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»

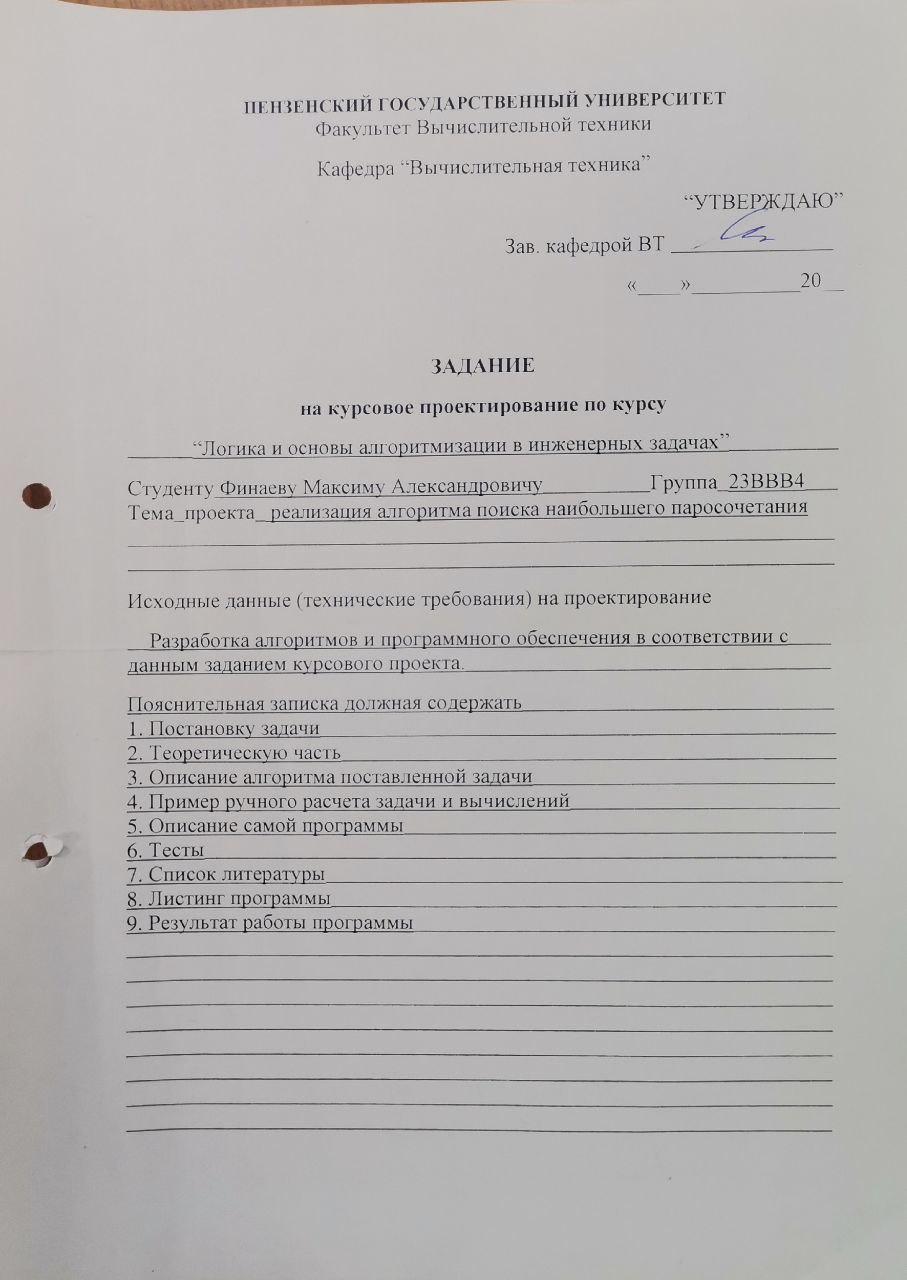
Выполнил:

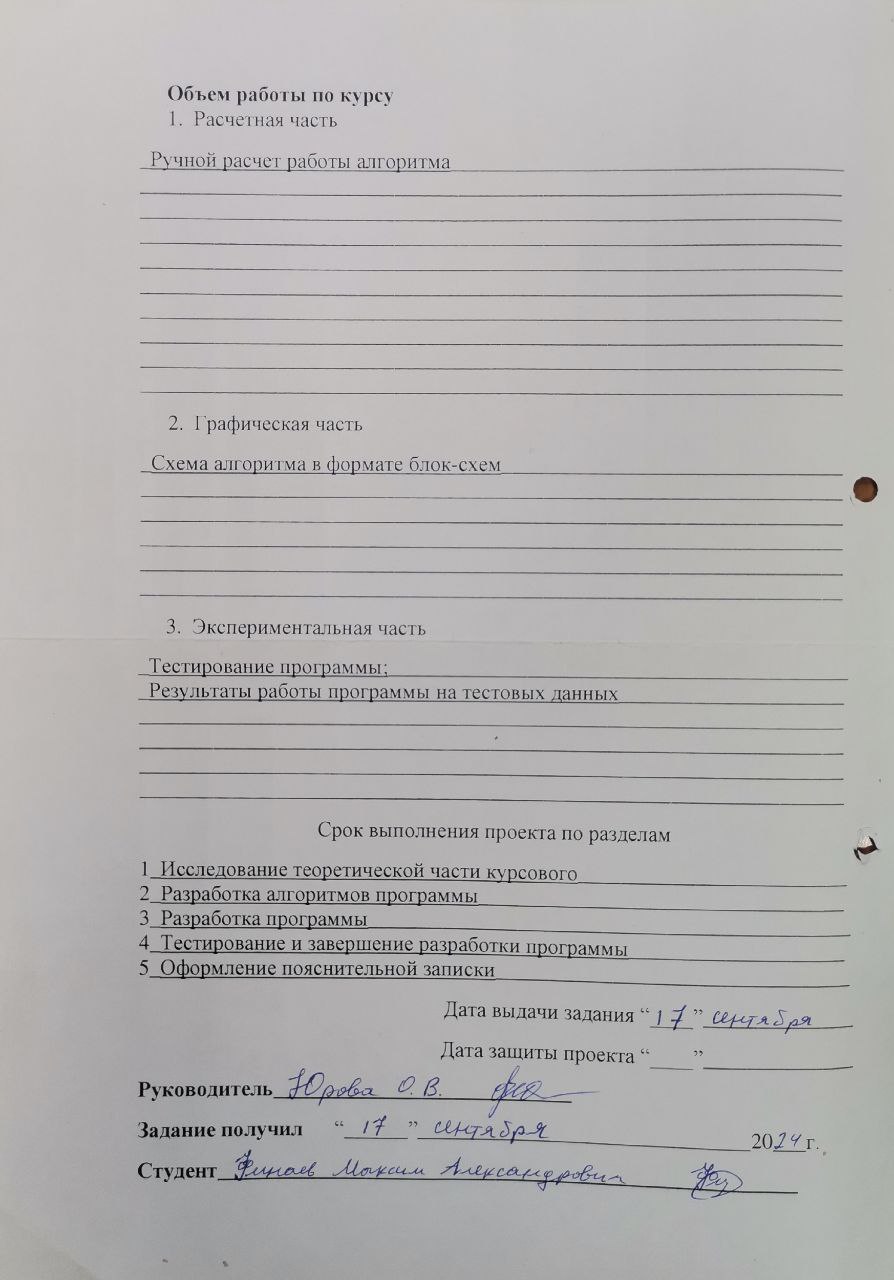
Ст. гр. 23ВВВ4 Финаев М. А.

