Министерство науки и высшего образования Российской Федерации
Пензенский государственный университет
Кафедра «Вычислительная техника»

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

К курсовому проектированию
по курсу «Логика и основы алгоритмизации
в инженерных задачах» на тему
«Реализация алгоритма поиска наибольшего паросочетания»

28.12.23
отлично

Выполнил:
студент группы 22ВВВ3
Байков А. В.

Приняли:
Юрова О. В.
Акифьев И. В.

Пенза 2023

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 Байкову Алексею Владимировичу. Группа 22ВВВ3
Тема_проекта реализация алгоритма поиска наибольшего подпоследовательности.

Исходные данные (технические требования) на проектирование

Разработка алгоритмов и программного обеспечения в соответствии с данным заданием курсового проекта. Пояснительная записка должна содержать:
1) Постановку задачи;
2) Теоретическую часть задания;
3) Описание алгоритма поставленной задачи
4) Пример ручного расчета задачи и вычислений (на небольшом участке работы алгоритма);
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 Исследование теоретической части курсового
2 Разработка алгоритмов программы.
3 Разработка программы.
4 Тестирование и отладление разработанной программы.
5 Оформление пояснительной записки
6
7
8

Дата выдачи задания "6" сентября

Дата защиты проекта "___" _____

Руководитель Юсова Е.В.

Задание получил "6" сентября 2023 г.

Студент Байков Алексей Владимирович Байков

Содержание

Реферат

Отчет 39 стр, 16 рисунков.

ГРАФ, ОРГАФ, ПОИСК, НАИБОЛЬШЕЕ ПАРОСОЧЕТАНИЕ, АЛГОРИТМ КУНА.

Цель исследования – разработка программы, способная осуществлять алгоритм поиска наибольшего паросочетания со случайными числами и работа с пользовательским графом путём считывания его из файла.

В данной работе представлен модифицированный алгоритм Куна для работы как с ориентированным так не ориентированным графом.

Введение

В современном мире компьютерных наук, эффективное сопоставление объектов, группировка или распределение ресурсов играют непрерывно важную роль. Особенно в задачах, связанных с оптимизацией, сетями и социальными системами, поиск наибольшего паросочетания становится ключевой операцией.

Алгоритмы поиска наибольшего паросочетания — это мощный инструмент для распределения пар объектов в коллективе таким образом, чтобы максимизировать количество таких пар. В этой курсовой работе глубже исследую эту область и представлю алгоритм, которые позволит эффективно и точно найти наибольшее паросочетание в заданном графе.

Алгоритм поиска наибольшего паросочетания позволяет построить обход ориентированного и неориентированного графа. Также даёт возможность создания графа, как со случайными числами, так и работа с пользовательским графом путём считывания его из файла.

В качестве среды разработки мною была выбрана среда MicrosoftVisualStudio2022, язык программирования – Си.

Задача данной курсовой работы заключается в разработке программы на языке Си. Он и будет использоваться в данном курсовом проекте для реализации алгоритма поиска наибольшего паросочетания с помощью модифицированного алгоритма Куна.

Постановка задачи

Требуется разработать программу, которая выделит компоненты сильной связности орграфа, используя алгоритм поиска наибольшего паросочетания.

Первоочередной задачей является абстрагирование графа в программной среде. Это представление может быть основано на структурах данных, моделирующих вершины и рёбра графа. Граф может быть представлен как ориентированный или неориентированный, в зависимости от вводимых данных.

Решение задачи о максимальном паросочетании может включать использование известных алгоритмов, таких как алгоритм Куна для неориентированных графов или алгоритмы поиска увеличивающих путей для ориентированных графов. Данные методы обеспечивают эффективный поиск наибольшего паросочетания в графе.

Программа предоставляет пользователю опцию: генерация случайного графа или использование матрицы смежности из предоставленного файла. При выборе второго варианта, программа должна корректно считать и обработать данные из файла для дальнейшего использования в алгоритмах поиска паросочетаний.

После успешного поиска максимального паросочетания, программа предоставляет информацию о найденных рёбрах, составляющих паросочетание. При необходимости, программа также может выводить все возможные паросочетания для заданного графа.

Устройство ввода – клавиатура и мышь.

Задания выполняются в соответствии с вариантом №27.

Теоретическая часть задания

Пусть дан граф 6 = (V, E). Паросочетание M в графе G — это множество попарно несмежных рёбер, иначе говоря, рёбер у которых нет общих вершин. Мощностью паросочетания M называется количество рёбер, входящих в это паросочетание. Мощность паросочетания M обозначается как |M|.

Паросочетание M графа G называется максимальным, если оно не содер- жится ни в каком другом паросочетании графа G, иными словами, к этому паросочетанию невозможно добавить никакое другое ребро графа G, которое бы не являлось смежным со всеми другими рёбрами этого паросочетания.

Паросочетание M графа G, которое содержит максимальное количество рёбер называется наибольшим паросочетанием. То есть если M наибольшее паросочетание графа G, то в графе G не существует такого паросочетания M1, что |M1| > |M].

Любое наибольшее паросочетание является максимальным, однако не каждое максимальное паросочетание является наибольшим.

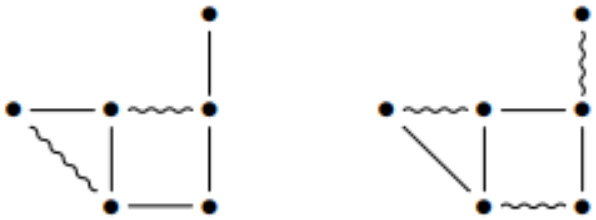

Рисунок 1 – Пример паросочетания

Пример того, что не каждое максимальное паросочетание является наибольшим. Слева на рисунке 1 изображено (волнистыми рёбрами) максимальное паросочетание, однако оно не является наибольшим. Справа на рисунке 1 изображено уже наибольшее паросочетание.

Задачу поиска максимального паросочетания чаще всего можно решить за полиномиальное время с помощью техники жадного алгоритма. Задача же поиска мощности наибольшего паросочетания является полиномиально разрешимой [13].

