

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 независимых множеств  
вершин графа A»

Выполнил:

Студент группы 22ВВП2

Майоров Н.А.

Принял:

Акифьев И.В.

  
Хорошо  
26.12.23г.

ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ  
Факультет Вычислительной техники

Кафедра "Вычислительная техника"

"УТВЕРЖДАЮ"

Зав. кафедрой ВТ

«    »    20   

ЗАДАНИЕ

на курсовое проектирование по курсу

«Логика и основы алгоритмизации вычислительных задач»  
Студенту Майорову Никите Александровичу Группа 22 ВВВЗ  
Тема проекта Валификация алгоритма поиска подмножеств вершин графа

Исходные данные (технические требования) на проектирование

- Разработка алгоритмов и программного обеспечения в соответствии с данными заданиями курсового проекта. Пояснительная записка должна содержать:
- 1) Постановку задачи;
  - 2) Исходные данные задачи;
  - 3) Описание алгоритма постановки задачи;
  - 4) Пример ручного расчета задачи и вычисления на компьютере (учетные данные алгоритма);
  - 5) Описание программы;
  - 6) Источники;
  - 7) Список литературы;
  - 8) Листинг программы;
  - 9) Результаты работы программы.

Объем работы по курсу

1. Расчетная часть

Ручной расчет работы алгоритма

*[Handwritten signature]*

2. Графическая часть

Схема алгоритма в формате  
"блок-схема"

3. Экспериментальная часть

Тестирование программы.  
Результаты работы программы на  
тестовых данных.

Срок выполнения проекта по разделам

- 1 Исследование теоретических основ
- 2 курсовая к
- 3 Разработка алгоритмов программы к
- 4 Разработка программы к
- 5 Тестирование, и завершение разработки
- 6 программы к
- 7 Оформление печатной работы к
- 8

Дата выдачи задания "25" сентября

Дата защиты проекта " " "

Руководитель И.В. Акифьев

*[Handwritten signature]*

Задание получил "25" сентября

2023г.

Студент Майоров Никита Александрович

# 1. Содержание

1. Содержание .....	4
2. Реферат .....	5
3. Введение .....	6
4. Постановка задачи.....	7
5. Теоретическая часть задания .....	8
6. Описание алгоритма программы.....	9
7. Описание программы.....	14
8. Тестирование .....	17
9. Ручной расчет задачи .....	20
10. Заключение .....	21
11. Список используемых источников .....	22
12. Приложение А. Листинг программы.....	23

## **2. Реферат**

Отчет 27 стр, 10 рисунков.

**ГРАФ, ТЕОРИЯ ГРАФОВ, МНОЖЕСТВА, НЕЗАВИСЫЕ ВЕРШИНЫ.**

Цель исследования – разработка программы, способная искать независимые множества вершин графа, используя алгоритм поиска независимых множеств вершин графа.

В работе рассмотрены правила поиска независимых вершин. Поиск независимых вершин графа осуществляется с использованием рекурсивного алгоритма.

### 3. Введение

Алгоритм поиска независимых множеств вершин использует алгоритм поиска в глубину (DFS) для систематического исследования вершин графа. Для каждой вершины проверяется, не связана ли она с вершинами уже включенными в текущее независимое множество. Если вершина независима, она добавляется в текущее множество, и рекурсивно вызывается для следующей вершины. В результате формируется вектор множеств, представляющих все найденные независимые множества вершин графа.

Отличие поиска в глубину от поиска в ширину заключается в том, что (в нашем случае граф неориентированный) результатом алгоритма поиска в глубину является некоторый маршрут, следуя которому можно обойти последовательно все вершины графа, доступные из начальной вершины.

В качестве среды разработки мною была выбрана среда *Microsoft Visual Studio 2010*, язык программирования – C++.

Целью данной курсовой работы является разработка программы на языке C++, который является широко используемым по всему миру. С его помощью в курсовом проекте реализован алгоритм поиска независимых множеств вершин графа.

#### **4. Постановка задачи**

Требуется разработать программу, которая найдет независимые множества вершин графа, используя алгоритм поиска в глубину.