Приняла:

к.т.н. Юрова О. В.

Пенза 2024





**Содержание**

[Реферат 5](#_Toc185201400)

[Введение 6](#_Toc185201401)

[1. Постановка задачи 7](#_Toc185201402)

[2. Теоретическая часть задания 8](#_Toc185201403)

[3. Описание алгоритма программы 9](#_Toc185201404)

[4. Описание программы 13](#_Toc185201405)

[5. Тестирование 17](#_Toc185201406)

[6. Ручная проверка 22](#_Toc185201407)

[Заключение 23](#_Toc185201408)

[Список литературы 24](#_Toc185201409)

[Приложение А. Листинг программы. 25](#_Toc185201410)

Реферат

Отчет 23 стр, 23 рисунка, 1 таблица, 1 приложение

Алгоритм Хопкрофта-Карпа, теория графов, двудольный граф, паросочетание, матрица смежности.

Цель работы – разработка программы реализации алгоритма Хопкрофта-Карпа для нахождения наибольшего паросочетания в двудольном графе

Данная курсовая работа посвящена реализации алгоритма поиска максимального паросочетания в двудольном графе с использованием алгоритма Хопкрофта-Карпа. Задача максимального паросочетания является важной в теории графов и имеет множество практических приложений, например, в задачах распределения работы между работниками, поиска оптимальных путей и т.д. В курсовой работе описан алгоритм Хопкрофта-Карпа, а также приведена его реализация на языке программирования Python. Основное внимание уделено разработке программы, ее тестированию и анализу производительности.

Введение

Алгоритм Хопкрофта-Карпа — алгоритм, принимающий на вход [двудольный граф](https://ru.wikipedia.org/wiki/%D0%94%D0%B2%D1%83%D0%B4%D0%BE%D0%BB%D1%8C%D0%BD%D1%8B%D0%B9_%D0%B3%D1%80%D0%B0%D1%84) и возвращающий максимальное [по мощности](https://ru.wikipedia.org/wiki/%D0%9C%D0%BE%D1%89%D0%BD%D0%BE%D1%81%D1%82%D1%8C_%D0%BC%D0%BD%D0%BE%D0%B6%D0%B5%D1%81%D1%82%D0%B2%D0%B0) [паросочетание](https://ru.wikipedia.org/wiki/%D0%9F%D0%B0%D1%80%D0%BE%D1%81%D0%BE%D1%87%D0%B5%D1%82%D0%B0%D0%BD%D0%B8%D0%B5), то есть наибольшее множество рёбер, таких что никакие два не имеют общую вершину.

Алгоритм был создан [Джоном Хопкрофтом](https://ru.wikipedia.org/wiki/%D0%A5%D0%BE%D0%BF%D0%BA%D1%80%D0%BE%D1%84%D1%82,_%D0%94%D0%B6%D0%BE%D0%BD) и [Ричардом Карпом](https://ru.wikipedia.org/wiki/%D0%9A%D0%B0%D1%80%D0%BF,_%D0%A0%D0%B8%D1%87%D0%B0%D1%80%D0%B4_%D0%9C%D1%8D%D0%BD%D0%BD%D0%B8%D0%BD%D0%B3) в 1973 году. Как и ранее созданные алгоритмы (такие как [венгерский алгоритм](https://ru.wikipedia.org/wiki/%D0%92%D0%B5%D0%BD%D0%B3%D0%B5%D1%80%D1%81%D0%BA%D0%B8%D0%B9_%D0%B0%D0%BB%D0%B3%D0%BE%D1%80%D0%B8%D1%82%D0%BC) и алгоритм из работы Эдмондса), алгоритм Хопкрофта-Карпа в цикле увеличивает паросочетание, находя увеличивающие пути ([цепи](https://ru.wikipedia.org/wiki/%D0%A6%D0%B5%D0%BF%D1%8C_%D0%B2_%D0%B3%D1%80%D0%B0%D1%84%D0%B5), рёбра которых попеременно принадлежат паросочетанию и не принадлежат ему, причём первая и последняя вершина не принадлежат паросочетанию; выполнив чередование паросочетания вдоль цепи, то есть убрав из паросочетания рёбра, бывшие в цепи, и добавив те, которых в нём не было, можно получить большее паросочетание). Вместо того чтобы находить один увеличивающий путь, алгоритм находит максимальное множество кратчайших увеличивающих путей. Та же идея используется в более сложных алгоритмах для поиска паросочетаний в недвудольных графах с тем же асимптотическим временем работы, как у алгоритма Хопкрофта — Карпа.

