

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

к курсовому проектированию
по курсу «Логика и основы алгоритмизации
в инженерных задачах»

на тему «Реализация алгоритма поиска максимальных паросочетаний.»

20.12.22
отлично
фед

Выполнил:

студент группы 21ВВ4
Нагорная Д. А.

Принял:

Акифьев И. В.
Юрова О. В.

"УТВЕРЖДАЮ"

Зав. кафедрой ВТ

«___» _____ 20__

ЗАДАНИЕ

на курсовое проектирование по курсу

Поиск и основы алгоритмизации в интернетных задачах.
Студенту Чалдрной Дарье Александровне Группа 21ВВ4
Тема проекта Реализация алгоритма поиска
максимальных пересечений.

Исходные данные (технические требования) на проектирование

1. Текстовое или графическое меню.
2. Возможность задания пользователем размера графа (множества).
3. Возможность выбора графического (случайного) или ручного (с клавиатуры или из файла) задания графа (элементов множества).
4. Возможность сохранения работы программы.

Объем работы по курсу

1. Расчетная часть

Разработка программы.

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 теоретической части курсовой работы
- 2 Разработка алгоритмов программы
- 3 Разработка программы
- 4 Тестирование и завершение разработки программы
- 5 Оформление пояснительной записки.
- 6
- 7
- 8

Дата выдачи задания "10" сент. 2022

Дата защиты проекта " " "

Руководитель Ирина Г.В. Фро

Задание получил "10" сент 2022г.

Студент Чонорная Дарья Александровна

Содержание

<u>Реферат</u>	5
<u>Введение</u>	6
1. <u>Постановка задачи</u>	7
2. <u>Теоритическая часть задания</u>	8
3. <u>Описание алгоритма программы</u>	10
4. <u>Описание программы</u>	13
5. <u>Тестирование</u>	17
6. <u>Ручной расчёт задачи</u>	21
<u>Заключение</u>	23
<u>Список литературы</u>	24
<u>Приложение А. Листинг программы</u>	25

Реферат

Отчет 30 стр, 12 рисунков.

ГРАФ, ТЕОРИЯ ГРАФОВ, НЕОРИЕНТИРОВАННЫЙ ГРАФ, РЕБЕРНЫЙ ГРАФ, НЕЗАВИСИМЫЕ МНОЖЕСТВА ВЕРШИН, ПАРОСОЧЕТАНИЕ.

Цель исследования – разработка программы, осуществляющая нахождение максимальных паросочетаний, используя алгоритм с возвратом независимых множеств вершин реберного графа.

В работе рассмотрен алгоритм, на основе которого находятся максимальные паросочетания неориентированного графа. При помощи данного алгоритма возможно решение на оптимизацию или задачи о назначениях.

Введение

Среди проблем связанных с решением логических задач, пристальное внимание исследователей привлекает вопрос о применении к данному роду задач теории графов.

Теория графов в настоящее время является интенсивно развивающимся разделом дискретной математики. Это объясняется тем, что в виде графовых моделей описываются многие объекты и ситуации: коммуникационные сети, схемы электрических и электронных приборов, химические молекулы, отношения между людьми и многое другое.

Графовые задачи обладают рядом достоинств, позволяющих их использовать для развития воображения и улучшения логического мышления. Графовые задачи допускают изложение в занимательной, игровой форме.

Как уже было сказано, графы имеют очень широкое применение: с их помощью выбирают наиболее выгодное расположение зданий, графами представлены схемы метро.

Некоторые примеры применения графов:

1. Можно составить граф любой позиционной игры: шахмат, шашек, «крестиков – ноликов».
2. Лабиринт. Исследовать лабиринт - это найти путь в этом графе.
3. Блок-схема программы. Графами являются блок – схемы программ для ЭВМ, а также любые электрические цепи или электрическая сеть.

Список применения теории графов достаточно велик, но в данной курсовой работе теория графов используется для реализации алгоритма поиска максимальных паросочетаний.

Постановка задачи

Требуется разработать программу, которая позволит найти все максимальные паросочетания в заданном графе, используя алгоритм с возвратом для нахождения независимых множеств вершин реберного графа.

Исходный граф в программе должен задаваться матрицей смежности. Программа должна работать так, чтобы пользователь вводил количество вершин в матрице смежности. После обработки данных пользователю предлагается сделать выбор вывода матрицы: генерация матрицы, ввод элементов матрицы в консоле и чтения матрицы из файла. Далее исходный граф должен преобразовываться в реберный и также выводиться на экран. А затем программа должна находить все максимальные независимые множества, которые и будут являться максимальными паросочетаниями исходного графа. Результат программы должен быть сохранен в файл, название которого задается в консоле.

Исходный граф неориентированный и не имеет петель.

Устройство ввода – клавиатура и мышь.

Теоретическая часть задания

Графом называется набор точек (эти точки называются вершинами), некоторые из которых объявляются смежными (или соседними). Считается, что смежные вершины соединены между собой ребрами (или дугами). Таким образом, ребро определяется парой вершин. Два ребра, у которых есть общая вершина, также называются смежными (или соседними).

В теории графов рёберным графом $L(G)$ неориентированного графа G называется граф $L(G)$, представляющий соседство рёбер графа G .

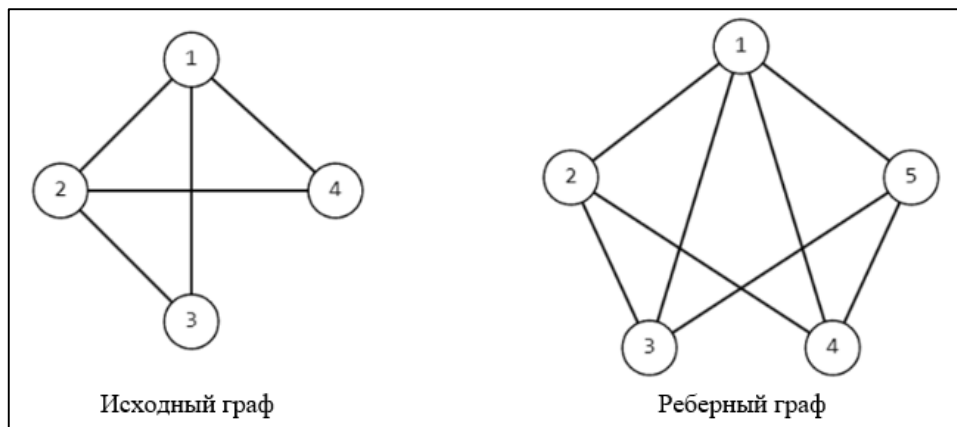


Рисунок 1 – Пример реберного графа

Независимое множество вершин (известное также как внутренне устойчивое множество) есть множество вершин графа G , такое, что любые две вершины в нем не смежны (т.е. никакая пара вершин не соединена ребром).