Исходные граф в программе должен задаваться матрицей смежности и случайно генерироваться. Также пользователю предоставлена возможность задать граф самому, указав имя файла с графом. Программа должна работать так, чтобы пользователь вводил количество вершин для генерации матрицы смежности. После обработки этих данных пользователь должен получить сгенерированный граф, а также все независимые множества вершин этого графа. Необходимо предусмотреть различные исходы поиска, чтобы программа не выдавала ошибок и работала правильно. Устройство ввода – клавиатура и мышь.

## 5. Теоретическая часть задания

Графы, в которых все ребра являются звеньями, то есть порядок двух концов ребра графа не существенен, называются неориентированными.

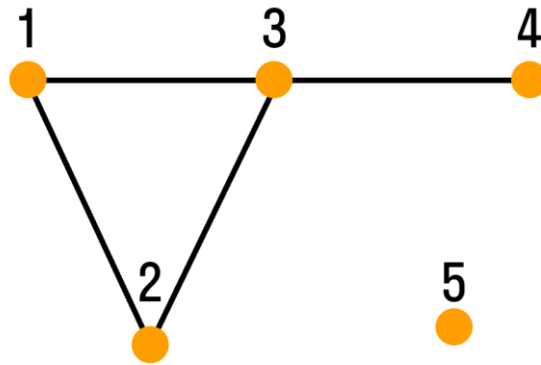


Рисунок 1 – Пример неориентированного графа

Существует много алгоритмов на графах, в основе которых лежит систематический перебор вершин графа, такой, что каждая вершина графа просматривается только один раз, и переход от одной вершины к другой осуществляется по ребрам графа. Мы остановимся на методе перебора – поиск в глубину. Он начинается с выбора одной вершины и затем идет "вглубь" по ребрам как можно дальше, прежде чем вернуться и продолжить поиск. Алгоритм использует стек для хранения информации о том, куда двигаться дальше.

Независимые вершины в графе - это множество вершин, в котором никакие две вершины не соединены ребром. Иными словами, это подмножество вершин графа, в котором отсутствуют ребра между любыми двумя вершинами этого подмножества.



## 6. Описание алгоритма программы

Программная реализация алгоритма поиска независимых множеств вершин в графе включает в себя использование различных массивов для управления процессом и хранения промежуточных результатов. AdjacencyMatrix (int) хранит в себе матрицу смежности Графа, vec (int) – используется для отметки посещенных вершин в процессе поиска, component (int) - хранит вершины, образующие текущую компоненту связности при поиске независимых множеств, v\_propysk (int) – хранит индексы вершин, которые необходимо пропустить при следующем обходе.

Вначале происходит инициализация пустого вектора independentSets (int), который хранит найденные независимые множества вершин. После вызывается рекурсивная функция findAllIndependentSetsHelper, которая рекурсивно исследует все возможные независимые множества, начиная с начальной вершиной 0, пустым текущим множеством и вектором для хранения результатов. В vertex находится вершина, с которой поиск начинается, currentSet представляет множество, которое построится в результате работы алгоритма. Проверка, является ли текущая вершина vertex независимой от вершин в текущем множестве currentSet. Если да, то добавление ее в текущее множество. Если вершин независима, добавляет её в текущее множество. После чего происходит возврат найденных независимых множеств.

Затем метод вызывается рекурсивно для следующей вершины, которая идет после текущей в порядке обхода графа. Рекурсивный вызов продолжается до тех пор, пока все вершины не будут рассмотрены. После завершения рекурсивного вызова для текущей вершины, она удаляется из currentSet для возврата к предыдущему состоянию и рассмотрения других вариантов. После обхода всех вершин алгоритм завершает свою работу.

Ниже представлен псевдокод функции findAllIndependentSetsHelper() и частично main ().

## **findAllIndependentSetsHelper()**

1. `isIndependent = true`
2. для `v` в `currentSet`
  3. если `adjacencyMatrix[vertex][v] > 0`
    4. `isIndependent = false`
    5. прервать выполнение
    6. конец цикла
7. конец условия
8. если `isIndependent`
  9. `currentSet.push_back(vertex);`
  10. `independentSets.push_back(set<int>(currentSet.begin(), currentSet.end()));`
  11. для (`int nextVertex = vertex + 1` пока `nextVertex < size` делать `++nextVertex`)
    12. `findAllIndependentSetsHelper(nextVertex, currentSet, independentSets);`
  13. конец условия
  14. `currentSet.pop_back();`
15. конец цикла
16. если `vertex + 1 < size`
  17. `findAllIndependentSetsHelper(vertex + 1, currentSet, independentSets);`
18. конец цикла

