Министерство науки и высшего образования РФ

Пензенский государственный университет

Кафедра «Вычислительной техники»

**ПОЯСНИТЕЛЬНАЯ ЗАПИСКА**

к курсовой работе

по курсу «Логика и основы алгоритмизации в инженерных задачах»

на тему «Реализация алгоритма поиска независимых множеств графа»

Выполнил:

Студент группы 23ВВВ2

Кокарев Д.С.

Принял:

Юрова О.В.

Пенза 2024

**Содержание**

Реферат……………………………………………………………………………3

Введение…………………………………………………………………………..4

1 Постановка задачи……………………………………………………………...5

2 Теоретическая часть задания…………………………………………………..6

3 Описание алгоритма программы………………………………………………7

4 Описание программы…………………………………………………………..9

5 Тестирование…………………………………………………………………...11

6 Ручной расчет задачи………………………………………………………….16

Заключение………………………………………………………………………18

Список используемой литературы……………………………………………..19

Приложение А. Листинг программы…………………………………………..20

**Реферат**

Отчет 30 стр, 11 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 как для ориентированных, так и для неориентированных графов.

В данной работе осуществляется глубокий анализ и реализация алгоритмов для выявления независимых множеств вершин в графах. Независимое множество вершин 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 в графах, как ориентированных, так и неориентированных, представленных матрицей смежности.

Независимое множество 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.

Для реализации была выбрана среда разработки Microsoft Visual Studio с использованием языка программирования C++. Целью данной курсовой работы является разработка программы, способной эффективно определять все возможные независимые множества вершин 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 тремя способами ввода матрицы смежности графа: случайная генерация, ручной ввод, или чтение из файла.

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, а также графическое представление графа, если это возможно.

Предусмотреть возможность сохранения результатов работы программы в файл, при этом пользователь должен иметь возможность выбора между сохранением в новый файл или добавлением результатов в существующий.

Реализовать функционал для просмотра предыдущих результатов из файлов.  
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 зависимости от характера рёбер графы бывают ориентированными и неориентированными. В ориентированном графе (орграфе) рёбра имеют направление, в то время как 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 вершин в графе. Независимое множество вершин 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 понадобятся функции, которые будут определять независимость множеств в графе, находить все независимые множества вершин и выводить их на экран.

На вход подается матрица смежности. После осуществляется перебор всех вершин и проверка на наличие ребер между ними. Если ребро между вершинами существует, программа возвращает сведение о наличии ребра.

После идет определение количества вершин в графе. Инициализация вектора для хранения всех независимых множеств. Перебор всех подмножеств вершин графа с использованием битовой маски. Для каждого подмножества проверка его на независимость с помощью. Добавление независимых множеств в вектор.

Проверка, содержит ли вектор какие-либо множества. Для каждого независимого множества, перебор и вывод всех его вершин.

Ниже представлен псевдокод функций «isIndependentSet», «findAllIndependentSets», «printAllIndependentSets».

**isIndependentSet (adjacency\_matrix, vertices, isDirected)**  
1. для каждой вершины vertex в vertices  
 2. для каждой смежной вершины adjacent\_vertex в vertices  
 3. если vertex != adjacent\_vertex  
 4. если isDirected и (adjacency\_matrix[vertex][adjacent\_vertex] == 1 или adjacency\_matrix[adjacent\_vertex][vertex] == 1)  
 5. вернуть false  
 6. если !isDirected и adjacency\_matrix[vertex][adjacent\_vertex] == 1  
 7. вернуть false  
8. вернуть true  
9. Конец Функции

**findAllIndependentSets (adjacency\_matrix, isDirected)**  
1. num\_vertices = размер adjacency\_matrix - 1  
2. all\_independent\_sets = пустой вектор множеств  
3. для каждой маски от 1 до (1 « num\_vertices)  
 4. current\_set = пустое множество  
 5. для i от 0 до num\_vertices  
 6. если маска & (1 « i) не равно 0  
 7. добавить i + 1 в current\_set  
 8. если isIndependentSet(adjacency\_matrix, current\_set, isDirected)  
 9. добавить current\_set в all\_independent\_sets  
10. вернуть all\_independent\_sets  
11. Конец Функции

**printAllIndependentSets (all\_independent\_sets)**  
1. если all\_independent\_sets пуст  
 2. вывод "Нет независимых множеств."  
3. иначе  
 4. вывод "Все независимые множества:"  
 5. для каждого independent\_set в all\_independent\_sets  
 6. вывод "{ "  
 7. для каждой вершины vertex в independent\_set  
 8. вывод vertex  
 9. вывод " }"  
10. Конец Функции

Полный код программы расположен в Приложении А.

