

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»


Выполнил:

студент группы 22ВВП2

Куракин Н.Н.

Принял:

Акифьев И.В.

  
Отм. 26.12.23г.

Пенза, 2023

ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ  
Факультет Вычислительной техники  
Кафедра "Вычислительная техника"

"УТВЕРЖДАЮ"

Зав. кафедрой ВТ \_\_\_\_\_

« \_\_\_\_ » \_\_\_\_\_ 20 \_\_\_\_

ЗАДАНИЕ

на курсовое проектирование по курсу

"Поиск и основы алгоритмизации в инженерных задачах"  
Студенту Хирожкину Никите Николовичу Группа 22 ВВВ 3  
Тема проекта Реализация алгоритма поиска независимых множеств вершин графа

Исходные данные (технические требования) на проектирование

Разработка алгоритмов и программного обеспечения в соответствии с данными задания курсового проекта. Техническая записка должна содержать:

1. Постановку задачи;
2. Теоретическую часть задания;
3. Описание алгоритма поставленной задачи;
4. Пример ручного расчета задачи и вычисления (на определенном этапе работы алгоритма);
5. Описание программы;
6. Тесты;
7. Список литературы;
8. Листинг программы;
9. Результаты работы программы.

## Объем работы по курсу

### 1. Расчетная часть

Путь и расчет работы алгоритма.

### 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 Изучение теоретической части курсового к
- 2
- 3 Разработка алгоритмов программы к
- 4 Разработка программы к
- 5 Тестирование и завершение разработки программы
- 6 к
- 7 Оформление пояснительной записки к
- 8

Дата выдачи задания "25" сентября

Дата защиты проекта " " "

Руководитель Акиерьев И.В.

Задание получил "25" сентября 2023 г.

Студент Курочкин Никита Николаевич Кир

## Содержание

1. Реферат .....	5
2. Введение .....	6
3. Постановка задачи.....	7
4. Теоретическая часть задания .....	8
5. Описание алгоритма программы.....	12
FindNotSmej() .....	13
FindSet() .....	13
6. Описание программы .....	16
7. Тестирование .....	24
8. Ручной расчет задачи .....	28
9. Заключение .....	31
10. Список литературы .....	32
11. Приложение А. ....	33
12. Листинг программы. ....	33
Файл source.h .....	33
Файл menu.h.....	37
Файл source.c .....	55

# **1. Реферат**

Отчет 54 стр., 44 рисунка.

Цель исследования – разработка программы поиска независимых множеств вершин в неориентированном графе, используя алгоритм поиска независимых множеств вершин графа.

В работе рассмотрены способ поиска независимых вершин графа и этапы алгоритма нахождения независимых множеств вершин графа.

Установлено, что с помощью данного алгоритма можно найти все независимые множества вершин графа.

## 2. Введение

Для полного понимания алгоритма поиска независимых множеств вершин графа следует дать определение искомым множествам. Множество вершин графа называется независимым, если никакие две вершины этого множества не являются смежными. Под смежностью подразумевается связь/соединение между вершинами, поэтому в данном множестве никакие две вершины не соединены друг с другом ребром. Следовательно, подграф, порожденный независимым множеством, будет представлять из себя пустой граф. Под пустым графом понимается такой граф, который не содержит ни одного ребра.

Общий смысл алгоритма поиска независимых множеств вершин графа заключается в поиске таких наборов множеств, подграф которых будет представлять из себя пустой граф. Этим он эффективен для графов, представленных в программе в виде матрицы смежности: каждая вершина предстает в виде набора нулей и единиц, и требуется лишь реализация логических выражений, которые будут осуществлять поставленный смысл алгоритма.

С другой стороны, алгоритм требует некоторых изменений для графов, представленных в виде структур смежности и матриц инцидентности. Зато имеется возможность реализовать данный алгоритм рекурсией, что позволит ускорить работу программы.

В качестве среды разработки мною была выбрана среда Microsoft Visual Studio 2022, язык программирования – Си.

Целью работы является написание программы на языке Си, который отличается универсальностью и скоростью выполнения.

### **3. Постановка задачи**

Требуется разработать программу, которая выполняет поиск независимых множеств вершин графа.

Исходный граф в программе должен задаваться матрицей смежности, при его генерации должны быть предусмотрены граничные условия. Программа должна работать так, чтобы пользователь генерировал графы(с вводом их порядка) или загружал их из файла. После пользователю предлагается выбор: просмотр матриц смежности графов(с возможностью выборочного удаления) и их сохранение в виде файла, поиск независимых множеств вершин сгенерированных графов и сохранение результатов поиска. Необходимо предусмотреть различные исходы поиска, чтобы программа не выдавала ошибок и работала правильно.

Устройство ввода – клавиатура и мышь.

#### 4. Теоретическая часть задания

Дан граф  $G$  (см. рисунок 1), который задается множеством вершин  $X_1, X_2, X_3, \dots, X_7$  и множеством ребер, соединяющих между собой определенные вершины.

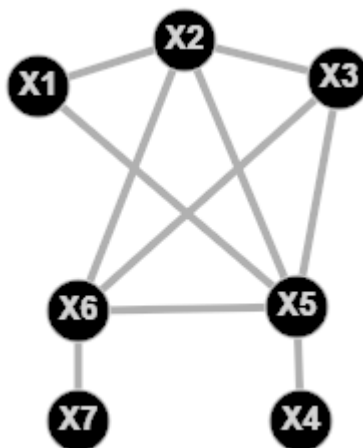


Рисунок 1.

При представлении графа в матрицу смежности он примет следующий вид:

	$X_1$	$X_2$	$X_3$	$X_4$	$X_5$	$X_6$	$X_7$
$X_1$	0	1	0	0	1	0	0
$X_2$	1	0	1	0	1	1	0
$X_3$	0	1	0	0	1	1	0
$X_4$	0	0	0	0	1	0	0
$X_5$	1	1	1	1	0	1	0
$X_6$	0	1	1	0	1	0	1
$X_7$	0	0	0	0	0	1	0

Рисунок 2.

Задача состоит в нахождении всех независимых множеств вершин графа через матрицу смежности. Для наглядности алгоритма разберем поиск относительно вершины  $X_1$ .



Выделим из матрицы смежности вектор(одномерный массив), соответствующий вершине  $X_1$  (см. рисунок 3). Обратим внимание, что  $\{X_1\}$  – независимое множество.

	$X_1$	$X_2$	$X_3$	$X_4$	$X_5$	$X_6$	$X_7$
$X_1$	0	1	0	0	1	0	0

Рисунок 3.

Из определения матрицы смежности следует, что каждый элемент выделенного массива соответствует вершине графа. Необходимо найти такие вершины, которые не смежны с вершиной  $X_1$ . Для этого проверим на не смежность вершины, начиная со следующей( $X_2, X_3, \dots, X_7$ ). Получаем:

	$X_1$	$X_2$	$X_3$	$X_4$	$X_5$	$X_6$	$X_7$
$X_1$	0	1	0	0	1	0	0

Рисунок 4. – Вершины  $X_3, X_4, X_6, X_7$  не смежны с вершиной  $X_1$

Из определения независимого множества вершин графа следует, что множества  $\{X_1, X_3\}$ ,  $\{X_1, X_4\}$ ,  $\{X_1, X_6\}$ ,  $\{X_1, X_7\}$  – независимые.

Полученные несмежные вершины рассмотрим отдельно для поиска множеств большей меры(размера). Начнем с вершины  $X_3$ . При этом необходимо выполнить отождествление вершин  $X_1$  и  $X_3$ . Получим:

	$X_1$	$X_2$	$X_3$	$X_4$	$X_5$	$X_6$	$X_7$
$X_1$	0	1	0	0	1	0	0
$X_3$	0	1	0	0	1	1	0
$X_1 \vee X_3$	0	1	0	0	1	1	0

Рисунок 5. – Вершина  $X_1 \vee X_3$  – результат отождествления.

Получена вершина, для которой необходимо найти несмежные вершины, начиная со следующей после  $X_3$  вершины( $X_4, X_5, \dots, X_7$ ). Имеем:

	$x_1$	$x_2$	$x_3$	$x_4$	$x_5$	$x_6$	$x_7$
$x_1 \vee x_3$	0	1	0	0	1	1	0

Рисунок 6. – Вершины  $x_4$  и  $x_7$  не смежны с вершиной  $x_1 \vee x_3$ .

Снова, из определения независимого множества вершин графа следует, что множества  $\{x_1, x_3, x_4\}$ ,  $\{x_1, x_3, x_7\}$  – независимые. Как мы видим, поиск относительно одной вершины сводится к рекурсии, состоящей из следующих этапов:

- 1) Получение вершины  $x_i$ .
- 2) Поиск ей несмежных вершин и запись независимых множеств вершин, включающих в себя те, которые были задействованы.
- 3) Если найдены несмежные вершины  $x_j, \dots, x_k$ , то они по отдельности отождествляются с  $x_i$  и полученная вершина передается в шаг 1).

При отсутствии таковых или достижения последней вершины мы возвращаемся к предыдущим несмежным вершинам (При достижении первоначальной вершины  $x_i$  поиск считается завершенным).

Дальнейший поиск независимых множеств вершин графа будет рассмотрен более коротко в связи с тем, что многие шаги повторяются.

	$x_1$	$x_2$	$x_3$	$x_4$	$x_5$	$x_6$	$x_7$
$x_1 \vee x_3$	0	1	0	0	1	1	0
$x_4$	0	0	0	0	1	0	0
$x_1 \vee x_3 \vee x_4$	0	1	0	0	1	1	0

Рисунок 7. – Получили вершину  $x_4$ . Вершина  $x_1 \vee x_3 \vee x_4$  – результат отождествления. Вершина  $x_7$  не смежна с вершиной  $x_1 \vee x_3 \vee x_4$ . Получили множество  $\{x_1, x_3, x_4, x_7\}$ . Достигнута последняя вершина  $x_7$ , возвращаемся к вершине  $x_1 \vee x_3$  (На этой вершине также найдены все независимые множества, поэтому возвращаемся к вершине  $x_1$ ).