## **main()**

1. size = 0
2. вывод “Введите размер графа (множества)”
3. ввод size
4. если size <= 0
  5. вывод “Граф такого размера не может быть создан”
  6. return 1;
7. конец цикла
8. вывод “Выберите способ задания графа (1 - из файла, 2 случайный):”
9. ввод choice
10. если choice == 1
  11. string filename;
  12. вывод “Введите имя файла:”
  13. ввод filename
  14. graph.inputFromFile(filename);
15. конец цикла
16. иначе если choice == 2
  17. для int i = 0 пока i < size делать ++i
    18. для int j = i + 1 пока j < size делать ++j
      19. если rand() % 2 == 0
        20. graph.addEdge(i, j);
    20. конец цикла
  21. конец условия
  22. конец условия
23. конец цикла
24. иначе
  25. вывод “Некорректный выбор.”
  26. return 1;

27. конец цикла
28. `int menuChoice;`
29. ВЫПОЛНЯТЬ
  30. `graph.displayMenu();`
  31. вывод “Выберите действие:”
  32. ввод `menuChoice`
  33. выбор `menuChoice`
  34. случай 1:
    35. `graph.printGraph();`
  36. прервать выполнение
  37. случай 2:
    38. `vector<set<int>> independentSets =`  
`graph.findAllIndependentSets();`
    39. вывод “Все найденные независимые множества  
 вершин графа:”
    40. для `size_t i = 0` пока `i < independentSets.size()`  
 делать `++i`
      41. вывод “Множество”
      42. для `const int& vertex :`  
`independentSets[i]`
      43. вывод `vertex`
      44. конец условия
    45. конец условия
  46. прервать выполнение
  47. случай 3:
    48. `vector<set<int>> independentSets =`  
`graph.findAllIndependentSets();`
    49. `graph.saveResults(independentSets);`

50. прервать выполнение

51. случай 4:

52. вывод “Программа завершена. Хорошего дня!”

53. прервать выполнение

54. иначе

55. вывод “Некорректный выбор. Попробуйте еще раз.”

56. конец условия

57. пока menuChoice != 4

58. конец выбора

Полный код программы можно увидеть в Приложении А.

## 7. Описание программы

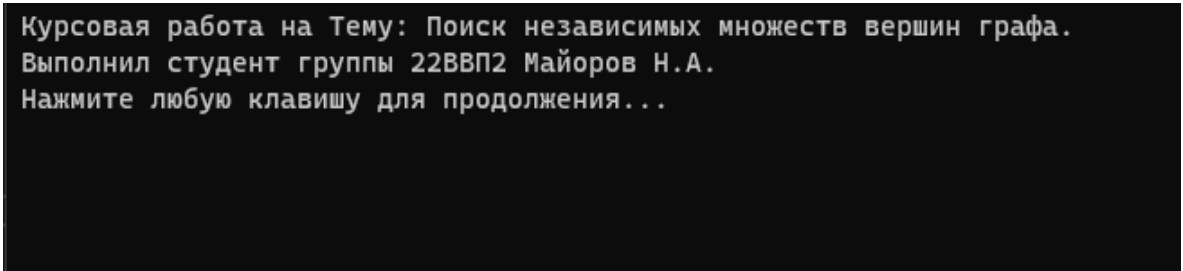
Для написания данной программы использован язык программирования C++. Язык программирования C++ - один из самых популярных языков программирования, который обрел популярность из-за возможности создания программ от простых консольных утилит до сложных игровых движков.

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

Данная программа является многомодульной, поскольку состоит из нескольких функций: `printGraph`, `findAllIndependentSetsHelper`, `saveResults`, `inputFromKeyboard`, `inputFromFile`, `displayMenu`, `main`.

Программа встречает пользователя текстом, который содержит в себе информацию о том, кто её разрабатывал. После появляется выбор задания графа – из файла или случайным образом. Если выбран способ задания графа случайным образом, в таком случае программа требует ввести размер графа от пользователя.