**4 Описание программы**

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

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

Работа программы начинается с того, что пользователю предлагается ряд операций.

1. Сгенерировать матрицу.

Граф представлен в виде матрицы смежности. Пользователь имеет возможность сгенерировать матрицу смежности автоматически, ввести её вручную, а также загрузить из файла. В случае автоматической генерации, предусмотрена функция для определения вероятности наличия ребра между вершинами.

После генерации или ввода матрицы смежности, программа использует функцию findAllIndependentSets для нахождения всех независимых множеств вершин в графе. Эти множества определяются на основе проверки каждого возможного подмножества вершин графа функцией isIndependentSet.

Затем, с использованием функции printAllIndependentSets, программа выводит все найденные независимые множества вершин на экран.

После пользователю предлагается сохранить результат в файл.

2. При выборе '2' пользователь сможет посмотреть предыдущие результаты работы.

3. При '3' пользователь может очистить файл с результатами, не заходя в него.

4. При использовании '4' очищается консоль.

5. '5' служит для завершения программы.

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.

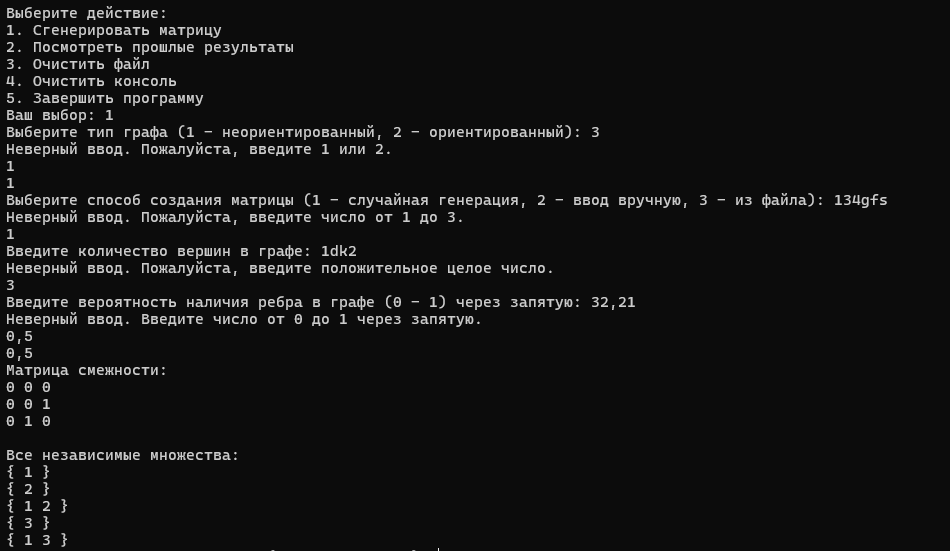
**5 Тестирование**

Среда разработки 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 ввода разных видов матрицы, сохранения и вывода результатов программы, вывода всех независимых множеств вершин графа. На рисунках 3 и 6 представлены результаты сохранения и загрузки матрицы в файл. На рисунках 8 и 10 продемонстрированы ручные расчеты, подтверждающие корректность работы программы. Итоговые результаты тестирования представлены в таблице 1.

Рисунок 1 - Тестирование программы на корректность создания матрицы и нахождение независимых вершин с помощью случайной генерации