Внутренне устойчивое множество называется максимальным, если оно не является собственным подмножеством некоторого другого независимого множества.

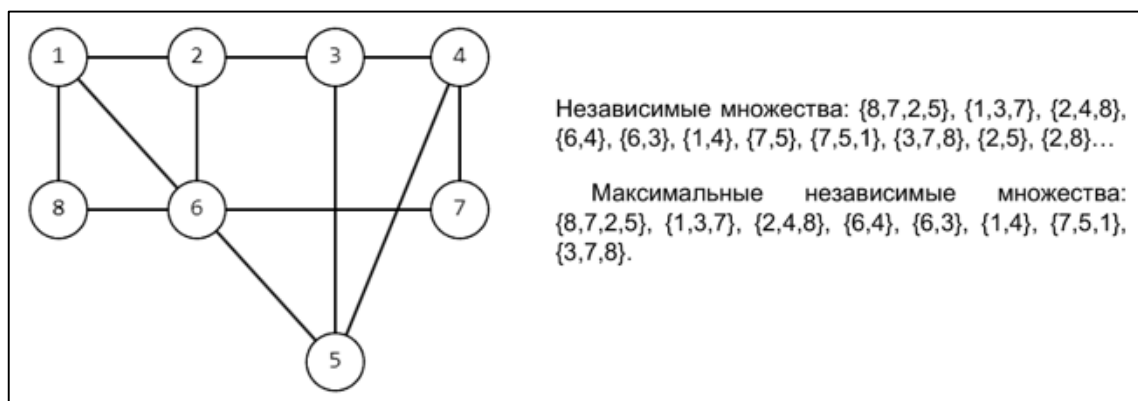


Рисунок 2 – Пример независимых множеств вершин и максимальных независимых множеств

В теории графов паросочетание или независимое множество рёбер в графе — это набор попарно несмежных рёбер.

Максимальное паросочетание — это такое паросочетание M в графе G , которое не содержится ни в каком другом паросочетании этого графа, то есть к нему невозможно добавить ни одно ребро, которое бы являлось несмежным ко всем рёбрам паросочетания.

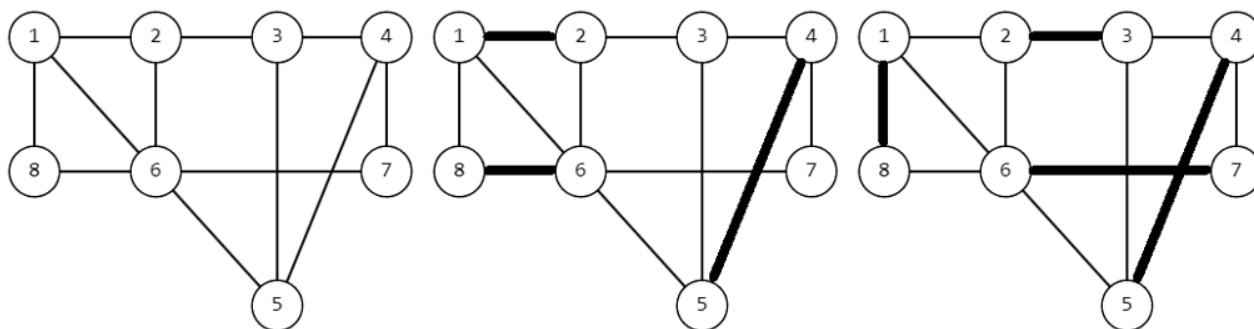


Рисунок 3 – Пример максимальных паросочетаний

Описание алгоритма программы

Для начала пользователь должен ознакомиться с меню, где ему предстоит выбрать вывод исходной матрицы, после чего необходимо задать название файла, в который будет записываться результат. Далее необходимо ввести порядок графа, чтобы сформировать матрицу смежности. Затем, вызывается функция `edge_matrix`, которая осуществляет преобразование исходного графа в реберный граф. Функция создает массив `edge` размером, равным количеству ребер в исходном графе и заполняет его нулями. После чего она проверяет, какие ребра являются смежными в исходном графе, и на основе этих данных заполняет новую матрицу единицами. Далее сформированная матрица смежности реберного графа выводится на экран. Функция возвращает указатель на массив `edge`, который хранит значения матрицы смежности реберного графа.

Затем вызывается функция `maximal_independent_set`, которая осуществляет поиск максимальных независимых множеств в реберном графе. В функцию передается вершина, которая помечается как посещенная в массиве `vis[]` и заносится в массив `set[]`. Далее происходит построчный проход по матрице смежности реберного графа, и, в случае нахождения вершины не смежной с теми, что находятся в массиве `set[]`, она помечается как посещенная и также заносится в массив `set[]`. После того, как все комбинации множеств с данной стартовой вершиной были перебраны и выведены на экран, в функцию передается следующая вершина. Так происходит до тех пор, пока все вершины не будут перебраны. Функция возвращает значение указателя на индекс текущего элемента массива `set[]`.

