Министерство науки и высшего образования Российской Федерации Пензенский государственный университет Кафедра «Вычислительной техники»

Пояснительная записка

к курсовому проектированию

по дисциплине: «Логика и основы алгоритмизации в инженерных задачах» на тему: «Реализация поиска независимых множеств вершин графа»

Выполнил студент группы 23ВВВ2:

Попков С.А

Приняли:

Митрохин М.А.

Юрова О.В.

Пенза 2024

Реферат……………………………………………………………………………5

Введение…………………………………………………………………………..6

1 Постановка задачи……………………………………………………………...7

2 Теоретическая часть задания…………………………………………………..8

3 Описание алгоритма программы………………………………………………9

4 Описание программы………………………………………………………….11

5 Тестирование…………………………………………………………………...13

6 Ручной расчет задачи………………………………………………………….19

Заключение………………………………………………………………………22

Список используемой литературы……………………………………………..23

Приложение А. Листинг программы…………………………………………..24

**Реферат**

Отчет 31 стр, 10 рисунков.

ГРАФ, МАТРИЦА СМЕЖНОСТИ, НЕЗАВИСИМЫЕ МНОЖЕСТВА, ОРИЕНТИРОВАННЫЙ ГРАФ, НЕОРИЕНТИРОВАННЫЙ ГРАФ, АЛГОРИТМ ПОИСКА НЕЗАВИСИМЫХ МНОЖЕСТВ.

Цель исследования - разработка программного решения для определения всех независимых множеств вершин в графе, представленном матрицей смежности, и анализа данного подхода для неориентированных графов.

В данной работе осуществляется глубокий анализ и реализация алгоритмов для выявления независимых множеств вершин в графах. Независимое множество в графе определяется как набор вершин, в котором никакие две вершины не соединены между собой. Исследование направлено на изучение способов генерации графов с использованием различных параметров, таких как количество вершин и вероятность наличия ребра, а также на анализ возможности и точности нахождения всех независимых множеств в этих графах.

Работа включает в себя рассмотрение механизмов работы с матрицей смежности для представления графов, разработку функций для генерации случайных графов, а также создание эффективных алгоритмов для вычисления независимых множеств. Особое внимание уделяется анализу производительности алгоритмов в зависимости от размера и сложности графа, а также их применимости как в теоретических, так и в практических задачах.

**Введение**

Алгоритмы, используемые для работы с графами, занимают ключевую позицию в области компьютерных наук и программирования. Они находят применение в самых разнообразных задачах, начиная от анализа социальных сетей и заканчивая сложными научными исследованиями. В данной курсовой работе основное внимание уделяется задаче нахождения независимых множеств вершин в неориентированных графах, представленных матрицей смежности.

Независимое множество вершин в графе – это набор вершин, в котором никакие две вершины не соединены ребром. Этот концепт широко применяется в различных областях, таких как теория сетей, оптимизация и даже в теоретической химии.

В качестве среды разработки была выбрана платформа Microsoft Visual Studio с использованием языка программирования Cи. Целью данной курсовой работы является разработка программы, способной эффективно определять все возможные независимые множества в заданном графе. Эта задача представляет особый интерес, так как алгоритмы для работы с графами часто требуют значительных вычислительных ресурсов, особенно при обработке больших графов.

В процессе работы будет проведен анализ различных подходов к решению данной задачи, их эффективности и применимости в различных ситуациях. В конечном итоге, целью является не только реализация рабочего программного обеспечения, но и понимание особенностей и ограничений алгоритмов поиска в графах, а также их практическое применение.

**1 Постановка задачи**

Создать программу, предоставляющую пользователю выбор между двумя способами ввода матрицы смежности графа: случайная генерация или ручной ввод.

Реализовать алгоритмы для определения, является ли данный набор вершин независимым множеством в графе.

Разработать функцию для нахождения всех независимых множеств в графе.

Обеспечить вывод на экран результатов работы программы: отображение матрицы смежности, всех найденных независимых множеств.

Предусмотреть возможность сохранения результатов работы программы в файл.

