Министерство науки и высшего образования Российской Федерации
Пензенский государственный университет
Кафедра «Вычислительная техника»

Пояснительная записка

к курсовой работе по курсу «Логика и основы алгоритмизации в инженерных задачах» на тему «Реализация алгоритма поиска независимых множеств вершин графа»

Выполнил:

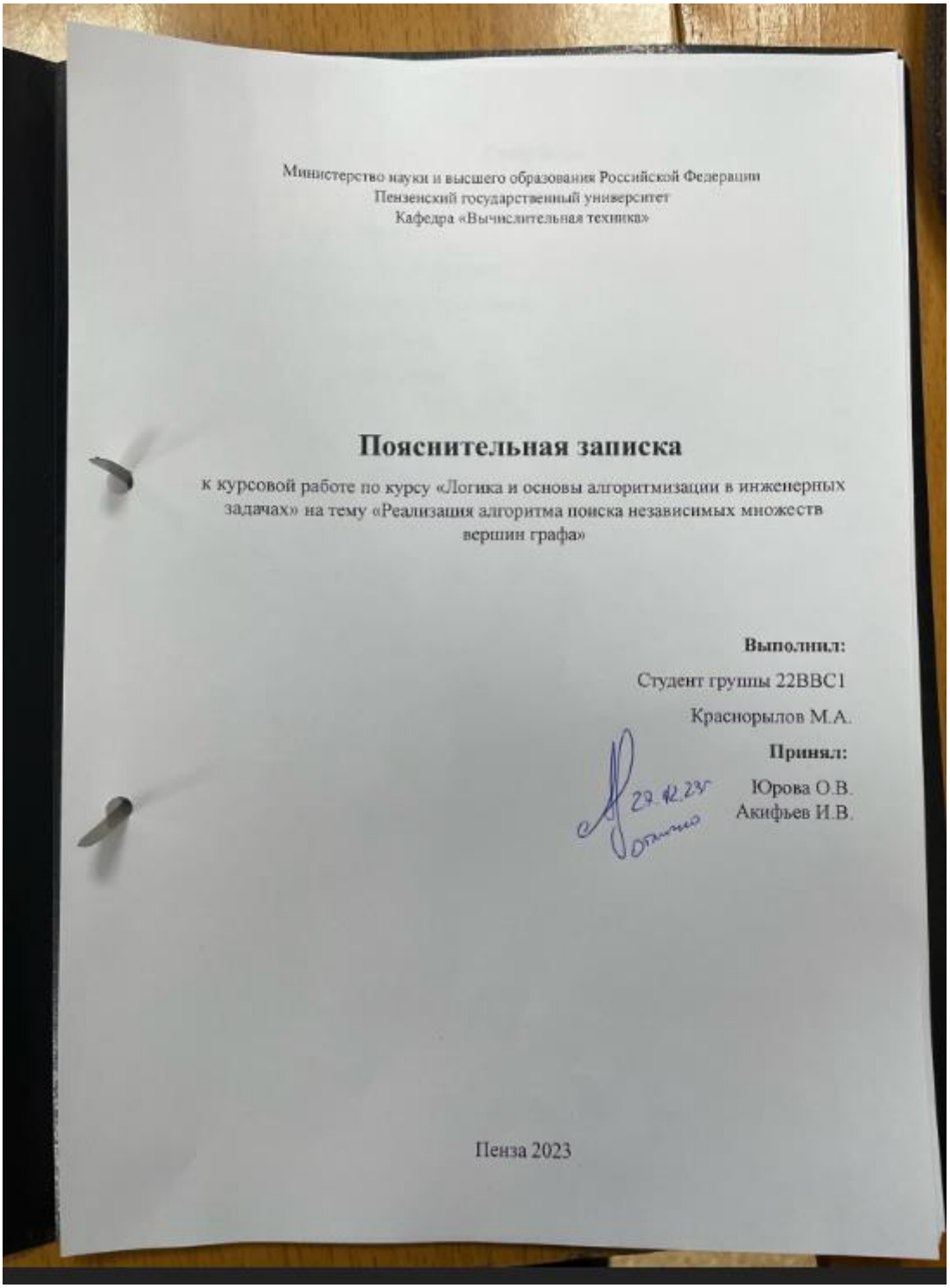
Студент группы 22ВВС1

Краснорылов М.А.

