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 множеств независимых рёбер

графа»

Выполнил:

студент группы 24ВВВ3

Азаров М.С.

Приняла, к.т.н., доцент:

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 Дорову Максиму Олеговичу Группа 21ИВТ-3

Тема_проекта Реализация алгоритма поиска независимых множеств рёбер графа

Исходные данные (технические требования) на проектирование

Разработка алгоритмов и программного обеспечения в соответствии с данными задачами курсового проекта.

Пояснительная записка должна содержать

1) Постановку задачи.
2) Теоретическую часть задания.
3) Описание алгоритма
4) Пример ручного расчёта задачи
5) Описание самой программы
6) Тесты.
7) Список лит-ры.
8) Листинг программы.
9) Результаты программы.

Объем работы по курсу

1. Расчетная часть

Ручной расчёт работы алгоритма

2. Графическая часть

Схема алгоритма в формате блок - схемы

3. Экспериментальная часть

Тестирование программы
Результаты работы программы на тестовых данных

Срок выполнения проекта по разделам

1 Исследование предметной области
2 Разработка алгоритмов программы
3 Разработка программы
4 Тестирование и доведение разработки программы
5 Оформление пояснительной записки
6
7
8

Дата выдачи задания "11" сентября

Дата защиты проекта "___" _____

Руководитель Юрова Ольга Викторовна

Задание получил "12" сентября 2025 г.

Студент Азаров Максим Сергеевич

.

Содержание

Реферат

Отчёт 31 стр, 14 рисунков, 2 таблицы, 1 приложение.
Ключевые слова: множества, паросочетания, матрица смежности, матрица инцидентности .

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

В работе рассмотрен способ рекурсивного поиска всех максимальных независимых множеств рёбер графа, который выражен при помощи матрицы инцидентности, основанная на матрице смежности. Для эффективной проверки смежности рёбер используется матрица инцидентности, построенная на основе матрицы смежности. Программа работает в двух режимах: со случайной генерацией матрицы или с её ручным вводом, и пользователь имеет выбор, какой из вариантов использовать. Результат работы программы — матрица смежности, матрица инцидентности, размер и состав найденного паросочетания - сохраняется в указанный пользователем выходной файл и одновременно выводится в консоль.

Введение

Курсовая работа посвящена разработке и реализации алгоритма жадного поиска максимального независимого множества рёбер в неориентированных графах. Графы являются фундаментальной структурой данных и применяются для моделирования сетей связи, социальных взаимодействий, молекулярных структур, транспортных маршрутов и множества других систем, где ключевое значение имеют связи между объектами.

Ключевым понятием работы выступает независимое множество рёбер (максимальное паросочетание) — такое подмножество рёбер графа, в котором никакие два ребра не имеют общей вершины. Подобные задачи оптимизации востребованы в различных прикладных областях: при распределении задач, планировании ресурсов без конфликтов, построении коммуникационных сетей и в биоинформатике, где требуется выделять непересекающиеся связи.

Алгоритм поиска максимального независимого множества рёбер графа, реализованный в рамках данной работы, использует жадный подход на основе матрицы инцидентности. Программа последовательно анализирует каждое ребро, выбирая доступные рёбра и блокируя смежные с ними, что позволяет эффективно формировать максимальное паросочетание. Результаты работы - матрица смежности, матрица инцидентности и найденное независимое множество рёбер - выводятся как в консоль, так и сохраняются в указанный пользователем выходной файл.

В качестве среды разработки была выбрана Microsoft Visual Studio 2022, а язык программирования - C, который остаётся широко используемым и эффективным инструментом для создания высокопроизводительных решений.

Целью данной курсовой работы является разработка программы на языке C, реализующей алгоритм поиска максимального независимого множества рёбер графа на основе матрицы инцидентности, с возможностью выбора способа ввода графа и сохранением всех результатов в выходной файл.

1. Постановка задачи

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

Исходный граф в программе задаётся матрицей смежности. Программа работает в двух режимах: со случайной генерацией симметричной матрицы смежности или с ручным вводом элементов верхнего треугольника. Пользователь выбирает нужный вариант и указывает размер графа. При генерации результат сохраняется в указанный пользователем текстовый файл. Итоги работы программы - матрицы смежности и инцидентности, найденные паросочетания - также сохраняются в этот же файл. Устройство ввода - клавиатура. Устройство вывода - монитор.

2. Теоретическая часть задания

Граф *G* (рисунок 1) задаётся множеством вершин X_1, X_2, X_3 и так далее и множеством рёбер, которые соединяют между собой определённые вершины. Ребра из множества A не ориентированы. Такой граф называется неориентированным.