В качестве среды разработки был выбран PyCharm 2023, язык программирования – Python. Целью данной курсовой работы является разработка программы на языке Python. Именно с его помощью реализуется алгоритм Хопкрофта – Карпа, осуществляющий нахождения наибольшего паросочетания в двудольном графе.

1. Постановка задачи

Требуется разработать программу, которая найдёт наибольшее паросочетание двудольного графа.

Задана матрица смежности двудольного графа, где одна часть вершин обозначена как U, а другая — как V. Требуется найти максимальное паросочетание, то есть максимальное количество рёбер, соединяющих вершины из двух частей графа так, чтобы каждое ребро соединяло различные вершины, и ни одна вершина не встречалась в нескольких рёбрах.

Интерфейс ­– консоль.

Устройства ввода – клавиатура и мышь.

1. Теоретическая часть задания

Граф G (рисунок 1) является двудольным графом с множеством вершин U и V соответственно.

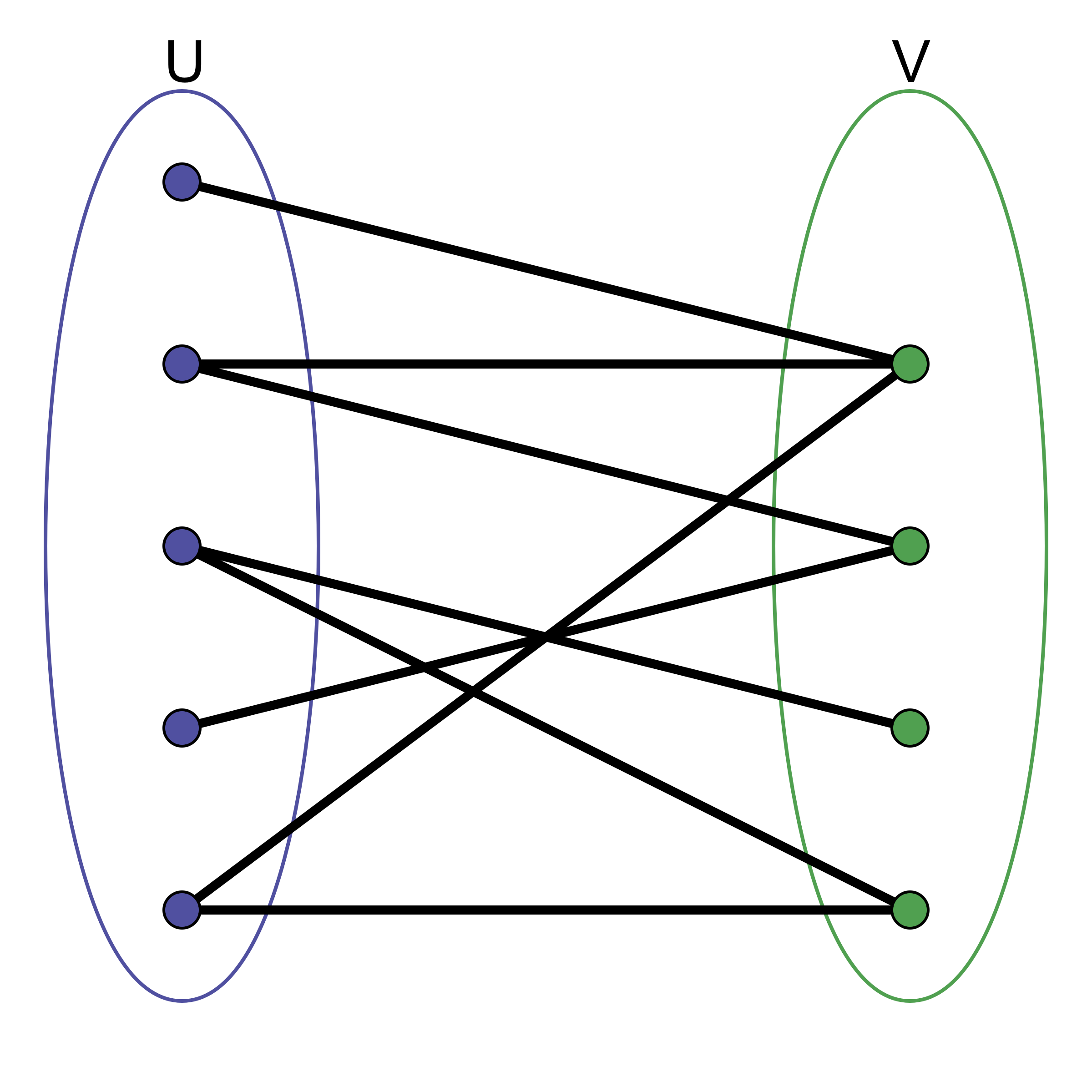


Рисунок 1 – Пример двудольного графа

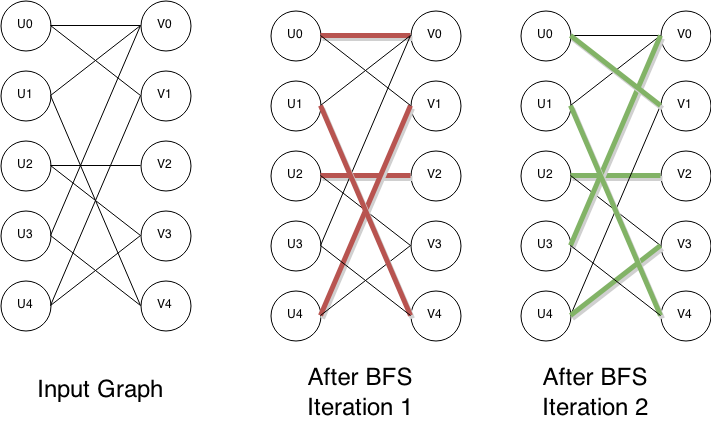
Алгоритм Хопкрофта-Карпа используется для нахождения максимального паросочетания в двудольных графах. Он работает за время O(√V \* E), где V — количество вершин в графе, а E — количество рёбер. Алгоритм использует два основных этапа: поиск в ширину (BFS) и поиск в глубину (DFS). BFS используется для нахождения максимального расстояния в графе, а DFS — для построения паросочетаний. Алгоритм оптимизирует процесс поиска паросочетаний, минимизируя количество лишних обходов графа.