Пусть M — это паросочетание в графе G. Обозначим граф, порождённый множеством вершин рёбер в M, как G [м].

Паросочетание M называется совершенным, если |G| = |G[M]]. Нетрудно видеть, что любое совершенное паросочетание является

наибольшим.

Почти совершенным паросочетанием называется такое паросочетание M, что |G| -1 = |G[M]|.

M называется P-паросочетанием если G[M] имеет свойство P, где P — это какое-то свойство в графе G (например двудольность). Наибольшим P- паросочетанием в графе G называется наибольшее (по количеству рёбер) па- росочетание, среди тех паросочетаний, для которых выполняется свойство P. Максимальным P-паросочетанием в графе G называется P-паросочетание, которое не содержится ни водному другом P-паросочетании графа C. Мощность наибольшего P-паросочетания будем обозначим как Bp(G) и будем называть числом P-паросочетания. Мощность наименьшего P-паросочетания среди всех максимальных P-паросочетаний в графе G будем обозначать $^{-}$Bp(G).

Описание алгоритма программы

Функция dfs

Функция рекурсивно ищет увеличивающий путь. Для каждой вершины v смежной с текущей вершиной u, если вершина v еще не была посещена (!visited[v]) и между u и v есть ребро (graph[u][v]), она помечает v как посещенную и проверяет:

- Если v еще не имеет пары (matchR[v] < 0) или для вершины v можно найти другую пару, вызывается рекурсивно та же функция для вершины matchR[v].
- Если удается найти увеличивающий путь, функция устанавливает новую пару для вершины v (matchR[v] = u) и возвращает true.

Функция продолжает искать увеличивающий путь для каждой вершины правой доли до тех пор, пока не обойдет все вершины или не достигнет конца рекурсии.

Функция printMatching используется для вывода одного паросочетания, которое содержится в массиве matchR[]. Она принимает на вход матрицу смежности graph, массив пар matchR[] и количество вершин vertices.

- Создается массив visited, который отслеживает, была ли вершина посещена. Все элементы массива устанавливаются в false.

- Затем происходит проход по массиву matchR[]. Если у вершины i есть пара (то есть matchR[i] != -1) и эта пара еще не была посещена (!visited[i]), то эта пара выводится на экран в формате (matchR[i], i).

- После вывода пары текущие вершины помечаются как посещенные (visited[i] = true и visited[matchR[i]] = true).

В результате функция выводит на экран одно паросочетание из графа.

Функция printAllMatchingsUtil используется для поиска всех паросочетаний, начиная с вершины u и используя рекурсивный подход. Она принимает на вход матрицу смежности graph, вершину u, массив посещенных вершин visited[], массив пар matchR[] и количество вершин vertices.

- Для текущей вершины u происходит проход по всем вершинам

графа (for (int v = 0; v < vertices; v++)).

-
- Если между вершиной u и вершиной v есть ребро и вершина v еще не посещена (graph[u][v] && !visited[v]), то выполняются действия:

- Помечаем вершину v как посещенную (visited[v] = true).

- Если у вершины v еще нет пары (matchR[v] < 0) или ее пара уже посещена (!visited[matchR[v]]), то создается новая пара (matchR[v] = u) и вызывается функция printMatching для вывода найденной пары.

- Рекурсивно вызывается printAllMatchingsUtil, чтобы продолжить поиск паросочетаний, начиная с вершины v.

- После завершения поиска для текущей вершины u, помечаем ее как непосещенную, чтобы использовать ее в других комбинациях пар.

Функция printAllMatchingsUtil в основном используется для нахождения всех паросочетаний, начиная с одной вершины и продолжая рекурсивный поиск через другие вершины.

printAllMatchings служит для поиска всех возможных паросочетаний в графе. Вот пошаговое описание этой функции:

- Выделение памяти: Функция начинается с выделения памяти под массив matchR[], который хранит пары вершин, и массив visited[], отслеживающий посещенные вершины.

- Инициализация массивов: Все элементы matchR[] устанавливаются в -1 (обозначая отсутствие пары), а visited[] устанавливается в false (обозначая, что ни одна вершина еще не посещена).

- Поиск всех паросочетаний: Для каждой вершины в графе (от 0 до vertices - 1), функция вызывает printAllMatchingsUtil. Это позволяет начать поиск паросочетаний с каждой вершины в графе.

- Освобождение памяти: После завершения поиска всех паросочетаний освобождаются выделенные ресурсы (matchR[] и visited[]).

Таким образом, printAllMatchings является оберткой, которая инициализирует массивы и вызывает функцию printAllMatchingsUtil для поиска всех паросочетаний в графе.

Ниже представлен псевдокод функции bpm(), printMatching(), printAllMatchingsUtil(), printAllMatchings()

bpm()

1. для v пока v < vertices делать v++

2. если graph[u][v] и !visited[v]

3. visited[v] = true

4. если matchR[v] < 0 или вызвать функцию bpm(graph, matchR[v], visited, matchR, vertices)

5. matchR[v] = u

6. Вернуть true

7. Конец условия

8. Конец условия .

9. Конец цикла

10. Вернуть false

printMatching()

1. Выделить память visited

2. для i = 0 пока i < vertices делать i++

3. Если matchR[i] != -1 и !visited[i]

4. Вывод matchR, i

5. visited[i] = true

6. visited[matchR[i]] = true

7. Конец условия

8. Конец цикла

9. Переход на новую строку

10. Освободить память visited

printAllMatchingsUtil()

1. Для v = 0 пока v < vertices делать v++

2. Если graph[u][v] и !visited[v]

3. visited = true

4. Если matchR[v] < 0 или !visited[matchR[v]]

5. matchR[v] = u

6. Вызвать функцию printMatching(graph, matchR, vertices)

7. Конец условия

8. Конец условия

9. Конец цикла

10. visited[u] = false

printAllMatchings()