	$x_1$	$x_2$	$x_3$	$x_4$	$x_5$	$x_6$	$x_7$
$x_1$	0	1	0	0	1	0	0
$x_4$	0	0	0	0	1	0	0
$x_1 \vee x_4$	0	1	0	0	1	0	0

Рисунок 8. – Получили вершину  $x_4$ . Вершина  $x_1 \vee x_4$  – результат отождествления. Вершины  $x_6$  и  $x_7$  не смежны с вершиной  $x_1 \vee x_4$ . Получили множества  $\{x_1, x_4, x_6\}$ ,  $\{x_1, x_4, x_7\}$ .

	$x_1$	$x_2$	$x_3$	$x_4$	$x_5$	$x_6$	$x_7$
$x_1 \vee x_4$	0	1	0	0	1	0	0
$x_5$	0	1	1	0	1	0	1
$x_1 \vee x_4 \vee x_5$	0	1	1	0	1	0	1

Рисунок 9. – Получили вершину  $x_6$ . Вершина  $x_1 \vee x_4 \vee x_6$  – результат отождествления. Не найдено несмежных вершин, возвращаемся к вершине  $x_1 \vee x_4$  (На этой вершине также найдены все независимые множества, поэтому возвращаемся к вершине  $x_1$ ).

	$x_1$	$x_2$	$x_3$	$x_4$	$x_5$	$x_6$	$x_7$
$x_1$	0	1	0	0	1	0	0
$x_5$	0	1	1	0	1	0	1
$x_1 \vee x_5$	0	1	1	0	1	0	1

Рисунок 10. Получили вершину  $x_6$ . Вершина  $x_1 \vee x_6$  – результат отождествления. Не найдено несмежных вершин, возвращаемся к вершине  $x_1$  (На этой вершине также найдены все независимые множества, поэтому поиск относительно  $x_1$  окончен).

Дальнейший поиск выполняется относительно  $x_2$  и т.д. аналогично шагам выше.

## 5. Описание алгоритма программы

Для программной реализации вышеописанного алгоритма понадобится два массива: `queue (int*)` – одномерный массив размером кол-ва вершин, в котором будут храниться номера пройденных вершин, `nSmejSet (int*)` – одномерный массив размером кол-ва вершин, в котором будут храниться все несмежные вершины относительно рассматриваемой вершины, `line (int*)` – одномерный массив размером кол-ва вершин, в котором будет храниться результат отождествления вершин из массива `queue`. Для сохранения найденных независимых множеств вершин графа будет использоваться структура `struct setNI`, включающая следующие элементы: `set (int*)` – одномерный массив размером кол-ва вершин, в котором будет храниться независимое множество вершин графа, `next (struct setNI*)` – указатель на структуру, в которой будет храниться следующее множество. Итак, имеется граф  $G$  с вершинами  $X_1, X_2, \dots, X_n$ . Поиск выполняется относительно каждой вершины, сам поиск разделен на два этапа: поиск всех несмежных вершин и поиск независимых множеств вершин графа.

Пусть поиск выполняется относительно вершины  $X_i$ . Вместе с номером вершины передаются такие параметры, как кол-во вершин графа `size (int)`, указатель на матрицу смежности графа `matrix (int*)`, структуру хранения независимого множества вершин графа `nodeCurrent (struct setNI*)` и указатель на счетчик кол-ва множеств `setsNum (int*)`. Выделив место в памяти для одномерных массивов `line (int*)` и `nSmejSet (int*)`, из матрицы смежности в матрицу `line (int*)` заносится вектор(одномерный массив), соответствующий вершине  $X_i$ . После в матрицу `nSmejSet (int*)` заносятся все несмежные относительно  $X_i$  вершины. Передавав матрицу `nSmejSet (int*)`, начинается второй этап алгоритма – поиск независимых множеств.

Выделив место в памяти для одномерных массивов `line (int*)` и `queue (int*)`, заносим в `queue[0]` номер вершины  $X_i$  и по отдельности будем заносить в `queue[1]` каждую вершину из `nSmejSet (int*)`. Пусть  $X_j$  – не смежна вершине

$X_i$ . Множество  $\{X_i, X_j\}$  будет являться независимым, следовательно программа занесет множество в структуру. Занесем  $X_j$  в `queue[1]`, после, используя массив `line (int*)`, произведем отождествление вершин  $X_i$  и  $X_j$ . Рекурсией вернемся к началу второго этапа, передав `line (int*)` и смещение `shift(int)` для `queue (int*)`. При достижении последней вершины или отсутствии несмежных вершин, мы возвращаемся к предыдущим несмежным вершинам. В конце концов, мы вернемся к вершине  $X_i$  и, достигнув относительно её последней вершины, завершим поиск независимых множеств вершин. графа.

Ниже представлен псевдокод функций `FindNotSmej()` – функция поиска несмежных вершин и `FindSet()` – поиск независимых множеств вершин графа.

### **FindNotSmej()**

Алгоритм `FindNotSmej(matrix, size, vertex, nodeCurrent, setsNum)`

1. НАЧ;
2. Выделить  $4 * \text{size}$  байт памяти для `line` и для `nSmejSet`;
3. `temp = -1`;
4. ДЛЯ  $i = 0$  ДО `size` ВЫПОЛНЯТЬ
5.     `line[i] = matrix[vertex * size + i]`;
6. ДЛЯ  $i = \text{vertex}$  ДО `size` ВЫПОЛНЯТЬ
7.     ЕСЛИ `matrix[vertex * size + i] == 0` ТО
8.         `nSmejSet[++temp] = i + 1`;
9. `FindSet(matrix, nSmejSet, size, 0, NULL, nodeCurrent, setsNum)`;
10. Освободить блок памяти `line` и `nSmejSet`;
11. КОН алгоритма `FindNotSmej()`;

### **FindSet()**

Алгоритм `FindSet(matrix, set, size, shift, queueCurrent, nodeCurrent, setsNum)`

1. НАЧ;
2. Выделить  $4 * \text{size}$  байт памяти для `line` и для `queue`;
3. `node = nodeCurrent`;

```

4. ЕСЛИ queueCurrent != 0 ТО
5.     ДЛЯ i = 0 ДО size ВЫПОЛНЯТЬ
6.         queue[i] = queueCurrent[i];
7. currentShift = shift + 1; cond = 0;
8. ДЛЯ i = shift ДО ((ЕСЛИ shift == 0 ТО 1 ИНАЧЕ size) И set[i] != 0)
    ВЫПОЛНЯТЬ
9. {
10.     ЕСЛИ shift >= size ТО
11.         КОН алгоритма FindNotSmej()
12.     queue[shift] = set[i];
13.     ДЛЯ j = 0 ДО (queue[j] != 0 И j < size) ВЫПОЛНЯТЬ
14.         ДЛЯ k = 0 ДО size ВЫПОЛНЯТЬ
15.             line[k] = line[k] И matrix[(queue[j] - 1) * size + k];
16.     ДЛЯ j = 0 ДО (queue[j] != 0 И j < size) ВЫПОЛНЯТЬ
17.         ЕСЛИ line[queue[j] - 1] == 1 И queue[j] <= queue[j - 1] ТО
18.             {
19.                 cond = 1;
20.                 Прервать цикл;
21.             }
22.     ЕСЛИ cond == 1
23.     {
24.         cond = 0;
25.         Пропустить шаг цикла;
26.     }
27.     ЕСЛИ shift >= 0
28.     {
29.         Выделить 4*size байт памяти для node->set;
30.         ДЛЯ j = 0 ДО (queue[j] != 0 И j < size) ВЫПОЛНЯТЬ
31.             node->set[j] = queue[j];
32.         Выделить 16 байт памяти для node->next;

```

```

33.         setsNum = setsNum + 1;
34.         FindSet(matrix, set, size, currentShift, queue, node->next,
           setsNum);
35.     }
36.     ИНАЧЕ
37.     {
38.         FindSet(matrix, set, size, currentShift, queue, node, setsNum);
39.     }
40.     ПОКА node->next != 0
41.         node = node->next;
42. }
43. Освободить блок памяти line и queue;
44. КОН алгоритма FindSet();

```

## 6. Описание программы

Для написания данной программы использован язык программирования Си. Этот ЯП является простым для понимания, а также универсальным для различных системных архитектур.

Проект был создан в виде консольного приложения Win32.

Данная программа является многомодульной, поскольку включает несколько файлов, содержащих код(menu.h, source.h, source.c).

Работа программы начинается с заставки (см. рисунок 11), показывающей курс и тему работы, группу и инициалы студента.