Принял:

Юрова О.В.
Акифьев И.В.

Пенза 2023



Министерство науки и высшего образования Российской Федерации
Пензенский государственный университет
Кафедра «Вычислительная техника»

Пояснительная записка

к курсовой работе по курсу «Логика и основы алгоритмизации в инженерных задачах» на тему «Реализация алгоритма поиска независимых множеств вершин графа»

Выполнил:

Студент группы 22ВВС1

Краснорылов М.А.

Принял:

29.12.23
Отлично

Юрова О.В.
Акифьев И.В.

Пенза 2023

ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
Факультет Вычислительной техники

Кафедра "Вычислительная техника"

"УТВЕРЖДАЮ"

Зав. кафедрой ВТ _____

«___»_____20__

ЗАДАНИЕ

на курсовое проектирование по курсу

"Логика и основы алгоритмизации в инженерных задачах"
Студенту Краснопольову Максиму Александровичу Группа 22ВВВ1
Тема_проекта Реализация алгоритма нахождения кратчайшего пути в графе

Исходные данные (технические требования) на проектирование

Разработка алгоритмов и программного обеспечения в соответствии с заданием текущего курсового проекта.
Пояснительная записка должна содержать:
1. Постановку задачи;
2. Теоретическую часть задания;
3. Описание алгоритма поставленной задачи
4. Функции ручного расчета задачи и вычислительного (описание работы алгоритма);
5. Описание основных программ;
6. Тесты;
7. Список литературы
8. Листинг программы
9. Результаты работы программы

Объем работы по курсу

1. Расчетная часть

Пример расчет работы алгоритма

2. Графическая часть

Схема алгоритма в программе блок-схем

3. Экспериментальная часть

Тестирование программы;
Результаты работы программы на тестовых данных

Срок выполнения проекта по разделам

1 Исследование предметной области задачи курсового
2 Разработка алгоритма программы
3 Тестовые программы
4 Тестирование и отладочные разработанной программы
5 Оформление пояснительной записки
6
7
8

Дата выдачи задания "6" октября 2023

Дата защиты проекта "___" _____

Руководитель Ярова Ольга Викторовна

Задание получил "26" сентября 2023 г.

Студент Краснослов Михаил Александрович

Содержание

Введение

Во многих прикладных задачах требуется найти в конечном множестве объектов систему объектов попарно не связанных друг с другом. Формулировки подобных задач на языке теории графов приводят к понятиям независимости.

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.

1. Постановка задачи

Создать программу, предоставляющую пользователю выбор между двумя способами ввода матрицы смежности графа: случайная генерация или генерация матрицы в файл и чтение из него.

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 и матрицы предков, всех найденных независимых множеств.

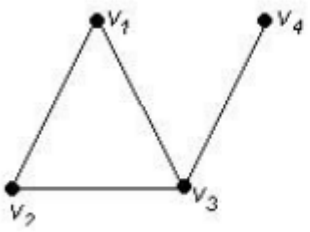
Предусмотреть возможность сохранения результатов работы программы в файл.

Обеспечить корректный ввод входных данных пользователем для корректной работы программы.

Разработать пользовательский интерфейс для ввода данных с клавиатуры и мыши.

2. Теоретическая часть задания

Граф G (рисунок 1) задается множеством вершин V1, V2, ..., Vn. и множеством ребер, соединяющих между собой определенные вершины.



$$\grave{A}=\begin{bmatrix} 0 & 1 & 1 & 0 \\ 1 & 0 & 1 & 0 \\ 1 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix}$$

Рисунок 1 – Пример графа, заданного матрицей смежности

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 в графе. Независимое множество в графе — это подмножество вершин, ни одна из которых не соединена рёбрами с другой вершиной в этом же множестве.

3. Описание алгоритма программы

Для программной реализации алгоритма понадобятся три матрицы: *a(int)* – являющаяся матрицей смежности, *vers(int)* – матрица предков построенная по матрице смежности, в нее записываются несвязные для каждой вершины, которые и нужно будет проверять, *visited(int),* отвечающий за посещение вершины, и вектор data использующийся для проверки связей. Итак, имеется граф *G = (V, E).* Каждая из вершин, входящих во множество V, изначально отмечена как не посещенная.

Для удобства восприятия мы проходим по графу начиная с нулевой вершины, проверяем наличие связей и записываем результат в матрицу предков, записывая только номера несвязных вершин.