Обеспечить обработку граничных условий и исключений для корректной работы программы при любых входных данных.

Разработать пользовательский интерфейс для ввода данных с клавиатуры и мыши.

Задания выполняются в соответствии с вариантом №19.

**2 Теоретическая часть задания**

Граф G представляет собой набор вершин X1, X2, ..., Xn и множество рёбер, соединяющих эти вершины. В неориентированном графе рёбра двунаправленны.

Матрица смежности в графах.

Матрица смежности — это способ представления графа, где элементы матрицы отражают наличие или отсутствие рёбер между парами вершин. Элемент матрицы равен 1, если между вершинами существует ребро, и 0 в противном случае. Этот метод представления эффективен как для ориентированных, так и для неориентированных графов.

Независимые множества в графах.

Основная задача, решаемая в данном проекте, — это поиск всех независимых множеств в графе. Независимое множество в графе — это подмножество вершин, ни одна из которых не соединена рёбрами с другой вершиной в этом же множестве. Поиск всех независимых множеств является важной задачей в теории графов и имеет множество приложений, от алгоритмов раскраски графа до различных оптимизационных задач.

**3 Описание алгоритма программы**

Для реализации алгоритма понадобится функции, которые будут находить все независимые множества и выводить их, а после сохранять их в файл.

На вход подается матрица смежности, которая описывает граф, и количество вершин в графе. Затем программа инициализирует массив для хранения текущего независимого множества и начинает рекурсивный перебор всех подмножеств вершин.

В ходе рекурсивного перебора для каждой вершины проверяется, можно ли включить её в текущее подмножество. Это делается путем проверки на наличие рёбер между текущей вершиной и уже выбранными вершинами множества. Если между вершинами нет рёбер, то вершина может быть добавлена в подмножество. Если хотя бы одно ребро есть, то вершина исключается из текущего подмножества. Этот процесс повторяется для всех вершин графа, после чего результат работы алгоритма выводится в консоль, а также по желанию его можно сохранить в файл.

После того как все независимые множества найдены, программа завершает свою работу. Таким образом, результатом работы программы является запись всех независимых множеств в файл, где каждое множество представлено списком индексов вершин.

Ниже представлен псевдокод функций «findAllIndependentSets», «findIndependentSetsToFile».

**findAllIndependentSets ()**

1. если index == vertices (достигнут конец списка вершин)

2. если currentSize > 0 (есть хотя бы одна вершина в текущем множестве)

3. записать "{" в файл

4. для каждой вершины i в currentSet

5. записать i в файл

6. записать "}" в файл

7. вернуть из функции

8. установить canInclude в true

9. для каждой вершины i от 0 до currentSize-1

10. если adjacency\_matrix[currentSet[i]][index] == 1 (есть ребро между текущей вершиной и индексом)

11. установить canInclude в false

12. выйти из цикла

13. если canInclude == true

14. добавить index в currentSet[currentSize]

15. вызвать рекурсивно findAllIndependentSets с currentSize + 1 и index + 1

16. вызвать рекурсивно findAllIndependentSets с тем же currentSize и index + 1 (исключаем текущую вершину)

**findIndependentSetsToFile()**

1. записать "Множества независимых вершин:" в файл

2. создать пустой массив currentSet

3. вызвать findAllIndependentSets с currentSet, начальным currentSize = 0, начальным index = 0, и file

**4 Описание программы**

Для реализации данной программы был выбран язык программирования Cи. Это мощный универсальный язык программирования, обладающий как высокоуровневыми, так и низкоуровневыми возможностями. Программа представляет собой консольное приложение, разработанное в среде Visual Studio.

Программа состоит из нескольких функций: findAllIndependentSets, findIndependentSetsToFile, а также дополнительных функций для взаимодействия с пользователем и обработки данных. Основная цель программы – выявление и вывод всех независимых множеств в заданном графе.

Работа программы начинается с заставки, в которой представлена информация о курсовой работе и студенте, выполнившим эту работу, после чего предлагается ряд операций.

1. Ввести размер графа. Пользователь задает изначальное количество вершин в графе.