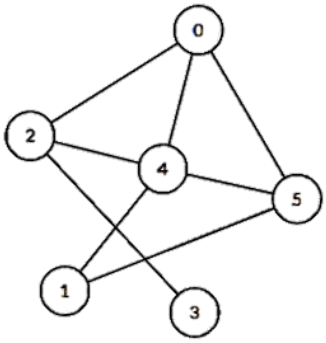

Рисунок 1 – Пример графа.

При представлении графа матрицей смежности информация о рёбрах хранится в квадратной матрице, где присутствие пути от одной вершиной обозначается единицей, иначе нулём. Матрица инцидентностей представляет матрицу, с количеством строк равному количеству вершин, а столбцов количеством рёбер. 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).

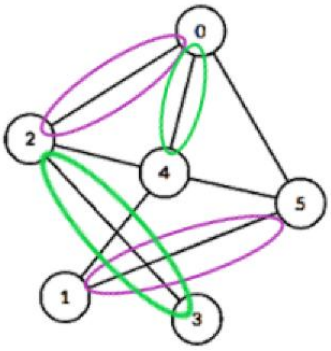

Рисунок 2 - Пример независимых рёбер.

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 find_matching:

```
ИНИЦИАЛИЗИРОВАТЬ matching[reb_count] = 0
ИНИЦИАЛИЗИРОВАТЬ available[reb_count] = 1
matching_size = 0
ДЛЯ i от 0 ДО reb_count-1:
    ЕСЛИ available[i]:
        matching[matching_size++] = i
        ДЛЯ j от i+1 ДО reb_count-1:
            ЕСЛИ available[j] И edges_share_vertex(i, j):
                available[j] = 0
            КОНЕЦ ЕСЛИ
        КОНЕЦ ДЛЯ
    КОНЕЦ ЕСЛИ
```

КОНЕЦ ДЛЯ

Функция edges_share_vertex(edge1, edge2) проверяет наличие общей вершины между двумя рёбрами по матрице инцидентности. После выполнения алгоритма результаты выводятся в консоль функцией print_matching_console() и сохраняются в файл функцией print_result_file().

Полный код программы можно увидеть в Приложении А.

Ниже представлен псевдокод функции menu_naiti:

```
ФУНКЦИЯ menu_naiti():
  ЕСЛИ graf не существует:
    ВЫВЕСТИ "Сначала создайте граф"
    ВЕРНУТЬ
  КОНЕЦ ЕСЛИ
  ВЫВЕСТИ матрицу смежности в консоль
  ЗАПИСАТЬ матрицу смежности в файл
  ПОСТРОИТЬ матрицу инцидентности:
    reb_count = 0
    ДЛЯ i от 0 ДО n-1:
      ДЛЯ j от i+1 ДО n-1:
        ЕСЛИ graf[i][j] = 1:
          reb_count++
        КОНЕЦ ЕСЛИ
      КОНЕЦ ДЛЯ
    КОНЕЦ ДЛЯ
```

```
    ВЫДЕЛИТЬ память для inc[n][reb_count] и available[reb_count]
    edge_id = 0
    ДЛЯ i от 0 ДО n-1:
        ДЛЯ j от i+1 ДО n-1:
            ЕСЛИ graf[i][j] = 1:
                inc[i][edge_id] = 1
                inc[j][edge_id] = 1
                edge_id++
            КОНЕЦ ЕСЛИ
        КОНЕЦ ДЛЯ
    КОНЕЦ ДЛЯ
КОНЕЦ ПОСТРОЕНИЯ

ЕСЛИ reb_count = 0:
    ВЫВЕСТИ "В графе нет рёбер"
    ВЕРНУТЬ
КОНЕЦ ЕСЛИ

ВЫВЕСТИ матрицу инцидентности в консоль
ЗАПИСАТЬ матрицу инцидентности в файл

ВЫВЕСТИ "Жадный поиск максимального паросочетания"
```

ВЫЗВАТЬ find_matching()

ВЫВЕСТИ результат паросочетания в консоль

ЗАПИСАТЬ результат в файл

ВЫВЕСТИ "Результаты сохранены в файл: filename"

КОНЕЦ ФУНКЦИИ

Эта функция координирует весь процесс: от построения матриц до вывода результатов.

Полный код программы можно увидеть в Приложении А.