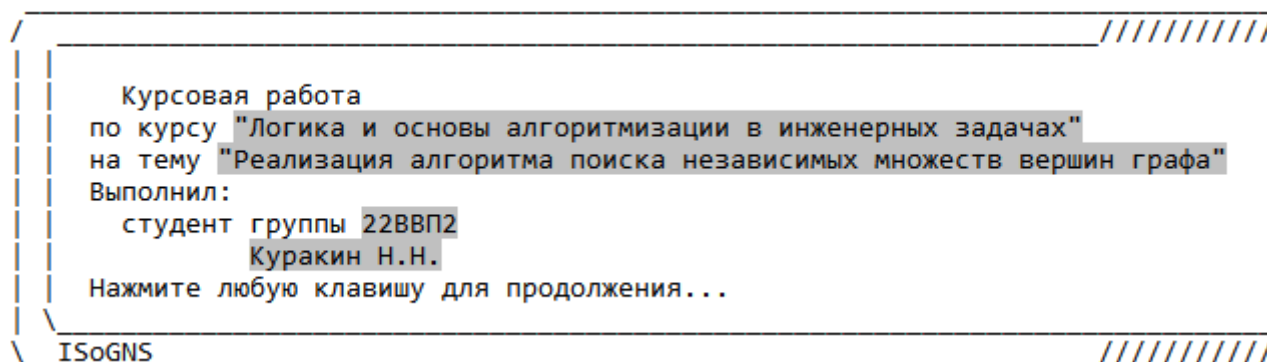


Рисунок 11. Заставка.

При нажатии любой клавиши на клавиатуре программа переходит в раздел меню (см. рисунок 12). Перед пользователем предстает выбор из 8 пунктов: Генерация – генерация графа произвольного порядка, задаваемого пользователем; Просмотр – просмотр хранящихся в программе графов с возможностью их выборочного удаления; Сохранение/загрузка – сохранение и загрузка графов с использованием файлов формата .gr; Поиск – поиск независимых множеств вершин во всех хранящихся в программе графах; Просмотр – просмотр хранящихся в программе независимых множеств вершин; Сохранение – сохранение независимых множеств вершин в файл формата .st; Настройки – изменяет название файла для быстрого



сохранения/загрузки графов; Выход – выход из программы. Рассмотрим каждый из пунктов.

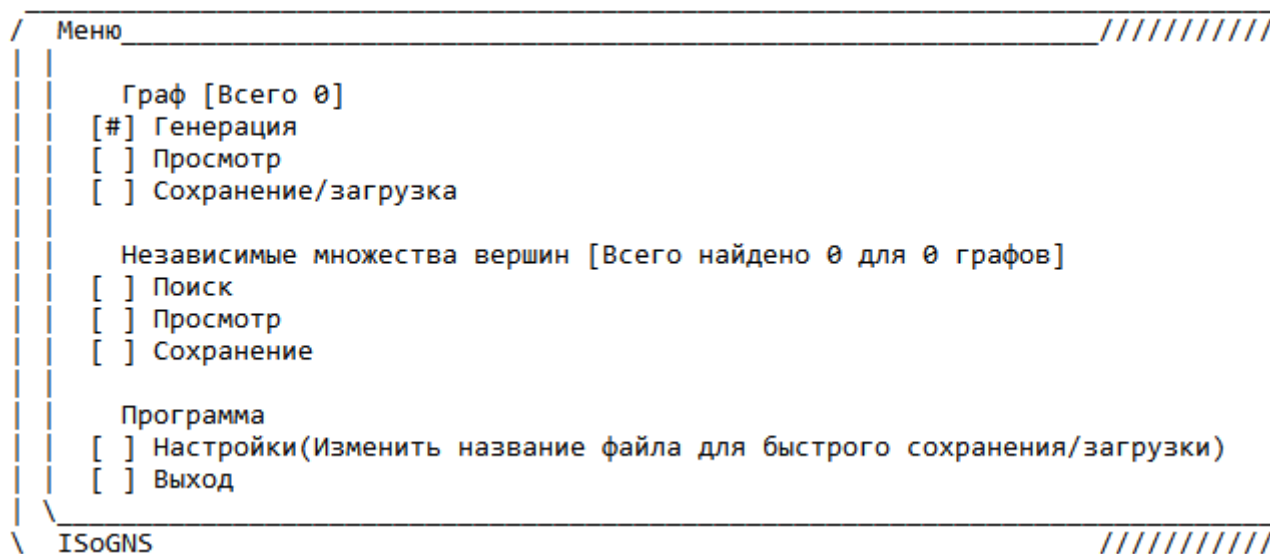


Рисунок 12. Меню

- 1) Генерация. При выборе пункта меню пользователю предлагается ввести количество вершин генерируемого графа(см. рисунок 13). После ввода произвольного числа программа выведет сообщение, соответствующее правильности введенного числа(см. рисунки 14, 15).

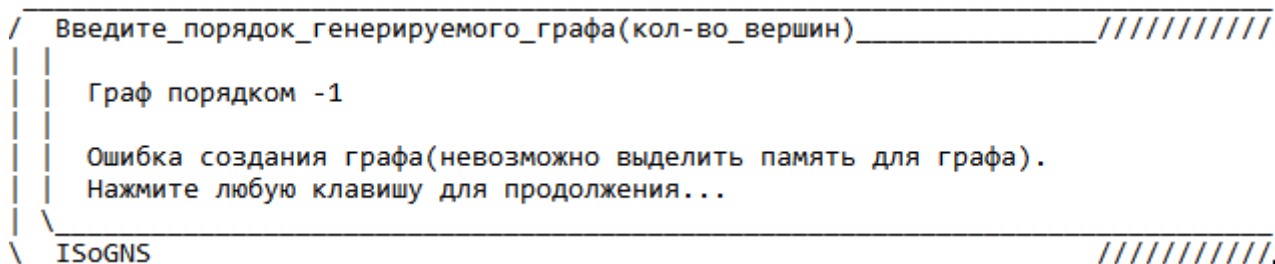


Рисунок 14. Введено неверное число вершин графа. При нажатии любой клавиши программа вернется в меню.

```

/ Введите_порядок_генерируемого_графа(кол-во_вершин)_____//
|
| Граф порядком 5
|
| Граф создан успешно.
| Хотите перейти к просмотру графов? Y/N
\
\ ISoGNS_____//

```

Рисунок 15. Введено верное число вершин графа. При нажатии Y программа перейдет к пункту 2), иначе вернется в меню.

- 2) Просмотр. Дальнейшее после выбора пункта поведение программы зависит от того, имеется ли в памяти программы хотя бы один хранящийся граф или нет. Программа выведет соответствующее сообщение, если инициализированных графов в программе не обнаружено (см. рисунок 16). Иначе пользователю выведется интерактивный интерфейс просмотра матриц смежности графов (см. рисунок 17).

```

/ Просмотр_графов_____//
|
| Нет инициализированных графов для просмотра. Хотите создать граф? Y/N
\
\ ISoGNS_____//

```

Рисунок 16. В программе нет инициализированных графов. При нажатии Y программа перейдет к пункту 1), иначе вернется в меню.

```

/ Просмотр_графов_____//
|
| Граф 1
| 0 1 1 0 1 |1
| 1 0 0 0 0 |2
| 1 0 0 0 1 |3
| 0 0 0 0 0 |4
| 1 0 1 0 0 |5
|
\
\ ISoGNS_/[-]Пред. [R]Удалить [Q] Меню [+]След.\_____//

```

Рисунок 17. В программе есть инициализированные графы. Навигация: “-” – показать предыдущий граф, “+” – показать следующий граф, “R” – удалить выбранный граф, “Q” – возврат в меню(в пункт 1)).

3) Сохранение/загрузка. При выборе пункта меню пользователю выводится подменю с пунктами Быстрое сохранение/Быстрая загрузка, Сохранение/Загрузка, Меню(см. рисунок 18). Смысл быстрого сохранения или загрузки заключается в том, что для этих операций используется название файла по умолчанию, следовательно, нет необходимости каждый раз вводить название файла. При выборе пункта сохранение/Быстрое сохранение дальнейшее поведение программы зависит от того, имеется ли в памяти программы хотя бы один хранящийся граф или нет. Программа выведет соответствующее сообщение, если инициализированных графов в программе не обнаружено (см. рисунок 19). Иначе программа сохранит файл и будет ждать дальнейшей команды (см. рисунок 20).

```

/ Сохранение/загрузка_графов_/////////////////
|
|   Сохранение графов в файл
|   [#] Быстрое сохранение [default.gr]
|   [ ] Сохранение
|
|   Загрузка графов из файла
|   [ ] Быстрая загрузка [default.gr]
|   [ ] Загрузка
|
|   [ ] Меню
|
\_____I
\_____I SoGNS_/////////////////

```

Рисунок 18. Подменю сохранения/загрузки графов.

```

/ Сохранение/загрузка_графов_/////////////////
|
|   Нет инициализированных графов для сохранения. Хотите создать граф? Y/N
|
\_____I
\_____I SoGNS_/////////////////

```

Рисунок 19. В программе нет инициализированных графов. При нажатии Y программа перейдет к пункту 1), иначе вернется в меню.

```

/ Сохранение/загрузка_графов_____//
|
| Успешно сохранено. Хотите открыть файл? Y/N
|
\
\ ISoGNS_____//

```

Рисунок 20. Графы были успешно сохранены. При нажатии Y программа откроет файл, иначе вернется в меню.

При выборе пункта загрузка/Быстрая загрузка дальнейшее поведение программы зависит от того, имеется ли в памяти программы хотя бы один хранящийся граф или нет. Программа выведет соответствующее сообщение, если обнаружены инициализированные графы в программе (см. рисунок 21). Иначе программа загрузит графы из файла и вернется к пункту 1).

```

/ Сохранение/загрузка_графов_____//
|
| В программе найдены инициализированные графы. Хотите продолжить? Y/N
|
\
\ ISoGNS_____//

```

Рисунок 21. В программе обнаружены инициализированные графы. При нажатии Y происходит загрузка файла, иначе программа вернется в меню.