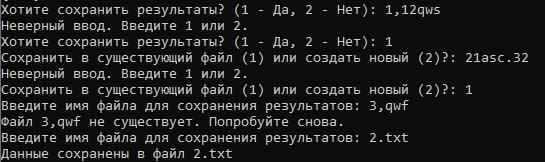


Рисунок 2 - Тестирование программы на сохранение результата программы в файл

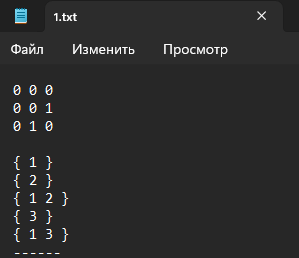
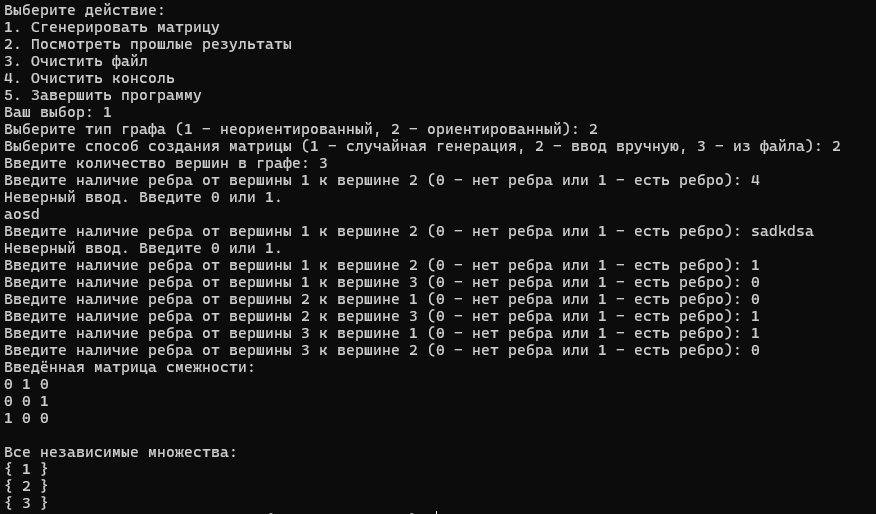
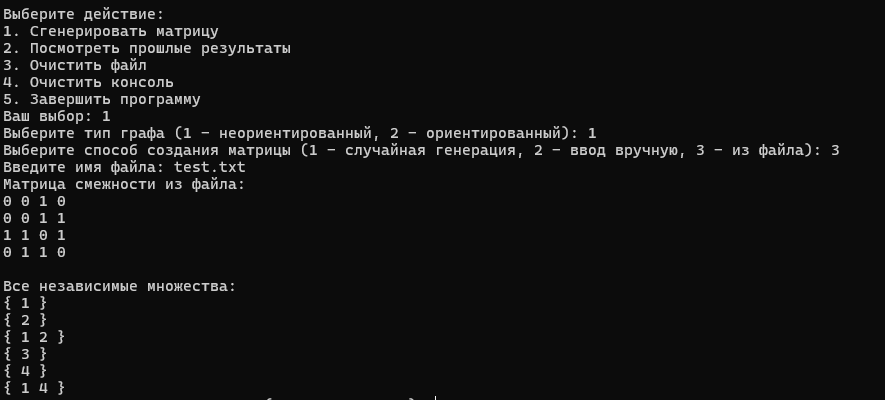


Рисунок 3 - Результат сохранения матрицы и независимых множеств в файл

Рисунок 4 - Тестирование программы на корректность создания матрицы и нахождение независимых вершин с помощью ручного ввода данных

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

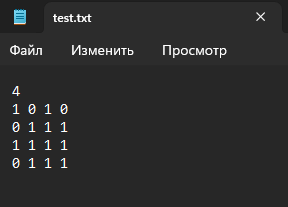


Рисунок 6 - Загруженная матрица из файла

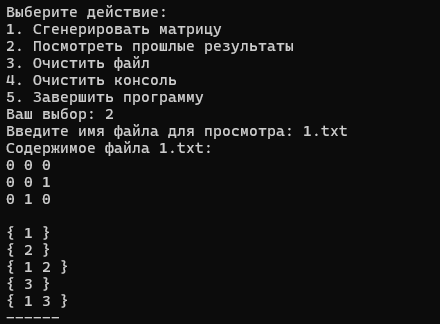


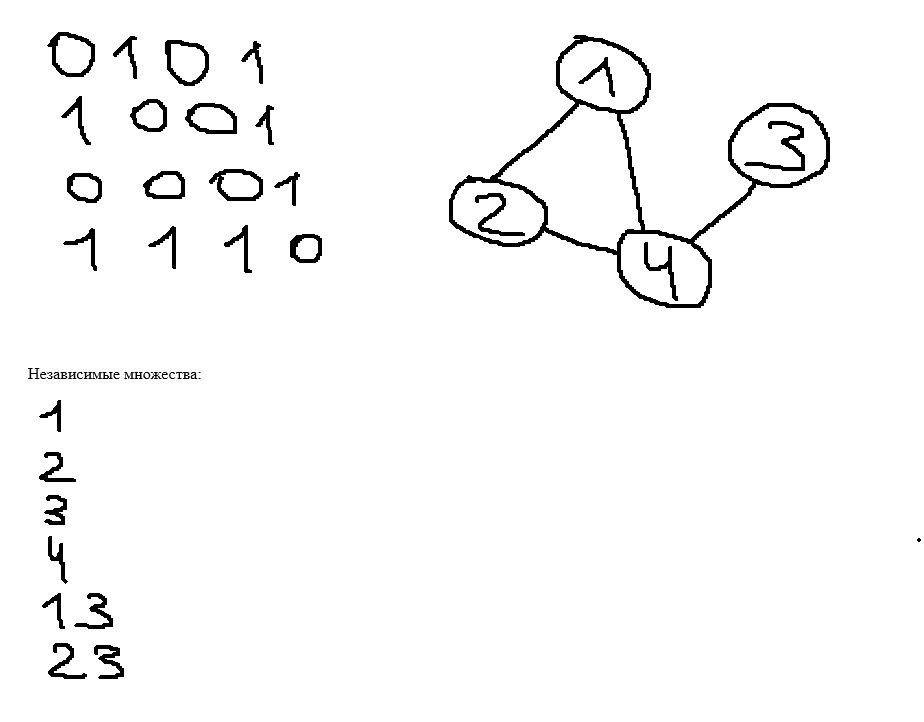
Рисунок 7 - Вывод сохраненного результата из файла