Далее выводится в меню, в котором можно выбрать действия: 1) Вывести матрицу смежности. 2) Найти все независимые множества вершин. 3) Сохранить результаты в файл. 4) Выйти из программы. При выборе первого варианта выводится сгенерированный граф. Нажав 2, на экран выводятся все независимые множества вершин, найденные в графе. При выборе третьего варианта результаты охраняются в файл `independent_sets.txt`. Последний вариант завершает работу программы.



```
Курсовая работа на Тему: Поиск независимых множеств вершин графа.  
Выполнил студент группы 22ВВП2 Майоров Н.А.  
Нажмите любую клавишу для продолжения...
```

Рисунок 2 – Заставочный экран программы

```

Введите размер графа (множества): 3
Выберите способ задания графа (1 – из файла, 2 – случайный): 2

Меню:
1. Вывести матрицу смежности
2. Найти все независимые множества вершин
3. Сохранить результаты в файл
4. Выйти из программы
Выберите действие: 1
Матрица смежности графа:
0 8 2
8 0 9
2 9 0

Меню:
1. Вывести матрицу смежности
2. Найти все независимые множества вершин
3. Сохранить результаты в файл
4. Выйти из программы
Выберите действие: |

```

Рисунок 3 – Случайная генерация графа и его вывод в консоль

```

Введите размер графа (множества): 3
Выберите способ задания графа (1 – из файла, 2 – случайный): 1
Введите имя файла: 111.txt

Меню:
1. Вывести матрицу смежности
2. Найти все независимые множества вершин
3. Сохранить результаты в файл
4. Выйти из программы
Выберите действие: 1
Матрица смежности графа:
0 2 0 0 10
2 0 2 0 0
0 2 0 0 0
0 0 0 0 2
10 0 0 2 0

Меню:
1. Вывести матрицу смежности
2. Найти все независимые множества вершин
3. Сохранить результаты в файл
4. Выйти из программы
Выберите действие:

```

111.txt
✕
+

Файл
Изменить
Просмотр

```

5
0 1
1 2
2 3
3 4
4 0

```

Рисунок 4 – Задание графа через файл и вывод его в консоль

```
4. Выйти из программы
Выберите действие: 2
Все найденные независимые множества вершин графа:
Множество 1: 0
Множество 2: 0 2
Множество 3: 0 2 3
Множество 4: 0 3
Множество 5: 0 2
Множество 6: 0 2 3
Множество 7: 0 3
Множество 8: 0 3
Множество 9: 1
Множество 10: 1 3
Множество 11: 1 4
Множество 12: 1 3
Множество 13: 1 4
Множество 14: 1 4
Множество 15: 2
Множество 16: 2 3
Множество 17: 2 4
Множество 18: 2 4
Множество 19: 3
Множество 20: 4

Меню:
1. Вывести матрицу смежности
2. Найти все независимые множества вершин
3. Сохранить результаты в файл
4. Выйти из программы
Выберите действие: |
```

Рисунок 5 – Выбрано действие найти независимые множества вершин

```
Меню:
1. Вывести матрицу смежности
2. Найти все независимые множества вершин
3. Сохранить результаты в файл
4. Выйти из программы
Выберите действие: 3
Результаты сохранены в файле independent_sets.txt
```

Рисунок 6 – Выбрано действие сохранить результаты в файл

```
Меню:
1. Вывести матрицу смежности
2. Найти все независимые множества вершин
3. Сохранить результаты в файл
4. Выйти из программы
Выберите действие: 4
Программа завершена. Хорошего дня!
```

Рисунок 7 – Выход из программ



## 8. Тестирование

Среда разработки Microsoft Visual Studio 2010 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.

Ниже продемонстрирован результат тестирования программы при вводе пользователем различных размеров и способов задания графа.

```
Введите размер графа (множества): 4
Выберите способ задания графа (1 – из файла, 2 – случайный): 2

Меню:
1. Вывести матрицу смежности
2. Найти все независимые множества вершин
3. Сохранить результаты в файл
4. Выйти из программы
Выберите действие: 1
Матрица смежности графа:
0 0 3 0
0 0 0 0
3 0 0 1
0 0 1 0

Меню:
1. Вывести матрицу смежности
2. Найти все независимые множества вершин
3. Сохранить результаты в файл
4. Выйти из программы
Выберите действие: 2
Все найденные независимые множества вершин графа:
Множество 1: 0
Множество 2: 0 1
Множество 3: 0 1 3
Множество 4: 0 1 3
Множество 5: 0 3
Множество 6: 0 3
Множество 7: 0 3
Множество 8: 1
```