1. Выделить память matchR

2. Если matchR == NULL

3. Вывести Ошибка выделения памяти

4. Переход на новую строку

5. Вернуть

6. Конец условия

7. Для i = 0 пока i < vertices делать i++

8. matchR[i] = -1

9. Конец цикла

10. Выделить память visited

11. Если matchR == NULL

12. Вывести Ошибка выделения памяти

13. Переход на новую строку

14. Освободить память matchR

15. Вернуть

16. Конец условия

17. Для i = 0 пока i < vertices делать i++

18. visited = false

19. Конец цикла

20. Для u = 0 пока u < vertices делать u++

21. Вызвать функцию printAllMatchingsUtil(graph, u, visited, matchR, vertices)

22. Конец цикла

23. Освободить память matchR

24. Освободить память visited

Описание программы

Язык программирования C — это мощный и эффективный инструмент, позволяющий создавать быстрые и эффективные программы. Его простота и низкоуровневая природа делают его идеальным выбором для разработки системного программного обеспечения и встроенных систем, где требуется управление ресурсами на низком уровне. Богатство библиотек и долгая история использования делают язык C одним из самых надежных и широко используемых инструментов в программировании. Его переносимость позволяет создавать программы, которые могут выполняться на различных платформах без изменений, что делает его универсальным выбором для множества задач различной сложности. Наконец, C обеспечивает прямой доступ к памяти, что позволяет разработчикам управлять ресурсами и оптимизировать производительность программы в большей степени, чем другие языки. Поэтому он был выбран для написания курсового проекта.

Проект был создан в виде консольного приложения Win32 (Visual C++).

Данная программа является многомодульной, поскольку состоит из нескольких функций: bpm, printMatching, printAllMatchings, printAllMatchingsUtil, memoryAllocation, generateRandomGraph, printGraph, isDirected, isUndirected.

При запуске программы пользователя встречает меню. В котором можно сделать выбор.

1) Работу со случайным графом;
2) Работу с пользовательским графом путем считывания из файла;
3) Завершить работу.

Если будет выбран 1-й пункт(Работа со случайным графом), то пользователь должен будет выбрать, количество вершин для графа. Происходит проверка введенного числа, для корректной работы программы.

```
// Выбор пользователем режима
        int typeMode;
        scanf("%d", &typeMode);

        // Цикл устранения некорректного ввода пользователем
        while (typeMode != 1 && typeMode != 2 && typeMode != 3)
        {   // Вывод предупреждения
            printf("\nОшибка! Введите номер соответствующего пункту
пункта: ");
            while (getchar() != '\n'); // Устранение зацикливания
            scanf("%d", &typeMode); // Повторный ввод
```

```
        }
        // Если пользователь выбрал работу со случайным графом
        if (typeMode == 1)
        {
            // Цикл устранения некорректного ввода пользователем
            bool num = false; // Переменная для работы цикла
            while (!num)
            {
                // Ввод вершин графа
                printf("\nВведите количество вершин в графе: ");
                if (scanf("%d", &vertices) == 1 && vertices > 0 &&
vertices <= 1000)  // Если введено число, которое больше 0
                {
                    num = true; // Выход из цикла
                }
                else // Иначе продолжаем обработку
                {   // Вывод предупреждения
                    printf("\nОшибка!   Размер   матрицы   должен
задаваться числом, более 0 и менее 1000. Повторите попытку.\n\n");
                    while  (getchar()  !=  '\n');  //  Устранение
зацикливания
                }
            }
```

Затем ему необходимо выбрать тип графа(ориентированный или неориентированный). Происходит проверка введенного числа, для корректной работы программы.

```
 int graphType; // Тип графа
            printf("\nВыберите тип графа:\n");
            printf("1) Ориентированный граф\n");
            printf("2) Неориентированный граф\n");

            // Ввод пользователем типа графа
            printf("Ваш выбор: ");
            scanf("%d", &graphType);

            // Цикл устранения некорректного ввода пользователем
            while (graphType != 1 && graphType != 2)
            {   // Вывод предупреждения
                printf("\nОшибка! Введите 1 или 2 для дальнейшей
работы программы: ");
                while   (getchar()   !=   '\n');   //   Устранение
зацикливания
                scanf("%d", &graphType); // Повторный ввод
            }
```

Происходит выделение памяти под матрицу графа, с помощью malloc по количеству вершин (vertices x vertices) за это отвечает функция memoryAllocation. Ее заполнение случайными числами, функция generateRandomGraph и вывод, функция printGraph.

```
memoryAllocation(graph, vertices); // Вызов функции, которая выделяет память под матрицу смежности

            generateRandomGraph(graph, vertices, graphType); // Вызов функции, отвечающую за генерацию матриц смежности

            printGraph(graph, vertices); // Вызов функции, отвечающую за генерацию матриц смежности
```

Происходит вывод всех найденных паросочетаний, включая наибольшее, это выполняет функция printAllMatchings. используется для поиска всех возможных паросочетаний в графе, начиная с определенной вершины u. Она просматривает все вершины графа, связанные с u, и если есть свободная вершина (v), устанавливает соответствие между u и v, после чего вызывает функцию printMatching для вывода текущих пар. Затем она продолжает исследовать другие возможные сочетания, помечая посещенные вершины и отмечая вершину u как непосещенную для дальнейших исследований.

```
// Вывод наибольшего паросочетания, Вызов функции, отвечающую за нахождение наибольшего паросочетания
            printf("\nПаросочетания и наибольшее:\n");
            printAllMatchings(graph, vertices); // Вызов функции, отвечающую за вывод паросочетаний
            printf("\n");
```

Если был выбран 2 пункт (работа с пользовательским файлом), то происходит ввод имени файла. Происходят проверка на его существование и не пустоту.

```
// Ввод имени файла
            printf("\nВведите имя файла, содержащего матрицу графа: ");
            scanf("%s", &fileName);

            // Проверка на наличие файла в цикле
            fp = fopen(fileName, "r");
            while (fp == NULL)
            {
                printf("\nОшибка: Файл не найден!\n");
                printf("\nВведите имя файла, содержащего матрицу графа: ");
                while (getchar() != '\n'); // Устранение зацикливания
                scanf("%s", &fileName);
                fp = fopen(fileName, "r");
            }
            fclose(fp);

            // Проверка на пустоту файла
            fp = fopen(fileName, "r");
```

```
            fseek(fp, 0, SEEK_END);
            if (ftell(fp) == 0)
            {
                printf("\nФайл пуст.\n\nПопробуйте заполнить его и
повторите попытку.\n\nУдачи ;)\n");
                return 1;
            }
            fclose(fp);
```