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 |
| Вывод предыдущих результатов программы | Выводит результаты предыдущих расчетов из файла | Верно |

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

**6 Ручной расчет задачи**

Рисунок 8 - Поиск независимых вершин вручную для неориентированного графа

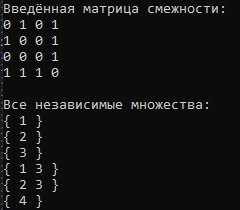
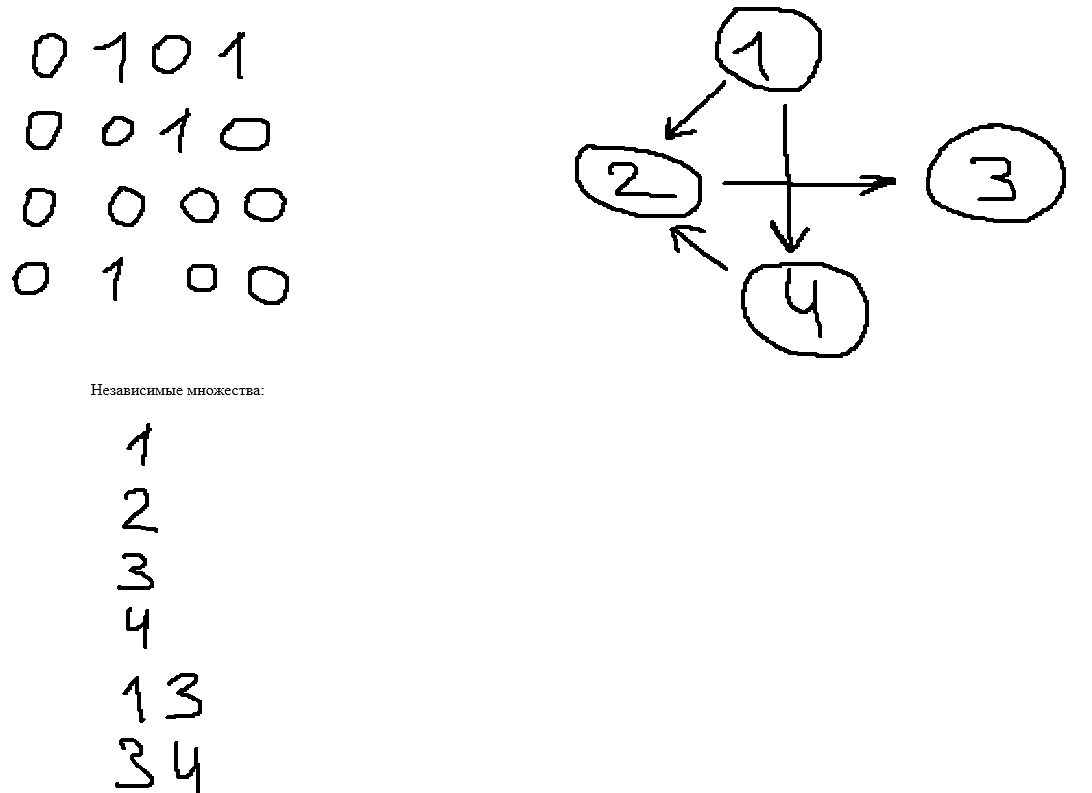


Рисунок 9 - Проверка на корректную работу программы для неориентированного графа

Рисунок 10 - Поиск независимых вершин вручную для ориентированного графа

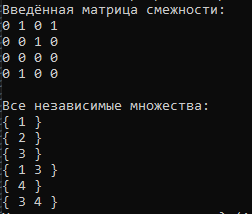


Рисунок 11 - Проверка на корректную работу программы для ориентированного графа

**Заключение**

Курсовой проект успешно демонстрирует глубокое понимание принципов работы с графами и алгоритмических подходов к решению задач теории графов. Разработанная программа обладает функционалом для создания как ориентированных, так и неориентированных графов, позволяет проверять их на валидность, а также эффективно находит все независимые множества вершин.

В процессе работы над проектом были усовершенствованы навыки программирования на C++, особенно в области работы с контейнерами STL, файловым вводом/выводом и обработкой данных. Также были углублены знания в области алгоритмических основ теории графов, что является важным вкладом в профессиональное развитие.