Далее происходит обработка матрицы предков, мы проверяем все ли вершины независимы. В ходе проверки, при нахождении связи, в вектор записываем 1, если же связь не найдена 0. Если проверка показала хотя бы одну связь, в файл записывается пара независимых вершин. Чтобы была возможность выводить не только пары, используем *visited(int)*, в него записываем вершину, которую уже проверяли.

4. Описание программы

Для написания данной программы использован язык программирования Си. Язык программирования Си - универсальный язык программирования, который завоевал особую популярность у программистов, благодаря сочетанию возможностей языков программирования высокого и низкого уровней.

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

Работа программы начинается с запроса генерации матрицы. Пользователю предлагается выбрать количество вершин в графе. Затем нужно выбрать между автоматической генерацией матрицы и считыванием с файла.

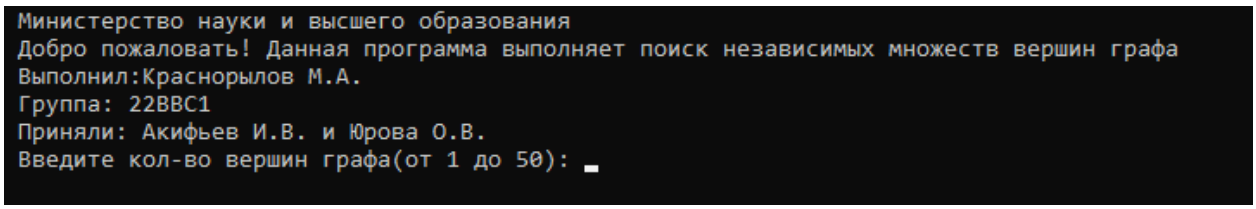


Рисунок 2 – Меню

На экран выводится матрица смежности.

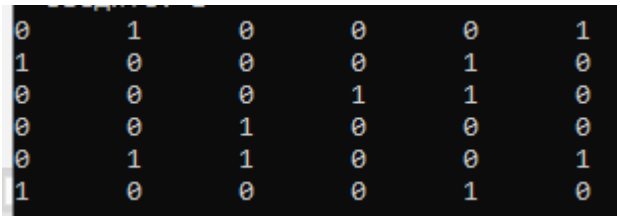


Рисунок 3 – Матрица

По матрице смежности мы генерируем матрицу предков.

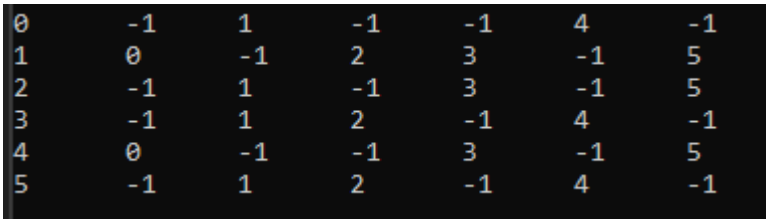


Рисунок 4 – Матрица предков

Для удобства первый столбик матрицы предков заполняем номерами вершин. Далее проходим по матрице a[i][j] и в случае отсутствия связи, заносим номер вершины в матрицу предков.

Каждую вершину, записанную в матрицу предков, проверяем на связь с элементами, находящимися в той же строчке, для корректного вывода помечаем вершину посещенной.

Чтобы проверить независимость вершин используем вектор, в случае наличия связи записываем в вектор 1, иначе 0. Если в векторе найдется хотя бы одна 1, вывод в файл произведется попарно(например 02)

5. Тестирование

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

Тестирование проводилось в рабочем порядке, в процессе разработки, после завершения написания программы. В ходе тестирования было выявлено и исправлено множество проблем, связанных с вводом данных, изменением дизайна выводимых данных, алгоритмом программы, взаимодействием функций.

Ниже продемонстрирован результат тестирования программы при вводе пользователем различных количеств вершин и вывода результата.