Пользователю не нужно вводить количество вершин поскольку программа сама понимаем сколько вершин в графе. Переменная countALL нужна для подсчета ребер между вершинами с пробелами, для последующего прохода по файлу и копированием данных в матрицу смежности. Переменная verticesFile нужна для выделения памяти под матрицу смежности, что и происходит в функции memoryAllocation в соответствии с количеством вершин.

```
char c; // Переменная для подсчета вершин
            // Определяем количество вершин
            fp = fopen(fileName, "r");
            while ((c = fgetc(fp)) != '\n' && c != EOF) // Пока не
дошли до перехода на новую строку и до конца файла
            {  // Если есть пробел
                if (c == ' ')
                {
                    countAll++;  //  Увеличиваем  счетчик  вершин  с
мусором
                    continue; // Пропускаем итерацию
                }
                verticesFile++; // Увеличиваем счетчик вершин без
мусора
            }
            fclose(fp);
memoryAllocation(graph, verticesFile); // Вызов функции, которая
выделяет память под матрицу смежности
            printf("\n");
```

Происходит вывод отсканированного графа и поиск наибольшего паросочетания.

```
printGraph(graph, verticesFile); // Вывод графа

            //  Вывод  наибольшего  паросочетания,  Вызов  функции,
отвечающую за нахождение наибольшего паросочетания
            printf("\nПаросочетания и наибольшее:\n");
            printAllMatchings(graph,   verticesFile);   //   Вызов
функции, отвечающую за вывод паросочетаний
            printf("\n");

            //  Освобождение  выделенной  памяти,  выделенной  для
```

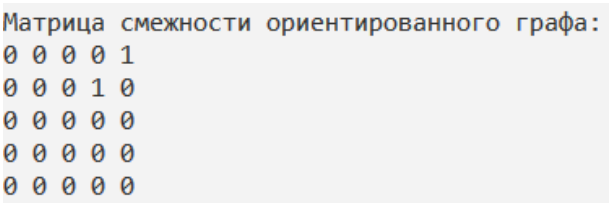

Рисунок 4 - Вывод сгенерированной матрицы

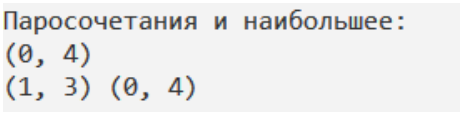

Рисунок 5 - Вывод паросочетаний и наибольшего

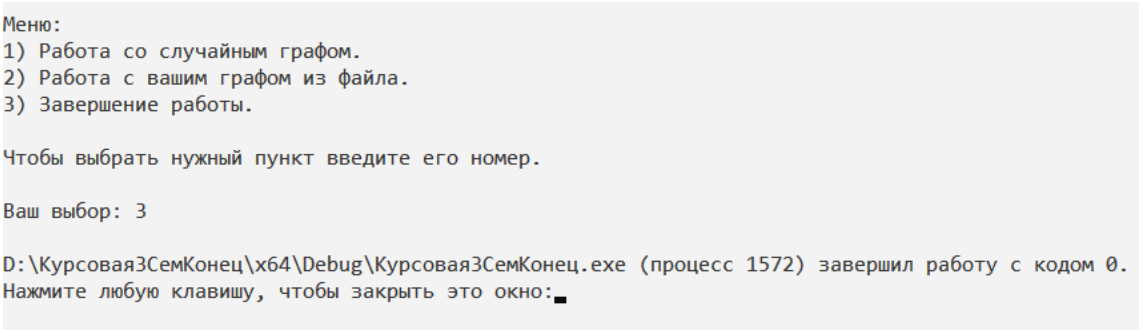

Рисунок 6 - Завершение работы программы

Тестирование

Если был выбран первый пункт

```
Меню:
1) Работа со случайным графом.
2) Работа с вашим графом из файла.
3) Завершение работы.

Чтобы выбрать нужный пункт введите его номер.

Ваш выбор: 764

Ошибка! Введите номер соответсвуюгено пункта: н

Ошибка! Введите номер соответсвуюгено пункта: S

Ошибка! Введите номер соответсвуюгено пункта: s

Ошибка! Введите номер соответсвуюгено пункта: 1

Введите количество вершин в графе:
```

Рисунок 7 - Корректный ввод типа работы

```
Введите количество вершин в графе: 0

Ошибка! Размер матрицы должен задаваться числом, более 0 и менее 1000. Повторите попытку.

Введите количество вершин в графе: 1001

Ошибка! Размер матрицы должен задаваться числом, более 0 и менее 1000. Повторите попытку.

Введите количество вершин в графе: тысяча

Ошибка! Размер матрицы должен задаваться числом, более 0 и менее 1000. Повторите попытку.

Введите количество вершин в графе: thounds

Ошибка! Размер матрицы должен задаваться числом, более 0 и менее 1000. Повторите попытку.

Введите количество вершин в графе: 6
```

Рисунок 8 - Корректный ввод количества вершин

```
Выберите тип графа:
1) Ориентированный граф
2) Неориентированный граф
Ваш выбор: 543

Ошибка! Введите 1 или 2 для дальнейшей работы программы: ореинтированный

Ошибка! Введите 1 или 2 для дальнейшей работы программы: 1
```