4. Описание программы

Данная программа реализована на языке программирования C. C — мощный системный язык программирования, обеспечивающий прямой доступ к памяти и высокую производительность, что делает его идеальным для реализации алгоритмов работы с графами.

Программа создана в виде консольного приложения для Microsoft Visual Studio 2022, работающего под Windows.

Работа программы начинается с запроса имени выходного файла для сохранения результатов. Затем пользователь выбирает режим работы из меню.

Часть кода, ответственная за выбор функций:

```
printf("\n--- MENYU ---\n");
printf("1 - Avto generaciya\n");
printf("2 - Ruchnoy vvod\n");
printf("3 - Naiti nezavisimykh reber\n");
printf("4 - Vykhod\n");
```

Эта часть кода выводит меню с четырьмя пунктами:

1) Авто генерация матрицы;

2) Ручной ввод матрицы;

3) Поиск независимых рёбер;

4) Выход.

Выбор пользователя сохраняется в переменную choice.

Часть кода, ответственная за автогенерацию графа (menu_avto):

```
printf("Razmer grafa n x n: ");
scanf_s("%d", &n);
allocate_graph();
srand((unsigned)time(NULL));
for (int i = 0; i < n; i++)
    for (int j = i + 1; j < n; j++)
        graf[i][j] = graf[j][i] = rand() % 2;
```

Пользователя просят ввести размер графа n, выделяется память под матрицу смежности graf[n][n], затем верхний треугольник заполняется случайными значениями 0/1 с симметризацией.

Часть кода, ответственная за ручной ввод графа (menu_ruchnoy):

```
printf("Razmer grafa n x n: ");
scanf_s("%d", &n);
allocate_graph();
printf("Vvedite matricu smezhnosti (0/1):\n");
for (int i = 0; i < n; i++) {
    for (int j = i + 1; j < n; j++) {
        int val;
        do {
            printf("a[%d][%d] = ", i + 1, j + 1);
            scanf_s("%d", &val);
        } while (val < 0 || val > 1);
        graf[i][j] = graf[j][i] = val;
    }
}
```

Пользователя просят ввести размер графа и элементы верхнего треугольника матрицы смежности с проверкой корректности (0 или 1).

Часть кода, отвечающая за обработку неверно выбранного режима работы:

```
default:
    printf("Nekorrektnyy vybor!\n");
```

При некорректном выборе выводится сообщение об ошибке.

Часть кода, ответственная за основной алгоритм поиска (menu_naiti):

```
    if (!graf) {
        printf("Snachala sozdaite graf (punkt 1 ili 2).\n");
        return;
    }

    print_adjacency_matrix_console();
    print_adjacency_matrix_file();

    build_incidence_matrix();
    if (reb_count == 0) {
        printf("\nV grafe net reber, nezavisimykh reber net.\n");
        return;
    }

    print_incidence_matrix_console();
    print_incidence_matrix_file();

    printf("\nAlgoritm: zhadnyy poisk maksimalnogo nezavisimogo mnozhestva
reber\n");
    find_matching();
    print_matching_console();
    print_result_file();

    printf("\nRezultaty sokhranyeny v fayle: %s\n", filename);
}
```

Эта часть приводится в действие при выборе пункта 3. Выводится матрица смежности, строится и выводится матрица инцидентности, выполняется жадный алгоритм find_matching(), результаты сохраняются в файл и выводятся в консоль. Завершается сообщением о сохранении результатов в файл.

На рисунке 3 можно увидеть что перед началом работы основной программы нужно вести название файла для сохранения дальнейших результатов алгоритма.

Рисунок 3 – Ввод названия файла

На рисунке 4 можно увидеть оформление начального запроса и дальнейшие действия с ним.

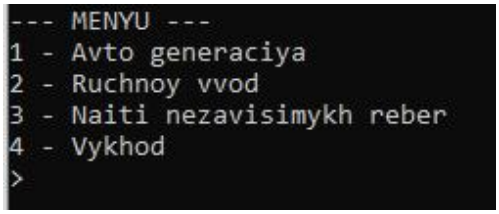

Рисунок 4 – Стартовое меню программы.

Выбор пункта с авто генерацией графа и запрос на ввод количества вершин представлены на рисунке 5.

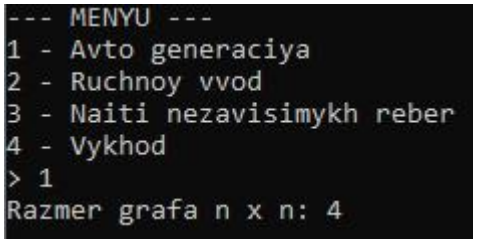

Рисунок 5 – Авто генерация графа.

