

**Реферат**

Отчет 25 страниц

ГРАФ, ТЕОРИЯ ГРАФОВ, ДВУДОЛЬНЫЙ ГРАФ, ОБХОД В ГЛУБИНУ, ПОИСК В ГЛУБИНУ

Цель исследования – разработка программы, способной находить наибольшее паросочетание в двудольном графе, используя алгоритм обхода в глубину.

В работе рассмотрены правила нахождения наибольшего паросочетания, добавлена функция чтения из файла исходных данных, а также записи в файл конечных значений.

Оглавление

[Введение 3](#_Toc122536945)

[1. Теоретическая часть 4](#_Toc122536946)

[1.1. Постановка задачи 4](#_Toc122536947)

[2. Описание алгоритма 5](#_Toc122536948)

[2.1. Описание алгоритма поиска в глубину 5](#_Toc122536949)

[3. Практическая часть 6](#_Toc122536950)

[3.1. Выбор среды 6](#_Toc122536951)

[3.2. Описание функций и классов 6](#_Toc122536952)

[3.3. Пример ручного расчета задачи 8](#_Toc122536952)

[4. Тесты 10](#_Toc122536953)

[5. Список литературы 13](#_Toc122536954)

[6. Листинг программы 14](#_Toc122536955)

[7. Результаты работы программы 25](#_Toc122536955)

# Введение

В [теории графов](https://ru.wikipedia.org/wiki/%D0%A2%D0%B5%D0%BE%D1%80%D0%B8%D1%8F_%D0%B3%D1%80%D0%B0%D1%84%D0%BE%D0%B2) паросочетание или независимое множество рёбер в графе — это набор попарно несмежных рёбер.

Максимальное паросочетание – это такое паросочетание *M* в графе *G*, которое не содержится ни в каком другом паросочетании этого графа, то есть к нему невозможно добавить ни одно ребро, которое бы являлось несмежным ко всем рёбрам паросочетания. Другими словами, паросочетание *M* графа *G* является максимальным, если любое ребро в *G* имеет непустое пересечение, по крайней мере, с одним ребром из *M*. Ниже приведены примеры максимальных паросочетаний (красные рёбра) в трёх графах.

Наибольшее паросочетание – это такое паросочетание, которое содержит максимальное количество рёбер. Число паросочетания графа – это число рёбер в наибольшем паросочетании. У графа может быть множество наибольших паросочетаний. При этом любое наибольшее паросочетание является максимальным, но не любое максимальное будет наибольшим.

В данной курсовой работе необходимо составить программу, которая будет наибольшее паросочетание в двудольном графе.

В качестве среды разработки мною была выбрана среда MicrosoftVisualStudio 2019, язык программирования – С++. В ходе выполнения данной курсовой работы были приобретены навыки работы с WinApi в среде MicrosoftVisualStudio 2019, навыки работы с проектами и многомодульными программами.

# Теоретическая часть

# Постановка задачи

Задание для курсового проектирования заключается в создании программы «Нахождение наибольшего паросочетания в двудольном графе». Программа должна выполнять следующие функции: генерация графа, графическое отображение на экране, должна осуществлять поиск всех паросочтений и производить выборку паросочетания с максимальным числом ребер.

# Описание алгоритма

Для реализации программы были использованы алгоритмы работы поиска в глубину.

# Описание алгоритма поиска в глубину

Стандартная реализация поиска в глубину помещает каждую вершину графа в одну из двух категорий:

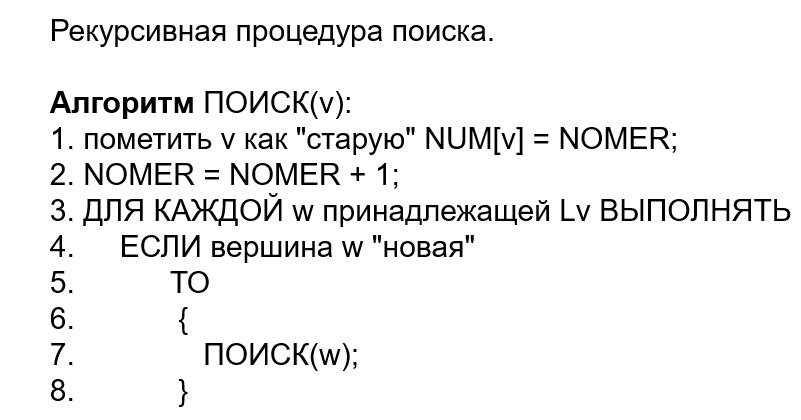
1. Посещенные.
2. Не посещенные.

Цель алгоритма - пометить каждую вершину, как посещенную, избегая циклов.

Алгоритм поиска в глубину работает следующим образом:

1. Пойти в какую-нибудь смежную вершину, не посещенную ранее.
2. Запустить из этой вершины алгоритм обхода в глубину
3. Вернуться в начальную вершину.
4. Повторить пункты 1-3 для всех не посещенных ранее смежных вершин.

Псевдокод алгоритма:



