://code-basics.com/ru/languages/csharp (дата обращения: 10.05.2024).

Приложение А.

Листинг программы.

```
using System;
using System.IO;

namespace IndependentSetsFinder
{
    class Program
    {
        private static int[,] graph;
        private static int numVertices;
        private const int MAX_VERTICES = 100;

        static void Main(string[] args)
        {
            Console.OutputEncoding = System.Text.Encoding.UTF8;

            Console.WriteLine("Программа поиска независимых множеств в графе");

Console.WriteLine("==========================================\n");

            bool running = true;

            while (running)
            {
                Console.WriteLine("\nГлавное меню:");
                Console.WriteLine("1. Считать граф из файла");
                Console.WriteLine("2. Сгенерировать случайный граф");
                Console.WriteLine("3. Найти независимые множества");
                Console.WriteLine("4. Выход");

                Console.Write("\nВыберите действие: ");
                string input = Console.ReadLine();

                switch (input)
                {
                    case "1":
                        LoadGraphFromFile();
                        break;
                    case "2":
                        GenerateRandomGraph();
                        break;
                    case "3":
                        if (graph == null)
                        {
                            Console.WriteLine("\nОшибка: граф не загружен!");
                            Console.WriteLine("Сначала загрузите или
сгенерируйте граф.");
                        }
                        else
                        {
                            FindIndependentSets();
                        }
                        break;
                    case "4":
                        running = false;
                        Console.WriteLine("\nЗавершение работы программы.");
                        break;
```

```
                    default:
                        Console.WriteLine("\nНеверный выбор. Пожалуйста,
выберите 1-4.");
                        break;
                }
            }
        }

        static void LoadGraphFromFile()
        {
            Console.Clear();
            Console.WriteLine("Загрузка графа из файла");
            Console.WriteLine("======================\n");

            Console.Write("Введите имя файла: ");
            string filename = Console.ReadLine();

            if (ReadGraphFromFile(filename))
            {
                Console.WriteLine("\nГраф успешно загружен.");
                ShowGraphMatrix();
            }
        }

        static bool ReadGraphFromFile(string filename)
        {
            try
            {
                if (!File.Exists(filename))
                {
                    Console.WriteLine("Файл не найден!");
                    return false;
                }

                using (StreamReader reader = new StreamReader(filename))
                {
                    string firstLine = reader.ReadLine();
                    if (firstLine == null)
                    {
                        Console.WriteLine("Файл пустой!");
                        return false;
                    }

                    if (!int.TryParse(firstLine.Trim(), out numVertices))
                    {
                        Console.WriteLine("Некорректный формат количества
вершин!");
                        return false;
                    }

                    if (numVertices <= 0 || numVertices > MAX_VERTICES)
                    {
                        Console.WriteLine("Количество вершин должно быть от 1
до " + MAX_VERTICES);
                        return false;
                    }

                    graph = new int[numVertices, numVertices];
```

```
                    for (int i = 0; i < numVertices; i++)
                    {
                        string line = reader.ReadLine();
                        if (line == null)
                        {
                            Console.WriteLine("Недостаточно строк в файле!
Ожидалось " + numVertices);
                            return false;
                        }

                        string[] values = line.Trim().Split(' ',
StringSplitOptions.RemoveEmptyEntries);

                        if (values.Length != numVertices)
                        {
                            Console.WriteLine("Ошибка в строке " + (i + 2) + ":
ожидалось " + numVertices + " чисел");
                            return false;
                        }

                        for (int j = 0; j < numVertices; j++)
                        {
                            if (!int.TryParse(values[j], out int value))
                            {
                                Console.WriteLine("Некорректное число: " +
values[j]);
                                return false;
                            }

                            if (value != 0 && value != 1)
                            {
                                Console.WriteLine("Значение должно быть 0 или
1: " + value);
                                return false;
                            }

                            graph[i, j] = value;
                        }
                    }
                }

                return true;
            }
            catch (Exception ex)
            {
                Console.WriteLine("Ошибка при чтении файла: " + ex.Message);
                return false;
            }
        }

        static void GenerateRandomGraph()
        {
            Console.Clear();
            Console.WriteLine("Генерация случайного графа");
            Console.WriteLine("============================\n");

            Console.Write("Введите количество вершин: ");
            if (!int.TryParse(Console.ReadLine(), out numVertices))
            {
```

```
                Console.WriteLine("Неверное количество вершин!");
                return;
            }

            if (numVertices <= 0 || numVertices > MAX_VERTICES)
            {