Рисунок 8 – Тестирование при вводе размера 4 и случайной генерации

```
Введите размер графа (множества): 5
Выберите способ задания графа (1 – из файла, 2 – случайный): 1
Введите имя файла: 111.txt
```

Меню:

1. Вывести матрицу смежности
2. Найти все независимые множества вершин
3. Сохранить результаты в файл
4. Выйти из программы

Выберите действие: 1

Матрица смежности графа:

```
0 9 0 0 3
9 0 10 0 0
0 10 0 0 0
0 0 0 0 10
3 0 0 10 0
```

Меню:

1. Вывести матрицу смежности
2. Найти все независимые множества вершин
3. Сохранить результаты в файл
4. Выйти из программы

Выберите действие: 2

Все найденные независимые множества вершин графа:

```
Множество 1: 0
Множество 2: 0 2
Множество 3: 0 2 3
Множество 4: 0 3
Множество 5: 0 2
Множество 6: 0 2 3
Множество 7: 0 3
Множество 8: 0 3
Множество 9: 1
Множество 10: 1 3
Множество 11: 1 4
Множество 12: 1 3
Множество 13: 1 4
Множество 14: 1 4
Множество 15: 2
Множество 16: 2 3
Множество 17: 2 4
Множество 18: 2 4
Множество 19: 3
Множество 20: 4
```

Рисунок 9 – Тестирование при вводе размера 5 и задание графа из файла

Таблица 1 – Описание поведения программы при тестировании

Описание теста	Ожидаемый результат	Полученный результат
Запуск программы	Вывод приветственного сообщения. Требование ввести размера графа.	Верно
Выбор способа задания графа	Вывод сообщения о выборе способа задания графа: из файла или случайная генерация	Верно
Выбор вывести матрицу смежности	Вывод матрицы смежности заданного графа	Верно
Выбор найти все независимые множества вершин	Вывод всех независимых множеств вершин	Верно
Выбор сохранить результаты в файл	Сохранение в файл результатов, которые выполнила программа	Верно
Выбор выйти из программы	Выполнение выхода из программы	Верно

В результате тестирования было выявлено, что программа работает без ошибок и находит независимые множества вершин.

## 9. Ручной расчет задачи

Проведем проверку программы посредством ручных вычислений на примере графа с размером 3, сгенерированным случайно.

```
Введите размер графа (множества): 3
Выберите способ задания графа (1 – из файла, 2 – случайный): 2

Меню:
1. Вывести матрицу смежности
2. Найти все независимые множества вершин
3. Сохранить результаты в файл
4. Выйти из программы
Выберите действие: 1
Матрица смежности графа:
0 0 0
0 0 4
0 4 0

Меню:
1. Вывести матрицу смежности
2. Найти все независимые множества вершин
3. Сохранить результаты в файл
4. Выйти из программы
Выберите действие: 2
Все найденные независимые множества вершин графа:
Множество 1: 0
Множество 2: 0 1
Множество 3: 0 2
Множество 4: 0 2
Множество 5: 1
Множество 6: 2
```

Рисунок 10 – Граф для ручного расчета

Начинаем проверять с 0 вершины и идем в последующие, попутно проверяя, соединена ли вершина с другими вершинами. Вершина 0 – не соединена с другими вершинами. Вершина 2 – не соединена с другими вершинами. Вершина 3 – не соединена с другими вершинами. Множества размером 2: Вершины 0 и 1, так как они не соединены ребром; вершины 0 и 2, так как они не соединены ребром; вершины 1 и 2, так как они не соединены ребром. Из этого делаем вывод, что в данном случае вершины изолированы.

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

## **10. Заключение**

Таким образом, в процессе создания данного проекта разработана программа, реализующая алгоритм поиска независимых множеств вершин графа в Microsoft Visual Studio 2010.

При выполнении данной курсовой работы были получены навыки разработки программ и освоены приемы создания матриц смежностей, основанных на теории орграфов. Приобретены навыки по осуществлению алгоритма поиска независимых вершин графа. Усовершенствованы знания о графах. Углублены знания языка программирования C++.

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

Программа имеет небольшой, но достаточный для использования функционал возможностей. В дальнейшем программу можно улучшить, добавив интерфейс и возможность работы сразу с 2 графами.