# Практическая часть

# Выбор среды

Для удовлетворения требований задачи был выбрал язык C и C++. C/C++  — [компилируемый](https://ru.wikipedia.org/wiki/%D0%9A%D0%BE%D0%BC%D0%BF%D0%B8%D0%BB%D0%B8%D1%80%D1%83%D0%B5%D0%BC%D1%8B%D0%B9_%D1%8F%D0%B7%D1%8B%D0%BA_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D1%8F), [статически типизированный](https://ru.wikipedia.org/wiki/%D0%A1%D1%82%D0%B0%D1%82%D0%B8%D1%87%D0%B5%D1%81%D0%BA%D0%B0%D1%8F_%D1%82%D0%B8%D0%BF%D0%B8%D0%B7%D0%B0%D1%86%D0%B8%D1%8F) [язык программирования](https://ru.wikipedia.org/wiki/%D0%AF%D0%B7%D1%8B%D0%BA_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D1%8F) общего назначения. Язык имеет богатую стандартную библиотеку, которая включает в себя распространённые контейнеры и алгоритмы, ввод-вывод, регулярные выражения, поддержку многопоточности и другие возможности. C++ сочетает свойства как [высокоуровневых](https://ru.wikipedia.org/wiki/%D0%92%D1%8B%D1%81%D0%BE%D0%BA%D0%BE%D1%83%D1%80%D0%BE%D0%B2%D0%BD%D0%B5%D0%B2%D1%8B%D0%B9_%D1%8F%D0%B7%D1%8B%D0%BA_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D1%8F), так и [низкоуровневых языков](https://ru.wikipedia.org/wiki/%D0%9D%D0%B8%D0%B7%D0%BA%D0%BE%D1%83%D1%80%D0%BE%D0%B2%D0%BD%D0%B5%D0%B2%D1%8B%D0%B9_%D1%8F%D0%B7%D1%8B%D0%BA_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D1%8F). В сравнении с его предшественником — языком [C](https://ru.wikipedia.org/wiki/%D0%A1%D0%B8_(%D1%8F%D0%B7%D1%8B%D0%BA_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D1%8F)), — наибольшее внимание уделено поддержке [объектно-ориентированного](https://ru.wikipedia.org/wiki/%D0%9E%D0%B1%D1%8A%D0%B5%D0%BA%D1%82%D0%BD%D0%BE-%D0%BE%D1%80%D0%B8%D0%B5%D0%BD%D1%82%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%BD%D0%BE%D0%B5_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5) и [обобщённого программирования](https://ru.wikipedia.org/wiki/%D0%9E%D0%B1%D0%BE%D0%B1%D1%89%D1%91%D0%BD%D0%BD%D0%BE%D0%B5_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5).

C/C++ широко используется для разработки программного обеспечения, являясь одним из самых популярных языков программирования. Область его применения включает создание [операционных систем](https://ru.wikipedia.org/wiki/%D0%9E%D0%BF%D0%B5%D1%80%D0%B0%D1%86%D0%B8%D0%BE%D0%BD%D0%BD%D0%B0%D1%8F_%D1%81%D0%B8%D1%81%D1%82%D0%B5%D0%BC%D0%B0), разнообразных прикладных программ, [драйверов](https://ru.wikipedia.org/wiki/%D0%94%D1%80%D0%B0%D0%B9%D0%B2%D0%B5%D1%80) устройств, приложений для встраиваемых систем, высокопроизводительных серверов, а также развлекательных приложений (игр). Существует множество реализаций языка C++, как бесплатных, так и коммерческих и для различных платформ. Например, на платформе x86 это [GCC](https://ru.wikipedia.org/wiki/GNU_Compiler_Collection), [Visual C++](https://ru.wikipedia.org/wiki/Visual_C%2B%2B), [Intel C++ Compiler](https://ru.wikipedia.org/wiki/Intel_C%2B%2B_Compiler), [Embarcadero (Borland) C++ Builder](https://ru.wikipedia.org/wiki/Embarcadero_C%2B%2B_Builder) и другие.

Синтаксис C++ унаследован от языка [C](https://ru.wikipedia.org/wiki/%D0%A1%D0%B8_(%D1%8F%D0%B7%D1%8B%D0%BA_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D1%8F)). Одним из принципов разработки было сохранение совместимости с C. Тем не менее, C++ не является в строгом смысле надмножеством C; множество программ, которые могут одинаково успешно транслироваться как [компиляторами](https://ru.wikipedia.org/wiki/%D0%9A%D0%BE%D0%BC%D0%BF%D0%B8%D0%BB%D1%8F%D1%82%D0%BE%D1%80) C, так и компиляторами C++, довольно велико, но не включает все возможные программы на C.

# Описание функций и классов

Для удобства и простоты пользования программой, а также для обеспечения наибольшего быстродействия разделим программу на 11 функций.

1. Функция makeGraf() генерирует массив матриц смежности для исходного неориентированного двудольного графа. Хранится в динамическом массиве. При создании граф проверяется на изолированные вершины. Если изолированная вершина найдена, то происходит повторная генерация для этой вершины.
2. Функция makeTransPut() создает массив матрицы смежности для ориентированного графа, который представлен не взвешенной транспортной сетью (максимальный вес ребра равен 1).
3. Функция makeCoord() создает массивы, которые хранят координаты вершин исходного графа и графа транспортной сети. По этим координатам происходит графическая отрисовка и анимация поиска.
4. Функция drawPoint () отрисовываем окружности вершин с помощью функции Ellipse() по координатам взятых из массива координат вершин. Сначала рисуется черный круг, который представляет собой границу круга, после уже отрисовывается цветной круг.
5. Функция drawLine() рисует с помощью LineTo, MoveTo линии между 2-мя указанными вершинами, либо всеми инцидентными ребрами этой вершины. При прорисовке ориентированного графа к линии добавляется стрелка, которая показывает направление ребра.
6. Функция searchDepth() подготавливает данные к поиску и выводит результат. Обнуляется матрицы смежности и счетчик количества паросочетаний.
7. Функция searchDepthRec() выполняет рекурсивный поиск в глубину. Поиск всегда начинается с истока (0-ой вершины) вершины, заканчивается в стоке (последняя вершина). Цикл выполняется, пока есть вершины, смежные истоку.
8. Функция resetMatrix() обнуляет массив, который передан как входные параметры. Заносит «0» во все элементы массива.
9. Функция makeMatrixAdjac() копирует один массив в другой. В первом массиве находится матрица двудольного смежности графа, которая задана следующим образом: строки массива – это вершины левой части двудольного графа, а столбцы – вершины правой части. Пересечение строк и столбцов – это ребро между вершинами. Во втором массиве расположена матрица смежности, у которой количество строк и столбцов одинаковое, и равное общему числу вершин в графе. В функции происходит копирование первого массива во второй со смещением, чтобы не изменить структуру графа.
10. Функция outPutResult() выводит результат работы программы: число максимальных паросочетаний и матрицу смежности,
11. Функция printMus(int\* ptr, char Size1, char Size2, HDC hdc, int x, int y, char a, char b) печатает массив, которые подается как входной параметр. (ptr = указатель на массив, Size - кол-во строк и столбцов, x и y - смещение относительно (0,0), a и b - индексы элемента в массиве, который необходимо выделить красным)