Выбор пункта с ручным вводом матрицы смежности показан на рисунке 6.

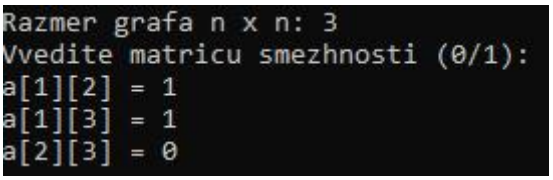

Рисунок 6 – Ручной ввод матрицы.

Ввод названия файла для записи результата работы программы, а также вывод результата (матрица смежности, матрица инцидентности и количество независимых множеств рёбер) в консоль показаны на рисунке 7.

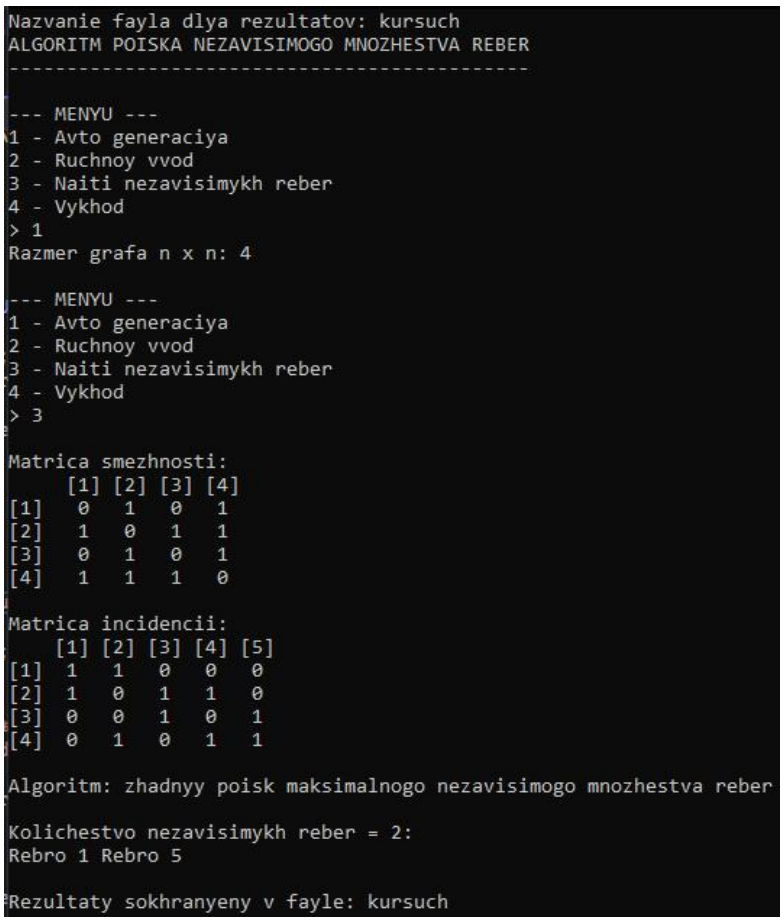

Рисунок 7 – Поиск независимых множеств рёбер.

5. Тестирование

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

Тестирование проводилось поэтапно: в процессе разработки и после завершения написания программы. В ходе тестирования было выявлено и исправлено множество проблем, связанных с динамическим выделением памяти, построением матрицы инцидентности, алгоритмом поиска паросочетания и 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 2022	Запрос имени файла для результатов
Ввод имени файла для результатов	Программа запущена	Ввести корректное имя файла (например, "res1")	Создание файла, вывод основного меню с 4 пунктами
Ввод некорректного выбора в меню	Открыто основное меню	Ввести число не из диапазона 1-4 (например, 5)	Сообщение "Некорректный выбор!" и повторный вывод меню

Продолжение таблицы 1

Описание теста	**Предусловие**	**Тестирование**	**Ожидаемый результат**
Выбор автогенерации	Открыто основное меню	Выбрать пункт 1 (Авто генерация)	Запрос размера графа n x n
Ввод размера графа	Запрошен размер графа	Ввести положительное целое число (например, 4)	Создание графа, возврат в главное меню
Ввод некорректного размера	Запрошен размер графа	Ввести нечисловое значение или 0	Некорректный размер! Введите положительное число:
Выбор ручного ввода	Открыто основное меню	Выбрать пункт 2(ручной ввод)	Запрос размера графа n x n
Ввод матрицы смежности	Введён размер графа	Последовательно вводить 0/1 для каждой пары вершин	Построение графа, возврат в главное меню
Ввод некорректных значений	Запрошен элемент матрицы	Ввести число не 0/1	Ошибка! Введите только 0/1:

Продолжение таблицы 1

Описание теста	**Предусловие**	**Тестирование**	**Ожидаемый результат**
Поиск без создания графа	Открыто основное меню, граф не создан	Выбрать пункт 3 (Найти независимых рёбер)	Сообщение "Сначала создайте граф (пункт 1 или 2)"
Поиск с пустым графом	Создан граф без рёбер (все 0)	Выбрать пункт 3	Вывод матрицы смежности, сообщение "В графе нет рёбер, независимых рёбер нет"
Поиск с нормальным графом	Создан граф с рёбрами	Выбрать пункт 3	Последовательный вывод: матрицы смежности, матрицы инцидентности, результата алгоритма
Проверка записи в файл	Успешно выполнен поиск	Проверить файл результатов	Файл содержит: заголовок, матрицу смежности, матрицу инцидентности, количество независимых рёбер

Проверка записи в файл авто генерации графа, представлена на рисунке 8.

Рисунок 8 – Запись авто генерации графа в файл.

Проверка записи в файл результатов работы программы, представлена на рисунке 9.

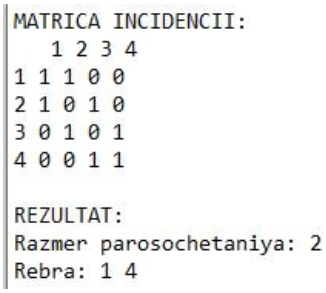

Рисунок 9 – Запись результатов работы программы в файл.

Таблица 2 – Поведение программы при тестировании.

Описание теста	**Ожидаемый результат**	**Полученный результат**
Ввод корректного имени файла	Успешное создание файла, вывод главного меню с 4 пунктами	Верно
Ввод некорректного имени файла	Ошибка открытия файла! Попробуйте другое название: Название файла для результатов:	Верно
Выбор авто генерации графа	Запрос размера графа n x n	Верно
Выбор ручного ввода графа	Запрос размера графа n x n	Верно
Ввод некорректного значения элемента матрицы (не 0/1)	Повторный запрос до ввода корректного значения	Верно
Выбор поиска без создания графа (пункт 3)	Сообщение "Сначала создайте граф (пункт 1 или 2)"	Верно

6. Ручной расчёт задачи

Проведём проверку программы посредством ручных вычислений на примере графа с 4 вершинами (рисунок 10).

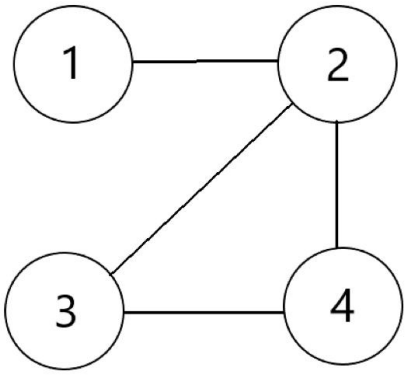

Рисунок 10 – Граф.

```
   1 2 3 4
1  0 1 0 0
2  1 0 1 1
3  0 1 0 1
4  0 1 1 0
```

Рисунок 11 – Результат генерации.

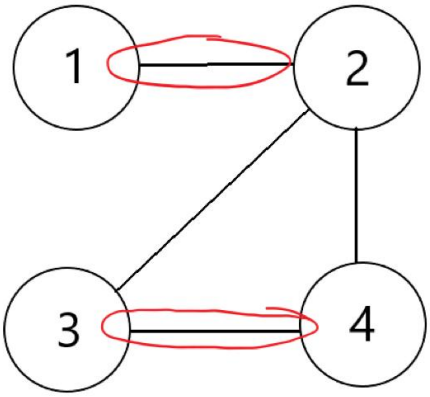

Рисунок 12 – Граф с выделенными рёбрами.

Прямо по графу можно увидеть два независимых ребра (рисунок 12), но неизвестно под какими цифрами, берём их произвольно и подписываем ребра на графе (рисунок 13):

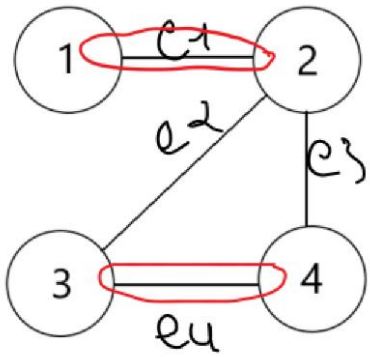

Рисунок 13 –Граф с подписанными рёбрами.