Основным недостатком программы является отсутствие графического пользовательского интерфейса, что делает ее менее интуитивно понятной для конечного пользователя. Однако, в рамках академической задачи, программа выполняет свои функции эффективно, демонстрируя корректность и точность в решении поставленной задачи поиска независимых множеств вершин 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 <cstdio>

#include <vector>

#include <ctime>

#include <cstdlib>

#include <locale.h>

#include <set>

#include <cstdlib>

#include <ctime>

#include <string>

#include <fstream>

#include <iostream>

#include <windows.h>

using namespace std;

// Функция проверяет, является ли набор вершин 'vertices' независимым множеством в графе, представленном матрицей смежности 'adjacency\_matrix'.

// 'isDirected' указывает, является ли граф ориентированным.

bool isIndependentSet(const vector < vector < int>>& adjacency\_matrix, const set<int>& vertices, bool isDirected) {

for (int vertex : vertices) { // Перебор всех вершин в наборе

for (int adjacent\_vertex : vertices) { // Перебор всех смежных вершин

if (vertex != adjacent\_vertex) { // Исключение самопетель

// Проверка на наличие ребра между 'vertex' и 'adjacent\_vertex'

if (isDirected && (adjacency\_matrix[vertex][adjacent\_vertex] == 1 || adjacency\_matrix[adjacent\_vertex][vertex] == 1)) {

return false; // Найдено ребро в ориентированном графе, множество не независимо

}

if (!isDirected && adjacency\_matrix[vertex][adjacent\_vertex] == 1) {

return false; // Найдено ребро в неориентированном графе, множество не независимо

}

}

}

}

return true; // Множество вершин независимо

}

// Функция находит все независимые множества в графе, представленном матрицей смежности 'adjacency\_matrix'.

// 'isDirected' указывает, является ли граф ориентированным.

vector < set < int>> findAllIndependentSets(const vector < vector < int>>& adjacency\_matrix, bool isDirected) {

int num\_vertices = adjacency\_matrix.size() - 1; // Количество вершин в графе

vector < set < int>> all\_independent\_sets; // Вектор для хранения всех независимых множеств

// Перебор всех подмножеств вершин графа

for (int mask = 1; mask < (1 << num\_vertices); ++mask) {

set<int> current\_set; // Текущее рассматриваемое подмножество вершин

// Создание множества вершин на основе битовой маски 'mask'

for (int i = 0; i < num\_vertices; ++i) {

if (mask & (1 << i)) {

current\_set.insert(i + 1); // Добавление вершины в множество

}

}

// Проверка, является ли текущее множество независимым

if (isIndependentSet(adjacency\_matrix, current\_set, isDirected)) {

all\_independent\_sets.push\_back(current\_set); // Добавление независимого множества в список

}

}

return all\_independent\_sets; // Возврат списка всех независимых множеств

}

// Функция выводит все найденные независимые множества, содержащиеся в 'all\_independent\_sets'.

void printAllIndependentSets(const vector<set<int>>& all\_independent\_sets) {

if (all\_independent\_sets.empty()) {

printf("\nНет независимых множеств.\n");

}

else {

printf("\nВсе независимые множества:\n");

for (const auto& independent\_set : all\_independent\_sets) { // Перебор всех независимых множеств

printf("{ ");

for (int vertex : independent\_set) { // Перебор вершин каждого множества

printf("%d ", vertex); // Вывод номера вершины

}

printf("}\n");

}

}

}

// Функция генерирует случайную матрицу смежности для графа с 'num\_vertices' вершинами и вероятностью наличия ребра 'edge\_probability'.

// 'graphType' указывает на тип графа: 1 для неориентированного, 2 для ориентированного.

vector<vector<int>> generateAndPrintMatrix(int num\_vertices, double edge\_probability, int graphType) {

vector<vector<int>> adjacency\_matrix(num\_vertices + 1, vector<int>(num\_vertices + 1, 0)); // Матрица смежности

// Генерация матрицы смежности

for (int i = 1; i <= num\_vertices; ++i) {

for (int j = (graphType == 1 ? i + 1 : 1); j <= num\_vertices; ++j) {

if (i != j && static\_cast<double>(rand()) / RAND\_MAX < edge\_probability) {

adjacency\_matrix[i][j] = 1; // Установка ребра

if (graphType == 1) {

adjacency\_matrix[j][i] = 1; // Для неориентированного графа устанавливаем ребро в обоих направлениях