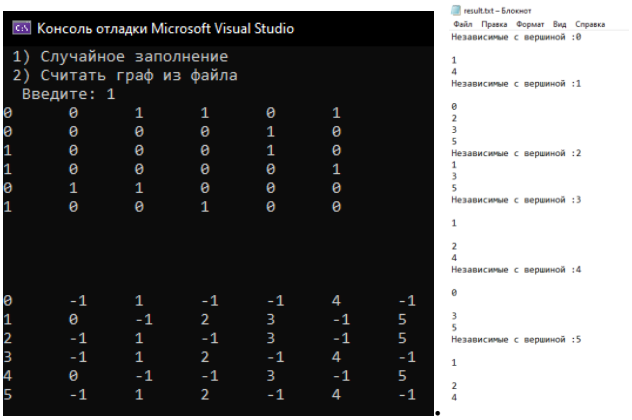


Рисунок 5 – Тестирование при случайном заполнении графа при 6 вершинах

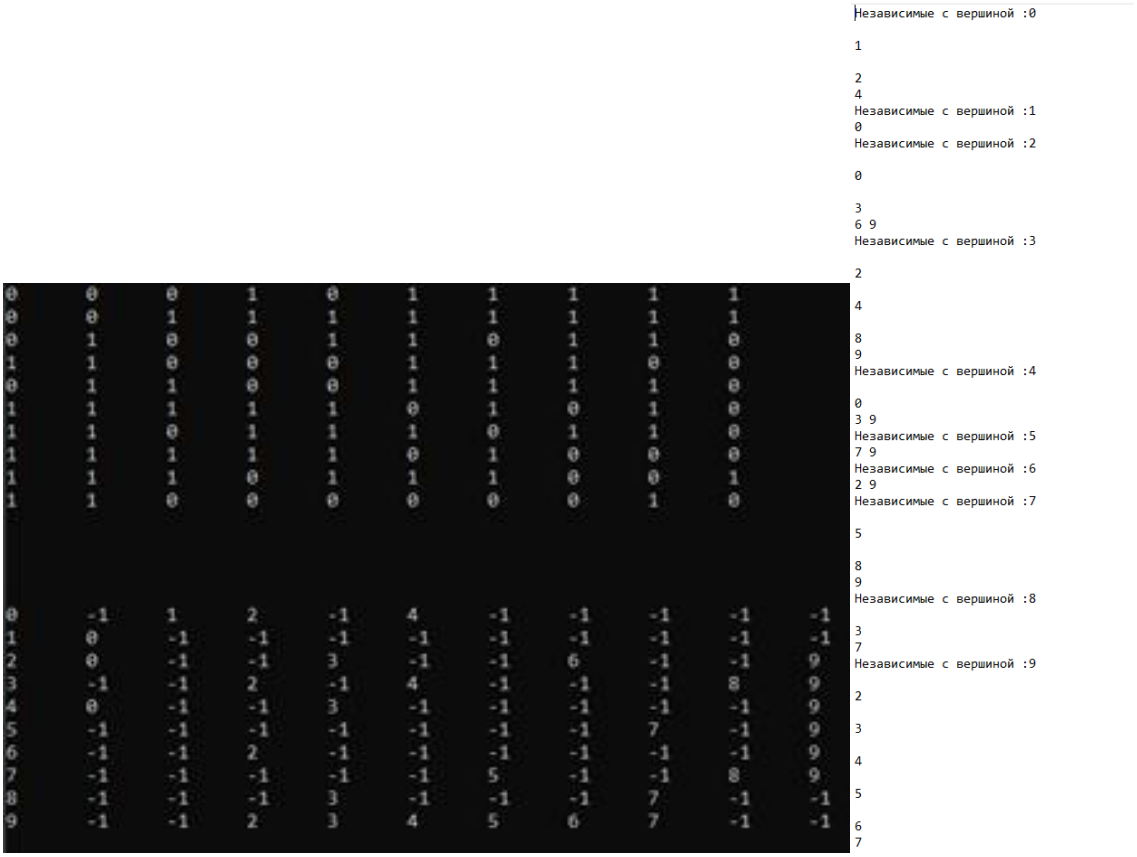


Рисунок 6 – Тестирование при случайном заполнении графа при 10 вершинах

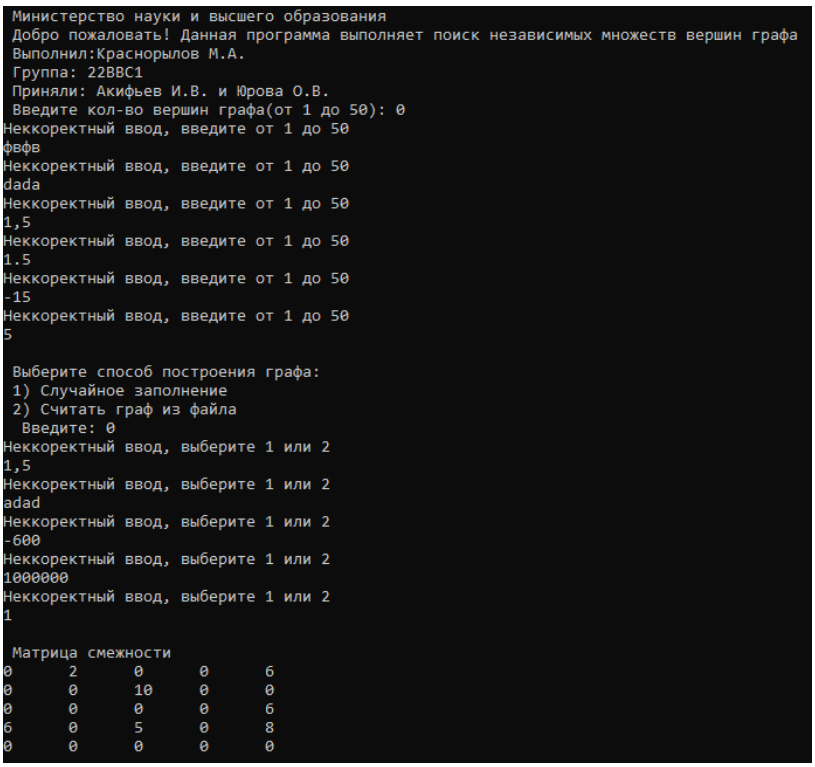


Рисунок 7 – Тестирование при различных вариациях некорректного ввода

Таблица 1 – Таблица тестирования

Описание теста	Ожидаемый результат	Полученный результат
Запуск программы	Вывод сообщения о вводе кол-ва вершин	Верно
Выбор генерации матрицы	Вывод меню программы и выбор пользователем нужного пункта	Верно
Заполнение матрицы случайными числами	Вывод матрица смежности	Верно
Заполнение матрицы предков	Проверка отсутствия связей в матрице смежностей	Верно
Проверка на независимость вершин	Нахождение множества независимых вершин	Верно
Вывод в файл	Корректная запись в файл	Верно