Рисунок 2 – Пример работы алгоритма Хопкрофта-Карпа

1. Описание алгоритма программы

Для программной реализации понадобиться объявить такие методы, как bfs, dfs, hopcroft\_Karp, generate\_random\_graph, input\_graph, save\_results.

Основной цикл программы:

* Пользователь выбирает: ввести матрицу смежности вручную, рандомно либо же завершить программу
* Пользователь вводит с клавиатуры количество вершин левой и правой частей графа.
* Матрица смежности заполняется либо рандомно, либо вручную задаётся пользователем в зависимости от выбора в первом пункте
* Выполняется алгоритм Хопкрофта-Карпа для нахождения наибольшего паросочетания
* Матрица смежности, наибольшее количество паросочетаний и связь между вершинами данного графа выводятся на экран
* Пользователь выбирает завершить программу или повторить ее с первого пункта

Глобальные переменные:

graph[] – массив, который задаёт матрицу смежности

pairU[] – массив пар для левой части графа

pairV[] – массив пар для правой части графа

dist[] – массив расстояний

Переменные n и m хранят количество вершин в двух частях двудольного графа.

Ниже псевдокодом представлены методы bfs(), dfs(u), hopcroft\_Karp(), generate\_random\_graph(), input\_graph(), save\_results(max\_matching)и main()

**bfs():**

1) Создать пустую очередь

2) Для каждого i от 0 до n-1:

3) Если pairU[i] == -1:

4) dist[i] = 0

5) Добавить i в очередь

6) Иначе:

7) dist[i] = бесконечность

8) dist[n] = бесконечность

9) front = 0

10)Пока front < длина очереди:

11) u = очередь[front]

12) front = front + 1

13) Если dist[u] < dist[-1]:

14) Для каждого v от 0 до m-1:

15) Если graph[u][v] == 1 и dist[pairV[v]] == бесконечность:

16) dist[pairV[v]] = dist[u] + 1

17) Добавить pairV[v] в очередь

18) Вернуть True, если dist[n] != бесконечность, иначе False

**dfs(u):**

1) Если u != -1:

2) Для каждого v от 0 до m-1:

3) Если graph[u][v] == 1 и dist[pairV[v]] == dist[u] + 1:

4) Если dfs(pairV[v]):

5) pairV[v] = u

6) pairU[u] = v

7) Вернуть True

8) dist[u] = бесконечность

9) Вернуть False

10) Вернуть True

**hopcroft\_karp():**

1) pairU = список из n элементов, равных -1

2) pairV = список из m элементов, равных -1

3) dist = список из n элементов, равных 0

5) matching = 0

6) Пока bfs() возвращает True:

7) Для каждого u от 0 до n-1:

8) Если pairU[u] == -1 и dfs(u) возвращает True:

9) matching = matching + 1

10) Вернуть matching

**generate\_random\_graph():**

1) Инициализировать генератор случайных чисел с помощью текущего времени

2) Для каждого i от 0 до n-1:

3) Для каждого j от 0 до m-1:

4) graph[i][j] = случайное число 0 или 1

**input\_graph():**

1) Напечатать "Введите матрицу смежности (где 1 — ребро, 0 — отсутствие ребра):"

2) Для каждого i от 0 до n-1:

3) Для каждого j от 0 до m-1:

4) Повторять до корректного ввода:

5) Запросить у пользователя значение для graph[i][j]

6)Если значение не равно 0 или 1:

7) Напечатать "Некорректный ввод. Введите 0 или 1."

8) Иначе:

9) Записать введенное значение в graph[i][j] и выйти из цикла

**save\_results(max\_matching):**

1) Открыть файл "results.txt" для добавления данных

2) Записать количество вершин в левой и правой части графа

3) Записать матрицу смежности в файл

4) Записать размер максимального паросочетания

5) Записать связи между вершинами в файл

6) Напечатать матрицу смежности и размер максимального паросочетания в консоль

7) Напечатать связи между вершинами в консоль

**main():**

1) cont = 1

2) Пока cont равно 1:

3) Напечатать меню с опциями:

3) 1. Ввести матрицу смежности вручную

4) 2. Генерировать случайную матрицу смежности

5) 3. Выход

6) Запросить у пользователя выбор опции

7) Если опция == 3:

8) Выйти из программы

9) Если опция != 1 и опция != 2:

10) Напечатать "Неверный выбор. Попробуйте снова."

11) Перейти к следующей итерации

12) Запросить количество вершин в левой и правой части графа

13) Инициализировать graph как пустую матрицу размером n x m

14) Если опция == 1:

15) Ввести матрицу вручную с помощью input\_graph()

16) Иначе, если опция == 2:

17) Сгенерировать случайную матрицу с помощью generate\_random\_graph()

18) Вычислить максимальное паросочетание с помощью hopcroft\_karp()

19) Сохранить результат в файл с помощью save\_results()

20) Запросить у пользователя, хочет ли он повторить выполнение программы

21) Если пользователь не хочет продолжить, выйти из программы

22) Напечатать "Программа завершена."