}

}

}

}

// Вывод матрицы смежности

printf("Матрица смежности:\n");

for (int i = 1; i <= num\_vertices; ++i) {

for (int j = 1; j <= num\_vertices; ++j) {

printf("%d ", adjacency\_matrix[i][j]); // Вывод элемента матрицы

}

printf("\n");

}

return adjacency\_matrix; // Возврат сгенерированной матрицы

}

// Функция проверяет, является ли матрица 'matrix' валидной матрицей смежности для графа.

bool isValidMatrix(const vector<vector<int>>& matrix) {

int num\_vertices = matrix.size() - 1; // Количество вершин, предполагаемое по размеру матрицы

// Проверка каждого элемента матрицы

for (int i = 1; i <= num\_vertices; ++i) {

if (matrix[i].size() != num\_vertices + 1) {

return false; // Несоответствие размеров матрицы

}

for (int j = 1; j <= num\_vertices; ++j) {

if (matrix[i][j] != 0 && matrix[i][j] != 1) {

return false; // Недопустимые значения в матрице (должны быть 0 или 1)

}

}

}

return true; // Матрица валидна

}

// Проверяет, является ли данная матрица валидной матрицей смежности для графа.

bool isMatrixValid(const vector<vector<int>>& matrix, bool isDirected) {

int num\_vertices = matrix.size() - 1; // Число вершин в графе

for (int i = 1; i <= num\_vertices; ++i) {

for (int j = 1; j <= num\_vertices; ++j) {

// Для ориентированного графа: проверка, что на диагонали нули

if (isDirected && i == j && matrix[i][j] != 0) return false;

// Для неориентированного графа: проверка симметричности матрицы

if (!isDirected && matrix[i][j] != matrix[j][i]) return false;

}

}

return true;

}

// Читает матрицу смежности из файла.

vector<vector<int>> readMatrixFromFile(const string& filename, int& num\_vertices, bool isDirected) {

ifstream file(filename); // Поток для чтения файла

vector<vector<int>> matrix; // Матрица смежности

if (file.is\_open()) {

file >> num\_vertices; // Чтение числа вершин

matrix.resize(num\_vertices + 1, vector<int>(num\_vertices + 1, 0)); // Инициализация матрицы

for (int i = 1; i <= num\_vertices; ++i) {

for (int j = 1; j <= num\_vertices; ++j) {

file >> matrix[i][j]; // Заполнение матрицы значениями из файла

}

}

// Обнуляем главную диагональ для любого типа графа

for (int i = 1; i <= num\_vertices; ++i) {

matrix[i][i] = 0;

}

// Делаем матрицу симметричной для неориентированного графа

if (!isDirected) {

for (int i = 1; i <= num\_vertices; ++i) {

for (int j = i + 1; j <= num\_vertices; ++j) {

matrix[j][i] = matrix[i][j];

}

}

}

// Вывод матрицы смежности

printf("Матрица смежности из файла:\n");

for (int i = 1; i <= num\_vertices; ++i) {

for (int j = 1; j <= num\_vertices; ++j) {

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

}

printf("\n");

}

}

else {

printf("Не удалось открыть файл.\n");

num\_vertices = 0; // Обнуляем количество вершин в случае ошибки

}

return matrix; // Возвращаем матрицу

}

// Сохраняет матрицу в файл.

void saveMatrixToFile(const vector < vector < int>>& matrix, const string& filename, ios\_base::openmode mode) {

ofstream file(filename, mode); // Открываем файл с указанным режимом

if (file.is\_open()) {

// Начинаем с 1, если матрица индексируется с 1

for (int i = 1; i < matrix.size(); i++) {

for (int j = 1; j < matrix[i].size(); j++) {

file << matrix[i][j] << " ";

}

file << "\n";

}

}

else {

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

}

}

// Сохраняет наборы в файл.

void saveSetsToFile(const vector < set < int>>& sets, const string& filename, ios\_base::openmode mode) {

ofstream file(filename, mode); // Открывает файл с указанным режимом

if (file.is\_open()) {

file << "\n";

for (const auto& set : sets) {

file << "{ ";

for (const auto& element : set) {

file << element << " "; // Записывает элементы набора в файл

}

file << "}\n"; // Новая строка после каждого набора

}

file << "------\n";

}

else {

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

}

}

// Показывает предыдущие результаты из файла.