Ниже представлены псевдокоды функций `edge_matrix` и `maximal_independent_set`, с помощью которых осуществляется поиск всех максимальных паросочетаний в программе.

edge_matrix()

1. нач
2. edge = динамическое выделение памяти для хранения матрицы смежности реберного графа
3. цикл от i=0 до i<num [шаг i++]
4. цикл от j=0 до j<num [шаг j++]
5. edge[i][j] = 0
6. конец цикла
7. конец цикла
8. iter = 0
9. цикл от i=0 до i<count [шаг i++]
10. цикл от j=i до j<count [шаг j++]
11. если matrix [i][j]=1 то
12. edge_coord[0][iter] = iter
13. edge_coord[1][iter] = i
14. edge_coord[2][iter] = j
15. iter++
16. конец условия
17. конец цикла
18. конец цикла
19. цикл от i=0 до i<num [шаг i++]
20. цикл от j=i+1 до j<num [шаг j++]
21. если (edge_coord[1][i]=edge_coord[1][j]) или
 edge_coord[1][i]=edge_coord[2][j] или
 edge_coord[2][i]=edge_coord[1][j] или
 edge_coord[2][i]=edge_coord[2][j])
22. edge[i][j] = 1
23. edge[j][i] = 1
24. конец условия
25. конец цикла

26.конец цикла

27.возврат значения edge

28.кон

maximal_independent_set()

1. нач

2. цикл от i=start до i<num [шаг i++]

3. если matrix [start][i]=0 и vis[i]=0 то

4. test = 0

5. цикл от j=0 до j<step+1 [шаг j++]

6. если matrix [set[j]][i]=0 то

7. test++

8. конец условия

9. конец цикла

10. если test=action+1 то

11. action ++

12. set[action] = i

13. vis[i] = 1

14. action = maximal_independent_set

15. конец условия

16. конец условия

17.конец цикла

18.вывод массива set[] на экран

19.vis[set[action]] = 0

20.action --

21.возврат значения action

22.кон

Описание программы

Для написания данной программы использован язык программирования Си. Си - язык программирования общего назначения, который очень известен своей эффективностью, экономичностью ресурсов и переносимостью на других программные платформы. Все эти преимущества обеспечивают качество разработки программного обеспечения, их быстроту и сравнительно небольшой размер.

В качестве среды программирования был выбран программный продукт Microsoft Visual Studio 2019. Проект был создан в виде консольного приложения Win32 (Visual C++).

Данная программа является многомодульной, поскольку состоит из нескольких функций: `execution`, `print_matrix`, `zero_matrix`, `generation_matrix`, `manual_matrix`, `file_matrix`, `edge_matrix`, `maximal_independent_set`, `main`.

Работа программы начинается с небольшого меню, с помощью которого пользователю необходимо выбрать способ вывода матрицы смежности. Перед этим необходимо задать название файла, чтобы сохранить наши результаты:

```
cout << "\n\t~~~ М Е Н Ю ~~~\n\n";
cout << "1. Сгенерировать матрицу смежности.\n";
cout << "2. Задать матрицу смежности с клавиатуры.\n";
cout << "3. Вывести матрицу смежности из файла.\n";

cout << "\nВведите имя файла результатов: ";
cin >> file_name;
file_write.open(file_name);

cout << "\nПункт меню: ";
cin >> item;

if (item == '1')
{
    do
    {
        printf("\nВведите порядок исходного графа: ");
        scanf_s("%d", &size);
        if (size < 1)
        {
            printf("Невозможно считать данные! Попробуйте заново!\n");
        }
    } while (size < 1);
    mass_vertex = generation_matrix(size);
    execution(mass_vertex, size);
}

if (item == '2')
{
    do
    {
```

```

        printf("\nВведите порядок исходного графа: ");
        scanf_s("%d", &size);
        if (size < 1)
        {
            printf("Невозможно считать данные! Попробуйте заново!\n");
        }
    } while (size < 1);
    mass_vertex = manual_matrix(size);
    execution(mass_vertex, size);
}

if (item == '3')
{
    do
    {
        printf("\nВведите порядок исходного графа: ");
        scanf_s("%d", &size);
        if (size < 1)
        {
            printf("Невозможно считать данные! Попробуйте заново!\n");
        }
    } while (size < 1);
    mass_vertex = file_matrix(size);
    execution(mass_vertex, size);
}

if ((item != '1') && (item != '2') && (item != '3'))
{
    printf("\nНеверный ввод! Попробуйте заново!\n\n");
}

file_write.close();

```

Далее выводятся матрица смежности вершин и матрица смежности ребер:

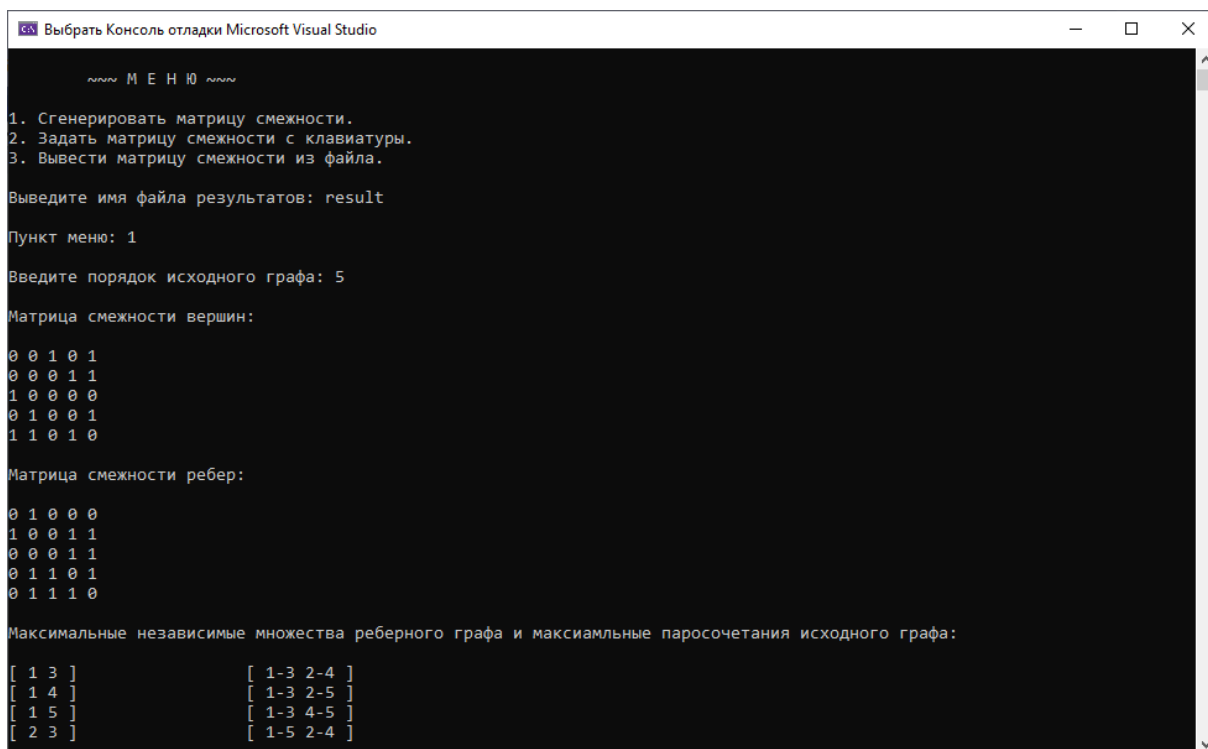
```

void print_matrix(int** matrix, int size)
{
    for (int i = 0; i < size; i++)
    {
        for (int j = 0; j < size; j++)
        {
            printf("%d ", matrix[i][j]);
            file_write << " " << matrix[i][j];
        }
        printf("\n");
        file_write << "\n";
    }
}