В результате тестирования было выявлено, что программа успешно выдает результат.

6. Ручной просчет задачи

Проведем проверку программы посредством ручного просчета на примере графа с 6 вершинами. (Рисунок 8). Для наглядности была создана модель.

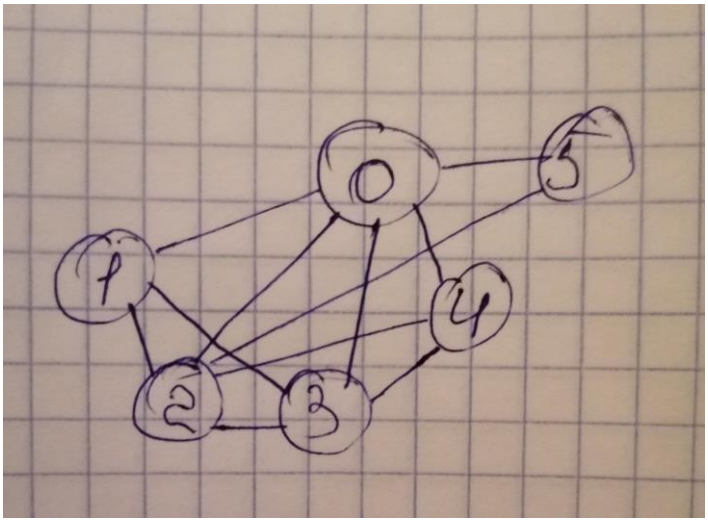


Рисунок 8 – Модель графа

```
0     1     1     1     1     1
1     0     1     1     0     0
1     1     0     1     1     1
1     1     1     0     1     0
1     0     1     1     0     0
1     0     1     0     0     0



0     -1    -1    -1    -1    -1    -1
1     -1    -1    -1    -1    4     5
2     -1    -1    -1    -1    -1    -1
3     -1    -1    -1    -1    -1    5
4     -1    1     -1    -1    -1    5
5     -1    1     -1    3     4     -1
```

Рисунок 9 – Матрицы по графу

Мы сразу видим, что вершины 0 и 2 связаны со всеми вершинами, так что они не будут входить в множество.