4) Поиск. Дальнейшее после выбора пункта поведение программы зависит от того, имеется ли в памяти программы хотя бы один хранящийся граф или нет. Программа выведет соответствующее сообщение, если инициализированных графов в программе не обнаружено (см. рисунок 22). Иначе пользователю выведется сообщение со статистикой по поиску независимых множеств графа (см. рисунок 23).

```

/ Поиск_независимых_множеств_вершин_____//
|
| Нет инициализированных графов для поиска множеств. Хотите создать граф? Y/N
|
\
\ ISoGNS_____//

```

Рисунок 22. В программе нет инициализированных графов. При нажатии Y программа перейдет к пункту 1), иначе вернется в меню.

```

/ Поиск_независимых_множеств_вершин_/////////////////
|
| Поиск завершен успешно. Статистика:
|
| Граф 1, кол-во независимых множеств = 5
| Граф 2, кол-во независимых множеств = 3
|
| Хотите перейти к просмотру множеств? Y / N
|
\ ISoGNS_/////////////////////////////////

```

Рисунок 23. Программа выполнила поиск независимых множеств вершин и вывела статистику по поиску(кол-во независимых вершин каждого графа).

При нажатии Y программа перейдет к пункту 5), иначе вернется в меню.

5) Просмотр. Дальнейшее после выбора пункта поведение программы зависит от того, имеется ли в памяти программы хотя бы один хранящийся граф или нет, и имеются ли в памяти программы независимые множества вершин графов. Программа выведет соответствующее сообщение, если инициализированных графов в программе не обнаружено (см. рисунок 24) или независимых множеств вершин графов не обнаружено (см. рисунок 25). Иначе пользователю выведется интерактивный интерфейс просмотра независимых множеств вершин графов (см. рисунок 26).

```

/ Просмотр_независимых_множеств_вершин_/////////////////
|
| Нет инициализированных графов для поиска множеств. Хотите создать граф? Y/N
|
\ ISoGNS_/////////////////////////////////

```

Рисунок 24. В программе нет инициализированных графов. При нажатии Y программа перейдет к пункту 1), иначе вернется в меню.

```

/ Просмотр_независимых_множеств_вершин_/////////////////
|
| Нет множеств, которые можно просмотреть. Хотите произвести поиск множеств? Y/N
|
\ ISoGNS_/////////////////////////////////

```

Рисунок 25. В программе нет независимых множеств вершин графа. При нажатии Y программа перейдет к пункту 4), иначе вернется в меню.

б) Сохранение. Дальнейшее после выбора пункта поведение программы зависит от того, имеется ли в памяти программы хотя бы один хранящийся граф или нет, и имеются ли в памяти программы независимые множества вершин графов. Программа выведет соответствующее сообщение, если инициализированных графов в программе не обнаружено (см. рисунок 26) или независимых множеств вершин графов не обнаружено (см. рисунок 27). Иначе программа предложит ввести название файла для сохранения и выполнит сохранение в указанный файл. (см. рисунок 28).

```

/ Сохранение_множеств_____//
|
| Нет инициализированных графов для сохранения. Хотите создать граф? Y/N
|
\_____//
ISOGNS_____//

```

Рисунок 26. В программе нет инициализированных графов. При нажатии Y программа перейдет к пункту 1), иначе вернется в меню.

```

/ Сохранение_множеств_____//
|
| Нет множеств, которые можно сохранить. Хотите произвести поиск множеств? Y/N
|
\_____//
ISOGNS_____//

```

Рисунок 27. В программе нет независимых множеств вершин графа. При нажатии Y программа перейдет к пункту 4), иначе вернется в меню.

```

/ Сохранение_множеств_____//
|
| Успешно сохранено. Хотите открыть файл? Y/N
|
\_____//
ISOGNS_____//

```

Рисунок 28. Множества были успешно сохранены. При нажатии Y программа откроет файл, иначе вернется в меню.

7) Настройки. При выборе пункта программа предложит ввести название файла, используемого для быстрого сохранения/загрузки графов (см. рисунок 29).

```
/  Сохранение/загрузка_графов_////////////////////  
|  |  
|  | Введите название файла(без формата): ■
```

Рисунок 29. Ввод названия файла.

8) Выход. При выборе пункта происходит выход из программы.

## 7. Тестирование

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

Тестирование проводилось в процессе разработки и после завершения написания программы. В ходе тестирования было выявлено и исправлено множество проблем, связанных с выделением и освобождением блоков памяти, хранением графов, вводом/выводом данных, включая загрузку и сохранение, взаимодействием функций.

Ниже продемонстрированы поведение программы при различных сценариях тестирования и соответствующая таблица:

```
/ Введите_порядок_генерируемого_графа(кол-во_вершин)_____//  
|  
| Граф порядком 999  
| Генерировать граф с порядком больше 50 не рекомендуется: возможен риск  
| переполнения памяти при поиске независимых множеств вершин.  
| Вы хотите продолжить? Y/N
```

Рисунок 30. Ввод большого порядка в поле генерации графа. Пользователь может продолжить генерацию, не смотря на предупреждение.

```
/ Сохранение/загрузка_графов_____//  
|  
| Ошибка, не удалось открыть файл для загрузки.  
| Нажмите любую клавишу для продолжения...  
|  
| ISoGNS_____//
```

Рисунок 31. Файл, указанный пользователем, не существует/доступен только для чтения. При нажатии любой клавиши происходит возврат в меню.

```
/ Сохранение/загрузка_графов_____//  
|  
| Ошибка, некорректный файл. Нажмите любую клавишу для продолжения...  
|  
| ISoGNS_____//
```

Рисунок 32. Файл, указанный пользователем, поврежден. При нажатии любой клавиши происходит возврат в меню.



```

/ Сохранение/загрузка_графов_/////////////////
|
| Ошибка, не удалось создать/открыть файл для сохранения.
| Нажмите любую клавишу для продолжения...
\
\ ISoGNS_/////////////////////////////////

```

Рисунок 33. Файл, указанный пользователем, не удается создать(недостаточно места или файл существует, но доступен только для чтения). При нажатии любой клавиши происходит возврат в меню.

```

/ Сохранение_множеств_/////////////////
|
| Введите название файла(без формата): 22
| Ошибка, не удалось создать/открыть файл для сохранения.
| Нажмите любую клавишу для продолжения...
\
\ ISoGNS_/////////////////////////////////

```

Рисунок 34. Файл, указанный пользователем, не удается создать(недостаточно места или файл существует, но доступен только для чтения). При нажатии любой клавиши происходит возврат в меню.

```

/ Поиск_независимых_множеств_вершин_/////////////////
|
| Поиск завершен успешно. Статистика:
|
| Граф 1, кол-во независимых множеств = 3
|
| Хотите перейти к просмотру множеств? Y / N
\
\ ISoGNS_/////////////////////////////////

```

Рисунок 35. Поиск независимых множеств вершин в полном графе. Из определений полного графа(граф, в котором все вершины связаны между собой) и независимого множества вершин можно сделать вывод, что количество независимых множеств вершин в полном графе будет равно порядку этого графа.

```

/  Просмотр_независимых_множеств_вершин_////////////////
|
|  Граф 1
|  {1} {1 2} {1 2 3} {1 3}
|  {2} {2 3}
|  {3}
|
\  ISoGNS_/[-]Пред. [Q] Меню [+]След.\////////////////

```

Рисунок 36. Поиск независимых множеств вершин в пустом графе(в данном случае, порядок графа соответствует 4-ём). Из определения пустого графа(граф, в котором все вершины изолированы) и независимого множества вершин можно сделать вывод, что в пустом графе будет максимальное количество независимых множеств вершин в отличие от других графов тех же порядков.

Таблица 1 – Описание поведения программы при тестировании

Описание теста	Ожидаемый результат	Полученный результат
Запуск программы	Вывод заставки, после заставки вывод меню.	Верно
Выбор генерации графа	Вывод поля ввода порядка генерируемого графа, вывод сообщения при вводе неверного или слишком большого порядка.	Верно
Выбор просмотра графов	Вывод интерактивного меню с возможностью просмотра графов, перемещением между графами и выборочным удалением.	Верно
Выбор сохранение/загрузка графов	Вывод подменю с функциями быстрого сохранения/загрузки и обычного сохранения/загрузки. Правильность сохранения и загрузки.	Верно

Выбор поиска независимых множеств вершин графов	Вывод статистики поиска при успешном выполнении операции.	Верно
Выбор просмотра независимых множеств вершин графов	Вывод интерактивного меню с возможностью просмотра независимых множеств вершин графов, перемещением между графами.	Верно
Выбор сохранения/загрузки графов	Вывод поля ввода названия файла для сохранения независимых множеств вершин графа.	Верно
Выбор настроек	Вывод поля ввода названия файла для быстрого сохранения независимых множеств вершин графа.	Верно
Выход из программы	Выход из программы	Верно

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

## 8. Ручной расчет задачи

Проведем проверку программы посредством ручного поиска независимых множеств вершин на конкретном графе.

Дан граф  $G$  (см. рисунок 37), который задается множеством вершин  $X_1, X_2, X_3, X_4$  и множеством ребер, соединяющих между собой определенные вершины.

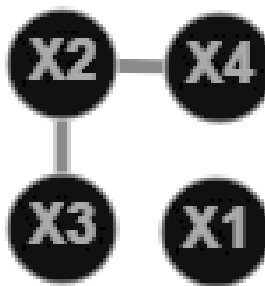


Рисунок 37.

При представлении графа в матрицу смежности он примет следующий вид:

	$X_1$	$X_2$	$X_3$	$X_4$
$X_1$	0	0	0	0
$X_2$	0	0	1	1
$X_3$	0	1	0	0
$X_4$	0	1	0	0

Рисунок 38.

	$X_1$	$X_2$	$X_3$	$X_4$
$X_1$	0	0	0	0

Рисунок 39. – Вершины  $X_2, X_3, X_4$  несмежны с вершиной  $X_1$ . Получены независимые множества вершин  $\{X_1, X_2\}, \{X_1, X_3\}, \{X_1, X_4\}$ .