2. Граф представлен в виде матрицы смежности. При выборе 2 пользователь имеет возможность заполнить её вручную (1 или 0).

3. При выборе 3 пользователь заполняет матрицу случайными числами 0 – нет ребра между вершинами или 1 – ребро есть.

4. При выборе 4 начинает работать функция findAllIndependentSets для нахождения всех независимых множеств в графе. Эти множества определяются на основе проверки каждого возможного подмножества вершин графа.

Затем программа выводит все найденные независимые множества на экран.

5. При выборе 5 пользователь сможет сохранить результаты работы с помощью функции saveResultsToFile, которая сохраняет независимые множества вершин, а также матрицу смежности.

6. При 6 пользователь может посмотреть текущую матрицу смежности

7. 7 служит для завершения программы и выводе финальной заставки.

Таким образом, данная программа представляет собой эффективный инструмент для анализа графов, особенно в контексте выявления независимых множеств, что может быть полезно в различных областях, включая оптимизацию и теоретическую информатику.

Результаты работы программы представлены на рисунке 1.

**5 Тестирование**

Среда Microsoft Visual Studio 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, алгоритмом программы.

Ниже продемонстрирован результат тестирования программы с помощью разных видов ввода матрицы, сохранение и вывод результатов программы, вывод всех независимых множеств. На рисунках 4 представлены результаты сохранения в файл. На рисунках 7 и 9 продемонстрированы ручные расчеты, подтверждающие корректность работы программы. Итоговые результаты тестирования представлены в таблице 1.