Проанализировав оставшиеся связи, мы можем сделать вывод, что складываются такие множества: 145, 35, 51, 53, 54.

Что мы и видим из результата.

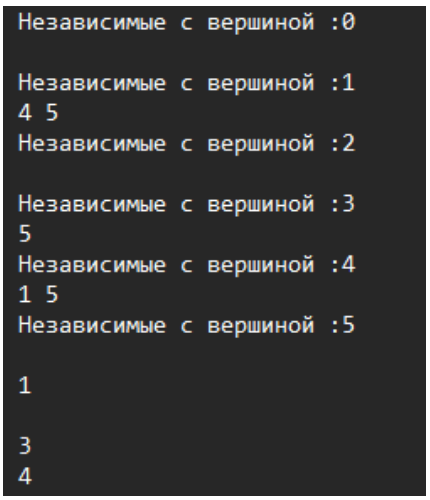


Рисунок 10 – Проверка с записанным результатом

Заключение

Во время создания данного проекта была разработана программа реализующая алгоритм поиска независимого множества вершин в графе в Microsoft Visual Studio 2022.

При выполнении данной работы были получены навыки разработки многомодульных программ и освоены приемы создания матриц смежностей, основанных на теории орграфов. Углублены знания языка программирования Си/Си++. Были улучшены навыки отладки больших работ в среде Microsoft Visual Studio.

Список литературы

1.Кристофидес Н. «Теория графов. Алгоритмический подход» - Мир, 1978, 296 с.

2. Зыков А.А. Основы теории графов. - М.:Наука, 1987, 384 с.

3. Мельников О.И. Теория графов в занимательных задачах. Изд.3, испр. и доп. 2009. 232 с.

4. Динман М.И. С++. Освой на примерах. – СПБ.: БХВ – Петербург, 2006 – 384 с.

5. Харари Ф. Теория графов / Пер.с англ. и предисл. В. П. Козырева. Под ред. Г. П. Гаврилова. Изд. 2-е. - М.: Едиториал УРСС, 2003. - 296 с.