```

Затем происходит поиск максимальных независимых множеств реберного графа и максимальных паросочетаний исходного графа. Выше были представлены псевдокоды этих функций.

Ниже можно увидеть оформление начального запроса и дальнейшие действия с ним.



```
Выбрать Консоль отладки Microsoft Visual Studio

М Е Н Ю

1. Сгенерировать матрицу смежности.
2. Задать матрицу смежности с клавиатуры.
3. Вывести матрицу смежности из файла.

Выведите имя файла результатов: result

Пункт меню: 1

Введите порядок исходного графа: 5

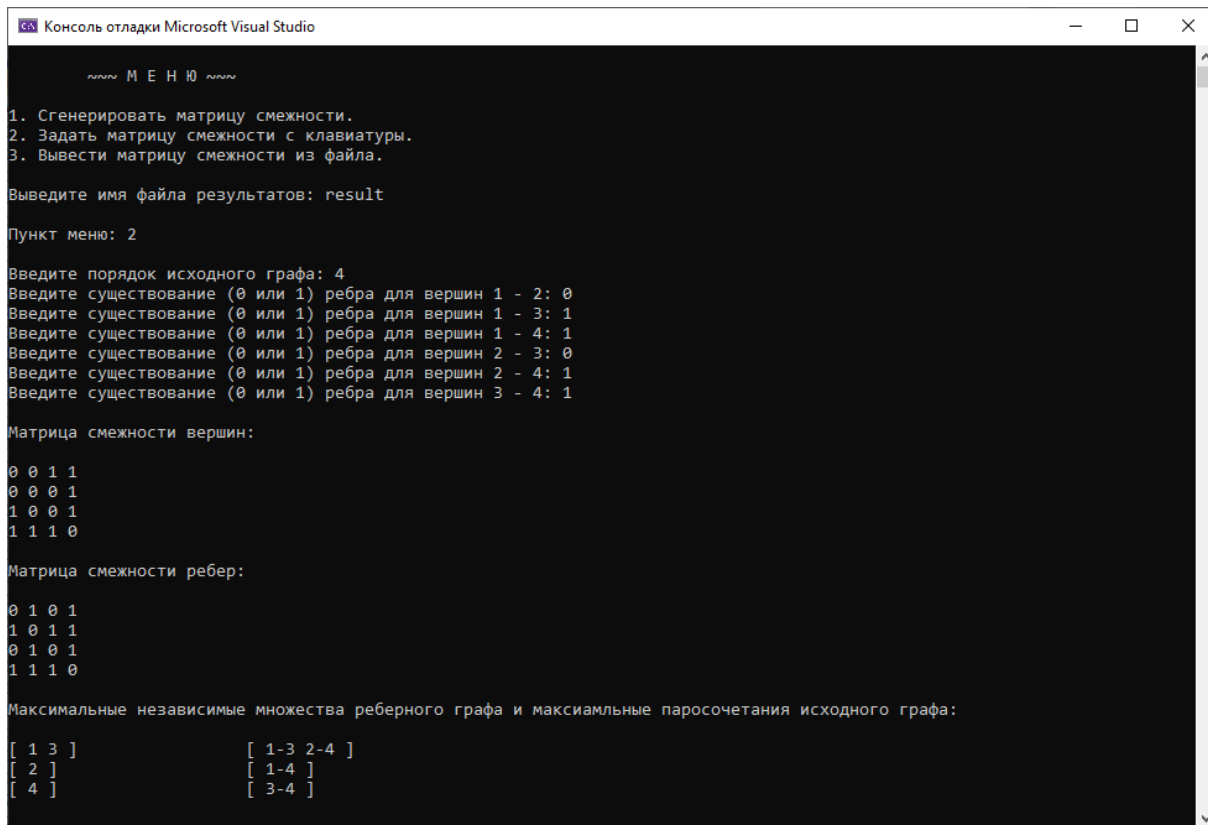
Матрица смежности вершин:
0 0 1 0 1
0 0 0 1 1
1 0 0 0 0
0 1 0 0 1
1 1 0 1 0

Матрица смежности ребер:
0 1 0 0 0
1 0 0 1 1
0 0 0 1 1
0 1 1 0 1
0 1 1 1 0

Максимальные независимые множества реберного графа и максимаьльные паросочетания исходного графа:

[ 1 3 ]      [ 1-3 2-4 ]
[ 1 4 ]      [ 1-3 2-5 ]
[ 1 5 ]      [ 1-3 4-5 ]
[ 2 3 ]      [ 1-5 2-4 ]
```