Так же можно найти независимые множества рёбер по матрице инцидентности (рисунок 14) для этого нужно сначала её построить:

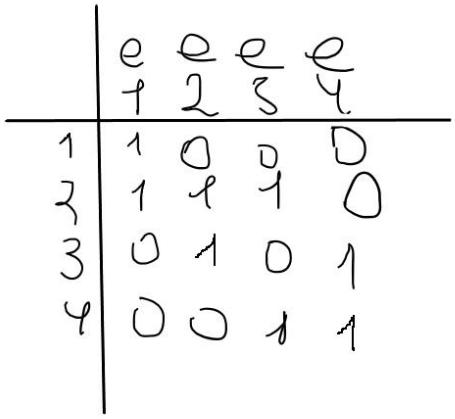

Рисунок 14 – Матрица инцидентности.

По этой матрице мы видим что вершины рёбер e1 и рёбер e4 никак не пересекаются, следовательно они независимы. Сравниваем с результатом программы (рисунок15), всё сходится, программа работает верно.

```
Matrica smezhnosti:
     [1] [2] [3] [4]
[1]   0   1   0   0
[2]   1   0   1   1
[3]   0   1   0   1
[4]   0   1   1   0

Matrica incidencii:
    [1] [2] [3] [4]
[1]  1   0   0   0
[2]  1   1   1   0
[3]  0   1   0   1
[4]  0   0   1   1

Algoritm: poisk maksimalnogo nezavisimogo mnozhestva reber

Kolichestvo nezavisimykh reber = 2:
Rebro 1 Rebro 4
```

Рисунок 15 –Результат работы программы.

Заключение

Таким образом, в процессе создания данного проекта разработана программа, реализующая поиск множеств рёбер графа в Microsoft Visual Studio 2022 на языке программирования C.

При выполнении данной курсовой работы были получены навыки разработки программ и освоены построение матрицы инцидентности, и реализации алгоритмов графов. Углублены знания языка программирования C.

Недостатком разработанной программы является примитивный пользовательский интерфейс и высокая сложность алгоритма, что влечёт 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. Кнут Д. Искусство программирования, том 3. Сортировка и поиск. — М.: Вильямс, 2007. — 832 с (Алгоритмы работы с графами, матрицы смежности и инцидентности).

2. Кормен Т., Лейзерсон Ч., Ривест Р., Стайн К. Алгоритмы: построение и анализ. — М.: Вильямс, 2013. — 1328 с (Жадные алгоритмы, паросочетания, независимые множества рёбер).

3. Дейкстра Э. Записки по программированию. — М.: Мир, 1981. — 320 с. (Структурированное программирование, управление памятью в С)

4. Строджер Дж., Хенси У. Программирование на языке С. — М.: Бином, 2018. — 432 с (Динамические массивы, указатели, работа с файлами в С)

5. Ахманов С.А., Семенов В.И. Математическое моделирование транспортных систем. — М.: ЛКИ, 2006. — 288 с. (Применение графов и паросочетаний в транспортных задачах)

Приложение А.