1. Описание программы

Для написания данной программы использован язык программирования Python. Python является высокоуровневым языком, который широко используется в разных областях, включая разработку веб-приложений, автоматизацию, обработку данных и математическое моделирование. Он популярен среди разработчиков благодаря простоте синтаксиса, мощным библиотекам и активному сообществу.

Проект представляет собой консольное приложение, которое реализует алгоритм Хопкрофта-Карпа для нахождения максимального паросочетания в двудольном графе. Программа выполняется в терминале через среду разработки PyCharm, что позволяет удобно разрабатывать и отлаживать код. Программа состоит из нескольких методов, каждый из которых выполняет свою специфическую задачу: bfs, dfs, hopcroft\_Karp, generate\_random\_graph, input\_graph, save\_results и main.

bfs – функция поиска в ширину, находит кратчайшие пути в графе от не связных вершин левой части графа(U) до правой части(V) с помощью поиска в ширину

dfs – функция поиска в глубину, рекурсивно ищет путь увеличения для паросочетания, пытаясь сопоставить вершины левой части с правой.

hopcroft\_Karp – функция алгоритма Хопкрофта-Карпа, реализуется сам алгоритм Хопкрофта-Карпа для нахождения наибольшего паросочетания в двудольном графе

generate\_random\_graph – функция для генерации случайной матрицы смежности

input\_graph – функция для ручного ввода матрицы смежности пользователем

save\_results – функция для сохранения результатов в файл, а также вывода их в консоль

main - основная функция программы, содержит цикл, который предоставляет пользователю мену для выбора действия и управляет выполнением программы.

Работа программы начинается с вывода меню, в котором пользователь должен выбрать: вариант ввода матрицы смежности (вручную либо рандомно) или завершить программу. Если пользователь выбрал какой-либо ввод матрицы смежности, то он также вводит количество левой и правой частей двудольного графа. После этого заполняется сама матрица смежности. Выполняется алгоритм поиска наибольшего паросочетания Хопкрофта-Крапа, результат записывается в файл, а также выводится на экран.

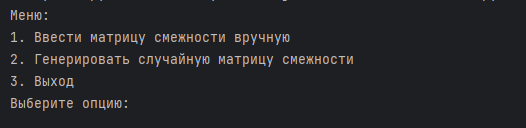
Далее показан начальный вид консоли при запуске программы:

Рисунок 3 – Вывод меню

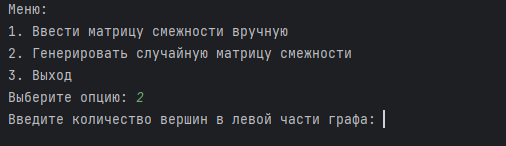
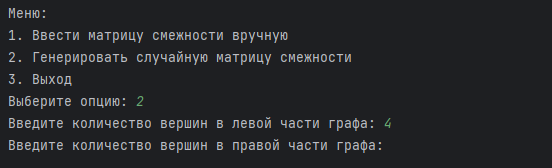


Рисунок 4 – Выбор варианта ввода матрицы смежности



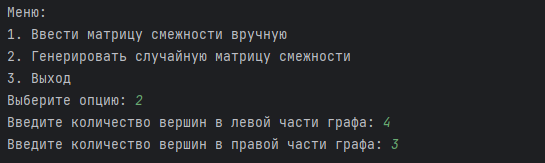
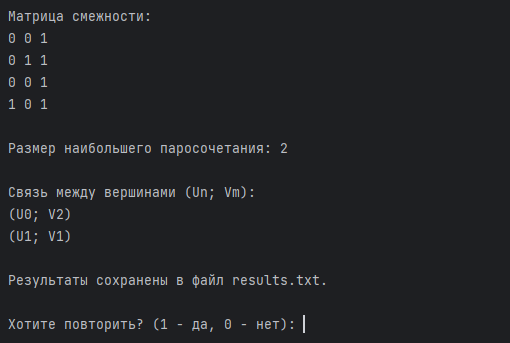
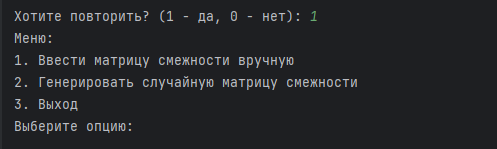
Рисунок 5 – Ввод количества вершин левой части графа

Рисунок 6 – Ввод количества вершин правой части графа



Рисунок 7 – Вывод результатов на экран и запрос на повторение программы

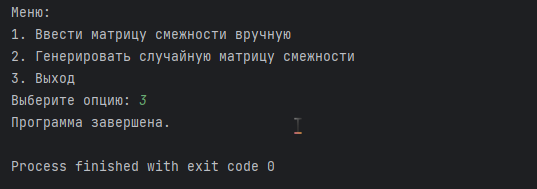
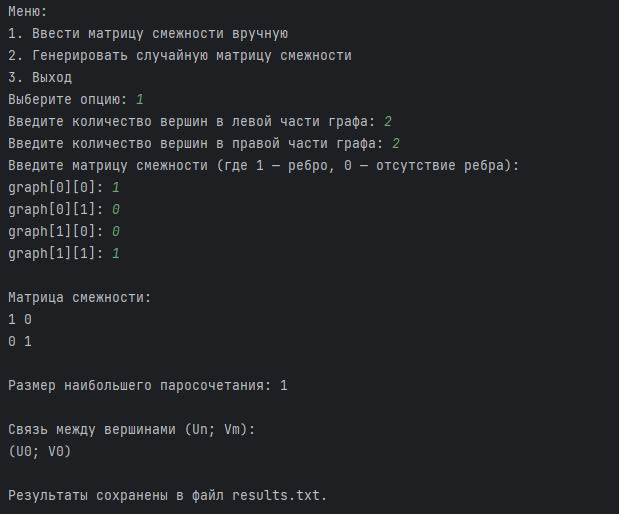
Рисунок 8 – Вывод меню при запросе на повторение программы

Рисунок 9 – Завершение работы программы из меню



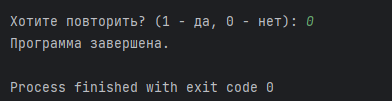
Рисунок 10 – Ручной ввод матрицы смежности

Рисунок 11 – Завершение работы программы при запросе на повтор

1. Тестирование

Среда разработки PyCharm представляет 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, поиском в глубину и ширину, сохранением результатов в файл, а также работой алгоритма Хопкрофта-Карпа.