## 11. Список используемых источников

1. Кристофидес Н. «Теория графов. Алгоритмический подход» - Мир, 1978 - URL: <https://studizba.com/files/show/pdf/53991-1-n-kristofides--teoriya-grafov.html> (дата обращения 01.12.2023)
2. Столяров А.В., Введение в язык Си++ / Андрей Михайлович Столяров, Пятое издание опубликовано издательством МАКС Пресс (Москва) в 2020 году – URL: <http://www.stolyarov.info/books/pdf/cppintro5.pdf> (дата обращения 03.12.2023)
3. Харви Дейтел, Пол Дейтел. Как программировать на C/C++. 2009 г. – URL: <http://ijevanlib.y-su.am/wp-content/uploads/2018/03/deytel.pdf> (дата обращения 10.12.2023)
4. Герберт Шилдт «Полный справочник по C++» - Вильямс, 2006 – URL: [https://sharpened.ucoz.ru/lib/polnyj\\_spravochnik\\_po\\_c-gerbert\\_shildt-2006.pdf](https://sharpened.ucoz.ru/lib/polnyj_spravochnik_po_c-gerbert_shildt-2006.pdf) (дата обращения 10.12.2023)

## 12. Приложение А.

### Листинг программы.

```
#include <iostream>
#include <fstream>
#include <vector>
#include <set>
#include <ctime>
#include <cstdlib>
#include <conio.h>

using namespace std;

class Graph {
private:
    int size;
    vector<vector<int>> adjacencyMatrix;

public:
    Graph(int n) : size(n), adjacencyMatrix(n, vector<int>(n, 0)) {}

    void addEdge(int u, int v) {
        int weight = rand() % 11; // Генерация случайного веса от 0 до 10
        adjacencyMatrix[u][v] = weight;
        adjacencyMatrix[v][u] = weight;
    }

    void printGraph() {
        cout << "Матрица смежности графа:" << endl;
        for (int i = 0; i < size; ++i) {
            for (int j = 0; j < size; ++j) {
                cout << adjacencyMatrix[i][j] << ' ';
            }
            cout << endl;
        }
    }

    vector<set<int>> findAllIndependentSets() {
        vector<set<int>> independentSets;
        set<set<int>> uniqueSets;

        vector<int> currentSet;
        findAllIndependentSetsHelper(0, currentSet, independentSets,
uniqueSets);

        return independentSets;
    }

    void findAllIndependentSetsHelper(int vertex, vector<int>& currentSet,
vector<set<int>>& independentSets, set<set<int>>& uniqueSets) {
        bool isIndependent = true;
        for (int v : currentSet) {
            if (adjacencyMatrix[vertex][v] > 0) { // Проверка, что вершины
не связаны
                isIndependent = false;
                break;
            }
        }

        if (isIndependent) {
            currentSet.push_back(vertex);
        }
    }
}
```

```

        set<int> currentSetUnique(currentSet.begin(), currentSet.end());

        // Проверка на уникальность множества перед добавлением
        if (uniqueSets.find(currentSetUnique) == uniqueSets.end()) {
            uniqueSets.insert(currentSetUnique);
            independentSets.push_back(currentSetUnique);
        }

        for (int nextVertex = vertex + 1; nextVertex < size;
++nextVertex) {
            findAllIndependentSetsHelper(nextVertex, currentSet,
independentSets, uniqueSets);
        }

        currentSet.pop_back();
    }

    if (vertex + 1 < size) {
        findAllIndependentSetsHelper(vertex + 1, currentSet,
independentSets, uniqueSets);
    }
}

void saveResults(const vector<set<int>>& independentSets) {
    ofstream outputFile("independent_sets.txt");
    if (!outputFile.is_open()) {
        cout << "Ошибка при открытии файла для записи результатов." <<
endl;
        return;
    }

    for (const auto& independentSet : independentSets) {
        for (const int& vertex : independentSet) {
            outputFile << vertex << " ";
        }
        outputFile << endl;
    }

    cout << "Результаты сохранены в файле independent_sets.txt" << endl;
    outputFile.close();
}

void inputFromKeyboard() {
    cout << "Введите количество рёбер: ";
    int edges;
    cin >> edges;
    if (edges <= 0) {
        cout << "Такое количество рёбер быть не может" << endl;
        return;
    }

    cout << "Введите рёбра в формате 'вершина1 вершина2', по одной паре
на строку:" << endl;
    for (int i = 0; i < edges; ++i) {
        int u, v;
        cin >> u >> v;
        addEdge(u, v);
    }
}

void inputFromFile(const string& filename) {
    ifstream inputFile(filename);
    if (!inputFile.is_open()) {