	$X_1$	$X_2$	$X_3$	$X_4$
$X_1$	0	0	0	0
$X_3$	0	1	0	0
$X_1 \vee X_3$	0	1	0	0

Рисунок 40. – Вершины  $X_4$  несмежны с вершиной  $X_1 \vee X_3$ . Получены независимые множества вершин  $\{X_1, X_3, X_4\}$ .

	$X_1$	$X_2$	$X_3$	$X_4$
$X_3$	0	1	0	0

Рисунок 41. – Вершина  $X_4$  несмежна с вершиной  $X_3$ . Получена независимые множества вершин  $\{X_3, X_4\}$ .

В ходе ручного поиска независимых множеств вершин мы получили следующие множества:  $\{X_1\}, \{X_2\}, \{X_3\}, \{X_4\}, \{X_1, X_2\}, \{X_1, X_3\}, \{X_1, X_4\}, \{X_1, X_3, X_4\}, \{X_3, X_4\}$ .

Загрузим в программу граф и произведем поиск независимых множеств вершин графа.

```

/  Просмотр_графов_____//
|
|  Граф 1
|  0 0 0 0 |1
|  0 0 1 1 |2
|  0 1 0 0 |3
|  0 1 0 0 |4
|
\
\__ISoGNS_/[-]Пред. [R]Удалить [Q] Меню [+]След.\_____//

```

Рисунок 42. Граф в программе.

```

/  Поиск_независимых_множеств_вершин_____//
|
|  Поиск завершен успешно. Статистика:
|
|  Граф 1, кол-во независимых множеств = 9
|
|  Хотите перейти к просмотру множеств? Y / N
|
\
\__ISoGNS_____//

```

Рисунок 43. Поиск завершен успешно.

```

/  Просмотр_независимых_множеств_вершин_____//
|
|  Граф 1
|  {1} {1 2} {1 3} {1 3 4} {1 4}
|  {2}
|  {3} {3 4}
|  {4}
|
\
\__ISoGNS_/[-]Пред. [Q] Меню [+]След.\_____//

```

Рисунок 44. Найденные независимые множества вершин графа.

Результат поиска независимых множества вершин графа программой схож с ручным поиском. Из этого делаем вывод, что программа работает верно.

## **9. Заключение**

В ходе выполнения курсовой работы были получены навыки работы с графами, представленными в виде матриц смежности, и реализации алгоритмов, связанных с графами.

Результатом курсовой работы стала программа, реализующая алгоритм поиска независимых множеств вершин графа в Microsoft Visual Studio 2022.

Программа имеет достаточный для использования функционал возможностей, однако может быть улучшена в дальнейшем за счет расширения функционала: поиск независимых множеств вершин графа в ориентированных и взвешенных графах, визуализация графов в виде набора вершин и ребер, переход от консольного приложения к оконному приложению.

## **10. Список литературы**

1. Керниган Б. Ритчи Д. Язык программирования С. 1985 г.
2. Плаугер П. Стандартная библиотека языка С. 1992 г.
3. Касьянов В. Н., Евстигнеев В. А. Графы в программировании: обработка, визуализация и применение. 2003 г.
4. Кристофидес Н. Теория графов. Алгоритмический подход. 1978



# 11. Приложение А.

## 12. Листинг программы.

### Файл source.h

```
#define _CRT_SECURE_NO_WARNINGS
#include <stdio.h>
#include <conio.h>
#include <stdlib.h>
#include <locale.h>
#include <windows.h>
#include <time.h>

struct setNI {
    int* set;
    struct setNI* next;
};

struct graph {
    int* matrix, size;
    struct graph* next;
    struct setNI* set;
};

int* CreateGraph(int vertexes) {
    int* Mtrx = (int*)malloc(sizeof(int) * vertexes * vertexes);
    if (!Mtrx) return 0;
    for (int i = 0; i != vertexes; i++) {
        *(Mtrx + i * vertexes + i) = 0;
        for (int j = 0; j < i; j++) {
            *(Mtrx + i * vertexes + j) = rand() % 2;
            *(Mtrx + j * vertexes + i) = *(Mtrx + i * vertexes + j);
        }
    }
    return Mtrx;
}

void ShowGraph(struct graph* graph) {
    if (!graph) return;
    for (int i = 0; i != graph->size; i++) {
        printf(" | | ");
        for (int j = 0; j != graph->size; j++) {
            printf("%i ", *(graph->matrix + i * graph->size + j));
        }
        printf("%i\n", i + 1);
    }
}
```

```

}

void ClearLine(struct setNI* list, int vertexes, int count) {
    if (!list) return;
    int cur = count;
    if (list->next)
        ClearLine(list->next, vertexes, cur + 1);
    if (list->next)
        free(list->set);
    if (cur)
        free(list);
}

void DeleteGraph(struct graph** startGraph, struct graph* currentGraph) {
    if (!startGraph || !currentGraph) return;
    struct graph* temp = *startGraph;
    if (*startGraph == currentGraph) {
        if ((*startGraph)->next)
            *startGraph = (*startGraph)->next;
        else
            *startGraph = NULL;
        free((currentGraph)->matrix);
    }
    if ((currentGraph)->set) {
        for (int i = currentGraph->size - 1; i != 0; i--) {
            ClearLine(currentGraph->set + i, currentGraph->size, 0);
        };
    }

    free(currentGraph);
    return;
}

while (temp->next != currentGraph) {
    temp = temp->next;
}

if (!(currentGraph)->next)
    temp->next = NULL;
else {
    temp->next = (currentGraph)->next;
}

free((currentGraph)->matrix);
if ((currentGraph)->set) {
    for (int i = currentGraph->size - 1; i != 0; i--) {
        ClearLine(currentGraph->set + i, currentGraph->size, 0);
    };
}
}

```

```

        free(currentGraph);
        return;
    }
void DeleteGraphs(struct graph* ma, struct graph* current) {
    if (!current)
        return;
    if (current->next)
        DeleteGraphs(ma, current->next);
    DeleteGraph(&ma, current);
}
void FindSet(int* matrix, int* set, int size, int shift, int* queueCurrent, struct setNI* nodeCurrent, int*
setsNum) {
    int* queue = (int*)malloc(sizeof(int) * size), * line = (int*)malloc(sizeof(int) * size);
    struct setNI* node = nodeCurrent;
    if (!queueCurrent) {
        for (int i = 0; i < size; i++) {
            queue[i] = 0;
        }
    }
    else {
        for (int i = 0; i < size; i++) {
            queue[i] = queueCurrent[i];
        }
    }
    int currentShift = shift + 1, cond = 0;
    for (int i = shift; i < (!shift ? 1 : size) && set[i] != 0; i++) {
        if (shift >= size) break;
        for (int j = 0; j < size; j++) {
            line[j] = 0;
        }
        queue[shift] = set[i];
        for (int j = 0; queue[j] != 0 && j < size; j++) {
            for (int k = 0; k < size; k++) {
                line[k] = line[k] || matrix[(queue[j] - 1) * size + k];
            }
        }
        for (int j = 0; queue[j] != 0 && j < size; j++) {
            if (line[queue[j] - 1] || queue[j] <= queue[j - 1]) {
                cond = 1;
                break;
            }
        }
    }
}

```

```

    if (cond) {
        cond = 0;
        continue;
    }
    if (shift >= 0) {
        node->set = (int*)malloc(sizeof(int) * size);
        for (int j = 0; j < size; j++) {
            node->set[j] = 0;
        }
        for (int j = 0; queue[j] != 0 && j < size; j++) {
            node->set[j] = queue[j];
        }
        node->next = (struct setNI*)malloc(sizeof(struct setNI));
        node->next->next = NULL;
        (*setsNum)++;
        FindSet(matrix, set, size, currentShift, queue, node->next, setsNum);
    }
    else
        FindSet(matrix, set, size, currentShift, queue, node, setsNum);
    while (node->next) {
        node = node->next;
    }
}
free(queue);
free(line);
}

void FindNotSmej(int* matrix, int size, int vertex, struct setNI* nodeCurrent, int* setsNum) {
    int temp = -1, * line = (int*)malloc(sizeof(int) * size);
    for (int i = 0; i < size; i++) {
        line[i] = matrix[vertex * size + i];
    }
    int* nSmejSet = (int*)malloc(sizeof(int) * size);
    for (int i = 0; i < size; i++) {
        nSmejSet[i] = 0;
    }
    for (int i = vertex; i < size; i++) {
        if (!matrix[vertex * size + i])
            nSmejSet[++temp] = i + 1;
    }
    FindSet(matrix, nSmejSet, size, 0, NULL, nodeCurrent, setsNum);
    free(line);
    free(nSmejSet);
}

```

```

};

void ShowList(struct setNI* node, int size) {
    if (!node || !node->set) return;
    struct setNI* nodeCurrent = node;
    printf(" || ");
    do {
        printf("{");
        for (int i = 0; nodeCurrent->set[i] != 0 && i < size; i++) {
            printf("%d%c", nodeCurrent->set[i], (i == size - 1 || !nodeCurrent->set[i + 1]) ? 0 : ' ');
        }
        printf("} ");
        nodeCurrent = nodeCurrent->next;
    } while (nodeCurrent->next);
    printf("\n");
    return;
}

void ShowListAdv(struct setNI* node, int size, FILE* file) {
    if (!node || !node->set) return;
    struct setNI* nodeCurrent = node;
    do {
        fprintf(file, "{");
        for (int i = 0; nodeCurrent->set[i] != 0 && i < size; i++) {
            fprintf(file, "%d%c", nodeCurrent->set[i], (i == size - 1 || !nodeCurrent->set[i + 1]) ? 0 : ' ');
        }
        fprintf(file, " } ");
        nodeCurrent = nodeCurrent->next;
    } while (nodeCurrent->next);
    fprintf(file, "\n");
    return;
}