Ниже продемонстрирован результат тестирования функционала программы при выборе способа заполнения матрицы смежности, вводе пользователем количества вершин левой и правой частей двудольного графа.

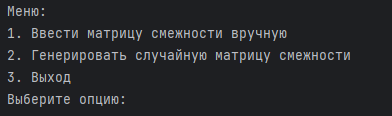


Рисунок 12 – тестирование: запуск программы

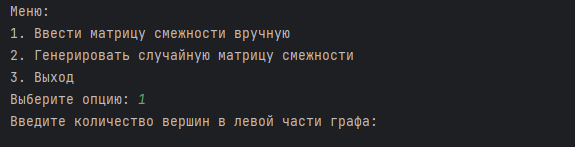


Рисунок 13 – тестирование: выбор ручного ввода матрицы смежности

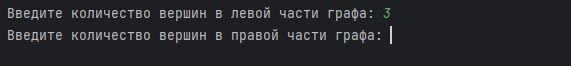


Рисунок 14 – тестирование: ввод количества вершин левой части графа

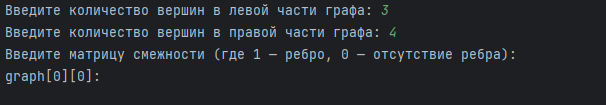


Рисунок 15 – тестирование: ввод количества вершин правой части графа

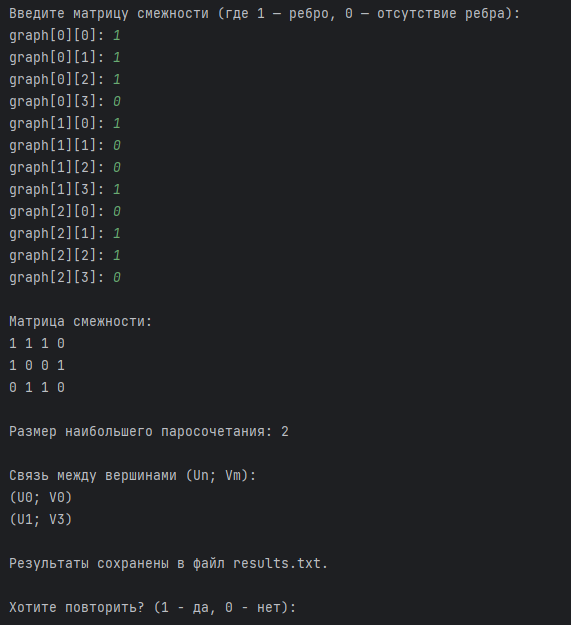


Рисунок 16 – тестирование: ввод матрицы смежности вручную

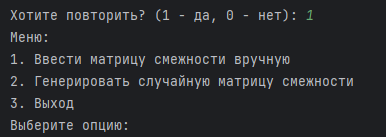


Рисунок 17 – тестирование: повторение программы после вывода результатов

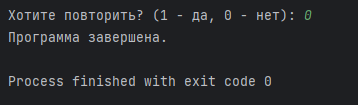


Рисунок 18 – тестирование: завершение работы программы после вывода результатов

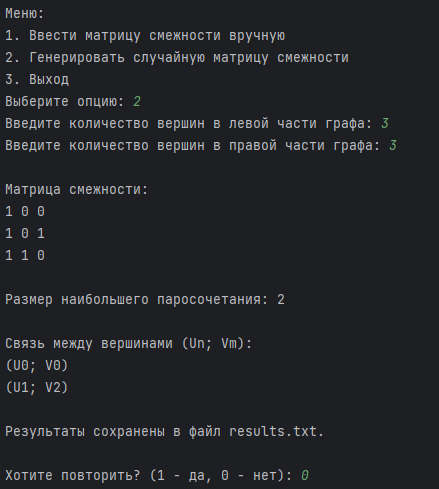


Рисунок 19 – тестирование: автоматическое заполнение матрицы смежности

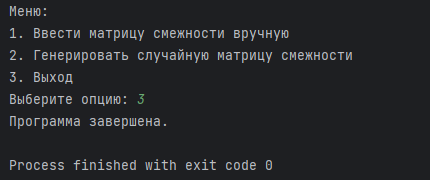


Рисунок 20 – тестирование: завершение работы программы из меню

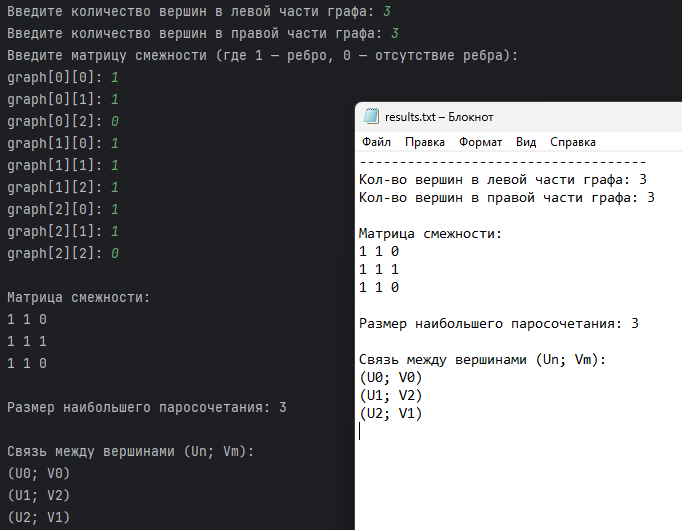


Рисунок 21 – тестирование: сохранение результатов в файл

Таблица 1 – описание поведения программы при тестировании

|  |  |  |
| --- | --- | --- |
| Описание теста | Ожидаемый результат | Полученный результат |
| Запуск программы | Вывод меню | Верно |
| Выбор ручного ввода матрицы смежности | Ожидание ввода количества вершин левой части графа | Верно |
| Ввод количества вершин левой части графа | Ожидание ввода количества вершин правой части графа | Верно |
| Ввод количества вершин правой части графа | Ожидание ручного ввода матрицы смежности | Верно |