```



```

        cout << "Ошибка при открытии файла для чтения." << endl;
        return;
    }

    inputFile >> size; // Используем size текущего объекта

    // Инициализация графа с учетом количества вершин
    adjacencyMatrix = vector<vector<int>>(size, vector<int>(size, 0));

    for (int i = 0; i < size; ++i) {
        int u, v;
        inputFile >> u >> v;
        addEdge(u, v); // Используем метод addEdge для добавления ребер
    }

    inputFile.close();
}

void displayMenu() {
    cout << "\nМеню:\n";
    cout << "1. Вывести матрицу смежности\n";
    cout << "2. Найти все независимые множества вершин\n";
    cout << "3. Сохранить результаты в файл\n";
    cout << "4. Выйти из программы\n";
}

};

int main() {
    srand(time(NULL));
    setlocale(LC_ALL, "Rus");
    cout << "Курсовая работа на Тему: Поиск независимых множеств вершин
графа." << endl;
    cout << "Выполнил студент группы 22ВВП2 Майоров Н.А." << endl;
    cout << "Нажмите любую клавишу для продолжения..." << endl;
    _getch(); // Ожидание нажатия клавиши

    system("cls"); // Очистка консоли

    Graph graph(0); // Инициализация пустого графа

    double choice;
    int proverka;
    do {
        while (true) {
            cout << "Выберите способ задания графа (1 - из файла, 2 -
случайный): ";
            cin >> choice;

            // Проверка на дробное число
            if (cin.fail() || choice != static_cast<int>(choice)) {
                cout << "Некорректный выбор. Пожалуйста, введите целое
число." << endl;
                cin.clear(); // Очистка флага ошибки
                cin.ignore(numeric_limits<streamsize>::max(), '\n'); //
Прочистка буфера ввода до конца строки
            }
            else {
                proverka = static_cast<int>(choice);
                break; // Выход из цикла при корректном вводе
            }
        }
    }
}

```

```

switch (static_cast<int>(choice)) {
case 1: {
    string filename;
    cout << "Введите имя файла: ";
    cin >> filename;
    graph.inputFromFile(filename);
    break;
}
case 2: {
    int size;
    do {
        cout << "Введите размер графа (множества): ";
        cin >> size;
        if (size <= 0) {
            cout << "Граф такого размера не может быть создан.
Пожалуйста, введите корректное значение." << endl;
        }
    } while (size <= 0);

    graph = Graph(size); // Регенерация графа

    // Автоматическое задание графа (случайные связи)
    for (int i = 0; i < size; ++i) {
        for (int j = i + 1; j < size; ++j) {
            if (rand() % 2 == 0) {
                graph.addEdge(i, j);
            }
        }
    }
    break;
}
default:
    cout << "Некорректный выбор. Пожалуйста, введите 1 или 2." <<
endl;
}
} while (static_cast<int>(choice) != 1 && static_cast<int>(choice) != 2);

int menuChoice;
do {
    // Отображение меню и получение выбора пользователя
    graph.displayMenu();
    cout << "Выберите действие: ";
    cin >> menuChoice;

    switch (menuChoice) {
case 1:
    graph.printGraph();
    break;
case 2:
    // Поиск всех независимых множеств и вывод результатов
    {
        vector<set<int>> independentSets =
graph.findAllIndependentSets();
        cout << "Все найденные независимые множества вершин графа:" <<
endl;

        for (size_t i = 0; i < independentSets.size(); ++i) {
            cout << "Множество " << i + 1 << ": ";
            for (const int& vertex : independentSets[i]) {
                cout << vertex << " ";
            }
            cout << endl;
        }
    }
}
}

```

```

        }
    }
    break;
case 3:
    // Сохранение результатов в файл
    {
        vector<set<int>> independentSets =
graph.findAllIndependentSets();
        graph.saveResults(independentSets);
    }
    break;
case 4:
    cout << "Программа завершена. Хорошего дня!" << endl;
    break;
default:
    cout << "Некорректный выбор. Попробуйте еще раз." << endl;
}

} while (menuChoice != 4);

return 0;
}

```