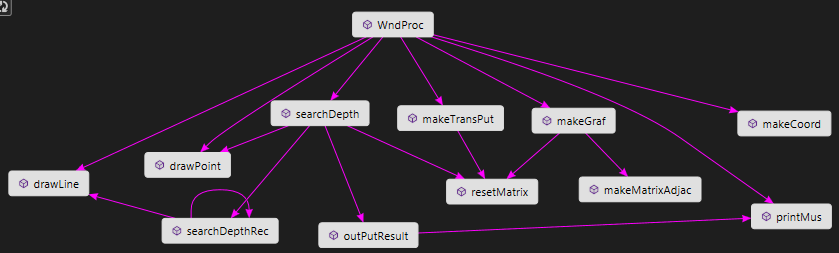


Рисунок 1. График вызовов функций

Из оконной процедуры WndProc() вызываются основные функции: makeGraf(), makeTransPut(), makeCoord() и searchDepth(). А также второстепенные: printMus(), drawPoint() и drawLine().

Функции searchDepth() , makeGraf() и makeTransPut() вызывают resetMatrix().

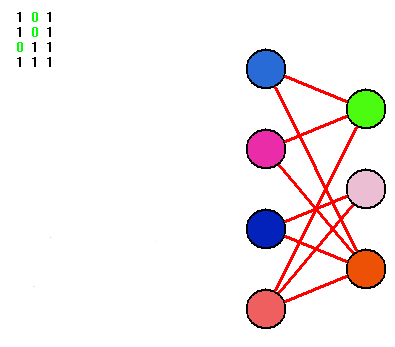
Функция makeGraf() использует makeMatrixAdjac(), для копирование матриц смежносте й.

Из функции searchDepth() вызываются функции searchDepthRec(), drawPoint, outPutResult(), а также рекурсивно вызывает себя.

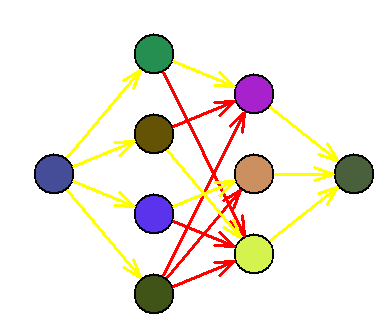
Функция outPutResult() вызывает printMus(), для вывода результата работы программы на экран.

* 1. **Пример ручного расчета задачи**

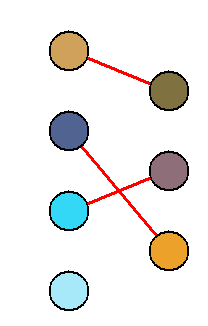
Проведем проверку работы программы ручным расчетом на примере графа с 3 вершинами слева и справа. Для простоты понимания приведем графическое представление графа вместе с матрицей смежности.



Исходя из этого перестроим граф под нашу задачу, найдя все известные нам пути:



Отсюда видно, что максимальное количество единичных путей – 3, соответственно наибольшее паросочтение будет так же 3.

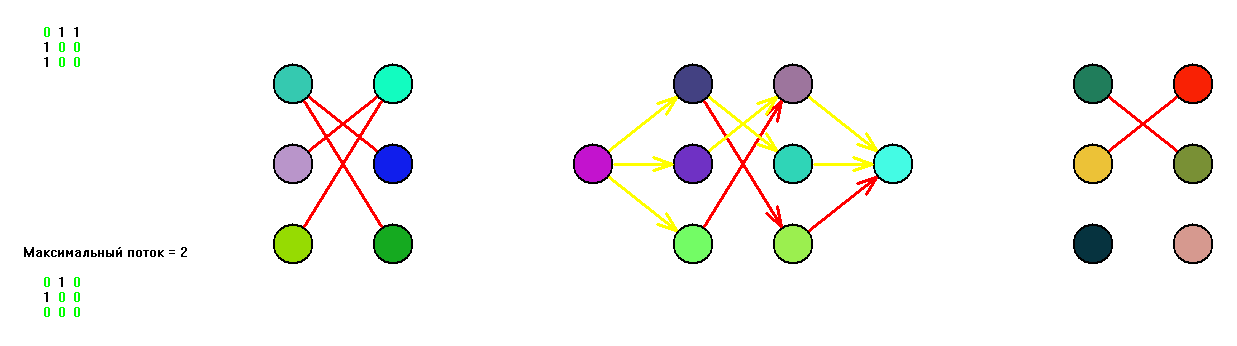
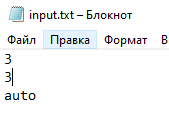
 

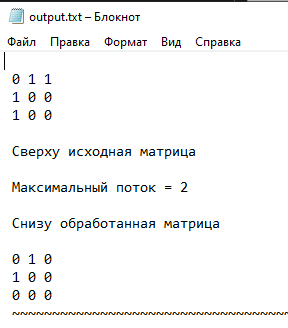
# Тесты

Тестирование программного обеспечения – процесс испытания программного продукта, имеющий следующие цели:

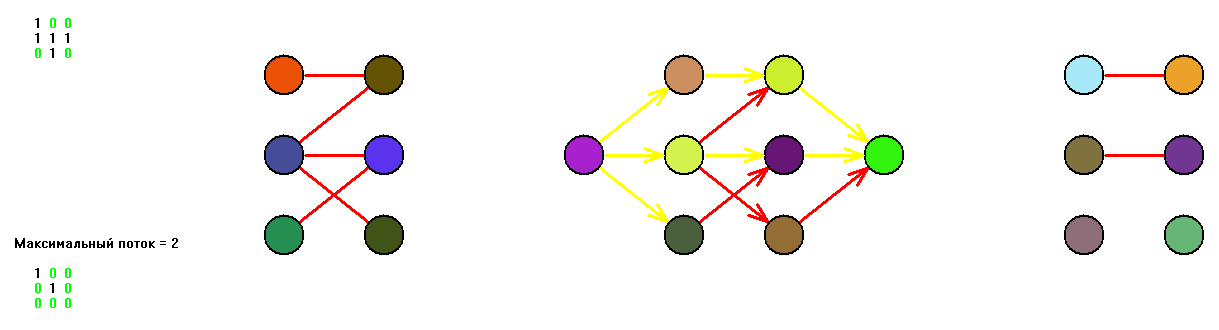
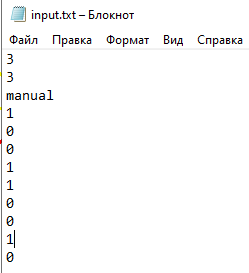
* Обнаружение дефектов;
* Повышение уверенности в уровне качества;
* Предоставление информации для принятия решений;
* Предотвращение дефектов;
* Продемонстрировать разработчикам и заказчикам, что программа соответствует требованиям.