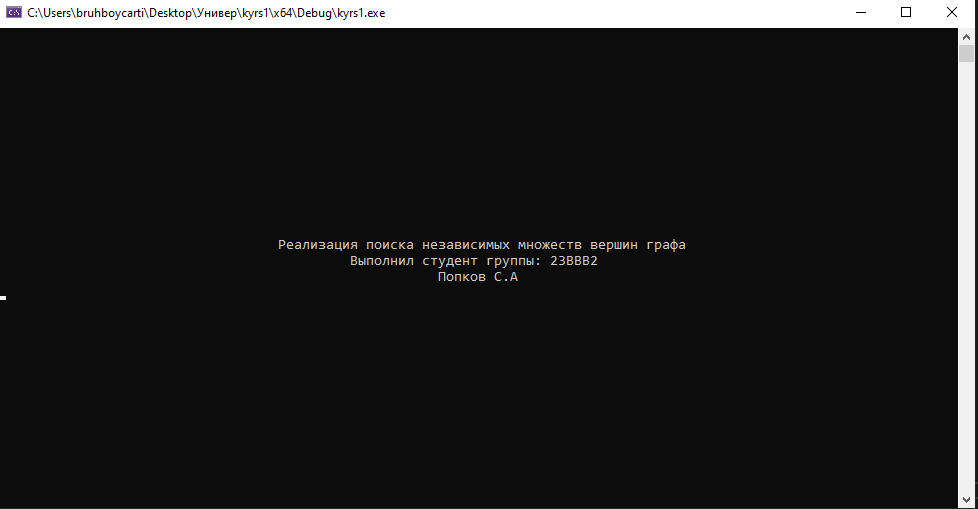


Рисунок 1 – Начальная заставка

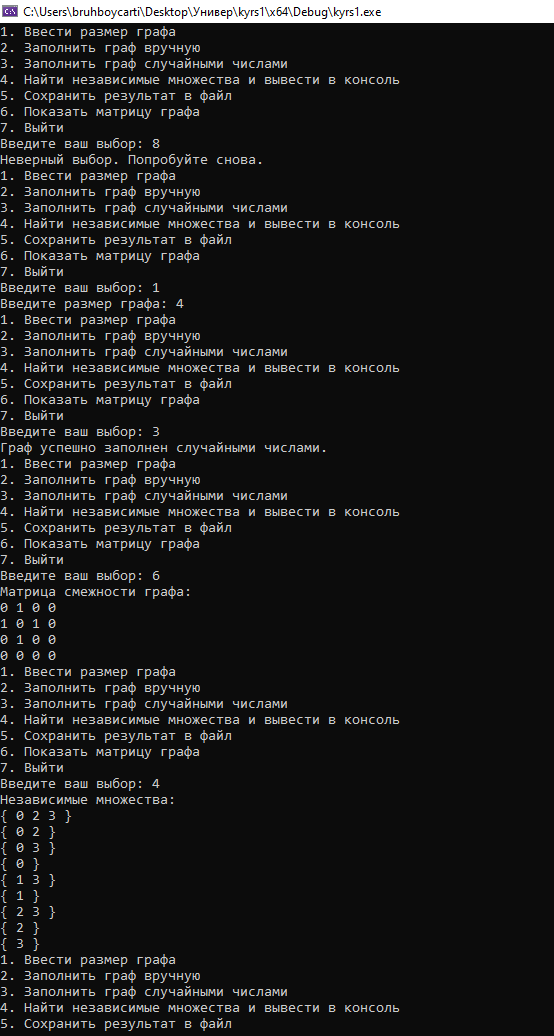


Рисунок 2 - Тестирование программы на корректность создания матрицы и нахождение независимых вершин с помощью случайной генерации

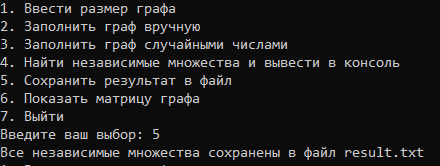


Рисунок 3 - Тестирование программы на сохранение результата программы в файл

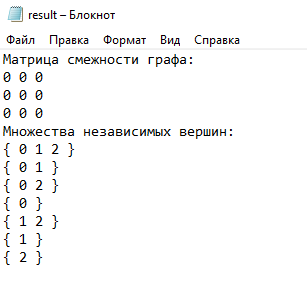


Рисунок 4 - Результат сохранения матрицы и независимых множеств в файл

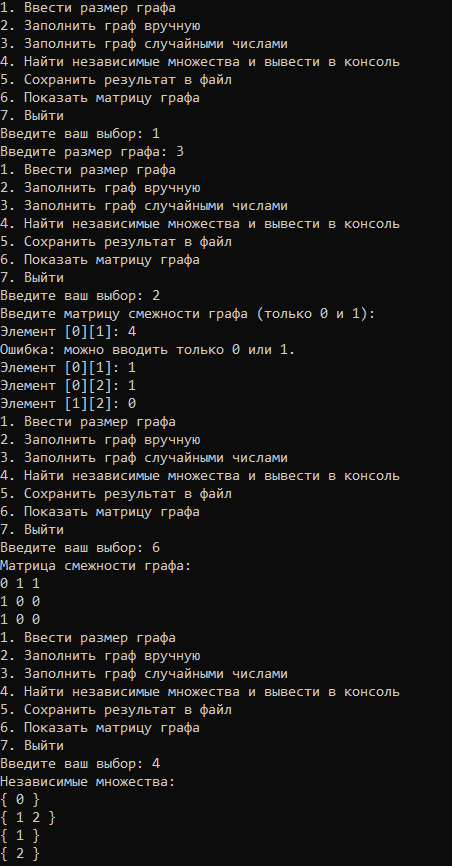


Рисунок 5 - Тестирование программы на корректность создания матрицы и нахождение независимых вершин с помощью выгрузки матрицы из файла

Таблица 1 - Описание поведения программы при тестировании

|  |  |  |
| --- | --- | --- |
| Описание теста | Ожидаемый результат | Полученный результат |
| Выбор операции | При некорректных значениях выводит ошибку и просит ввести нужное число | Верно |
| При выборе генерации | При некорректных значениях выводит ошибку и просит ввести нужное число | Верно |
| Генерация | Правильная генерация матрицы смежности | Верно |
| Нахождение независимых множеств | Находит все независимые множества для любых генераций и для любой матрицы | Верно |
| Сохранение результата | Сохраняет результат в виде матрицы смежности и независимых множеств в файл | Верно |

В результате тестирования, было выявлено, что программа выполняет все операции в соответствии с задачами.