Листинг программы

```
#define _CRT_SECURE_NO_WARNINGS
#include<vector>
#include<list>
#include<stack>
#include <sstream>
#include <fstream>
#include <stdio.h>
#include <iostream>
#include <stdlib.h>
#include "string"
#include <locale.h>
#include <windows.h>
using namespace std;
int* visited;
int** a;
int n = -1;
int main()
{
	COORD coord;
	SetConsoleDisplayMode(GetStdHandle(STD_OUTPUT_HANDLE), CONSOLE_FULLSCREEN_MODE, &coord);
	srand(time(0));
	int s = 0;
	string mes;
	setlocale(LC_ALL, "rus");
	FILE* file;
	FILE* some;
	int temp;
	system("cls");
	printf_s(" Министерство науки и высшего образования\n Добро пожаловать! Данная программа выполняет поиск
независимых множеств вершин графа\n Выполнил:Краснорылов М.А.\n Группа: 22ВВС1\n Приняли: Акифьев И.В. и Юрова О.В.\n");
	int i;
	while (true) { // бесконечный цикл
		printf_s(" Введите кол-во вершин графа(от 1 до 50): ");
		int n;
		while (!(cin >> n) || (cin.peek() != '\n') || n < 1 || n > 50)
		{
			cin.clear();
			while (cin.get() != '\n');
			cout << "Неккоректный ввод, введите от 1 до 50\n";
		}

		int** unchain;
		unchain = (int**)malloc(sizeof(int*) * n);
		for (int i = 0; i < n; i++)
		{
			unchain[i] = (int*)malloc(sizeof(int) * n);
			for (int j = 0; j < n; j++) {
				unchain[i][j] = -10;
			}
		}
		int* many = (int*)malloc(sizeof(int) * n);

		a = (int**)malloc(sizeof(int*) * n);//выделение памяти вод указатели
		for (int i = 0; i < n; i++)
		{
			a[i] = (int*)malloc(sizeof(int) * n);//выделение памяти под массив
		}

		printf_s("\n Выберите способ построения графа: \n 1) Случайное заполнение \n");
		printf_s(" 2) Считать граф из файла \n  Введите: ");
		while (!(cin >> s) || (cin.peek() != '\n') || s < 1 || s > 2)
		{
			cin.clear();
			while (cin.get() != '\n');
			cout << "Неккоректный ввод, выберите 1 или 2\n";
		}

		printf_s("\n");
		if (s == 1)
		{
			printf_s(" Матрица смежности ориентированного графа\n");
			for (int i = 0; i < n; i++)
			{

				for (int j = i; j < n; j++)
```

```
                {
                        a[i][j] = rand() % 3;//есть ребро или нет вероятность подкуручена чтобы более заполенный граф формировался

                        if (a[i][j] > 0)
                        {
                                temp = rand() % 2; //направление ребра
                                if (temp > 0) {
                                        a[i][j] = (rand() % 10) + 1;//вес ребра от 1 до 10
                                        a[j][i] = 0;
                                }
                                else {
                                        a[j][i] = (rand() % 10) + 1;
                                        a[i][j] = 0;
                                }
                        }
                        else {
                                a[i][j] = 0;
                                a[j][i] = 0;
                        }
                }
        }

        for (int i = 0; i < n; i++)
        {
                for (int j = 0; j < n; j++)
                {
                        printf_s("%-5d  ", a[i][j]);
                }
                printf_s("\n");
        }
}
if (s == 2)
{
        file = fopen("matrix.txt", "w");
        for (int i = 0; i < n; i++)
        {
                for (int j = i; j < n; j++)
                {
                        a[i][j] = rand() % 2;//есть ребро или нет
                        if (a[i][j] == 1)
                        {
                                temp = rand() % 2; //направление ребра
                                if (temp == 1) {
                                        a[i][j] = (rand() % 10) + 1;//вес ребра
                                        a[j][i] = 0;
                                }
                                else {
                                        a[i][j] = 0;
                                        a[j][i] = (rand() % 10) + 1;
                                }
                        }
                        else {
                                a[i][j] = 0;
                                a[j][i] = 0;
                        }
                        a[i][j] = 0;//обнуление главной диагонали
                }
        }
        for (int i = 0; i < n; i++) {
                for (int j = 0; j < n; j++) {
                        fprintf_s(file, "%-5d  ", a[i][j]);
                }
                fprintf_s(file, "\n");
        }
        fclose(file);

        file = fopen("matrix.txt", "r");
        if (!file)
                exit(EXIT_FAILURE);
        for (int i = 0; i < n && !feof(file); i++) {
                for (int j = 0; j < n && !feof(file); j++) {
                        fscanf(file, "%d", &a[i][j]);
                        printf_s("%-5d ", a[i][j]);
                }
                putchar('\n');
        }
        fclose(file);
```

```
}
printf_s("\n");
printf_s("\n");
printf_s("\n");
printf_s("\n");

int** vers = (int**)malloc(n * sizeof(int*));//выделение памяти под указатели(матрицу предков)

for (int i = 0; i < n; i++)
{
	vers[i] = (int*)malloc(n * sizeof(int));//выделение памяти под значения(матрицы предков)
}
for (int i = 0; i < n; i++)
{
	for (int j = 0; j <= n; j++)
	{
		vers[i][j] = -1;//инициализация значений в матрице
	}

}
for (int i = 0; i < n; i++)
{
	vers[i][0] = i;//нумерация первого столбца
}
int ch = 0;
for (int i = 0; i < n; i++)
{
	for (int j = 0; j <= n; j++)
	{

		if (a[i][j] == 0 && i != j)//если мы находим в таблице смежности ноль и это не главная диагональ(находим не связанные вершины)
		{
			if (a[i][j] == 0 && a[j][i] == 0)
			{
				vers[i][j + 1] = j;//то значение ячейки матрицы предков равно номеру столбца начиная с первого(не с нулевого)----(сохраняем что данная вершина i не связана с вершиной j)
			}
		}
	}
}
printf_s(" Матрица предков\n наличие связи - 1\n отсутствие связи - номер вершины \n");
for (int i = 0; i < n; i++)
{
	for (int j = 0; j <= n; j++)
	{
		printf_s("%-5d  ", vers[i][j]);//вывести матрицу предков

	}
	printf_s("\n");
}
visited = (int*)malloc(n * sizeof(int));//выделить память под массив посещение
for (int i = 0; i < n; i++)
{
	visited[i] = 0;//указать что ни одна еще не посещена
}
FILE* fp;
char name[] = "result.txt";
fp = fopen(name, "w");
vector<int>data;
int summ = 0;
for (int i = 0; i < n; i++)
{
	for (int k = 0; k < n; k++)
	{
		visited[k] = -1;//сбросить посещаемость
	}
	fprintf_s(fp, "Независимые с вершиной :%d\n", vers[i][0]);
	printf_s("Независимые с вершиной :%d\n", vers[i][0]);
	for (int j = 0; j < n; j++)
	{