Листинг программы.

```
#include <stdio.h>
#include <stdlib.h>
#include <Windows.h>
#include <locale.h>
#include <time.h>
#include <string.h>

int n;
int** graf = NULL;
int** inc = NULL;
int reb_count = 0;
int* matching = NULL;
int matching_size = 0;
int* available = NULL;
FILE* output_file = NULL;
char filename[256];

void free_graph() {
    if (graf) {
        for (int i = 0; i < n; i++) {
            if (graf[i]) free(graf[i]);
        }
        free(graf);
        graf = NULL;
    }
    if (inc) {
        for (int i = 0; i < n; i++) {
            if (inc[i]) free(inc[i]);
        }
        free(inc);
        inc = NULL;
    }
    if (matching) {
        free(matching);
        matching = NULL;
    }
    if (available) {
        free(available);
        available = NULL;
    }
    reb_count = 0;
    matching_size = 0;
}

void print_adjacency_matrix_console() {
    if (!graf) {
        printf("\nGraf eshche ne sozdan.\n");
        return;
    }
    printf("\nMatrica smezhnosti:\n");
    printf("     ");
    for (int i = 0; i < n; i++) printf("[%d] ", i + 1);
    printf("\n");
    for (int i = 0; i < n; i++) {
        printf("[%d] ", i + 1);
        for (int j = 0; j < n; j++) {
            printf("%3d ", graf[i][j]);
```

```
        }
        printf("\n");
    }
}

void print_adjacency_matrix_file() {
    if (!graf || !output_file) return;
    fprintf(output_file, "\nMATRICA SMEZHNOSTI:\n");
    fprintf(output_file, "   ");
    for (int i = 0; i < n; i++) fprintf(output_file, "%d ", i + 1);
    fprintf(output_file, "\n");
    for (int i = 0; i < n; i++) {
        fprintf(output_file, "%d  ", i + 1);
        for (int j = 0; j < n; j++) {
            fprintf(output_file, "%d ", graf[i][j]);
        }
        fprintf(output_file, "\n");
    }
}

void allocate_graph() {
    free_graph();
    graf = (int**)malloc(n * sizeof(int*));
    for (int i = 0; i < n; i++) {
        graf[i] = (int*)calloc(n, sizeof(int));
    }
}

void allocate_incidence() {
    inc = (int**)malloc(n * sizeof(int*));
    for (int i = 0; i < n; i++) {
        inc[i] = (int*)calloc(reb_count, sizeof(int));
    }
    available = (int*)calloc(reb_count, sizeof(int));
}

void build_incidence_matrix() {
    reb_count = 0;
    for (int i = 0; i < n; i++)
        for (int j = i + 1; j < n; j++)
            if (graf[i][j]) reb_count++;

    if (reb_count == 0) {
        printf("\nV grafe net reber.\n");
        return;
    }

    allocate_incidence();

    int edge_id = 0;
    for (int i = 0; i < n; i++) {
        for (int j = i + 1; j < n; j++) {
            if (graf[i][j]) {
                inc[i][edge_id] = 1;
                inc[j][edge_id] = 1;
                edge_id++;
            }
        }
    }
}

void print_incidence_matrix_console() {
```

```
    if (!inc) {
        printf("\nMatrica incidencii eshche ne postroena.\n");
        return;
    }
    printf("\nMatrica incidencii:\n");
    printf("    ");
    for (int i = 0; i < reb_count; i++) printf("[%d] ", i + 1);
    printf("\n");
    for (int i = 0; i < n; i++) {
        printf("[%d]", i + 1);
        for (int j = 0; j < reb_count; j++) {
            printf("%3d ", inc[i][j]);
        }
        printf("\n");
    }
}

void print_incidence_matrix_file() {
    if (!inc || !output_file) return;
    fprintf(output_file, "\nMATRICA INCIDENCII:\n");
    fprintf(output_file, "    ");
    for (int i = 0; i < reb_count; i++) fprintf(output_file, "%d ", i + 1);
    fprintf(output_file, "\n");
    for (int i = 0; i < n; i++) {
        fprintf(output_file, "%d ", i + 1);
        for (int j = 0; j < reb_count; j++) {
            fprintf(output_file, "%d ", inc[i][j]);
        }
        fprintf(output_file, "\n");
    }
}

int edges_share_vertex(int edge1, int edge2) {
    for (int v = 0; v < n; v++) {
        if (inc[v][edge1] && inc[v][edge2]) return 1;
    }
    return 0;
}

void find_matching() {
    if (!inc || reb_count == 0) {
        printf("\nNet reber dlya poiska parosochetaniya.\n");
        return;
    }

    if (matching) free(matching);
    matching = (int*)calloc(reb_count, sizeof(int));

    for (int i = 0; i < reb_count; i++) available[i] = 1;
    matching_size = 0;

    for (int i = 0; i < reb_count; i++) {
        if (available[i]) {
            matching[matching_size++] = i;
            for (int j = i + 1; j < reb_count; j++) {
                if (available[j] && edges_share_vertex(i, j)) {
                    available[j] = 0;
                }
            }
        }
    }
}
```

```
void print_matching_console() {
    if (!matching) {
        printf("\nParosochetanie eshche ne naideno.\n");