Рисунок 4 – Случайная генерация матрицы



```
Консоль отладки Microsoft Visual Studio

М Е Н Ю

1. Сгенерировать матрицу смежности.
2. Задать матрицу смежности с клавиатуры.
3. Вывести матрицу смежности из файла.

Выведите имя файла результатов: result

Пункт меню: 2

Введите порядок исходного графа: 4
Введите существование (0 или 1) ребра для вершин 1 - 2: 0
Введите существование (0 или 1) ребра для вершин 1 - 3: 1
Введите существование (0 или 1) ребра для вершин 1 - 4: 1
Введите существование (0 или 1) ребра для вершин 2 - 3: 0
Введите существование (0 или 1) ребра для вершин 2 - 4: 1
Введите существование (0 или 1) ребра для вершин 3 - 4: 1

Матрица смежности вершин:
0 0 1 1
0 0 0 1
1 0 0 1
1 1 1 0

Матрица смежности ребер:
0 1 0 1
1 0 1 1
0 1 0 1
1 1 1 0

Максимальные независимые множества реберного графа и максимаьльные паросочетания исходного графа:

[ 1 3 ]      [ 1-3 2-4 ]
[ 2 ]       [ 1-4 ]
[ 4 ]       [ 3-4 ]
```

Рисунок 5 – Ввод элементов матрицы с клавиатуры

Выбрать Консоль отладки Microsoft Visual Studio

М Е Н Ю

1. Сгенерировать матрицу смежности.
2. Задать матрицу смежности с клавиатуры.
3. Вывести матрицу смежности из файла.

Выведите имя файла результатов: result

Пункт меню: 3

Введите порядок исходного графа: 6

Матрица смежности вершин:

```
0 0 0 1 0 0
0 0 0 1 0 0
0 0 0 0 1 0
1 1 0 0 1 0
0 0 1 1 0 1
0 0 0 0 1 0
```

Матрица смежности ребер:

```
0 1 0 1 0
1 0 0 1 0
0 0 0 1 1
1 1 1 0 1
0 0 1 1 0
```

Максимальные независимые множества реберного графа и максимальные паросочетания исходного графа:

[1 3]	[1-4 3-5]
[1 5]	[1-4 5-6]
[2 3]	[2-4 3-5]
[2 5]	[2-4 5-6]
[4]	[4-5]

Рисунок 6 – Чтение матрицы из файла

result – Блокнот

Файл Правка Формат Вид Справка

Матрица смежности вершин:

```
0 0 0 1 0 0
0 0 0 1 0 0
0 0 0 0 1 0
1 1 0 0 1 0
0 0 1 1 0 1
0 0 0 0 1 0
```

Матрица смежности ребер:

```
0 1 0 1 0
1 0 0 1 0
0 0 0 1 1
1 1 1 0 1
0 0 1 1 0
```

Максимальные паросочетания исходного графа:

```
[ 1-4 3-5 ]
[ 1-4 5-6 ]
[ 2-4 3-5 ]
[ 2-4 5-6 ]
[ 4-5 ]
```

Стр 1, столб 1 100% Windows (CRLF) ANSI

Рисунок 7 – Файл результатов

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. Также были предусмотрены выводы предупреждений об ошибках при недопустимых действиях пользователя.

Ниже продемонстрирован результат тестирования программы.