```

## Файл menu.h

```

#include "source.h"

void Intro() {
    setlocale(LC_ALL, "RU");
    char menu[] =
        "
        _____\n"
        " /
        _____////////////////\n"
        " | \n"
        " | | \tКурсовая работа\n"

```

```

        " | по курсу \033[30;47m\"Логика и основы алгоритмизации в инженерных
задачах\" \033[0m\n"
        " | на тему \033[30;47m\"Реализация алгоритма поиска независимых множеств
вершин графа\" \033[0m\n"
        " | Выполнил:\n"
        " | \tстудент группы \033[30;47m22ВВП2\033[0m\n"
        " | \t\t\033[30;47mКуракин Н.Н.\033[0m\n"
        " | Нажмите любую клавишу для продолжения...\n"
        " |

\\_____ \n"
        "

\\_ISoGNS_____///////////////";

        printf(menu);
        _getch();
    }

int MainMenu(int graphNum, int setsNum, int graphSetsNum, int marker) {
    int markerPosition = marker;
    char menu[] =
        "
_____ \n"
        " /

Меню _____///////////////\n"
        " | \n"
        " | \tГраф [Всего %d]\n"
        " | [%c] Генерация\n"
        " | [%c] Просмотр\n"
        " | [%c] Сохранение/загрузка\n"
        " | \n"
        " | \tНезависимые множества вершин [Всего найдено %d для %d графов]\n"
        " | [%c] Поиск\n"
        " | [%c] Просмотр\n"
        " | [%c] Сохранение\n"
        " | \n"
        " | \tПрограмма\n"
        " | [%c] Настройки(Изменить название файла для быстрого сохранения/загрузки)\n"
        " | [%c] Выход\n"
        " |

\\_____ \n"
        "

\\_ISoGNS_____///////////////";

```

```

        printf(menu, graphNum, markerPosition == 0 ? '#' : '', markerPosition == 1 ? '#' : '', markerPosition
        == 2 ? '#' : '', setsNum, graphSetsNum, markerPosition == 3 ? '#' : '', markerPosition == 4 ? '#' : '', markerPosition ==
        5 ? '#' : '', markerPosition == 6 ? '#' : '', markerPosition == 7 ? '#' : '');
        for (;;) {
            if (_kbhit()) {
                int pressedButton = _getch();
                if (pressedButton != 13 && pressedButton != 72 && pressedButton != 80)
                    continue;
                system("cls");
                switch (pressedButton) {
                    case 13:
                        return markerPosition;
                        break;
                    case 72:
                        markerPosition = markerPosition == 0 ? 7 : markerPosition - 1;
                        break;
                    case 80:
                        markerPosition = markerPosition == 7 ? 0 : markerPosition + 1;
                        break;
                }
                printf(menu, graphNum, markerPosition == 0 ? '#' : '', markerPosition == 1 ? '#' : '
                ', markerPosition == 2 ? '#' : '', setsNum, graphSetsNum, markerPosition == 3 ? '#' : '', markerPosition == 4 ? '#' : '',
                markerPosition == 5 ? '#' : '', markerPosition == 6 ? '#' : '', markerPosition == 7 ? '#' : '');
            }
            Sleep(100);
        }
    }

    struct graph* OptionGenerateMatrix(struct graph* graph, int* graphNum, int* setsNum, int* graphSetsNum,
    struct graph* (*ReturnFunction)(struct graph*, int*, int*, int*)) {
        system("cls");
        int size = 0, * matrix = NULL, cond1 = 0, cond2 = 0, temp = 0;
        struct graph* cGraph = graph;
        char menu1[] =
            "
            _____\n"
            " / Введите_порядок_генерируемого_графа(кол-
            во_вершин)_____//\n"
            " | \n"
            " | | Граф порядком ",
            menu2[] =
            " | \n"
            " | | %s\n"

```

```

        " |
\\_____\\n"
        "
\\_ISoGNS_____//";
do { fseek(stdin, 0, SEEK_END); system("cls"); printf(menu1); } while (!scanf("%d", &size));
if (size > 50) {
    printf(" || Генерировать граф с порядком больше 50 не рекомендуется: возможен
риск\\n || переполнения памяти при поиске независимых множеств вершин.\\n || Вы хотите продолжить?
Y/N\\n");
    do {
        fseek(stdin, 0, SEEK_END);
        temp = _getch();
    } while (temp != 'Y' && temp != 'y' && temp != 'N' && temp != 'n' && temp != 141 &&
temp != 173 && temp != 146 && temp != 226);
    if (temp == 'N' || temp == 'n' || temp == 146 || temp == 226)
        return graph;
};
matrix = size > 0 ? CreateGraph(size) : matrix;
printf(menu2, (matrix && size > 0) ? "Граф создан успешно.\\n || Хотите перейти к просмотру
графов? Y/N" : "Ошибка создания графа(невозможно выделить память для графа).\\n || Нажмите любую
клавишу для продолжения...");
if (!matrix || !size) {
    (void)_getch();
    return graph;
}
*setsNum = 0;
*graphSetsNum = 0;
if (!graph || !*graphNum) {
    graph = (struct graph*)malloc(sizeof(struct graph));
    cGraph = graph;
    cond1 = 1;
}
else {
    while (cGraph->next) {
        cGraph = cGraph->next;
    };
    cGraph->next = (struct graph*)malloc(sizeof(struct graph));
    cGraph = cGraph->next;
}
cGraph->matrix = matrix;
cGraph->size = size;
cGraph->next = NULL;

```



```

cGraph->set = NULL;
(*graphNum)++;
if (!cond1) {
    do {
        fseek(stdin, 0, SEEK_END);
        temp = _getch();
    } while (temp != 'Y' && temp != 'y' && temp != 'N' && temp != 'n' && temp != 141 &&
temp != 173 && temp != 146 && temp != 226);
    return (temp == 'Y' || temp == 'y' || temp == 141 || temp == 173) ? ReturnFunction(graph,
graphNum, setsNum, graphSetsNum) : graph;
}
else {
    do {
        fseek(stdin, 0, SEEK_END);
        temp = _getch();
    } while (temp != 'Y' && temp != 'y' && temp != 'N' && temp != 'n' && temp != 141 &&
temp != 173 && temp != 146 && temp != 226);
    return (temp == 'Y' || temp == 'y' || temp == 141 || temp == 173) ? ReturnFunction(cGraph,
graphNum, setsNum, graphSetsNum) : cGraph;
}
}

struct graph* OptionShowGraph(struct graph* graph, int* graphNum, int* setsNum, int* graphSetsNum) {
    system("cls");
    char menu1[] =
        "
        _____\n"
        " /
Просмотр_графов_////////\n"
        " | \n"
        " | | Нет инициализированных графов для просмотра. Хотите создать граф? Y/N\n"
        " |
\\_____ \n"
        "
\\_ISoGNS_////////";
    if (!graph || !*graphNum) {
        int temp = 0;
        printf(menu1);
        do { fseek(stdin, 0, SEEK_END);
temp = _getch(); }
        while (temp != 'Y' && temp != 'y' && temp != 'N' && temp != 'n' && temp != 141 &&
temp != 173 && temp != 146 && temp != 226);

```

```

        return (temp == 'Y' || temp == 'y' || temp == 141 || temp == 173) ?
OptionGenerateMatrix(graph, graphNum, setsNum, graphSetsNum, OptionShowGraph) : graph;
    }
    int count = 0;
    char menu2[] =
        "
_____\\n"
        " /
Просмотр_графов_____////////\\n"
        " | \\n"
        " | | Граф %d\\n",
        menu3[] =
        " |
\\_____\\n"
        " \\__ISoGNS_/[ -]Пред. [R]Удалить [Q] Меню
[+]След.\\_____////////";
    printf(menu2, count + 1);
    ShowGraph(graph);
    printf(menu3);
    while (1) {
        struct graph* graphCurrent = graph;
        switch (_getch()) {
            case '+':
            case '=':
                count = count == (* graphNum - 1) ? 0 : count + 1;
                for (int i = 0; i < count; i++) {
                    graphCurrent = graphCurrent->next;
                }
                system("cls");
                printf(menu2, count + 1);
                ShowGraph(graphCurrent);
                printf(menu3);
                break;
            case '-':
            case '_':
                count = !count ? (* graphNum - 1) : count - 1;
                for (int i = 0; i < count; i++) {
                    graphCurrent = graphCurrent->next;
                }
                system("cls");
                printf(menu2, count + 1);
                ShowGraph(graphCurrent);

```

```

        printf(menu3);
        break;
    case 'R':
    case 'r':
    case 138:
    case 170:
        for (int i = 0; i < count; i++) {
            graphCurrent = graphCurrent->next;
        }
        DeleteGraph(&graph, graphCurrent);
        (*graphNum)--;
        *setsNum = 0;
        *graphSetsNum = 0;
        if (!graph)
            return OptionShowGraph(graph, graphNum, setsNum,
graphSetsNum);

        count = count ? count - 1 : count;
        graphCurrent = graph;
        for (int i = 0; i < count; i++) {
            graphCurrent = graphCurrent->next;
        }
        system("cls");
        printf(menu2, count + 1);
        ShowGraph(graphCurrent);
        printf(menu3);
        break;
    case 'Q':
    case 'q':
    case 137:
    case 169:
        return graph;
        break;
    }
}
return graph;
}

struct graph* OptionFindSet(struct graph* graph, int* graphNum, int* setsNum, int* graphSetsNum, struct
graph* (*ReturnFunction)(struct graph*, int*, int*, int*)) {
    system("cls");
    char menu1[] =
        "