**6 Ручной расчет задачи**

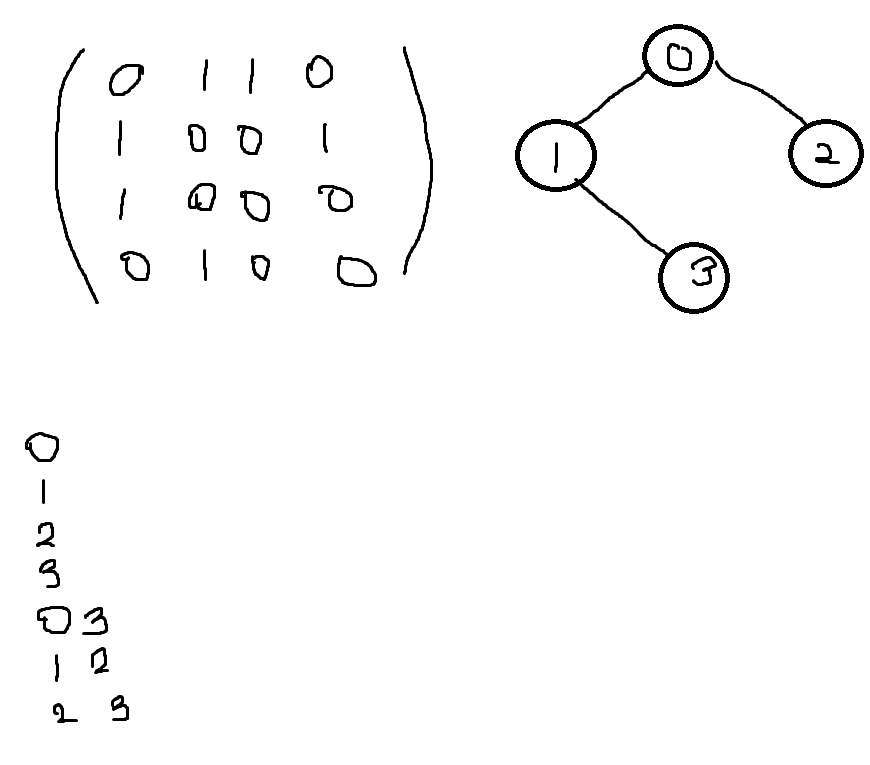


Рисунок 6 - Поиск независимых вершин вручную для неориентированного графа

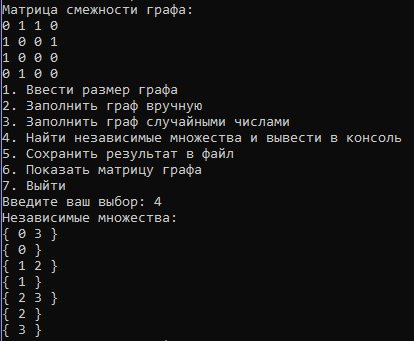


Рисунок 7 - Проверка на корректную работу программы для неориентированного графа

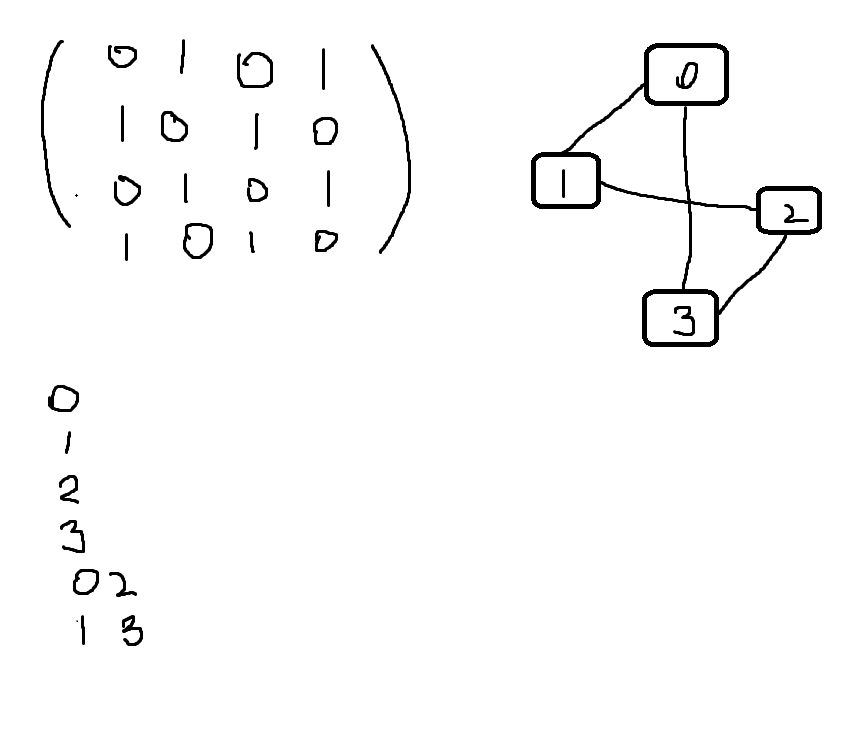


Рисунок 8 - Поиск независимых вершин вручную для ориентированного графа

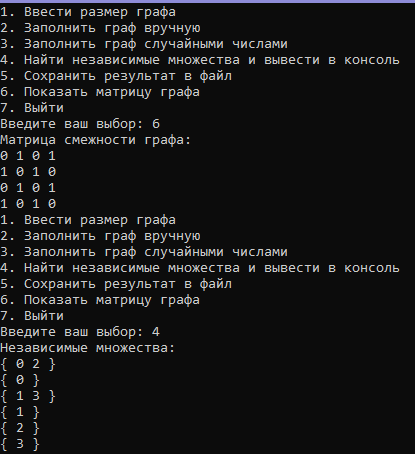


Рисунок 9 - Проверка на корректную работу программы для ориентированного графа

**Заключение**

Курсовой проект демонстрирует глубокое понимание принципов работы с графами и алгоритмических методов решения задач теории графов. Разработанная программа предоставляет функционал для создания неориентированных графов, проверки их эффективного нахождения всех независимых множеств вершин.

В процессе работы над проектом были улучшены навыки программирования на языке C, а также углублены знания в области алгоритмических основ теории графов, что является значительным вкладом в профессиональное развитие.

Основным недостатком программы является отсутствие графического пользовательского интерфейса, что делает ее менее интуитивно понятной для пользователей. Тем не менее, в рамках академической задачи программа выполняет свои функции эффективно, демонстрируя корректность и точность в решении задачи поиска независимых множеств в графе.