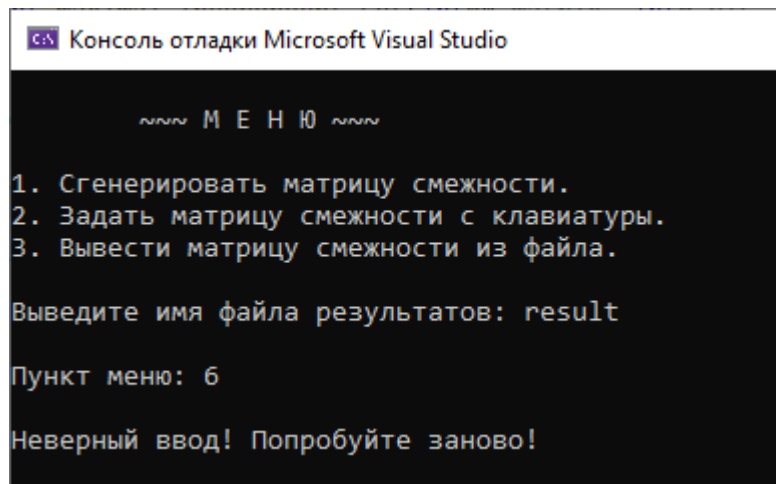


Рисунок 8 – Тестирование при некорректном вводе пункта меню

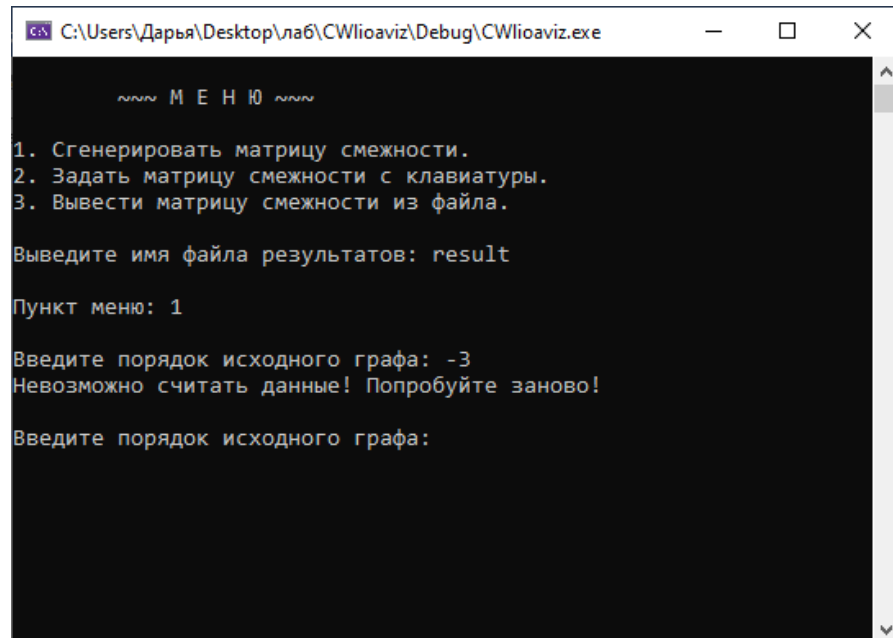


Рисунок 9 – Тестирование при некорректном вводе порядка матрицы

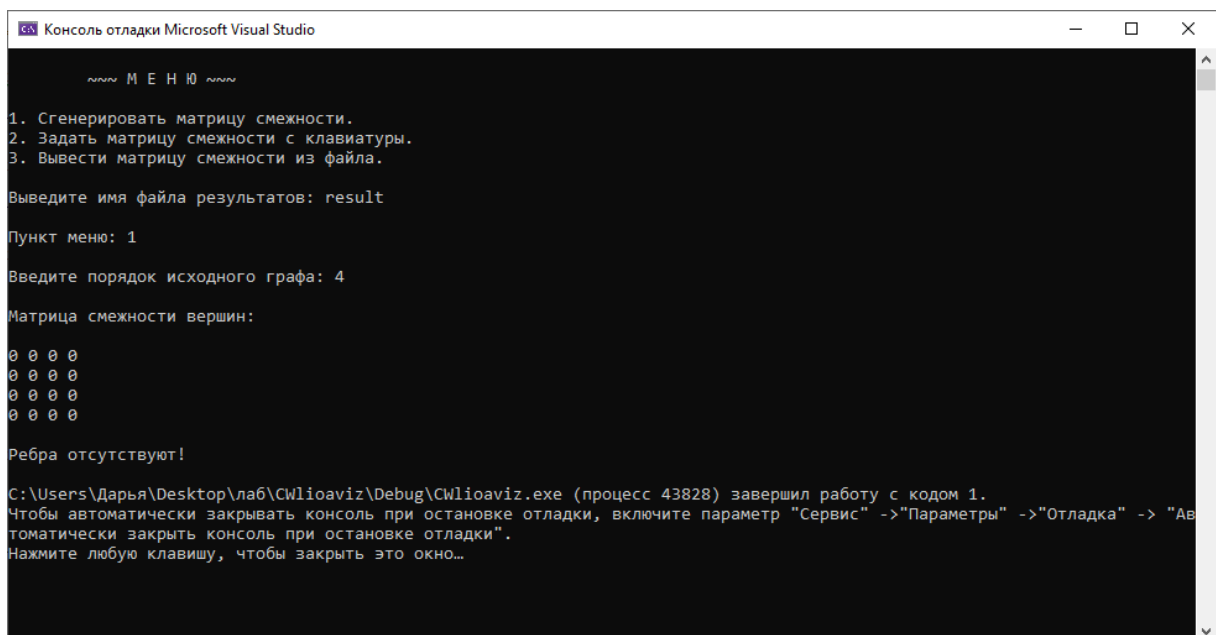


Рисунок 10 – Тестирование при генерации нулевой матрицы

```
Консоль отладки Microsoft Visual Studio

~*~ М Е Н Ю ~*~

1. Сгенерировать матрицу смежности.
2. Задать матрицу смежности с клавиатуры.
3. Вывести матрицу смежности из файла.

Выведите имя файла результатов: result

Пункт меню: 2

Введите порядок исходного графа: 4
Введите существование (0 или 1) ребра для вершин 1 - 2: 0
Введите существование (0 или 1) ребра для вершин 1 - 3: 1
Введите существование (0 или 1) ребра для вершин 1 - 4: 5
Неверный ввод! Матрица смежности включает в себя 0 или 1!

Введите существование (0 или 1) ребра для вершин 1 - 4: 1
Введите существование (0 или 1) ребра для вершин 2 - 3: 8
Неверный ввод! Матрица смежности включает в себя 0 или 1!

Введите существование (0 или 1) ребра для вершин 2 - 3: 0
Введите существование (0 или 1) ребра для вершин 2 - 4: 1
Введите существование (0 или 1) ребра для вершин 3 - 4: 1

Матрица смежности вершин:

0 0 1 1
0 0 0 1
1 0 0 1
1 1 1 0

Матрица смежности ребер:

0 1 0 1
1 0 1 1
0 1 0 1
1 1 1 0

Максимальные независимые множества реберного графа и максимаьные паросочетания исходного графа:

[ 1 3 ]          [ 1-3 2-4 ]
[ 2 ]           [ 1-4 ]
[ 4 ]           [ 3-4 ]
```

Рисунок 11 – Тестирование при некорректном вводе элементов матрицы в консоле

Таблица 1 – Описание поведения программы при тестировании

Описание теста	Ожидаемый результат	Полученный результат
Запуск программы	Вывод сообщения о выборе способа вывода матрицы	Верно
Некорректный ввод пункта меню	Сообщение об ошибке с просьбой попробовать заново	Верно
Некорректный ввод порядка матрицы	Сообщение об ошибке с просьбой попробовать заново	Верно
Генерация нулевой матрицы	Сообщение об отсутствии ребер	Верно
Некорректный ввод элементов матрицы в консоле	Сообщение об ошибке, ввести еще раз	Верно

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

Ручной расчет задачи

```
Консоль отладки Microsoft Visual Studio

~*~ М Е Н Ю ~*~

1. Сгенерировать матрицу смежности.
2. Задать матрицу смежности с клавиатуры.
3. Вывести матрицу смежности из файла.

Выведите имя файла результатов: result

Пункт меню: 1

Введите порядок исходного графа: 4

Матрица смежности вершин:

0 0 1 1
0 0 0 1
1 0 0 0
1 1 0 0

Матрица смежности ребер:

0 1 0
1 0 1
0 1 0

Максимальные независимые множества реберного графа и максимальные паросочетания исходного графа:

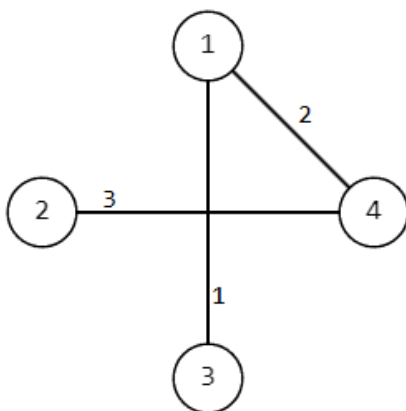
[ 1 3 ]           [ 1-3 2-4 ]
[ 2 ]             [ 1-4 ]
```

Рисунок 12 – Проверка на правильность работы программы

Матрица смежности вершин:

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

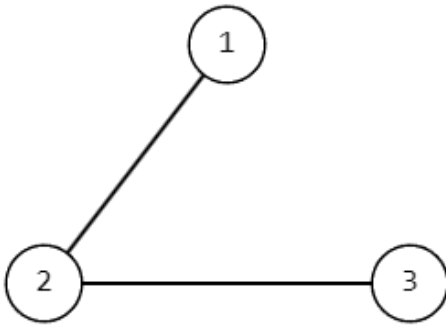
Граф:



Матрица смежности реберного графа:

	1	2	3
1	0	1	0
2	1	0	1
3	0	1	0

Граф:



Тогда найдем все максимальные независимые множества реберного графа:
[1 3] [2]

Так как граф реберный, то его максимальные независимые множества будут являться максимальными паросочетаниями исходного графа.