Рисунок 9 - Корректный ввод типа вершин

```
Пензенский государственный университет
Кафедра "Вычислительная техника"
Курсовая работа
По курсу "Логика и основы алгоритмизации в инженерных задачах"
На тему "Реализация алгоритма поиска наибольшего паросочетания"
Приняли: Юрова О.В. и Акифьев И.В.
Выполнил: Байков Алексей Владимирович, учебная группа 22ВВВ3(22ВВП2)

Меню:
1) Работа со случайным графом.
2) Работа с вашим графом из файла.
3) Завершение работы.

Чтобы выбрать нужный пункт введите его номер.

Ваш выбор: 1

Введите количество вершин в графе: 6

Выберите тип графа:
1) Ориентированный граф
2) Неориентированный граф
Ваш выбор: 1

Матрица смежности ориентированного графа:
0 0 0 0 1 0
0 0 0 0 0 0
0 0 0 0 0 1
0 0 0 0 1 0
0 0 0 0 0 0
0 1 1 0 0 0

Паросочетания и наибольшее:
(0, 4)
(0, 4) (2, 5)
(5, 1) (0, 4)
(5, 1) (5, 2) (0, 4)

Меню:
1) Работа со случайным графом.
2) Работа с вашим графом из файла.
3) Завершение работы.

Чтобы выбрать нужный пункт введите его номер.

Ваш выбор: 3

D:\КурсоваяЗСемКонец\x64\Debug\КурсоваяЗСемКонец.exe (процесс 13824) завершил работу с кодом 0.
Нажмите любую клавишу, чтобы закрыть это окно:
```

Рисунок 10 - Полная корректная работа 1 пункта

```
Пензенский государственный университет
Кафедра "Вычислительная техника"
Курсовая работа
По курсу "Логика и основы алгоритмизации в инженерных задачах"
На тему "Реализация алгоритма поиска наибольшего паросочетания"
Приняли: Юрова О.В. и Акифьев И.В.
Выполнил: Байков Алексей Владимирович, учебная группа 22ВВВ3(22ВВП2)

Меню:
1) Работа со случайным графом.
2) Работа с вашим графом из файла.
3) Завершение работы.

Чтобы выбрать нужный пункт введите его номер.

Ваш выбор: 2

Пример как выглядят матрицы:
Матрица неориентированного графа:       Матрица ориентированного графа:
           0 0 0 1                                 0 1 0 0
           0 0 1 1                                 0 0 1 0
           0 1 0 0                                 1 0 0 0
           1 1 0 0                                 0 0 1 0
```

Рисунок 11 - Оформление 2 пункта

```
Введите имя файла, содержащего матрицу графа: 5235.txt

Ошибка: Файл не найден!

Введите имя файла, содержащего матрицу графа:
```

Рисунок 12 - Устранение ошибки, если нет файла

```
Ошибка: Файл не найден!

Введите имя файла, содержащего матрицу графа: graph.txt

Файл пуст.

Попробуйте заполнить его и повторите попытку.

Удачи ;)
```

Рисунок 13 - Устранение ошибки, если файл пустой

```
Введите имя файла, содержащего матрицу графа: graph.txt

Ваша матрица не соотвествует графам.

Переделайте матрицу и запустите праграмму снова.

Удачи в следующих запусках ;)

D:\КурсоваяЗСемКонец\x64\Debug\КурсоваяЗСемКонец.exe (процесс 14072) завершил работу с кодом 0.
Нажмите любую клавишу, чтобы закрыть это окно:
```

Рисунок 14 - Устранение ошибки, если матрица не соответствует графам

```
Пензенский государственный университет
Кафедра "Вычислительная техника"
Курсовая работа
По курсу "Логика и основы алгоритмизации в инженерных задачах"
На тему "Реализация алгоритма поиска наибольшего паросочетания"
Приняли: Юрова О.В. и Акифьев И.В.
Выполнил: Байков Алексей Владимирович, учебная группа 22ВВВ3(22ВВП2)

Меню:
1) Работа со случайным графом.
2) Работа с вашим графом из файла.
3) Завершение работы.

Чтобы выбрать нужный пункт введите его номер.

Ваш выбор: 2

Пример как выглядят матрицы:
Матрица неориентированного графа:       Матрица ориентированного графа:
           0 0 0 1                              0 1 0 0
           0 0 1 1                              0 0 1 0
           0 1 0 0                              1 0 0 0
           1 1 0 0                              0 0 1 0

Введите имя файла, содержащего матрицу графа: orgraph.txt

Матрица орграфа:
0 1 0 0
0 0 0 1
0 1 0 0
1 0 0 0

Паросочетания и наибольшее:
(0, 1)
(0, 1) (1, 3)
(2, 1) (1, 3)
(3, 0) (2, 1)

Меню:
1) Работа со случайным графом.
2) Работа с вашим графом из файла.
3) Завершение работы.

Чтобы выбрать нужный пункт введите его номер.

Ваш выбор: 3

D:\КурсоваяЗСемКонец\x64\Debug\КурсоваяЗСемКонец.exe (процесс 1588) завершил работу с кодом 0.
Нажмите любую клавишу, чтобы закрыть это окно:
```

Рисунок 15 - Полная корректная работа 2 пункта

Таблица 1 – Описание поведения программы при тестировании

Описание теста	Ожидаемый результат	Полученный результат
Запуск программы	Вывод титульного листа. Вывод меню взаимодействия с пользователем	Верно
Выбор генерации матрицы	Создание случайными числами или считыванием из файла с автоматическим определением размера.	Верно
Устранение ошибок на этапе взаимодействия с пользователем	Учет возможного ошибочного ввода данных, некорректного файла	Верно
Работа с ориентированным и неориентированным графом	Успешная обработка матриц графа и орграфа	Верно
Вывод результатов	Информативный и корректный вывод	Верно

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

Ручной расчет задачи

Пошагово рассчитаем наибольшее паросочетание для данного графа, представленного в виде матрицы смежности:

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

Для нахождения всех наибольших паросочетаний в орграфе можно использовать алгоритм поиска максимального паросочетания, основанный на алгоритме поиска увеличивающих путей. Каждый увеличивающий путь добавляет одно ребро к текущему паросочетанию, пока больше увеличивающих путей не будет найдено.

Вершины 0 и 1 образуют паросочетание (0, 1), а вершины 1 и 3 образуют паросочетание (1, 3). Это два непересекающихся пути в орграфе. Также можно найти паросочетание (2, 1). Вы правы, в этой матрице есть несколько паросочетаний.