                Console.WriteLine("Количество вершин должно быть от 1 до " +
MAX_VERTICES);
                return;
            }

            Console.Write("Введите имя файла для сохранения графа: ");
            string filename = Console.ReadLine();

            if (GenerateRandomGraphFile(numVertices, filename))
            {
                Console.WriteLine("\nСлучайный граф успешно сгенерирован.");
                ShowGraphMatrix();
            }
        }

        static bool GenerateRandomGraphFile(int vertices, string filename)
        {
            try
            {
                numVertices = vertices;
                graph = new int[numVertices, numVertices];
                Random rand = new Random();

                string directory = Path.GetDirectoryName(filename);
                if (!string.IsNullOrEmpty(directory) &&
!Directory.Exists(directory))
                {
                    Directory.CreateDirectory(directory);
                }

                using (StreamWriter writer = new StreamWriter(filename))
                {
                    writer.WriteLine(numVertices);

                    int edgeCount = 0;
                    for (int i = 0; i < numVertices; i++)
                    {
                        for (int j = 0; j < numVertices; j++)
                        {
                            if (i == j)
                            {
                                graph[i, j] = 0;
                            }
                            else if (j > i)
                            {
                                int value = rand.Next(100) < 50 ? 1 : 0;
                                graph[i, j] = value;
                                graph[j, i] = value;

                                if (value == 1) edgeCount++;
                            }

                            writer.Write(graph[i, j]);
```

```
                        if (j < numVertices - 1) writer.Write(" ");
                    }
                    writer.WriteLine();
                }

                Console.WriteLine("Вершин: " + numVertices + ", Ребер: " +
edgeCount);
            }

            return true;
        }
        catch (Exception ex)
        {
            Console.WriteLine("Ошибка при генерации графа: " + ex.Message);
            return false;
        }
    }

    static void FindIndependentSets()
    {
        Console.Clear();
        Console.WriteLine("Поиск независимых множеств");
        Console.WriteLine("==========================\n");

        ShowGraphMatrix();

        Console.Write("\nВведите имя файла для сохранения результатов: ");
        string outputFile = Console.ReadLine();

        int[] subset = new int[MAX_VERTICES];
        int totalFound = 0;

        Console.WriteLine("\nНайденные независимые множества:");
        FindIndependentSetsRecursive(subset, 0, 0, outputFile, ref
totalFound);

        Console.WriteLine("Найдено множеств: " + totalFound);

        Console.WriteLine("\nНажмите любую клавишу для продолжения...");
        Console.ReadKey();
    }

    static void FindIndependentSetsRecursive(int[] subset, int index, int
count,
                                             string outputFile, ref int
totalFound)
    {
        if (index == numVertices)
        {
            bool independent = true;

            for (int i = 0; i < count && independent; i++)
            {
                for (int j = i + 1; j < count; j++)
                {
                    if (graph[subset[i], subset[j]] == 1)
                    {
                        independent = false;
                        break;
```

```
                        }
                    }
                }

                if (independent && count > 0)
                {
                    totalFound++;

                    Console.Write("{ ");
                    for (int i = 0; i < count; i++)
                    {
                        Console.Write(subset[i]);
                        if (i != count - 1) Console.Write(", ");
                    }
                    Console.WriteLine(" }");

                    try
                    {
                        using (StreamWriter writer = new
StreamWriter(outputFile, true))
                        {
                            for (int i = 0; i < count; i++)
                            {
                                writer.Write(subset[i]);
                                if (i != count - 1) writer.Write(" ");
                            }
                            writer.WriteLine();
                        }
                    }
                    catch (Exception ex)
                    {
                        Console.WriteLine("Ошибка при записи в файл: " +
ex.Message);
                    }
                }
                return;
            }

            subset[count] = index;
            FindIndependentSetsRecursive(subset, index + 1, count + 1,
outputFile, ref totalFound);
            FindIndependentSetsRecursive(subset, index + 1, count, outputFile,
ref totalFound);
        }

        static void ShowGraphMatrix()
        {
            if (graph == null)
            {
                Console.WriteLine("Граф не загружен.");
                return;
            }

            Console.WriteLine("Матрица смежности графа (" + numVertices + "
вершин):");
            Console.Write("   ");
            for (int j = 0; j < numVertices; j++)
            {
                Console.Write(j.ToString().PadLeft(3));
```

```
            }
            Console.WriteLine();

            Console.Write("   ");
            for (int j = 0; j < numVertices; j++)
            {
                Console.Write("---");
            }
            Console.WriteLine();

            for (int i = 0; i < numVertices; i++)
            {
                Console.Write(i.ToString().PadLeft(2) + "|");
                for (int j = 0; j < numVertices; j++)
                {
                    Console.Write(graph[i, j].ToString().PadLeft(3));
                }
                Console.WriteLine();
            }
        }
    }
}
```