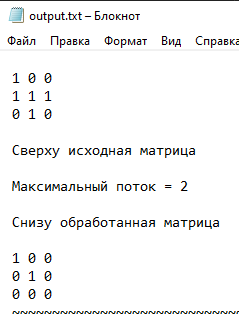
**Тестирование для 3 вершин слева и справа с генерированием:**

****

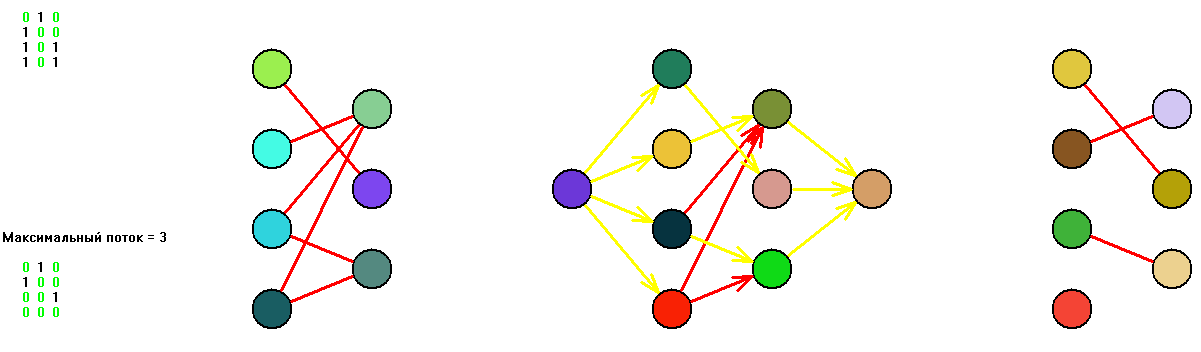
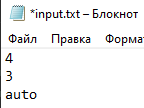
****

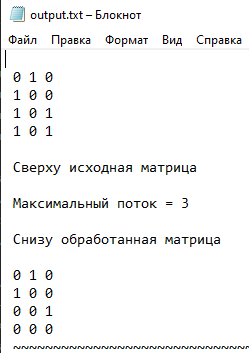
**Тестирование для 3 вершин слева и справа с ручным вводом:**

****

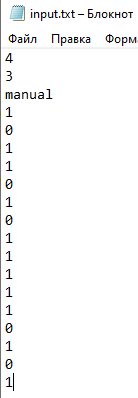
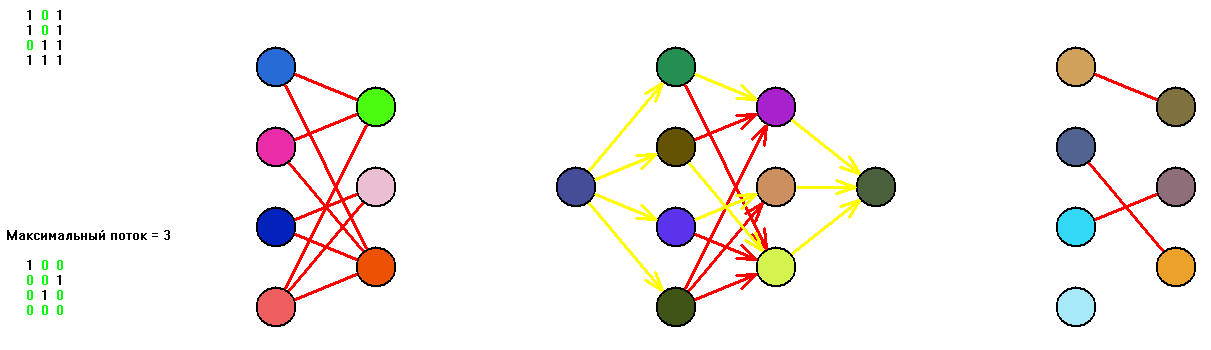
****

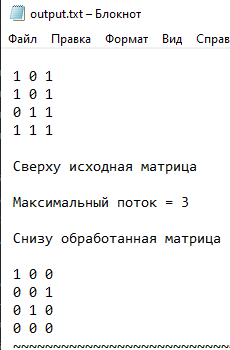
**Тестирование для 4 вершин слева и 3 справа с генерированием:**

****

****

**Тестирование для 4 вершин слева и 3 справа с ручным вводом:**



|  |  |  |
| --- | --- | --- |
| Описание теста | Ожидаемый результат | Полученный результат |
| Запуск программы с 3 вершинами слева и справа с автогенерированием значений | Вывод наибольшего паросочетания, вывод графической части и конечной матрицы смежности | Верно |
| Запуск программы с 3 вершинами слева и справа с ручным вводом значений | Вывод наибольшего паросочетания, вывод графической части и конечной матрицы смежности | Верно |
| Запуск программы с 4 вершинами слева и  3 справа с автогенерированием значений | Вывод наибольшего паросочетания, вывод графической части и конечной матрицы смежности | Верно |
| Запуск программы с 4 вершинами слева и  3 справа с ручным вводом значений | Вывод наибольшего паросочетания, вывод графической части и конечной матрицы смежности | Верно |

# Список литературы

1. Станислав Окулов. Дискретная математика. Теория и практика решения задач по информатике. Учебное пособие. — Litres, 2014-02-07. — С. 186. — 428 с.
2. Фуад Алескеров, Элла Хабина, Дмитрий Шварц. Бинарные отношения, графы и коллективные решения. — Litres, 2016-01-28. — С. 22. — 343 с.
3. Леонид Гладков, Владимир Курейчик, Виктор Курейчик, Павел Сороколетов. [Биоинспирированные методы в оптимизации](https://books.google.com/books?id=Ywp4CwAAQBAJ&pg=PA330). — Litres, 2016-01-28. — С. 330. — 381 с.
4. Зыков А.А. Основы теории графов. - М.:Наука, 1987.
5. Кристофидес Н. Теория графов. Алгоритмический подход. Пер. с анг. - М.:Мир, 1978.