Итак, наибольшие паросочетания:
(0, 1) (1, 3)
(2, 1) (1, 3)
(3, 0) (2, 1)
Они все наибольшие для данной матрицы орграфа.

```
Матрица орграфа:
0 1 0 0
0 0 0 1
0 1 0 0
1 0 0 0

Паросочетания и наибольшее:
(0, 1)
(0, 1) (1, 3)
(2, 1) (1, 3)
(3, 0) (2, 1)
```

Рисунок 16- Сравнение с работой программы

Ручной расчет подтверждает правильность работы программы

Заключение

В процессе создания данного проекта разработана программа, реализующая алгоритм поиска наибольшего паросочетания, как в ориентированном, так и в неориентированном графе в MicrosoftVisualStudio 2022.

Была реализована возможность создания графа, как со случайными числами, так и работа с пользовательским графом путём считывания его из файла. При выполнении данной курсовой работы были приобретены навыки по осуществлению алгоритма поиска наибольшего паросочетания. Углублены знания языка программирования Си.

Программа обладает достаточным для использования набором функций. Она имеет простой пользовательский интерфейс из-за своей консольной природы, что исключает сложности создания графического пользовательского интерфейса.

Список литературы

1. Брайн Керниган, Деннис Ритчи «Язык программирования С» - Вильямс, 2019 г. – 253с.
2. Томас Кормен, Чарльз Лейзерсон, Рональд Ривест «Алгоритмы. Построение и анализ» - Вильямс, 2013 г. – 1324 с.
3. В.Н. Сачков «Комбинаторика и теория графов» - Диалектика, 2009 г. - 376 с.
4. Герберт Шилдт «С. Полное руководство» - Вильямс, 2006 г. – 800с.
5. Г. Хаггарти «Дискретная математика для программистов» - Издание 2-е, 2001 г. – 401 с.
6. Мельников Б. Ф. Алгоритмы – М.: БХВ, 2003. -192с.
7. Никлаус Вирт. Алгоритмы и структуры данных – М.:ДМК-Пресс,2016. -272с.