Искомые максимальные паросочетания:
[1-3 2-4] [1-4]

Таким образом, можно сделать вывод, что программа работает верно

Заключение

Таким образом, в процессе создания данного проекта разработана программа, реализующая алгоритм нахождения всех максимальных паросочетаний в заданном графе с использованием алгоритма с возвратом для нахождения независимых множеств вершин реберного графа в Microsoft Visual Studio 2019 C++ 6.0.

При выполнении данной курсовой работы были получены навыки разработки программ и освоены приемы создания матриц, основанных на теории графов. Углублены знания языка программирования С.

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

В дальнейшем программу можно улучшить, добавив псевдографический интерфейс.

Список используемых источников

1. Википедия, «Паросочетания»
2. Марк Лутц, «Изучаем Python»
3. Методические указания по курсу «Логика и основы алгоритмизации в инженерных задачах»
4. Ф.А. Новиков. Дискретная математика для программистов, Питер, 2001.

Приложение А.

Листинг программы.

```
#include <iostream>
#include "windows.h"
#include "time.h"
#include <stack>
#include <fstream>

#define _CRT_SECURE_NO_WARNINGS

using namespace std;

void execution(int** mass_vertex, int size);

void print_matrix(int** matrix, int size);
void zero_matrix(int** matrix, int size);

int** generation_matrix(int size);
int** manual_matrix(int size);
int** file_matrix(int size);

int** edge_matrix(int** matrix, int** edge_coord, int size, int* num);

int maximal_independent_set(int** matrix, int* vis, int* set, int* num, int* vis_prnt, int
start, int action, int** edge_coord);

string file_name;
ofstream file_write;

void execution(int** mass_vertex, int size)
{
    int kolvo_edge = 0, zero, action = 0;
    int* num = &kolvo_edge, * vis, * set, * vis_prnt;
    int** edge, ** edge_coord;
    int** matrix;

    printf("\nМатрица смежности вершин:\n\n");
    file_write << "\nМатрица смежности вершин:\n\n";
    print_matrix(mass_vertex, size);
    zero_matrix(mass_vertex, size);

    for (int i = 0; i < size; i++)
    {
        for (int j = i; j < size; j++)
        {
            if (mass_vertex[i][j] == 1)
            {
                (*num)++;
            }
        }
    }

    edge_coord = (int**)malloc(3 * sizeof(int*));
    for (int i = 0; i < 3; i++) {
        edge_coord[i] = (int*)malloc(*num * sizeof(int));
    }
    edge = edge_matrix(mass_vertex, edge_coord, size, num);
    printf("\nМатрица смежности ребер:\n\n");
    file_write << "\nМатрица смежности ребер:\n\n";
    print_matrix(edge, *num);
    printf("\n");
}
```

```

vis = (int*)malloc(*num * sizeof(int));
set = (int*)malloc(*num * sizeof(int));
vis_prnt = (int*)malloc(*num * sizeof(int));
for (int i = 0; i < *num; i++) {
    vis_prnt[i] = 0;
}

printf("Максимальные независимые множества реберного графа и максимаьные
паросочетания исходного графа:\n\n");
file_write << "\nМаксиамльные паросочетания исходного графа:\n\n";
for (int i = 0; i < *num; i++)
{
    for (int j = 0; j < *num; j++)
    {
        vis[j] = 0;
    }
    vis[i] = 1;
    set[action] = i;
    action = maximal_independent_set(edge, vis, set, num, vis_prnt, i, action,
edge_coord);
    action = 0;
}
}

void print_matrix(int** matrix, int size)
{
    for (int i = 0; i < size; i++)
    {
        for (int j = 0; j < size; j++)
        {
            printf("%d ", matrix[i][j]);
            file_write << " " << matrix[i][j];
        }
        printf("\n");
        file_write << "\n";
    }
}

void zero_matrix(int** matrix, int size)
{
    int zero = 0;
    for (int i = 0; i < size; i++)
    {
        for (int j = 0; j < size; j++)
        {
            if (matrix[i][j] == 0) zero++;
        }
    }
    if (zero == size * size)
    {
        printf("\nРебра отсутствуют!\n");
        exit(1);
    }
}

int** generation_matrix(int size)
{
    int** matrix;
    matrix = (int**)malloc(size * sizeof(int*));
    for (int i = 0; i < size; i++)
    {
        matrix[i] = (int*)malloc(size * sizeof(int));
    }
    for (int i = 0; i < size; i++)
    {