# Листинг программы

# “framework.h”

#pragma once

#include "targetver.h"

#define WIN32\_LEAN\_AND\_MEAN

#include <windows.h>

#include <stdlib.h>

#include <stdio.h>

#include <malloc.h>

#include <memory.h>

#include <tchar.h>

#include <string.h>

#include <strstream>

#include <queue>

#include <time.h>

ATOM MyRegisterClass(HINSTANCE hInstance);

BOOL InitInstance(HINSTANCE, int);

LRESULT CALLBACK WndProc(HWND, UINT, WPARAM, LPARAM);

void makeGraf(void);

void makeTransPut(void);

void makeCoord(void);

void printMus(int\*, char, char, HDC, int, int, bool, char, char);

void drawPoint(char, HDC);

void drawLine(char, HDC, char, char, bool, DWORD);

bool searchDepthRec(HDC, char, char);

void outPutResult(HDC, char);

void makeMatrixAdjac(void);

void resetMatrix(int\*, char, char);

# “Loavis-kur.cpp”

#define \_CRT\_SECURE\_NO\_WARNINGS

#include "framework.h"

#include "Loavis-kur-syava.h"

#include <iostream>

#include <string>

std::ofstream out("output.txt");

std::ifstream in("input.txt");

COORD\* coord; // Указатель на массив координат исходного графа неориентированного

COORD\* coordTrans; // Указатель на массив координат графа транспортной сети

int\* mus; // Указатель на массив полной матрицы смежности исходного двудольного графа

int\* transWay; // Указатель на массив графа транспортной сети (двудольный + еще исток и сток)

int\* matrix; // Указатель на массив уменьшенной матрицы смежности, у которой строки это левая часть двухдольного графа, а столбцы - правая часть двудольного графа

char N, P; // N - кол-во вершин, P - кол-во вершин + исток и сток

char M1, M2; // M1 - кол-во вершин в левой части, М2 - кол-во вершин в правой части

char\* copy;

char\* copy2;

//====================================================================

// Создание матриц смежности (уменьшенной и полной), генерация значений и запись в матрицы

void makeGraf() {

std::string s; // сюда будем класть считанные строки

std::string stack1 = "auto";

std::string stack2 = "manual";

getline(in, s);

copy = strcpy(new char[s.length() + 1], s.c\_str());

std::cout << copy;

M1 = copy[0] - 48;

getline(in, s);

copy2 = strcpy(new char[s.length() + 1], s.c\_str());

std::cout << copy2;

M2 = copy2[0] - 48;

getline(in, s);

bool found = stack1.find(s) != std::string::npos;

bool found2 = stack2.find(s) != std::string::npos;

if (found) {

N = M1 + M2; // Всего вершин - сумма левой и правой части

P = N + 2; // В транспортной сети добавляется еще Исток и сток (+2)

char M3 = (M1 > M2) ? M1 : M2; // М3 - наиобольшее кол-во вершин в одной из частей

mus = (int\*)malloc(N \* N \* sizeof(int)); // Создание полной матрицы смежности вершин

matrix = (int\*)malloc(M3 \* M3 \* sizeof(int)); // Создание уменьшенной матрицы смежности вершин размером M3 \* M3 - двухмерный массив

{

bool badMus; // BadMus - флаг повторной генерация строки или столбца массива

for (char i = 0; i < M1; i++) {

badMus = true; // Обнуляем badMus

for (char j = 0; j < M2; j++) {

\*(matrix + i \* M1 + j) = rand() % 2 \* rand() % 2 \* rand() % 2; // вероятность появления 1 равна 0.125 или (1/8)

if (\*(matrix + i \* M1 + j) == 1) // Если имеется хотя бы одна 1, то перегенерация не нужна

badMus = false;

}

if (badMus) // Если все элементы равнялись = 0, повторно генерируем строку

i--;

}

for (char j = 0; j < M2; j++) {

badMus = true; // Обнуляем badMus

for (char i = 0; i < M1; i++)

if (\*(matrix + i \* M1 + j) == 1) // Если имеется хотя бы одна 1, то перегенерация не нужна

badMus = false;

if (badMus) { // Если все элементы равнялись = 0, повторно генерируем столбец

for (char i = 0; i < M1; i++)

\*(matrix + i \* M1 + j) = rand() % 2 \* rand() % 2 \* rand() % 2; // вероятность появления 1 равна 0.125 или (1/8)

j--; // Повторно проверяем стоблец

}

}

}

resetMatrix(&mus[0], N, N); // Сброс матрицы в 0

makeMatrixAdjac(); // Перенос уменьшенной матрицы смежности в полную матрицу смежности

}

if (found2) {

N = M1 + M2; // Всего вершин - сумма левой и правой части

P = N + 2; // В транспортной сети добавляется еще Исток и сток (+2)

char M3 = (M1 > M2) ? M1 : M2; // М3 - наиобольшее кол-во вершин в одной из частей

mus = (int\*)malloc(N \* N \* sizeof(int)); // Создание полной матрицы смежности вершин

matrix = (int\*)malloc(M3 \* M3 \* sizeof(int)); // Создание уменьшенной матрицы смежности вершин размером M3 \* M3 - двухмерный массив