Приложение А.
Листинг программы.

```
#define _CRT_SECURE_NO_WARNINGS
#include <stdio.h>
#include <stdlib.h>
#include <stdbool.h>
#include <time.h>
#include <locale.h>
#include <malloc.h>

// Объявление функции, отвечающей за обход графа в глубину
bool bpm(bool** graph, int u, bool visited[], int matchR[], int
vertices);

// Объявление функции, отвечающей за вывод всех паросочетаний
void printAllMatchings(bool** graph, int vertices);

// Объявление функции, отвечающей вывод
void printMatching(bool** graph, int matchR[], int vertices);

// Функция, отвечающие за поиск всех паросочетаний
void printAllMatchingsUtil(bool** graph, int u, bool visited[],
int matchR[], int vertices);

// Выделение памяти для матрицы смежности
int memoryAllocation(bool** graph, int vertices);

// Объявление функции, отвечающей за генерацию матриц графа и
оргграфа
void generateRandomGraph(bool** graph, int vertices, int
graphType);

// Объявление функции, отвечающей вывод матрицы смежности
void printGraph(bool** graph, int vertices);

// Объявление функции, которая определяет тип графа
bool isDirected(bool** graph, int verticesFile);

// Объявление функции, которая доопределяет тип графа
bool isUndirected(bool** graph, int vertices);

// Основная функция
int main()
{
    srand(time(NULL)); // Подключаем случайные числа
    setlocale(LC_ALL, "RUS"); // Подключение русского языка

    // Вывод титульного листа
    printf("Пензенский государственный университет\n");
    printf("Кафедра \"Вычислительная техника\"\n");
```

```
    printf("Курсовая работа\n");
    printf("По курсу \"Логика и основы алгоритмизации в
инженерных задачах\"\n");
    printf("На тему \"Реализация алгоритма поиска наибольшего
паросочетания\"\n");
    printf("Приняли: Юрова О.В. и Акифьев И.В.\n");
    printf("Выполнил: Байков Алексей Владимирович, учебная
группа 22ВВВ3(22ВВП2)\n\n");
    //

    int vertices; // Размер графа по количеству вершин при
работе со случайным графом
    bool work = true; // Переменная для цикла
    while (work)
    {   // Меню
        printf("Меню: \n");
        printf("1) Работа со случайным графом.\n");
        printf("2) Работа с вашим графом из файла.\n");
        printf("3) Завершение работы.\n\n");
        printf("Чтобы выбрать нужный пункт введите его
номер.\n\n");
        printf("Ваш выбор: ");
        //

        // Выбор пользователем режима
        int typeMode;
        scanf("%d", &typeMode);

        // Цикл устранения некорректного ввода пользователем
        while (typeMode != 1 && typeMode != 2 && typeMode != 3)
        {   // Вывод предупреждения
            printf("\nОшибка! Введите номер соответствующего
пункта: ");
            while (getchar() != '\n'); // Устранение
зацикливания
            scanf("%d", &typeMode); // Повторный ввод
        }
        // Если пользователь выбрал работу со случайным графом
        if (typeMode == 1)
        {
            // Цикл устранения некорректного ввода пользователем
            bool num = false; // Переменная для работы цикла
            while (!num)
            {
                // Ввод вершин графа
                printf("\nВведите количество вершин в графе: ");
                if (scanf("%d", &vertices) == 1 && vertices > 0
&& vertices <= 1000)  // Если введено число, которое более 0 и
менее 1000
                {
                    num = true; // Выход из цикла
                }
                else // Иначе продолжаем обработку
```

```
                {   // Вывод предупреждения
                    printf("\nОшибка! Размер матрицы должен
задаваться числом, больше 0. Повторите попытку.\n\n");
                    while (getchar() != '\n'); // Устранение
зацикливания
                }
            }

            int graphType; // Тип графа
            printf("\nВыберите тип графа:\n");
            printf("1) Ориентированный граф\n");
            printf("2) Неориентированный граф\n");

            // Ввод пользователем типа графа
            printf("Ваш выбор: ");
            scanf("%d", &graphType);

            // Цикл устранения некорректного ввода пользователем
            while (graphType != 1 && graphType != 2)
            {   // Вывод предупреждения
                printf("\nОшибка! Введите 1 или 2 для дальнейшей
работы программы: ");
                while (getchar() != '\n'); // Устранение
зацикливания
                scanf("%d", &graphType); // Повторный ввод
            }
            bool** graph = (bool**)malloc(vertices *
sizeof(bool*));

            memoryAllocation(graph, vertices); // Вызов функции,
которая выделяет память под матрицу смежности

            generateRandomGraph(graph, vertices, graphType); //
Вызов функции, отвечающую за генерацию матриц смежности

            printGraph(graph, vertices); // Вызов функции,
отвечающую за генерацию матриц смежности

            // Вывод наибольшего паросочетания, Вызов функции,
отвечающую за нахождение наибольшего паросочетания
            printf("\nПаросочетания и наибольшее:\n");
            printAllMatchings(graph, vertices); // Вызов
функции, отвечающую за вывод паросочетаний
            printf("\n");

            // Освобождение выделенной памяти, выделенной для
матрицы смежности
            for (int i = 0; i < vertices; i++)
            {
                free(graph[i]);
            }
            free(graph);
        } // Если пользователь выбрал работу со своим графом
```

```
        else if (typeMode == 2)
        {
            // Пример как выглядят матрицы графов
            printf("\n");
            printf("Пример как выглядят матрицы: \n");
            printf("Матрица неориентированного графа: \tМатрица
ориентированного графа: \n");
            printf("\t    0 0 0 1\t\t\t\t   0 1 0 0\n\t    0 0 1
1\t\t\t\t   0 0 1 0\n\t    0 1 0 0\t\t\t\t   1 0 0 0\n\t    ");
            printf("1 1 0 0\t\t\t\t   0 0 1 0\n");

            int countAll = 0; // Переменная для подсчета вершин
с мусором
            int verticesFile = 0; // Переменная для подсчета
вершин без мусора
            char fileName[50]; // Для имени файла
            FILE* fp;

            // Ввод имени файла
            printf("\nВведите имя файла, содержащего матрицу
графа: ");
            scanf("%s", &fileName);

            // Проверка на наличие файла в цикле
            fp = fopen(fileName, "r");
            while (fp == NULL)
            {
                printf("\nОшибка: Файл не найден!\n");
                printf("\nВведите имя файла, содержащего матрицу
графа: ");
                while (getchar() != '\n'); // Устранение
зацикливания
                scanf("%s", &fileName);
                fp = fopen(fileName, "r");
            }
            fclose(fp);

            // Проверка на пустоту файла
            fp = fopen(fileName, "r");
            fseek(fp, 0, SEEK_END);
            if (ftell(fp) == 0)
            {
                printf("\nФайл пуст.\n\nПопробуйте заполнить его
и повторите попытку.\n\nУдачи ;)\n");
                return 1;
            }
            fclose(fp);

            char c; // Переменная для подсчета вершин
            // Определяем количество вершин
            fp = fopen(fileName, "r");
            while ((c = fgetc(fp)) != '\n' && c != EOF) // Пока
не дошли до перехода на новую строку и до конца файла
```

```
            {  // Если есть пробел
                if (c == ' ')
                {
                    countAll++; // Увеличиваем счетчик вершин с
мусором
                    continue; // Пропускаем итерацию
                }
                verticesFile++; // Увеличиваем счетчик вершин
без мусора
            }
            fclose(fp);

            bool** graph = (bool**)malloc(verticesFile *
sizeof(bool*));

            memoryAllocation(graph, verticesFile); // Вызов
функции, которая выделяет память под матрицу смежности
            printf("\n");

            int num; // Переменная фильтрации пробела
            // Заполняем матрицу введенную из файла
            fp = fopen(fileName, "r");
            // Считываем матрицу из файла
            for (int i = 0; i < countAll + 1; i++)
            {
                for (int j = 0; j < countAll + 1; j++)
                {
                    fscanf(fp, "%d", &num);
                    if (num == ' ')  // Если пришел пробел
                    {
                        continue;  // Пропускаем итерацию
                    }
                    else