| Ввод матрицы смежности | Вывод результатов: матрица смежности, количество наибольших паросочетаний, связь между вершинами. | Верно |
| Повторение программы после вывода результата | Вывод первоначального меню | Верно |
| Завершение программы после вывода результатов | Завершение работы программы | Верно |
| Автоматическое заполнения матрицы смежности | Ввод левой, а затем правой частей графа, Вывод результатов: матрица смежности, количество наибольших паросочетаний, связь между вершинами. | Верно |
| Завершение программы из меню | Завершение работы программы | Верно |
| Сохранение результатов в файл | Обновление файла с результатами | Верно |

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

1. Ручная проверка



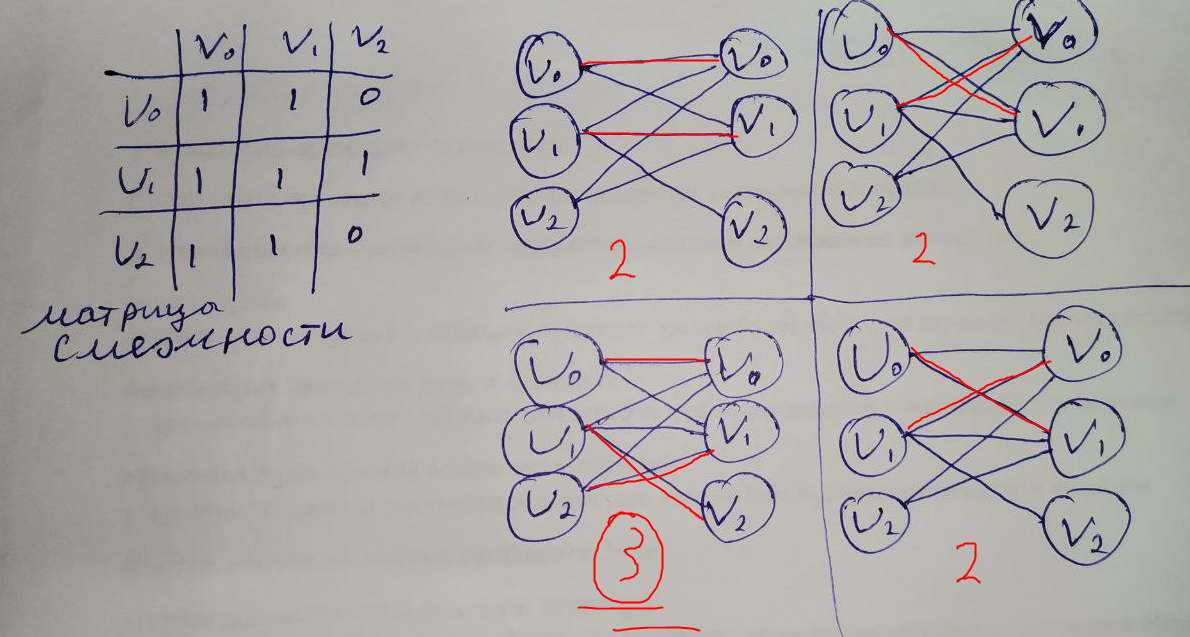


Рисунок 22 – Ручное нахождение наибольшего

паросочетания двудольного графа

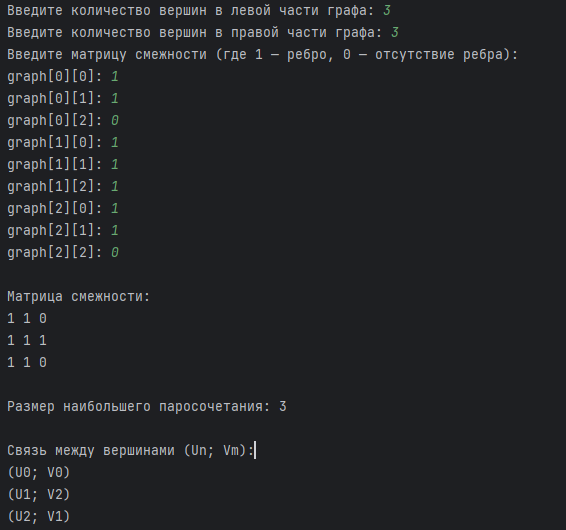


Рисунок 23 – Сравнение ручной проверки с работой пограммы

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

Заключение

Таким образом, в процессе выполнения данной курсовой работы была разработана программа, реализующая алгоритм Хопкрофта-Карпа для нахождения наибольшего паросочетния двудольного графа, в PyCharm 2023.

При выполнении были получены навыки разработки программ, закреплены методы обхода в глубину и ширину, освоен принцип работы алгоритма Хопкрофта-Карпа. Улучшены навыки владения языком программирования Python.

В будущем программу можно улучшить, добавив возможность работы с более крупными графами, а также поддерживать динамическое изменение графа (добавление или удаление рёбер) в процессе работы программы. Также можно рассмотреть использование других алгоритмов для поиска паросочетаний в графах, таких как алгоритм Прима или алгоритм Эдмондса.