{

bool badMus; // BadMus - флаг повторной генерация строки или столбца массива

for (char i = 0; i < M1; i++) {

badMus = true; // Обнуляем badMus

for (char j = 0; j < M2; j++) {

getline(in, s);

copy = strcpy(new char[s.length() + 1], s.c\_str());

std::cout << copy;

\*(matrix + i \* M1 + j) = copy[0] - 48; // вероятность появления 1 равна 0.125 или (1/8)

if (\*(matrix + i \* M1 + j) == 1) // Если имеется хотя бы одна 1, то перегенерация не нужна

badMus = false;

}

if (badMus) // Если все элементы равнялись = 0, повторно генерируем строку

i--;

}

for (char j = 0; j < M2; j++) {

badMus = true; // Обнуляем badMus

for (char i = 0; i < M1; i++)

if (\*(matrix + i \* M1 + j) == 1) // Если имеется хотя бы одна 1, то перегенерация не нужна

badMus = false;

if (badMus) { // Если все элементы равнялись = 0, повторно генерируем столбец

for (char i = 0; i < M1; i++)

\* (matrix + i \* M1 + j) = rand() % 2 \* rand() % 2 \* rand() % 2; // вероятность появления 1 равна 0.125 или (1/8)

j--; // Повторно проверяем стоблец

}

}

}

resetMatrix(&mus[0], N, N); // Сброс матрицы в 0

makeMatrixAdjac();

}

}

// Создание матрицы транспортной сети

void makeTransPut() {

transWay = (int\*)malloc(P \* P \* sizeof(int));

resetMatrix(&transWay[0], P, P); // Сброс транспортной матрицы в 0

for (char i = 0; i < P - 1; i++) {

for (char j = i + 1; j < P; j++)

if (i == 0 && j <= M1 || j == P - 1 && i > M1) // В 0-ой строке заполняется первые M1 вершин, которые относются к левой части двухдольного графа заносится 1 - так как еони смежные

\*(transWay + i \* P + j) = 1; // В последнем столбце в строках под номерами вершин из правой части двухдольного графа заносится 1 - так как еони смежные

else if (i > 0 && i < P - 1 && j > 0 && j < P - 1) // Копирование полной матрицы смежности во внутрь матрицы транспортной сети с 1 строки и столбца - по предпоследнюю строку и столбец

\*(transWay + i \* P + j) = \*(mus + (i - 1) \* N + j - 1);

}

}

// Создание массива координат

void makeCoord() {

if (coord != NULL) { // Удаление массивов координат,

delete[] coord; // если они были созданы ранее

delete[] coordTrans;

}

coord = new COORD[N]; // Создание новых массивов координат для полной матрицы смежности

coordTrans = new COORD[P]; // и для матрицы транспортной сети

char k1 = (M1 % 2 == 0) ? 40 : 0; // Если число вершин четное в левой и в правой части по отдельности, то смещаем относительно

char k2 = (M2 % 2 == 0) ? 40 : 0; // центра на половину растояния между вершинами : k1 - для левой, k2 - для правой части

POINT offset = { 300, 150 + 40 \* (int)((M1 > M2 ? M1 : M2) / 2) }; // Смещение координат относительно (0, 0)

for (char i = 0; i < N; i++) {

if (i < M1) { // Если вершина относится к левой части графа, то не смещается больше по Х

coord[i].X = offset.x;

coord[i].Y = offset.y + (i - (int)(M1 / 2)) \* 80 + k1;

}

else { // Если вершина относится к правой части, то смещается на + 100 по Х

coord[i].X = offset.x + 100;

coord[i].Y = offset.y + (i - M1 - (int)(M2 / 2)) \* 80 + k2; // (-М1) - чтобы вершины были на одном уровне, а так как номерация вершин продолжается, необходимо вычесть вершины из левой части

}

coordTrans[i + 1].X = coord[i].X + 400; // Копируем Полученные координаты для двухдольного графа полной матрицы смежности

coordTrans[i + 1].Y = coord[i].Y; // Но по Х смещается на 400, чтобы графы не пересекались

}

coordTrans[0].X = offset.x + 300; // Смещается на 400 и вычитается 100, так как это веришина истока, она находится левее левой части графа

coordTrans[0].Y = offset.y; // Координаты по У копируются

coordTrans[P - 1].X = offset.x + 600; // Смещается на 400 и добавляется еще 200, так как это вершина сток, и находится правее правой части графа

coordTrans[P - 1].Y = offset.y;

}

//====================================================================

// Печать массива на экран - (ptr = указатель на массив, Size1 - кол-во строк, Size2 - кол-во столбцов,

// x и y - смещение относительно (0,0), center - при = true - выделяет главную диагональ серым, a и b - индексы жлемента в массиве, который наадо выделить красным)

void printMus(int\* ptr, char Size1, char Size2, HDC hdc, int x, int y, bool center, char a, char b) {

for (char i = 0; i < Size1; i++)

{

out << "\n" << '\0';

for (char j = 0; j < Size2; j++) {

SetTextColor(hdc, (center && i == j) ? RGB(100, 100, 100) : ((i == a && j == b) ? RGB(255, 0, 0) : (\*(ptr + i \* Size1 + j) == 0) ? RGB(0, 255, 0) : RGB(0, 0, 0))); // Выбор цвета - если главная диагональ - то серым, если 0 - зеленый, если не 0 - черный, если выбранный элемент (a,b) - красным

std::strstream str;

str << (int) \* (ptr + i \* Size1 + j) << '\0';

out << (int) \* (ptr + i \* Size1 + j) << '\0';

TextOutA(hdc, x + j \* 15, y + 15 \* i, str.str(), strlen(str.str()));

}

}

}

// Копирование уменьшенной матрицы смежности в полную матрицу смежности графа

void makeMatrixAdjac() {

for (char i = 0; i < M1; i++) {

for (char j = 0; j < M2; j++)

if (\*(matrix + i \* M1 + j) == 1) { // Если элемент равен 1, то заносится в полную матрицу по смещенным индексам

\*(mus + i \* N + M1 + j) = 1;

\*(mus + (M1 + j) \* N + i) = \*(mus + i \* N + M1 + j); // Зеркалится относительно главной диагонали