                    {
                        graph[i][j] = num; // Иначе добавляем в
массив
                    }
                }
            }
            fclose(fp);

            if (isDirected(graph, verticesFile)) // Если матрица
соответствует орграфу
            {
                printf("Матрица орграфа: \n");
            }
            else if (isUndirected(graph, verticesFile)) // Если
матрица соответствует графу
            {
                printf("Матрица графа: \n");
            }
            else // Иначе вывод сообщения
            {
```

```
                printf("Ваша матрица не соответствует
графам.\n\nПеределайте матрицу и запустите программу
снова.\n\nУдачи в следующих запусках ;)\n");
                break; // Конец работы
            }
            printGraph(graph, verticesFile); // Вывод графа

            // Вывод наибольшего паросочетания, Вызов функции,
отвечающую за нахождение наибольшего паросочетания
            printf("\nПаросочетания и наибольшее:\n");
            printAllMatchings(graph, verticesFile); // Вызов
функции, отвечающую за вывод паросочетаний
            printf("\n");

            // Освобождение выделенной памяти, выделенной для
матрицы смежности
            for (int i = 0; i < verticesFile; i++)
            {
                free(graph[i]);
            }
            free(graph);
        }
        else
        {
            work = false;
        }
    }
    return 0; // Завершение работы
}

// Функция отвечающая за обход графа
bool bpm(bool** graph, int u, bool visited[], int matchR[], int
vertices)
{
    for (int v = 0; v < vertices; v++)
    {
        // Если есть ребро между сравниваемыми вершинами и была
ли вершина v посещена ранее.
        if (graph[u][v] && !visited[v])
        {
            visited[v] = true; // Отмечаем посещенной
            // Если вершина не имеет пару или можно найти другую
пару
            if (matchR[v] < 0 || bpm(graph, matchR[v], visited,
matchR, vertices))
            {
                matchR[v] = u; // Устанавливаем пару
                return true;
            }
        }
    }
    return false; // Если не удалось найти путь
}
```

```
// Функция, отвечающая за вывод паросочетания
void printMatching(bool** graph, int matchR[], int vertices)
{
    bool* visited = (bool*)calloc(vertices, sizeof(bool)); //
Выделяем память под массив посещенных вершин и заполняем 0

    for (int i = 0; i < vertices; i++)
    {
        if (matchR[i] != -1 && !visited[i]) // Если у вершины i
есть пара, и вершина i не была посещена
        {
            printf("(%d, %d) ", matchR[i], i); // Выводим пару
            visited[i] = true; // Помечаем вершину посещенной
            visited[matchR[i]] = true;
        }
    }
    printf("\n");

    free(visited); // Освобождаем память
}

// Функция, отвечающие за поиск всех паросочетаний
void printAllMatchingsUtil(bool** graph, int u, bool visited[],
int matchR[], int vertices)
{
    for (int v = 0; v < vertices; v++)
    {
        if (graph[u][v] && !visited[v]) // Если есть ребро и
вершина не посещена
        {
            visited[v] = true; // Помещаем вершину как
посещенную
            if (matchR[v] < 0 || !visited[matchR[v]]) // Если у
вершины ещё нет пары или у её пары есть другая свободная вершина
            {
                matchR[v] = u; // Создаем пару
                printMatching(graph, matchR, vertices); //
Выводим пары
            }
        }
    }
    visited[u] = false; // Помещаем не посещенные
}

// Функция, отвечающая за вывод всех паросочетания
void printAllMatchings(bool** graph, int vertices)
{
    int* matchR = (int*)malloc(vertices * sizeof(int)); //
Выделение памяти под массив пар
    // Если была ошибка при выделение памяти
    if (matchR == NULL)
    {
```

```
        printf("Ошибка выделения памяти.\n");
        return;
    }

    for (int i = 0; i < vertices; i++)
    {
        matchR[i] = -1; // Указываем, что никак паросочетаний
еще не было
    }

    bool* visited = (bool*)malloc(vertices * sizeof(bool)); //
Выделение памяти под массив текущих посещенных вершин
    // Проверяем на ошибку при выделение памяти
    if (visited == NULL)
    {
        printf("Ошибка выделения памяти.\n");
        free(matchR); // Освобождение памяти
        return;
    }

    for (int i = 0; i < vertices; i++)
    {
        visited[i] = false; // Отмечаем, что все вершины не
посещенные
    }

    for (int u = 0; u < vertices; u++)
    {
        printAllMatchingsUtil(graph, u, visited, matchR,
vertices); // Для каждой вершины находим пару через другие
вершины
    }

    free(matchR); // Освобождение памяти
    free(visited); // Освобождение памяти
}

// Функция, отвечающая за генерацию матриц
void generateRandomGraph(bool** graph, int vertices, int
graphType)
{
    // Если пользователь выбрал работу с орграфом
    if (graphType == 1)
    {
        printf("\nМатрица смежности ориентированного графа:\n");

        for (int i = 0; i < vertices; i++)
        {
            for (int j = 0; j < vertices; j++)
            {
                if (i == j) {
                    graph[i][j] = 0; // Нет петель
                }
```

```
                else {
                    int randomValue = rand() % 5;
                    if (randomValue == 0)
                    {
                        graph[i][j] = 1; // Устанавливаем ребро
от i к j
                    }
                    else {
                        graph[i][j] = 0;
                    }
                }
            }
        }
    } // Если пользователь выбрал работу с графом
    else
    {
        printf("\nМатрица смежности графа:\n");
        // Заполнение матрицы смежности случайными числами
        for (int i = 0; i < vertices; i++)
        {
            for (int j = i; j < vertices; j++)
            {
                // Устанавливаем на главной диагонали 0
                if (i == j)
                {
                    graph[i][j] = 0;
                }
                else
                {
                    int randomValue = rand() % 2;
                    graph[i][j] = randomValue;
                    graph[j][i] = randomValue; // Матрица
симметрична
                }
            }
        }
    }
}

// Функция, отвечающая за выделение памяти
int memoryAllocation(bool** graph, int vertices)
{
    // Выделение памяти под матрицу смежности с проверкой
    if (graph == NULL)
    { // Если произошла ошибка при выделение памяти, завершить
работу
        printf("Ошибка выделения памяти.\n");
        return 0;
    }
    // Выделение памяти под матрицу смежности с проверкой
    for (int i = 0; i < vertices; i++)
    {
        graph[i] = (bool*)malloc(vertices * sizeof(bool));
```

```
        if (graph[i] == NULL)
        { // Если произошла ошибка при выделение памяти,
завершить работу, освободить память
            printf("Ошибка выделения памяти.\n");
            for (int j = 0; j < i; j++)
            {
                free(graph[j]);
            }
            free(graph);
            return 0;
        }
    }
}

// Функция выводящая матрицу смежности
void printGraph(bool** graph, int vertices)
{
    // Проход по всей матрице
    for (int i = 0; i < vertices; i++)
    {
        for (int j = 0; j < vertices; j++)
        {
            printf("%d ", graph[i][j]); // Вывод каждого ребра
        }
        printf("\n");
    }
}

// Функция, отвечающая для определение типа графа
bool isDirected(bool** graph, int vertices)
{
    for (int i = 0; i < vertices; i++)
    {
        for (int j = 0; j < vertices; j++)
        {
            if (graph[i][j] && !graph[j][i])
            {
                return true; // Найдено ребро в одну сторону и
отсутствие обратного, граф ориентированный
            }
        }
    }
    return false; // Нет прямых отличий, граф неориентированный
}

// Функция, отвечающая за определение ошибочной матрицы
bool isUndirected(bool** graph, int vertices)
{
    for (int i = 0; i < vertices; i++)
    {
        if (graph[i][i] != 0)
        {
            return false;
```

```
        }
        for (int j = i; j < vertices; j++)
        {
            if (graph[i][j] != graph[j][i])
            {
                return false; // Найдено противоречие в
противоположных направлениях, граф неориентированный
            }
        }
    }
    return true; // Нет прямых отличий в противоположных
направлениях, граф ориентированный
```