```



```

        for (int j = 0; j < size; j++)
        {
            if (j > i)
            {
                matrix[i][j] = rand() % 100;
                matrix[j][i] = matrix[i][j];
            }
            else if (i == j)
            {
                matrix[i][j] = 0;
            }
            if (0 <= matrix[i][j] && matrix[i][j] <= 50)
            {
                matrix[i][j] = 0;
            }
            else if (matrix[i][j] >= 51 && matrix[i][j] <= 100)
            {
                matrix[i][j] = 1;
            }
        }
    }
    return matrix;
}

int** manual_matrix(int size)
{
    int** matrix = (int**)malloc(sizeof(int*) * (size));

    int i = 0;
    int j = 0;

    for (i = 0; i < size; ++i)
    {
        matrix[i] = (int*)malloc(sizeof(int) * size);
    }

    for (i = 0; i < size; ++i)
    {
        matrix[i][i] = 0;
        for (j = i + 1; j < size; ++j)
        {
            do {
                printf("Введите существование (0 или 1) ребра для вершин %d - %d:
", i + 1, j + 1);

                scanf_s("%d", &matrix[i][j]);
                if ((matrix[i][j] < 0) || (matrix[i][j] > 1))
                    printf("Неверный ввод! Матрица смежности включает в себя 0
или 1!\n\n");
            } while ((matrix[i][j] < 0) || (matrix[i][j] > 1));
            matrix[j][i] = matrix[i][j];
        }
    }
    return matrix;
}

int** file_matrix(int size)
{
    int** matrix;

    stack <int> graph;
    int buffer = 0;
    ifstream file_read("read.txt");
    if (!file_read.is_open())
    {
        _cputws(L"Не удалось открыть файл.");
    }
}

```

```

while (!file_read.eof())
{
    file_read >> buffer;
    graph.push(buffer);
}

file_read.close();
size = (int)sqrt(graph.size());

matrix = (int**)malloc(size * sizeof(int*));
for (int i = 0; i < size; i++)
    matrix[i] = (int*)malloc(size * sizeof(int));

for (int i = size - 1; i > -1; i--)
{
    for (int j = size - 1; j > -1; j--)
    {
        matrix[i][j] = graph.top();
        graph.pop();
    }
}
return matrix;
}

int** edge_matrix(int** matrix, int** edge_coord, int size, int* num) {
    int** edge;
    // выделение памяти для хранения матрицы смежности ребер и заполнение ее нулями
    edge = (int**)malloc(*num * sizeof(int*));
    for (int i = 0; i < *num; i++) {
        edge[i] = (int*)malloc(*num * sizeof(int));
    }
    for (int i = 0; i < *num; i++) {
        for (int j = 0; j < *num; j++) {
            edge[i][j] = 0;
        }
    }
    // цикл прохода по исходной матрице с целью задания им порядка и определения их
координат
    int iter = 0;
    for (int i = 0; i < size; i++) {
        for (int j = i; j < size; j++) {
            if (matrix[i][j] == 1) {
                edge_coord[0][iter] = iter;
                edge_coord[1][iter] = i;
                edge_coord[2][iter] = j;
                iter++;
            }
        }
    }
    // цикл прохода по массиву координат с целью заполнения реберной матрицы смежности
(если находится хотя бы одна одинаковая координата у двух ребер, то они смежные)
    for (int i = 0; i < *num; i++) {
        for (int j = i + 1; j < *num; j++) {
            if ((edge_coord[1][i] == edge_coord[1][j]) || (edge_coord[1][i] ==
edge_coord[2][j]) || (edge_coord[2][i] == edge_coord[1][j]) || (edge_coord[2][i] ==
edge_coord[2][j])) {
                edge[i][j] = 1;
                edge[j][i] = 1;
            }
        }
    }
    return edge;
}

```

```

int maximal_independent_set(int** matrix, int* vis, int* set, int* num, int* vis_prnt, int
start, int action, int** edge_coord) {

    int control = 0;

    for (int i = start; i < *num; i++) {
        if (matrix[start][i] == 0 && vis[i] == 0) {
            int test = 0;
            for (int j = 0; j < action + 1; j++) {
                if (matrix[set[j]][i] == 0) {
                    test++;
                }
            }
            if (test == action + 1) {
                control = 1;
                action++;
                set[action] = i;
                vis[i] = 1;
                vis_prnt[i] = 1;
                action = maximal_independent_set(matrix, vis, set, num, vis_prnt,
start, action, edge_coord);
            }
        }
    }

    if (control == 0 && (vis_prnt[set[0]] == 0 || action != 0)) {
        printf("[ ");
        for (int i = 0; i < action + 1; i++) {
            printf("%d ", set[i] + 1);
        }
        printf("]");
        file_write << "[ ";

        for (int i = 0; i < action + 1; i++) {
            printf("%d-%d ", edge_coord[1][set[i]] + 1, edge_coord[2][set[i]] + 1);
            file_write << edge_coord[1][set[i]] + 1 << "-" << edge_coord[2][set[i]]
+ 1 << " ";
        }
        printf("]\n");
        file_write << "]\n";
    }

    vis[set[action]] = 0;
    action--;
    return action;
}

int main()
{
    SetConsoleCP(1251);
    SetConsoleOutputCP(1251);
    srand(time(NULL));

    int size;
    int** mass_vertex;

    char item = '\0';

    cout << "\n\t~~~ М Е Н Ю ~~~\n\n";
    cout << "1. Сгенерировать матрицу смежности.\n";
    cout << "2. Задать матрицу смежности с клавиатуры.\n";
    cout << "3. Вывести матрицу смежности из файла.\n";

    cout << "\nВведите имя файла результатов: ";
    cin >> file_name;
}

```

```

file_write.open(file_name);

cout << "\nПункт меню: ";
cin >> item;

if (item == '1')
{
    do
    {
        printf("\nВведите порядок исходного графа: ");
        scanf_s("%d", &size);
        if (size < 1)
        {
            printf("Невозможно считать данные! Попробуйте заново!\n");
        }
    } while (size < 1);
    mass_vertex = generation_matrix(size);
    execution(mass_vertex, size);
}

if (item == '2')
{
    do
    {
        printf("\nВведите порядок исходного графа: ");
        scanf_s("%d", &size);
        if (size < 1)
        {
            printf("Невозможно считать данные! Попробуйте заново!\n");
        }
    } while (size < 1);
    mass_vertex = manual_matrix(size);
    execution(mass_vertex, size);
}

if (item == '3')
{
    do
    {
        printf("\nВведите порядок исходного графа: ");
        scanf_s("%d", &size);
        if (size < 1)
        {
            printf("Невозможно считать данные! Попробуйте заново!\n");
        }
    } while (size < 1);
    mass_vertex = file_matrix(size);
    execution(mass_vertex, size);
}

if ((item != '1') && (item != '2') && (item != '3'))
{
    printf("\nНеверный ввод! Попробуйте заново!\n\n");
}

file_write.close();
}

```