}

}

}

// Сброс матрицы, обнуление всех элементов

void resetMatrix(int\* ptr, char Size1, char Size2) { // ptr - указатель, Size1 - кол-во строк, Size2 - кол-во столбцов,

for (char i = 0; i < Size1; i++)

for (char j = 0; j < Size2; j++)

\*(ptr + i \* Size1 + j) = 0;

}

//====================================================================

// Задает начальные значения, обнуляет матрицы и вызывает функцию вывода резульата

void searchDepth(HWND hWnd, HDC hdc) {

resetMatrix(&mus[0], N, N); // Сброс матрицы mus в 0

for (char i = 0; i < N; i++) // Смещение координат исходного двухдольного

coord[i].X += 800; // графа на 800, для вывода результата

drawPoint(0, hdc); // Вывод вершин исходного графа, но без ребер

resetMatrix(&matrix[0], M1, M2); // Сброс уменьшенной матрицы matrix в 0

char count = 0; // count - счетчик паросочетания или максимального потока

for (char i = 0; i < M1; i++) // Запускаем цикл от 0 до M1 - проходим все вершин левой части двухдольного графа

if (searchDepthRec(hdc, 0, 0)) // Функция возравщает true|false. false - если функция не достигла стока, значит паросочетание не надено для этой вершины

count++; // При достижении стока, функция вернет true, значит путь найден и еще одно папросочетание

outPutResult(hdc, count); // Вывод результата. Кол-во паросочетаний и уменьшенную матрицу

makeMatrixAdjac(); // Перенос уменьшенной матрицы смежности в полную матрицу смежности

}

// Рекурсивная функция поиска в глубину. Стартовые верши - это вершины левой части двухдольного графа

bool searchDepthRec(HDC hdc, char st, char sd) {

for (char i = 0; i < P; i++) { // Поиск смежной вершины для текущей

if (\*(transWay + st \* P + i) != 0) { // Если вершина была найдена, то

drawLine(1, hdc, st, i, true, RGB(255, 255, 0)); // Отрисовывается ребро, как уже пройденное - анимация

Sleep(500); // Задержка на 0.5 сек

if (i == P - 1) // Если найденная вершина это стокк, то выход из функции с возвратом true

return true; // Так как путь от истока к стоку был найден

for (int j = 0; j <= M1; j++) // Если текщая вершина не сток, то удаляются все ребра, которые входят в текущую вершину

\*(transWay + j \* P + i) = 0; // чтобы исключить возможность, повторного захода в эту вершину

if (i > M1)

\*(matrix + (st - 1) \* M1 + i - M1 - 1) = 1; // Добавляется ребро в уменьшенную матрицу

if (searchDepthRec(hdc, i, sd)) // Рекурсивный вызов функции поиска в глубину

return true; // Если функция вернула - true, то выход и возврат true

}

}

return false; // Если сток не был достигнут, а вершин больше нет, знат паросочетания не существует для этой вершины, возврат false

}

// Вывод результата поиска

void outPutResult(HDC hdc, char count) {

out << "\r\n Сверху исходная матрица " << '\0';

out << "\r\n Максимальный поток = " << (int)count << " \n " << '\0';

out << "\n Снизу обработанная матрица " << " \n " << '\0';

SetTextColor(hdc, RGB(0, 0, 0)); // цвет текста в черный

std::strstream str;

str << "Максимальный поток = " << (int)count << " " << '\0';

TextOutA(hdc, 30, 270, str.str(), strlen(str.str())); // вывод на экран максимальный поток или кол-во Max паросочетаний

printMus(&matrix[0], M1, M2, hdc, 50, 300, false, -1, -1); // Печать уменьшенной матрицы смежности, в которой остались только найденные паросочетания

out << "\n ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~" << '\0';

}

//====================================================================

//Отрисовка ребер вершин графа

void drawLine(char index, HDC hdc, char cursor, char point, bool pow, DWORD colorPen) {

HPEN RPEN = CreatePen(PS\_SOLID, 3, colorPen); // Кисть рисования

SelectObject(hdc, RPEN);

int\* ptr = mus; // Указатель на полный массив матрицы смежности

char p = N; // Размер массива

COORD\* crd = coord; // Указатель на массив координат

if (index == 1) { // Если это матрица транспортной сети

ptr = transWay; // Указатель на матрицу транспротной сети

p = P; // Её размер

crd = coordTrans; // и координаты

}

double pi = 3.14155926535, anglePow = 20 \* pi / 180, widthPow = 20, rad = 21, angleLine, sinAngle, cosAngle;

if (point == -1) { // ЕСли не указана вторая вершина, значит печать всех инцидентных ребер

for (char i = cursor; i < p; i++) {

if (\*(ptr + cursor \* p + i) != 0) {

angleLine = atan2(crd[i].X - crd[cursor].X, crd[i].Y - crd[cursor].Y); // Вычисления угла нарисованной линии, оносительно оси Ox

sinAngle = sin(angleLine) \* rad; // Уменьшение линии, чтобы не заходила на круги

cosAngle = cos(angleLine) \* rad;

MoveToEx(hdc, crd[cursor].X + sinAngle, crd[cursor].Y + cosAngle, NULL); // Установка курсора на одну вершину

LineTo(hdc, crd[i].X - sinAngle, crd[i].Y - cosAngle); // Отрисовка линии до другой верины

if (pow) { // Если указаны стрелки, (Pow == true)

LineTo(hdc, crd[i].X - sinAngle - sin(angleLine + anglePow) \* widthPow, crd[i].Y - cosAngle - cos(angleLine + anglePow) \* widthPow);

MoveToEx(hdc, crd[i].X - sinAngle, crd[i].Y - cosAngle, NULL);

LineTo(hdc, crd[i].X - sinAngle - sin(angleLine - anglePow) \* widthPow, crd[i].Y - cosAngle - cos(angleLine - anglePow) \* widthPow);

}

}

}

}

else {

angleLine = atan2(crd[point].X - crd[cursor].X, crd[point].Y - crd[cursor].Y);

sinAngle = sin(angleLine) \* rad;

cosAngle = cos(angleLine) \* rad;