```

---

```

        \n"

```

```

" /
Поиск_независимых_множеств_вершин_////////\n"
" | \n"
" | | Нет инициализированных графов для поиска множеств. Хотите создать граф?
Y/N\n"
" |
\\_____ \n"
"
\\_ISoGNS_////////";
    if (!graph || !*graphNum) {
        int temp = 0;
        printf(menu1);
        do {
            fseek(stdin, 0, SEEK_END);
            temp = _getch();
        } while (temp != 'Y' && temp != 'y' && temp != 'N' && temp != 'n' && temp != 141 &&
temp != 173 && temp != 146 && temp != 226);
        return (temp == 'Y' || temp == 'y' || temp == 141 || temp == 173) ?
OptionGenerateMatrix(graph, graphNum, setsNum, graphSetsNum, OptionShowGraph) : graph;
    }
    char menu2[] =
        "
_____ \n"
        " /
Поиск_независимых_множеств_вершин_////////\n"
        " | \n"
        " | | Поиск завершен успешно. Статистика:\n"
        " | \n",
        menu3[] =
        " | | Граф %d, кол-во независимых множеств = %d\n",
        menu4[] =
        " | \n"
        " | | Хотите перейти к просмотру множеств? Y / N\n"
        " |
\\_____ \n"
        "
\\_ISoGNS_////////";
    struct graph* currentGraph = graph;
    *setsNum = 0;
    printf(menu2);
    int temp = 0, count = 0;
    do {

```

```

temp = *setsNum;
if (currentGraph->set)
    for (int i = currentGraph->size - 1; i != 0; i--) {
        ClearLine(currentGraph->set + i, currentGraph->size, 0);
    };
currentGraph->set = (struct setNI*)malloc(sizeof(struct setNI) * currentGraph->size);
for (int i = 0; i < currentGraph->size; i++) {
    (currentGraph->set[i]).next = NULL;
    (currentGraph->set[i]).set = NULL;
    FindNotSmej(currentGraph->matrix, currentGraph->size, i, currentGraph->set + i,
setsNum);
}
currentGraph = currentGraph->next;
printf(menu3, ++count, *setsNum - temp);
} while (currentGraph);
printf(menu4);
*graphSetsNum = *graphNum;
do {
    fseek(stdin, 0, SEEK_END);
    temp = _getch();
} while (temp != 'Y' && temp != 'y' && temp != 'N' && temp != 'n' && temp != 141 && temp !=
173 && temp != 146 && temp != 226);
return (temp == 'Y' || temp == 'y' || temp == 141 || temp == 173) ? ReturnFunction(graph, graphNum,
setsNum, graphSetsNum) : graph;
}

struct graph* OptionShowSet(struct graph* graph, int* graphNum, int* setsNum, int* graphSetsNum) {
    system("cls");
    char menu1[] =
        "
        _____\n"
        " /
Просмотр_независимых_множеств_вершин _____//\n"
        " | \n"
        " | | Нет инициализированных графов для поиска множеств. Хотите создать граф?
Y/N\n"
        " |
\\_____ \n"
        "
\\_ISoGNS_____//";
    char menu2[] =
        "
        _____\n"

```

```

" /
Просмотр_независимых_множеств_вершин_////////\n"
" | \n"
" | | Нет множеств, которые можно просмотреть. Хотите произвести поиск множеств?
Y/N\n"
" |
\\_____ \n"
"
\\_ISoGNS_////////";
    if (!graph && !*graphNum) {
        int temp = 0;
        printf(menu1);
        do {
            fseek(stdin, 0, SEEK_END); temp = _getch(); } while (temp != 'Y' && temp != 'y'
&& temp != 'N' && temp != 'n' && temp != 141 && temp != 173 && temp != 146 && temp != 226);
            return (temp == 'Y' || temp == 'y' || temp == 141 || temp == 173) ?
OptionGenerateMatrix(graph, graphNum, setsNum, graphSetsNum, OptionShowGraph) : graph;
        }

        if (!*setsNum || !*graphSetsNum) {
            int temp = 0;
            printf(menu2);
            do {
                fseek(stdin, 0, SEEK_END); temp = _getch(); } while (temp != 'Y' && temp != 'y'
&& temp != 'N' && temp != 'n' && temp != 141 && temp != 173 && temp != 146 && temp != 226);
                return (temp == 'Y' || temp == 'y' || temp == 141 || temp == 173) ? OptionFindSet(graph,
graphNum, setsNum, graphSetsNum, OptionShowSet) : graph;
            }
            int count = 0;
            char menu3[] =
                "
                _____ \n"
                " /
Просмотр_независимых_множеств_вершин_////////\n"
                " | \n"
                " | | Граф %d\n",
                menu4[] =
                " |
                _____ \n"
                " \\_ISoGNS_/[-]Пред. [Q] Меню
[+]След.\\_////////";
            printf(menu3, count + 1);
            for (int i = 0; i < graph->size; i++) {

```

```

        ShowList(graph->set + i, graph->size);
    }
    printf(menu4);
    while (1) {
        struct graph* graphCurrent = graph;
        switch (_getch()) {
            case '+':
            case '=':
                count = count == (*graphNum - 1) ? 0 : count + 1;
                for (int i = 0; i < count; i++) {
                    graphCurrent = graphCurrent->next;
                }
                system("cls");
                printf(menu3, count + 1);
                for (int i = 0; i < graphCurrent->size; i++) {
                    ShowList(graphCurrent->set + i, graphCurrent->size);
                }
                printf(menu4);
                break;
            case '-':
            case '_':
                count = !count ? (*graphNum - 1) : count - 1;
                for (int i = 0; i < count; i++) {
                    graphCurrent = graphCurrent->next;
                }
                system("cls");
                printf(menu3, count + 1);
                for (int i = 0; i < graphCurrent->size; i++) {
                    ShowList(graphCurrent->set + i, graphCurrent->size);
                }
                printf(menu4);
                break;
            case 'Q':
            case 'q':
            case 137:
            case 169:
                return graph;
                break;
        }
    }
    return graph;
}

```

```

void OptionGetFileName(char* fileNameHolder, int type) {
    system("cls");
    char menu1[] =
        "
        _____\n"
        " / Сохранение%s_____//\n"
        " | \n"
        " | | Введите название файла(без формата): ";
    do { fseek(stdin, 0, SEEK_END); system("cls"); printf(menu1, !type ? "/загрузка_графов__" :
        "_множеств_____"); } while (!scanf("%s", fileNameHolder));
    }

    struct graph* OptionSaveGraph(struct graph* graph, int* graphNum, int* setsNum, int* graphSetsNum,
    char* file) {
        char menu1[] =
            "
            _____\n"
            " /
Сохранение/загрузка_графов_____//\n"
            " | \n",
            menu2[] =
            " | | %s\n"
            " |
            _____\n"
            "
            _____//";
        if (!graph || !*graphNum) {
            int temp = 0;
            printf(menu1);
            printf(menu2, "Нет инициализированных графов для сохранения. Хотите создать
граф? Y/N");
            do {
                fseek(stdin, 0, SEEK_END);
                temp = _getch();
            } while (temp != 'Y' && temp != 'y' && temp != 'N' && temp != 'n' && temp != 141 &&
temp != 173 && temp != 146 && temp != 226);
            return (temp == 'Y' || temp == 'y' || temp == 141 || temp == 173) ?
OptionGenerateMatrix(graph, graphNum, setsNum, graphSetsNum, OptionShowGraph) : graph;
        }
        if (!file[0])
            OptionGetFileName(file, 0);
        char fileName[256] = "";
        strcpy(fileName, file);
    }
}

```



```

strcpy(strchr(fileName, '\0'), ".gr");
FILE* dbFile = fopen(fileName, "w");
if (!dbFile) {
    printf(menu1);
    printf(menu2, "Ошибка, не удалось создать/открыть файл для сохранения.\n | |
Нажмите любую клавишу для продолжения...");
    (void)_getch();
    return graph;
}
struct graph* graphCurrent = graph;
char graphInfo[] = "#####\n#%d#####\n#####\n";
do {
    fprintf(dbFile, graphInfo, graphCurrent->size);
    for (int i = 0; i < graphCurrent->size; i++) {
        for (int j = 0; j < graphCurrent->size; j++) {
            fprintf(dbFile, "%d, ", graphCurrent->matrix[i * graphCurrent->size + j]);
        }
        fprintf(dbFile, "\n");
    }
    graphCurrent = graphCurrent->next;
} while (graphCurrent);
fclose(dbFile);
system("cls");
printf(menu1);
printf(menu2, "Успешно сохранено. Хотите открыть файл? Y/N");;
int temp = 0;
do {
    fseek(stdin, 0, SEEK_END);
    temp = _getch();
} while (temp != 'Y' && temp != 'y' && temp != 'N' && temp != 'n' && temp != 141 && temp !=
173 && temp != 146 && temp != 226);
if (temp == 'Y' || temp == 'y' || temp == 141 || temp == 173) {
    char temp[264] = "notepad ";
    strcpy(strchr(temp, '\0'), fileName);
    system(temp);
}
return graph;
}

struct graph* OptionSaveSets(struct graph* graph, int* graphNum, int* setsNum, int* graphSetsNum) {
    //system("cls");
    char filename[256] = { 0 };
    char menu1[] =