Список литературы

1. Хопкрофт, Дж. Е., Карп, Р. М. (1973). "Алгоритм для нахождения максимальных паросочетаний в двудольных графах". SIAM Journal on Computing, 2(4), 225-231.
2. Кормен, Т. Х., Лейсерсон, Ч. Е., Ривест, Р. Л., Стейн, К. (2009). "Введение в алгоритмы" (3-е изд.). MIT Press.
3. Вазирани, В. В. (2001). "Приближенные алгоритмы". Springer.
4. Тарьян, Р. Э. (1983). "Структуры данных и алгоритмы для сетей". SIAM.
5. Берже, К. (1973). "Теория графов". North-Holland Publishing Company.
6. Вест, Д. Б. (2001). "Введение в теорию графов" (2-е изд.). Pearson.
7. Кнут, Д. Э. (1973). "Искусство программирования, том 1: Основы алгоритмов" (2-е изд.). Addison-Wesley.

Приложение А. Листинг программы.

import random  
import time  
  
# Глобальные переменные  
graph = [] # Матрица смежности  
pairU = [] # Массив пар для левой части графа  
pairV = [] # Массив пар для правой части графа  
dist = [] # Массив расстояний  
n, m = 0, 0 # Количество вершин в левой и правой части графа  
  
  
# Поиск в ширину (BFS)  
def bfs():  
 queue = []  
 for i in range(n):  
 if pairU[i] == -1:  
 dist[i] = 0  
 queue.append(i)  
 else:  
 dist[i] = float('inf')  
  
 dist[n] = float('inf')   
  
 front = 0  
 while front < len(queue):  
 u = queue[front]  
 front += 1  
 if dist[u] < dist[n]:   
 for v in range(m):  
 if graph[u][v] and dist[pairV[v]] == float('inf'):  
 dist[pairV[v]] = dist[u] + 1  
 queue.append(pairV[v])  
  
 return dist[n] != float('inf')   
  
# Поиск в глубину (DFS)  
def dfs(u):  
 if u != -1:  
 for v in range(m):  
 if graph[u][v] and dist[pairV[v]] == dist[u] + 1:  
 if dfs(pairV[v]):  
 pairV[v] = u  
 pairU[u] = v  
 return True  
 dist[u] = float('inf')  
 return False  
 return True  
  
  
# Алгоритм Хопкрофта-Карпа  
def hopcroft\_karp():  
 global pairU, pairV, dist  
 pairU = [-1] \* n  
 pairV = [-1] \* m  
 dist = [0] \* (n + 1)   
  
 matching = 0  
 while bfs():  
 for u in range(n):  
 if pairU[u] == -1 and dfs(u):  
 matching += 1  
 return matching  
  
  
# Функция для генерации случайной матрицы смежности  
def generate\_random\_graph():  
 random.seed(time.time())  
 for i in range(n):  
 for j in range(m):  
 graph[i][j] = random.randint(0, 1)  
  
  
# Функция для ввода матрицы вручную  
def input\_graph():  
 print("Введите матрицу смежности (где 1 — ребро, 0 — отсутствие ребра):")  
 for i in range(n):  
 for j in range(m):  
 while True: # Цикл для проверки правильности ввода  
 try:  
 value = int(input(f"graph[{i}][{j}]: "))  
 if value in [0, 1]:  
 graph[i][j] = value  
 break # Прерываем цикл, если ввод правильный  
 else:  
 print("Некорректный ввод. Введите 0 или 1.")  
 except ValueError:  
 print("Некорректный ввод. Пожалуйста, введите число 0 или 1.")  
  
  
  
# Функция для сохранения результата в файл results.txt  
def save\_results(max\_matching):  
 # Сохранение результатов в файл  
 with open("results.txt", "a") as file:  
 file.write("------------------------------------\n")  
 file.write(f"Кол-во вершин в левой части графа: {n}\n")  
 file.write(f"Кол-во вершин в правой части графа: {m}\n\n")  
  
 file.write("Матрица смежности:\n")  
 for i in range(n):  
 file.write(" ".join(map(str, graph[i])) + "\n")  
  
 file.write(f"\nРазмер наибольшего паросочетания: {max\_matching}\n")  
  
 file.write("\nСвязь между вершинами (Un; Vm):\n")  
 for i in range(n):  
 if pairU[i] != -1:  
 file.write(f"(U{i}; V{pairU[i]})\n")  
  
 # Вывод результатов в консоль  
 print("\nМатрица смежности:")  
 for i in range(n):  
 print(" ".join(map(str, graph[i]))) # Выводим строку матрицы в консоль  
 print(f"\nРазмер наибольшего паросочетания: {max\_matching}")  
 print("\nСвязь между вершинами (Un; Vm):")  
 for i in range(n):  
 if pairU[i] != -1:  
 print(f"(U{i}; V{pairU[i]})")  
  
 print("\nРезультаты сохранены в файл results.txt.")  
  
  
# Основная программа  
def main():  
 global n, m, graph  
  
 cont = 1  
  
 while cont:  
 print("Меню:")  
 print("1. Ввести матрицу смежности вручную")  
 print("2. Генерировать случайную матрицу смежности")  
 print("3. Выход")  
 option = int(input("Выберите опцию: "))  
  
 if option == 3:  
 break  
 elif option != 1 and option != 2:  
 print("Неверный выбор. Попробуйте снова.")  
 continue  
  
 n = int(input("Введите количество вершин в левой части графа: "))  
 m = int(input("Введите количество вершин в правой части графа: "))  
  
 # Инициализация графа  
 graph = [[0] \* m for \_ in range(n)]  
  
 if option == 1:  
 input\_graph()  
 elif option == 2:  
 generate\_random\_graph()  
  
 # Вычисление максимального паросочетания  
 max\_matching = hopcroft\_karp()  
  
 # Сохранение результата в файл  
 save\_results(max\_matching)  
  
 # Повторить?  
 cont = int(input("\nХотите повторить? (1 - да, 0 - нет): "))  
  
 print("Программа завершена.")  
  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 main()