Таким образом, курсовой проект успешно достиг своих образовательных и практических целей, продемонстрировав глубокие знания в области программирования и алгоритмической обработки данных в контексте теории графов.

**Список используемой литературы**

1. Бьярне Страуструп. **Язык программирования C++.** 2013г.

2. Томас Х. Кормен, Чарльз Е. Лейзерсон, Рональд Л. Ривест, Клиффорд Штайн. **Алгоритмы. Построение и анализ.** 2009г.

3. Бьярне Страуструп. Программирование. Принципы и практика с использованием C++. 2018г.

4. Седжвик, Роберт. Алгоритмы на C++. М.: Вильямс, 2013.

5. Харари, Франк. Теория графов. М.: Мир, 1973.

6. Гаврилов, М. В. Теория графов. М.: Физматлит, 2000.

7. Беллман, Р. Введение в теорию матриц. М.: Наука, 1970.

**Приложение А.**

**Листинг программы.**

#define \_CRT\_SECURE\_NO\_WARNINGS

#include <stdio.h>

#include <stdlib.h>

#include <time.h>

#include <locale.h>

#include <windows.h>

// Стартовая заставка с информацией о проекте

int z1() {

system("cls");

setlocale(LC\_ALL, "Rus");

char title[] = " Реализация поиска независимых множеств вершин графа\n";

char title2[] = "Выполнил студент группы: 23ВВВ2\n";

char title3[] = " Попков С.А\n";

int length = strlen(title);

int length2 = strlen(title2);

int length3 = strlen(title3);

int x, y;