```

```

"
_____\n"
" /
Сохранение_множеств_////////\n"
" | \n",
menu2[] =
" | | %s\n"
" |
\\_____\n"
"
\\_ISoGNS_////////";

printf(menu1);
if (!graph || !*graphNum) {
    int temp = 0;
    printf(menu2, "Нет инициализированных графов для сохранения. Хотите создать
граф? Y/N");

    do {
        fseek(stdin, 0, SEEK_END);
        temp = _getch();
    } while (temp != 'Y' && temp != 'y' && temp != 'N' && temp != 'n' && temp != 141 &&
temp != 173 && temp != 146 && temp != 226);
    return (temp == 'Y' || temp == 'y' || temp == 141 || temp == 173) ?
OptionGenerateMatrix(graph, graphNum, setsNum, graphSetsNum, OptionShowGraph) : graph;
}
if (!*setsNum || !*graphSetsNum) {
    int temp = 0;
    printf(menu2, "Нет множеств, которые можно сохранить. Хотите произвести поиск
множеств? Y/N");

    do {
        fseek(stdin, 0, SEEK_END); temp = _getch();
    } while (temp != 'Y' && temp != 'y' && temp != 'N' && temp != 'n' && temp != 141 &&
temp != 173 && temp != 146 && temp != 226);
    return (temp == 'Y' || temp == 'y' || temp == 141 || temp == 173) ? OptionFindSet(graph,
graphNum, setsNum, graphSetsNum, OptionShowSet) : graph;
}
char fileName[256] = "";
OptionGetFileName(fileName, 1);
strcpy(strchr(fileName, '\0'), ".st");
FILE* dbFile = fopen(fileName, "w");
if (!dbFile) {
    printf(menu2, "Ошибка, не удалось создать/открыть файл для сохранения.\n | |
Нажмите любую клавишу для продолжения...");
}

```

```

        (void)_getch();
        return graph;
    }
    int count = 1;
    struct graph* graphCurrent = graph;
    char graphInfo[] = "#####\n#%d#####\n#####\n";
    do {
        fprintf(dbFile, graphInfo, count++);
        for (int i = 0; i < graphCurrent->size; i++) {
            ShowListAdv(graphCurrent->set + i, graphCurrent->size, dbFile);
        }
        graphCurrent = graphCurrent->next;
    } while (graphCurrent);
    fclose(dbFile);
    system("cls");
    printf(menu1);
    printf(menu2, "Успешно сохранено. Хотите открыть файл? Y/N");
    int temp = 0;
    do {
        fseek(stdin, 0, SEEK_END);
        temp = _getch();
    } while (temp != 'Y' && temp != 'y' && temp != 'N' && temp != 'n' && temp != 141 && temp !=
173 && temp != 146 && temp != 226);
    if (temp == 'Y' || temp == 'y' || temp == 141 || temp == 173) {
        char temp[264] = "notepad ";
        strcpy(strchr(temp, '\0'), fileName);
        system(temp);
    }
    return graph;
}

struct graph* OptionLoadGraph(struct graph* graph, int* graphNum, int* setsNum, int* graphSetsNum,
char* file) {
    system("cls");
    char menu1[] =
        "
_____ \n"
        " /
Сохранение/загрузка_графов_____ ////////////// \n"
        " | \n",
    menu2[] =
        " | | %s\n"

```

```

        " |
\\_____\\n"
        "
\\_ISoGNS_____/////////";

    printf(menu1);
    if (graph || *graphNum) {
        int temp = 0;
        printf(menu2, "В программе найдены инициализированные графы. Хотите
продолжить? Y/N");
        do {
            fseek(stdin, 0, SEEK_END);
            temp = _getch();
        } while (temp != 'Y' && temp != 'y' && temp != 'N' && temp != 'n' && temp != 141 &&
temp != 173 && temp != 146 && temp != 226);
        if (temp == 'N' || temp == 'n' || temp == 146 || temp == 226) return graph;
    }
    if (!file[0])
        OptionGetFileName(file, 0);
    char fileName[256] = "";
    strcpy(fileName, file);
    strcpy(strchr(fileName, '\\0'), ".gr");
    FILE* dbFile = fopen(fileName, "r+");
    if (!dbFile) {
        system("cls");
        printf(menu1);
        printf(menu2, "Ошибка, не удалось открыть файл для загрузки.\\n | | Нажмите любую
клавишу для продолжения...");
        (void)_getch();
        return graph;
    }
    struct graph* cGraph = NULL, * temp = NULL;
    int tempCount = 0, tempSize = 0;
    while (1) {
        if (fscanf(dbFile, "#####\\n#%d#####\\n#####\\n", &tempSize) != 1) {
            fclose(dbFile);
            if (!tempCount || !cGraph) {
                system("cls");
                printf(menu1);
                printf(menu2, "Ошибка, некорректный файл. Нажмите любую
клавишу для продолжения...");
            }
            else {

```

```

        system("cls");
        printf(menu1);
        printf(menu2, "Графы были загружены успешно. Нажмите любую
клавишу для продолжения...");

        if (graph) {
            DeleteGraphs(graph, graph);
            graph->matrix = NULL;
            graph->next = NULL;
            graph->set = NULL;
            graph->size = 0;
        }
        *graphNum = tempCount;
        *setsNum = 0;
        *graphSetsNum = 0;
    }
    (void)_getch();
    return !cGraph ? graph : cGraph;
}

if (!cGraph) {
    cGraph = (struct graph*)malloc(sizeof(struct graph));
    temp = cGraph;
}
else {
    temp->next = (struct graph*)malloc(sizeof(struct graph));
    temp = temp->next;
}
tempCount++;
temp->size = tempSize;
temp->matrix = (int*)malloc(sizeof(int) * tempSize * tempSize);
for (int i = 0; i < tempSize * tempSize; i++) {
    temp->matrix[i] = 0;
}
temp->set = NULL;
temp->next = NULL;
for (int i = 0; i < tempSize; i++) {
    for (int j = 0; j < tempSize; j++) {
        fscanf(dbFile, "%d, ", (temp->matrix + i * tempSize + j));
    }
    fscanf(dbFile, "\n");
}
}
}

```

```

    struct graph* OptionManageGraph(struct graph* graph, int* graphNum, int* setsNum, int* graphSetsNum,
char* defaultName) {
        system("cls");
        int markerPosition = 0;
        char menu1[] =
            "
            _____\n"
            " /
Сохранение/загрузка_графов _____//\n"
            " | \n"
            " | \tСохранение графов в файл\n"
            " | | [%c] Быстрое сохранение [%s.gr]\n"
            " | | [%c] Сохранение\n"
            " | \n"
            " | \tЗагрузка графов из файла\n"
            " | | [%c] Быстрая загрузка [%s.gr]\n"
            " | | [%c] Загрузка\n"
            " | \n"
            " | | [%c] Меню\n"
            " |
            _____\n"
            "
            \\\_ISoGNS_____//",
            filename[256] = { 0 };
        printf(menu1, markerPosition == 0 ? '#' : ' ', defaultName, markerPosition == 1 ? '#' : ' ',
markerPosition == 2 ? '#' : ' ', defaultName, markerPosition == 3 ? '#' : ' ', markerPosition == 4 ? '#' : ' ');
        for (;;) {
            if (_kbhit()) {
                int pressedButton = _getch();
                if (pressedButton != 13 && pressedButton != 72 && pressedButton != 80)
                    continue;
                system("cls");
                switch (pressedButton) {
                    case 13:
                        switch (markerPosition) {
                            case 0:
                                return OptionSaveGraph(graph, graphNum, setsNum,
graphSetsNum, defaultName);
                            case 1:
                                return OptionSaveGraph(graph, graphNum, setsNum,
graphSetsNum, filename);
                            case 2:

```

```

return OptionLoadGraph(graph, graphNum, setsNum,
graphSetsNum, defaultName);

case 3:
return OptionLoadGraph(graph, graphNum, setsNum,
graphSetsNum, filename);

case 4:
return graph;
}
break;
case 72:
markerPosition = markerPosition == 0 ? 4 : markerPosition - 1;
break;
case 80:
markerPosition = markerPosition == 4 ? 0 : markerPosition + 1;
break;
}
printf(menu1, markerPosition == 0 ? '#' : ' ', defaultName, markerPosition == 1 ?
'#' : ' ', markerPosition == 2 ? '#' : ' ', defaultName, markerPosition == 3 ? '#' : ' ', markerPosition == 4 ? '#' : ' ');
}
Sleep(100);
}
}

```

## Файл source.c

```

//Поиск независимых множеств вершин графа
//independent sets of graph nodes search
#include "h\menu.h"
int main() {
    srand(time(NULL));
    Intro();
    system("cls");
    struct graph* matrixs = NULL;
    int graphNum = 0, setsNum = 0, graphSetsNum = 0, marker = 0;
    char defaultName[256] = "default";
    while (1) {
        marker = MainMenu(graphNum, setsNum, graphSetsNum, marker);
        switch (marker) {
            case 0:
                matrixs = OptionGenerateMatrix(matrixs, &graphNum, &setsNum,
&graphSetsNum, OptionShowGraph); //генерация
                break;
            case 1:

```

```

matrixs = OptionShowGraph(matrixs, &graphNum, &setsNum,
&graphSetsNum);//Просмотр
break;
case 2:
matrixs = OptionManageGraph(matrixs, &graphNum, &setsNum,
&graphSetsNum, defaultName);//Сохранение/загрузка1
break;
case 3:
matrixs = OptionFindSet(matrixs, &graphNum, &setsNum, &graphSetsNum,
OptionShowSet);//Поиск
break;
case 4:
matrixs = OptionShowSet(matrixs, &graphNum, &setsNum,
&graphSetsNum);//Просмотр
break;
case 5:
matrixs = OptionSaveSets(matrixs, &graphNum, &setsNum,
&graphSetsNum);//Сохранение2
break;
case 6:
OptionGetFileName(defaultName, 0);
break;
case 7:
system("cls");
exit(0);
break;
}
system("cls");
}
}

```