MoveToEx(hdc, crd[cursor].X + sinAngle, crd[cursor].Y + cosAngle, NULL);

LineTo(hdc, crd[point].X - sinAngle, crd[point].Y - cosAngle);

if (pow) {

LineTo(hdc, crd[point].X - sinAngle - sin(angleLine + anglePow) \* widthPow, crd[point].Y - cosAngle - cos(angleLine + anglePow) \* widthPow);

MoveToEx(hdc, crd[point].X - sinAngle, crd[point].Y - cosAngle, NULL);

LineTo(hdc, crd[point].X - sinAngle - sin(angleLine - anglePow) \* widthPow, crd[point].Y - cosAngle - cos(angleLine - anglePow) \* widthPow);

}

}

}

// Отрисовка вершин графа

void drawPoint(char index, HDC hdc) {

HPEN RPEN;

char p = N; // Размер массива

COORD\* crd = coord; // Указатель на массив координат

if (index == 1) { // Если это матрица транспортной сети

p = P; // Её размер

crd = coordTrans; // и координаты

}

for (char i = 0, rad; i < p; i++) {

RPEN = CreatePen(PS\_SOLID, 2, RGB(0, 0, 0));

SelectObject(hdc, RPEN);

rad = 20;

Ellipse(hdc, crd[i].X - rad, crd[i].Y + rad, crd[i].X + rad, crd[i].Y - rad); // Рисуем черный круг, с точкой границей

RPEN = CreatePen(PS\_SOLID, rad, RGB(rand() % 255, rand() % 255, rand() % 255));

SelectObject(hdc, RPEN);

rad = 9;

Ellipse(hdc, crd[i].X - rad, crd[i].Y - rad, crd[i].X + rad, crd[i].Y + rad); // Рисуем цветной круг до черного круг, чтобы был внутри черного

}

}

//====================================================================

// Функция обработки сообщений

LRESULT CALLBACK WndProc(HWND hWnd, UINT message, WPARAM wParam, LPARAM lParam) {

switch (message) {

case WM\_PAINT:

{

makeGraf(); // Создание матриц смежности

makeTransPut(); // Создание транспортной сети

makeCoord(); // Создание координат для матриц

PAINTSTRUCT ps;

HDC hdc = BeginPaint(hWnd, &ps);

printMus(&matrix[0], M1, M2, hdc, 50, 50, false, -1, -1); // Вывод на экран исходной сгенерированой уменьшенной матрицы

for (char index = 0; index < 3; index++) { // ПРорисовка 0 1 0

drawPoint(index, hdc); // Отрисовка вершин

for (char i = 0; i < ((index == 1) ? P : N); i++) // Отрисовка ребер для каждой вершины

drawLine(index, hdc, i, -1, (index == 1), RGB(255, 0, 0));

if (index == 1) // Начало поиска в глубину после вывода транспортной матрицы

searchDepth(hWnd, hdc);

}

EndPaint(hWnd, &ps);

}

break;

case WM\_DESTROY:

PostQuitMessage(0);

break;

default:

return DefWindowProc(hWnd, message, wParam, lParam);

}

return 0;

}

//===============================================================================================

//===============================================================================================

// Здесь ничего менять не надо!!

WCHAR szTitle[100];

WCHAR szWindowClass[100];

int APIENTRY wWinMain(\_In\_ HINSTANCE hInstance, \_In\_opt\_ HINSTANCE hPrevInstance, \_In\_ LPWSTR lpCmdLine, \_In\_ int nCmdShow) {

UNREFERENCED\_PARAMETER(hPrevInstance);

UNREFERENCED\_PARAMETER(lpCmdLine);

LoadStringW(hInstance, IDS\_APP\_TITLE, szTitle, 100);

LoadStringW(hInstance, IDC\_LOAVISKURSYAVA, szWindowClass, 100);

MyRegisterClass(hInstance);

if (!InitInstance(hInstance, nCmdShow)) {

return FALSE;

}

HACCEL hAccelTable = LoadAccelerators(hInstance, MAKEINTRESOURCE(IDC\_LOAVISKURSYAVA));

MSG msg;

//srand((unsigned)time(NULL));

while (GetMessage(&msg, nullptr, 0, 0)) {

if (!TranslateAccelerator(msg.hwnd, hAccelTable, &msg)) {

TranslateMessage(&msg);

DispatchMessage(&msg);

}

}

return (int)msg.wParam;

}

ATOM MyRegisterClass(HINSTANCE hInstance) {

WNDCLASSEXW wcex;

wcex.cbSize = sizeof(WNDCLASSEX);

wcex.style = CS\_HREDRAW | CS\_VREDRAW;

wcex.lpfnWndProc = WndProc;

wcex.cbClsExtra = 0;

wcex.cbWndExtra = 0;

wcex.hInstance = hInstance;

wcex.hIcon = LoadIcon(hInstance, MAKEINTRESOURCE(IDI\_LOAVISKURSYAVA));

wcex.hCursor = LoadCursor(nullptr, IDC\_ARROW);

wcex.hbrBackground = (HBRUSH)(COLOR\_WINDOW + 1);

wcex.lpszMenuName = MAKEINTRESOURCEW(IDC\_LOAVISKURSYAVA);

wcex.lpszClassName = szWindowClass;

wcex.hIconSm = LoadIcon(wcex.hInstance, MAKEINTRESOURCE(IDI\_SMALL));

return RegisterClassExW(&wcex);

}

BOOL InitInstance(HINSTANCE hInstance, int nCmdShow) {

HINSTANCE hInst = hInstance; // Сохранить маркер экземпляра в глобальной переменной

HWND hWnd = CreateWindowW(szWindowClass, szTitle, WS\_OVERLAPPEDWINDOW,

0, 0, 1300, 720, nullptr, nullptr, hInstance, nullptr);

if (!hWnd) {

return FALSE;

}

ShowWindow(hWnd, nCmdShow);

UpdateWindow(hWnd);

return TRUE;

}

1. **Результаты работы программы**