CONSOLE\_SCREEN\_BUFFER\_INFO csbi;

GetConsoleScreenBufferInfo(GetStdHandle(STD\_OUTPUT\_HANDLE), &csbi);

int console\_width = csbi.srWindow.Right - csbi.srWindow.Left + 1;

int console\_height = csbi.srWindow.Bottom - csbi.srWindow.Top + 1;

// рассчет отступа для каждой строки

int x\_title = (console\_width - length) / 2;

int x\_title2 = (console\_width - length2) / 2;

int x\_title3 = (console\_width - length3) / 2;

y = console\_height / 2;

// заголовок по центру

for (int i = 0; i < y - 2; i++) // Добавляем пустые строки чтобы был ближе к центру

printf("\n");

for (int i = 0; i < x\_title; i++)

printf(" ");

for (int j = 0; j < length; j++) {

printf("%c", title[j]);

Sleep(20);

}

Sleep(200);

// ФИО по центру

for (int i = 0; i < x\_title2; i++)

printf(" ");

for (int m = 0; m < length2; m++) {

printf("%c", title2[m]);

Sleep(60);

}

Sleep(600);

// информация о группе и курсе по центру

for (int l = 0; l < x\_title3; l++)

printf(" ");

for (int n = 0; n < length3; n++) {

printf("%c", title3[n]);

Sleep(60);

}

Sleep(500);

system("cls");

return 0;

}

void showMenu() {

printf("1. Ввести размер графа\n");

printf("2. Заполнить граф вручную\n");

printf("3. Заполнить граф случайными числами\n");

printf("4. Найти независимые множества и вывести в консоль\n");

printf("5. Сохранить результат в файл\n");

printf("6. Показать матрицу графа\n");

printf("7. Выйти\n");

}

// функция для создания пустой матрицы смежности

int\*\* createGraph(int size) {

int\*\* graph = (int\*\*)malloc(size \* sizeof(int\*));

for (int i = 0; i < size; i++) {

graph[i] = (int\*)malloc(size \* sizeof(int));

for (int j = 0; j < size; j++) {

graph[i][j] = 0; // инициализация нулями

}

}

return graph;

}

// функция для ручного ввода графа

void inputGraph(int\*\* graph, int size) {

printf("Введите матрицу смежности графа (только 0 и 1):\n");

for (int i = 0; i < size; i++) {

for (int j = 0; j < size; j++) {

if (i < j) { // проверяем, чтобы вводить только верхнюю часть матрицы

int value;

do {

printf("Элемент [%d][%d]: ", i, j);

scanf\_s("%d", &value);

if (value != 0 && value != 1) {

printf("Ошибка: можно вводить только 0 или 1.\n");

}

} while (value != 0 && value != 1);

// Устанавливаем значение симметрично для неориентированного графа

graph[i][j] = value;

graph[j][i] = value;

}

else if (i == j) {

graph[i][j] = 0; // Нет петель на вершину

}

}

}

}

// функция для заполнения графа случайными числами (только 0 и 1)

void randomGraph(int\*\* graph, int size) {

srand(time(NULL));

for (int i = 0; i < size; i++) {

for (int j = 0; j < size; j++) {

if (i != j) {

graph[i][j] = rand() % 2; // случайное значение 0 или 1

graph[j][i] = graph[i][j]; // обеспечиваем симметричность

}

else {

graph[i][j] = 0; // нет петель на вершину

}

}

}

printf("Граф успешно заполнен случайными числами.\n");

}

// функция для вывода матрицы смежности

void printGraph(int\*\* graph, int size) {

printf("Матрица смежности графа:\n");

for (int i = 0; i < size; i++) {

for (int j = 0; j < size; j++) {

printf("%d ", graph[i][j]);

}

printf("\n");

}

}

// Функция для поиска всех независимых множеств

void findAllIndependentSets(int\*\* graph, int size, int\* currentSet, int currentSize, int index, FILE\* file) {

if (index == size) {

if (currentSize > 0) {

fprintf(file, "{ ");

for (int i = 0; i < currentSize; i++) {

fprintf(file, "%d ", currentSet[i] + 1); // +1 для удобства отображения

}

fprintf(file, "}\n");