        return;
    }
    printf("\nKolichestvo nezavisimykh reber = %d:\n", matching_size);
    for (int i = 0; i < matching_size; i++) {
        printf("Rebro %d ", matching[i] + 1);
    }
    printf("\n");
}

void print_result_file() {
    if (!output_file || !matching) return;
    fprintf(output_file, "\nREZULTAT:\n");
    fprintf(output_file, "Razmer parosochetaniya: %d\n", matching_size);
    fprintf(output_file, "Rebra: ");
    for (int i = 0; i < matching_size; i++) {
        fprintf(output_file, "%d ", matching[i] + 1);
    }
    fprintf(output_file, "\n");
}

void menu_avto() {
    printf("Razmer grafa n x n: ");
    while (scanf_s("%d", &n) != 1 || n <= 0) {
        printf("Nekorrektnyy razmer! Vvedite polozhitelnoe chislo: ");
        while (getchar() != '\n');
    }
    allocate_graph();
    srand((unsigned)time(NULL));
    for (int i = 0; i < n; i++)
        for (int j = i + 1; j < n; j++)
            graf[i][j] = graf[j][i] = rand() % 2;
    if (output_file)
        fprintf(output_file, "\n--- Avtogen eraciya ---\n");
}

void menu_ruchnoy() {
    printf("Razmer grafa n x n: ");
    while (scanf_s("%d", &n) != 1 || n <= 0) {
        printf("Nekorrektnyy razmer! Vvedite polozhitelnoe chislo: ");
        while (getchar() != '\n');
    }
    allocate_graph();
    printf("Vvedite matricu smezhnosti (0/1):\n");
    for (int i = 0; i < n; i++) {
        for (int j = i + 1; j < n; j++) {
            int val;
            do {
                printf("a[%d][%d] = ", i + 1, j + 1);
                if (scanf_s("%d", &val) != 1 || val < 0 || val > 1) {
                    printf("Oshibka! Vvedite TOLKO 0 ili 1: ");
                    val = 2;
                    while (getchar() != '\n');
                }
            } while (val < 0 || val > 1);
            graf[i][j] = graf[j][i] = val;
        }
    }
    if (output_file)
```

```
        fprintf(output_file, "\n--- Ruchnoy vvod ---\n");
}

int open_output_file() {
    while (1) {
        printf("Nazvanie fayla dlya rezultatov: ");
        scanf_s("%s", filename, (unsigned)_countof(filename));

        int has_cyrillic = 0;
        for (int i = 0; filename[i]; i++) {
            if (filename[i] >= 0xC0 && filename[i] <= 0xFF) {
                has_cyrillic = 1;
                break;
            }
        }

        if (has_cyrillic) {
            printf("Preduprezhdenie! Ispolzuyte latinskie");
            while (getchar() != '\n');
            continue;
        }

        errno_t err = fopen_s(&output_file, filename, "w");
        if (err == 0 && output_file) {
            return 1;
        }
        printf("Oshibka otkrytiya fayla! Poprobuyte drugoe imya: ");
        while (getchar() != '\n');
    }
    return 0;
}

void menu_naiti() {
    if (!graf) {
        printf("Snachala sozdaite graf (punkt 1 ili 2).\n");
        return;
    }

    print_adjacency_matrix_console();
    print_adjacency_matrix_file();

    build_incidence_matrix();
    if (reb_count == 0) {
        printf("\nV grafe net reber, nezavisimykh reber net.\n");
        return;
    }

    print_incidence_matrix_console();
    print_incidence_matrix_file();

    printf("\nAlgoritm: poisk maksimalnogo nezavisimogo mnozhestva reber\n");
    find_matching();
    print_matching_console();
    print_result_file();

    printf("\nRezultaty sokhranyeny v fayle: %s\n", filename);
}

int main() {
    setlocale(LC_ALL, "Russian");

    open_output_file();
```

```
    fprintf(output_file, "ALGORITM POISKA NEZAVISIMOGO MNOZHESTVA REBER\n");
    fprintf(output_file, "----------------------------------------\n");

    printf("ALGORITM POISKA NEZAVISIMOGO MNOZHESTVA REBER\n");
    printf("----------------------------------------\n");

    int choice;
    do {
        printf("\n--- MENYU ---\n");
        printf("1 - Avto generaciya\n");
        printf("2 - Ruchnoy vvod\n");
        printf("3 - Naiti nezavisimykh reber\n");
        printf("4 - Vykhod\n");
        printf("> ");

        if (scanf_s("%d", &choice) != 1) {
            printf("Nekorrektnyy vvod! Vvedite chislo.\n");
            while (getchar() != '\n');
            continue;
        }

        switch (choice) {
        case 1:
            menu_avto();
            break;
        case 2:
            menu_ruchnoy();
            break;
        case 3:
            menu_naiti();
            break;
        case 4:
            printf("Vykhod\n");
            break;
        default:
            printf("Nekorrektnyy vybor! Vvedite 1-4.\n");
        }
    } while (choice != 4);

    free_graph();
    if (output_file) fclose(output_file);
    return 0;
}
```