		if (vers[i][j + 1] >= 0)//Проверка на отрицательные значения(нетронутые при генерации матрицы предков значения)
		{
			ch = vers[i][j + 1];//запись проверяемого значения в временную переменную
			visited[j] = ch;//отметка о посещении вершины

			for (int k = 0; k < n; k++)//обход каждой ячейки в ряде матрицы предков
```

```
                                        {
                                                int p = vers[i][k + 1];
                                                if (p >= 0 && ch != p && a[ch][p] != 1 && visited[k] == -1)
                                                {
                                                        data.push_back(0);//если 2 вершины Ср и р (vers[i][j + 1] и
vers[i][k + 1]) не связаны то закинуть в конец вектора 0
                                                }
                                                if (p >= 0 && ch != p && a[ch][p] == 1 && visited[k] == -1)
                                                {
                                                        data.push_back(1);//если связаны по матрице смежности то
закинуть в конец вектора 1
                                                }
                                        }
                                        for (int k = 0; k < data.size(); k++)
                                        {
                                                if (data[k] == 1)
                                                {
                                                        summ++;//подсчет единиц в векторе
                                                }
                                        }
                                        if (summ == 0)
                                        {
                                                printf_s("%d ", ch);
                                                unchain[vers[i][0]][j] = ch;
                                                fprintf_s(fp, "%d ", ch);//вывод в файл полученные множества
                                        }
                                        else {
                                                printf_s("\n%d \n", ch);
                                                unchain[vers[i][0]][j] = ch;
                                                fprintf_s(fp, "\n%d \n", ch);
                                        }
                                        summ = 0;
                                }
                                data.clear();//очистить вектор
                        }
                        fprintf_s(fp, "\n");
                        printf_s("\n");
                }
                data.clear();//очистить вектор
                //for (int i = 0; i < n; i++) //вывод массива
                //      {

                //              for (int j = 0; j < n; j++) {
                //                      printf("%d\t", unchain[i][j]);
                //              }
                //              printf("\n");
                //      }
                int tik;
                bool flaglocal = false;
                bool flaglobal = true;
                for (int t = 0; t < n; t++) {
                        int tak = 1;
                        data.push_back(t);
                        for (int i = 0; i < n; i++) {
                                for (int j = 0; j < n; j++) {
                                        if (unchain[i][j] != -10) {
                                                flaglobal = true;
                                                tik = tak;
                                                while (tik != 0) {
                                                        flaglocal = false;
                                                        for (int k = 0; k < n; k++) {
                                                                if (unchain[unchain[i][j]][k] == data[tik - 1]) {
                                                                        flaglocal = true;
                                                                        break;
                                                                }
                                                        }
                                                        flaglobal = flaglobal && flaglocal;
                                                        tik--;
                                                }
                                                if (flaglobal) {
                                                        data.push_back(unchain[i][j]);
                                                        tak++;
                                                }
                                        }
                                }
                        }

                        if (data.size() != 1) {
```

```
                                        if (data.size() > 2) {
                                                for (int i = 0; i < data.size(); i++) {
                                                        for (int j = i; j < data.size(); j++) {
                                                                if (i != j) {
                                                                        printf("Подмножество %d %d\n", data[i], data[j]);
                                                                }
                                                        }
                                                }
                                        }
                                        printf("Множество\n");
                                        while (!data.empty()) {

                                                printf("%d ", data[tak - 1]);
                                                data.pop_back();
                                                tak--;
                                        }
                                }
                                printf("\n");
                                data.clear();
                        }
                        cout << " Выполнить программу еще раз? (1 - да, 2 - нет)\n";
                        while (!(cin >> i) || (cin.peek() != '\n') || i < 1 || i > 2)
                        {
                                cin.clear();
                                while (cin.get() != '\n');
                                cout << "Неккоректный ввод, выберите 1 или 2\n";
                        }
                        if (i != 1) break;
                        system("cls"); // очищаем экран
                }
}
```