}

return;

}

// Включаем текущую вершину в независимое множество

int canInclude = 1;

for (int i = 0; i < currentSize; i++) {

if (graph[currentSet[i]][index] == 1) {

canInclude = 0;

break;

}

}

if (canInclude) {

currentSet[currentSize] = index;

findAllIndependentSets(graph, size, currentSet, currentSize + 1, index + 1, file);

}

// Исключаем текущую вершину из независимого множества

findAllIndependentSets(graph, size, currentSet, currentSize, index + 1, file);

}

// Функция для поиска независимых множеств и записи их в файл

void findIndependentSetsToFile(int\*\* graph, int size, FILE\* file) {

fprintf(file, "Множества независимых вершин:\n"); // Заголовок перед выводом множеств

// Динамическое выделение памяти для currentSet

int\* currentSet = (int\*)malloc(size \* sizeof(int));

if (currentSet == NULL) {

printf("Ошибка: не удалось выделить память для currentSet.\n");

return;

}

// Поиск независимых множеств

findAllIndependentSets(graph, size, currentSet, 0, 0, file);

// Освобождение памяти

free(currentSet);

}

// сохранение результатов в файл

void saveResultsToFile(int\*\* graph, int size) {

FILE\* file = fopen("result.txt", "w");

if (file == NULL) {

printf("Не удалось открыть файл для записи.\n");

return;

}

// сохраняем матрицу смежности

fprintf(file, "Матрица смежности графа:\n");

for (int i = 0; i < size; i++) {

for (int j = 0; j < size; j++) {

fprintf(file, "%d ", graph[i][j]);

}

fprintf(file, "\n");

}

// запускаем поиск независимых множеств и сохраняем в файл

findIndependentSetsToFile(graph, size, file);

// закрываем файл

fclose(file);

printf("Все независимые множества сохранены в файл result.txt\n");

}

// основная функция

int main() {

int size = 0;

int\*\* graph = NULL;

int choice;

z1(); // запуск заставки

while (1) {

showMenu();

printf("Введите ваш выбор: ");

scanf\_s("%d", &choice);

switch (choice) {

case 1:

printf("Введите размер графа: ");

scanf\_s("%d", &size);

if (size < 1) {

printf("Размер графа должен быть больше 0.\n");

return 1;

}

else {

graph = createGraph(size); // Создаем граф

}

break;

case 2:

if (size == 0) {

printf("Сначала задайте размер графа!\n");

}

else {

inputGraph(graph, size); // Заполняем граф вручную

}

break;

case 3:

if (size == 0) {

printf("Сначала задайте размер графа!\n");

}

else {

randomGraph(graph, size); // Заполняем граф случайными числами

}

break;

case 4:

if (size == 0) {

printf("Сначала заполните граф!\n");

}

else {

printf("Независимые множества:\n");

// Запрашиваем у пользователя количество вершин

int numVertices;

printf("Введите количество вершин: ");

scanf("%d", &numVertices);

// Динамическое выделение памяти для массива индексов вершин текущего независимого множества

int\* currentSet = (int\*)malloc(numVertices \* sizeof(int));

if (currentSet == NULL) {

printf("Ошибка выделения памяти!\n");

return 1; // Завершение программы в случае ошибки

}

findAllIndependentSets(graph, size, currentSet, 0, 0, stdout); // Выводим в консоль

// Освобождение выделенной памяти

free(currentSet);

}

break;

case 5:

if (size == 0) {

printf("Сначала найдите независимые вершины!\n");

}

else {

saveResultsToFile(graph, size); // Сохраняем результаты в файл

}

break;

case 6:

if (size == 0) {

printf("Сначала создайте матрицу!\n");

}

else {

printGraph(graph, size); // Показываем матрицу графа

}

break;

case 7:

for (int i = 0; i < size; i++)

free(graph[i]); // Освобождаем память

free(graph);

return 0;

default:

printf("Неверный выбор. Попробуйте снова.\n");

break;

}

}

return 0;

}