void viewPreviousResults(const string& filename) {

ifstream file(filename); // Поток для чтения файла

if (file.is\_open()) {

string line;

printf("Содержимое файла %s:\n", filename.c\_str());

while (getline(file, line)) {

cout << line << endl; // Выводит каждую строку из файла

}

}

else {

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

}

}

int main() {

srand(time(NULL));

setlocale(LC\_ALL, "");

vector<vector<int>> matrix;

int num\_vertices, i;

double edge\_probability;

char input[100];

char filename[100];

int choice;

char c;

int f;

printf("\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\n");

printf("| Министерство образования Российской Федерации |\n");

printf("| Пензенский государственный университет |\n");

printf("| Кафедра «Вычислительная техника» |\n");

printf("| |\n");

printf("| |\n");

printf("| |\n");

printf("| |\n");

printf("| ПОЯСНИТЕЛЬНАЯ ЗАПИСКА |\n");

printf("| К курсовому проектированию по курсу |\n");

printf("| «Логика и основы алгоритмизации в инженерных задачах» |\n");

printf("| на тему «Алгоритм поиска независимых множеств графа» |\n");

printf("| |\n");

printf("| |\n");

printf("| |\n");

printf("| |\n");

printf("| |\n");

printf("| |\n");

printf("| |\n");

printf("| |\n");

printf("| |\n");

printf("| Выполнил:|\n");

printf("| Студент группы 23ВВВ2|\n");

printf("| Кокарев Д.С|\n");

printf("| Принял:|\n");

printf("| Юрова О.В |\n");

printf("| |\n");

printf("| |\n");

printf("| Пенза 2024 |\n");

printf("\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\n");

time\_t startTime = time(nullptr);

while (difftime(time(nullptr), startTime) < 3) {}

system("cls");

while (true) {

printf("Выберите действие:\n");

printf("1. Сгенерировать матрицу\n");

printf("2. Посмотреть прошлые результаты\n");

printf("3. Очистить файл\n");

printf("4. Очистить консоль\n");

printf("5. Завершить программу\n");

printf("Ваш выбор: ");

if (scanf("%d%c", &choice, &c) != 2 || c != '\n' || choice < 1 || choice > 5) {

printf("Неверный ввод. Пожалуйста, введите число от 1 до 5.\n");

while (getchar() != '\n'); // Очистка входного буфера

continue;

}

switch (choice) {

case 1: {

int graphType, matrixCreationType, saveChoice, fileChoice;

int num\_vertices;

double edge\_probability;

// Выбор типа графа

do {

printf("Выберите тип графа (1 - неориентированный, 2 - ориентированный): ");

while (scanf("%d%c", &graphType, &c) != 2 || c != '\n' || graphType < 1 || graphType > 2) {

printf("Неверный ввод. Пожалуйста, введите 1 или 2.\n");

while (getchar() != '\n'); // Очистка входного буфера

}

} while (graphType != 1 && graphType != 2);

// Выбор способа создания матрицы

do {

printf("Выберите способ создания матрицы (1 - случайная генерация, 2 - ввод вручную, 3 - из файла): ");

while (scanf("%d%c", &matrixCreationType, &c) != 2 || c != '\n' || matrixCreationType < 1 || matrixCreationType > 3) {

printf("Неверный ввод. Пожалуйста, введите число от 1 до 3.\n");

while (getchar() != '\n'); // Очистка входного буфера

}

} while (matrixCreationType < 1 || matrixCreationType > 3);

// Обработка выбора пользователя

if (matrixCreationType == 1) {

// Запрос количества вершин

do {

printf("Введите количество вершин в графе: ");

while (scanf("%d%c", &num\_vertices, &c) != 2 || c != '\n' || num\_vertices <= 0) {

printf("Неверный ввод. Пожалуйста, введите положительное целое число.\n");

while (getchar() != '\n');

}

} while (num\_vertices <= 0);

// Запрос вероятности наличия ребра

do {

printf("Введите вероятность наличия ребра в графе (0 - 1) через запятую: ");

while (scanf("%lf%c", &edge\_probability, &c) != 2 || c != '\n' || edge\_probability < 0 || edge\_probability > 1) {

printf("Неверный ввод. Введите число от 0 до 1 через запятую.\n");

while (getchar() != '\n');

}

} while (edge\_probability < 0 || edge\_probability > 1);

matrix = generateAndPrintMatrix(num\_vertices, edge\_probability, graphType);

}

else if (matrixCreationType == 2) {

do {

printf("Введите количество вершин в графе: ");

while (scanf("%d%c", &num\_vertices, &c) != 2 || c != '\n' || num\_vertices <= 0) {

printf("Неверный ввод. Пожалуйста, введите положительное целое число.\n");

while (getchar() != '\n');

}

} while (num\_vertices <= 0);

matrix.resize(num\_vertices + 1, vector<int>(num\_vertices + 1, 0));

// Ввод матрицы смежности вручную

for (int i = 1; i <= num\_vertices; ++i) {

for (int j = 1; j <= num\_vertices; ++j) {

if (graphType == 1 && i < j) {

do {

printf("Введите наличие ребра между вершинами %d и %d (0 - нет ребра или 1 - есть ребро): ", i, j);

if (scanf("%d%c", &matrix[i][j], &c) == 2 && (c == '\n') && (matrix[i][j] == 0 || matrix[i][j] == 1)) {

break; // Валидный ввод (0 или 1)

}

else {

printf("Неверный ввод. Введите 0 или 1.\n");

while (getchar() != '\n'); // Очистка буфера ввода

}

} while (true);

matrix[j][i] = matrix[i][j]; // Для неориентированных графов

}

else if (graphType == 2 && i != j) {

do {

printf("Введите наличие ребра от вершины %d к вершине %d (0 - нет ребра или 1 - есть ребро): ", i, j);

if (scanf("%d%c", &matrix[i][j], &c) == 2 && (c == '\n') && (matrix[i][j] == 0 || matrix[i][j] == 1)) {

break; // Валидный ввод (0 или 1)

}

else {

printf("Неверный ввод. Введите 0 или 1.\n");

while (getchar() != '\n'); // Очистка буфера ввода

}

} while (true);

}

else if (i == j) {

matrix[i][j] = 0; // Диагональные элементы должны быть нулями

}

}

}

printf("Введённая матрица смежности:\n");

for (int i = 1; i <= num\_vertices; ++i) {

for (int j = 1; j <= num\_vertices; ++j) {

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

}

printf("\n");

}

}

else if (matrixCreationType == 3) {

char filename[100];

printf("Введите имя файла: ");

scanf("%s", filename);

int num\_vertices; // Переменная для хранения числа вершин

bool isDirected = (graphType == 2); // Переменная для хранения информации об ориентированности матрицы

matrix = readMatrixFromFile(filename, num\_vertices, isDirected);

}

bool isDirected = (graphType == 2);

vector < set < int>> all\_independent\_sets = findAllIndependentSets(matrix, isDirected);

printAllIndependentSets(all\_independent\_sets);

// Проверка на сохранение результатов

int saveResultsChoice;

do {

printf("Хотите сохранить результаты? (1 - Да, 2 - Нет): ");

if (scanf("%d%c", &saveResultsChoice, &c) == 2 && c == '\n' && (saveResultsChoice == 1 || saveResultsChoice == 2)) {

break; // Валидный ввод

}

else {

printf("Неверный ввод. Введите 1 или 2.\n");

while (getchar() != '\n'); // Очистка буфера ввода

}

} while (true);

if (saveResultsChoice == 1) {

int saveFileChoice;

do {

printf("Сохранить в существующий файл (1) или создать новый (2)?: ");

if (scanf("%d%c", &saveFileChoice, &c) == 2 && c == '\n' && (saveFileChoice == 1 || saveFileChoice == 2)) {

break; // Валидный ввод

}

else {

printf("Неверный ввод. Введите 1 или 2.\n");

while (getchar() != '\n'); // Очистка буфера ввода

}

} while (saveFileChoice != 1 && saveFileChoice != 2);

char filename[100];

bool fileExists;

do {

printf("Введите имя файла для сохранения результатов: ");

scanf("%s", filename);

ifstream testFile(filename);

fileExists = testFile.good();

testFile.close();

if (saveFileChoice == 1 && !fileExists) {

printf("Файл %s не существует. Попробуйте снова.\n", filename);

}

else if (saveFileChoice == 2 && fileExists) {

printf("Файл %s уже существует. Введите другое имя файла.\n", filename);

}

} while ((saveFileChoice == 1 && !fileExists) || (saveFileChoice == 2 && fileExists));

ofstream file(filename, ios::out | ios::app); // Открывает файл для добавления в конец

if (file.is\_open()) {

saveMatrixToFile(matrix, filename, ios::out | ios::app);

saveSetsToFile(all\_independent\_sets, filename, ios::out | ios::app);

printf("Данные сохранены в файл %s\n", filename);

}

else {

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

}

}

break;

}

case 2: {

char filename[100];

printf("Введите имя файла для просмотра: ");

scanf("%s", filename);

viewPreviousResults(filename);

break;

}

case 3: {

char filename[100];

printf("Введите имя файла для очистки: ");

scanf("%s", filename);

ifstream testFile(filename);

if (testFile.good()) { // Проверка на существование файла

testFile.close(); // Закрываем файл после проверки

ofstream file(filename, ios::out | ios::trunc); // Открывает файл для записи с очисткой содержимого

if (file.is\_open()) {

printf("Файл %s очищен.\n", filename);

file.close();

}

else {

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

}

}

else {

printf("Файл %s не существует.\n", filename);

}

break;

}

case 4:

system("cls");

break;

case 5:

return 0;

}

}

}