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 наибольшего паросочетания»

Выполнил:

студент группы 21ВВ1

Аляев А.О.

Принял:

к.т.н., доцент Юрова О.В.

Пенза 2022

**Содержание**

[Реферат](#_Toc57476845) 3

[Введение](#_Toc57476846) 4

1. [Постановка задачи](#_Toc57476847) 6

2. [Теоритическая часть задания](#_Toc57476848) 7

3. [Описание алгоритма программы.](#_Toc57476849) 10

4. [Описание программы](#_Toc57476850) 15

5. [Тестирование](#_Toc57476851) 22

6. [Ручной расчёт задачи](#_Toc57476854) 29

[Заключение](#_Toc57476852) 31

[Список литературы](#_Toc57476853) 32

[Приложение А. Листинг программы](#_Toc57476854) 33

**Реферат**

Отчет 32 стр, 25 рисунков.

ДВУДОЛЬНЫЙ ГРАФ,ТЕОРИЯ ГРАФОВ,ПАРОСОЧЕТАНИЕ, АЛГОРИТМ КУНА.

Цель исследования – разработка программы, способной определять наибольшее паросочетание, используя алгоритм Куна.

В работе рассмотрен алгоритм Куна, на основе которого

находится наибольшее паросочетание двудольного графа. При помощи данного алгоритма возможно решение задачи о назначениях.

**Введение**

**Графы**

Граф — это абстрактный математический объект, который

представляет собой множество вершин графа и набор рёбер, которые

соединяют вершины между собой. Например, за множество вершин можно

взять множество населенных пунктов, а в качестве рёбер представить дороги, соединяющие эти пункты.

Графы могут различаться направленностью, ограничениями на количество связей и дополнительными данными о вершинах или рёбрах. Многие структуры могут быть представлены графами. Например, строение компьютерных сетей можно смоделировать при помощи ориентированного графа, в котором вершины — это компьютеры, а дуги (ориентированные рёбра) —соединяющие их сети.

Различают несколько видов графов.

Граф, или неориентированный граф G —это упорядоченная пара G: =(V, E),где V —это непустое множество вершин или узлов, а E —множество пар (в случае неориентированного графа —неупорядоченных) вершин, называемых рёбрами. Ориентированный граф (сокращённо орграф) G —это упорядоченная пара G: =(V,A), где V —непустое множество вершин или узлов, и A —множество (упорядоченных) пар различных вершин, называемых ориентированными рёбрами (дугами).Смешанный граф G —

это граф, в котором некоторые рёбра могут быть ориентированными, а некоторые —неориентированными. Записывается упорядоченной тройкой G: =(V, E,A), где V, E и A определены так же, как выше.

Двудольный граф или биграф – это граф, множество вершин которого

можно разбить на две части таким образом, что каждое ребро графа

соединяет какую-то вершину из одной части с какой-то вершиной другой

части, то есть не существует ребра, соединяющего две вершины из одной и

той же части.

В данной курсовой работе будет рассматриваться двудольный граф.

**Паросочетания**

Паросочетанием в теории графов называют множество попарно несмежных рёбер, то есть рёбер, не имеющих общих вершин.

Выделяют несколько видов паросочетаний.

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.

Наибольшее паросочетание (или максимальное по размеру паросочетание)— это такое паросочетание, которое содержит максимальное количество рёбер. У графа может быть множество наибольших паросочетаний. При этом любое наибольшее паросочетание является максимальным, но не любое максимальное будет наибольшим.

Для нахождения наибольшего паросочетания можно использовать алгоритм Куна. Этот алгоритм можно описать так: сначала возьмём пустое паросочетание, а потом — пока в графе удаётся найти увеличивающую цепь, — будем выполнять чередование паросочетания вдоль этой цепи, и повторять процесс поиска увеличивающей цепи. Как только такую цепь найти не удалось — процесс останавливаем, — текущее паросочетание и есть максимальное.

**1. Постановка задачи**

Исходный граф в программе должен задаваться матрицей смежности,

причём при генерации данных должны быть предусмотрены граничные

условия.

Программа должна работать так, чтобы пользователь вводил

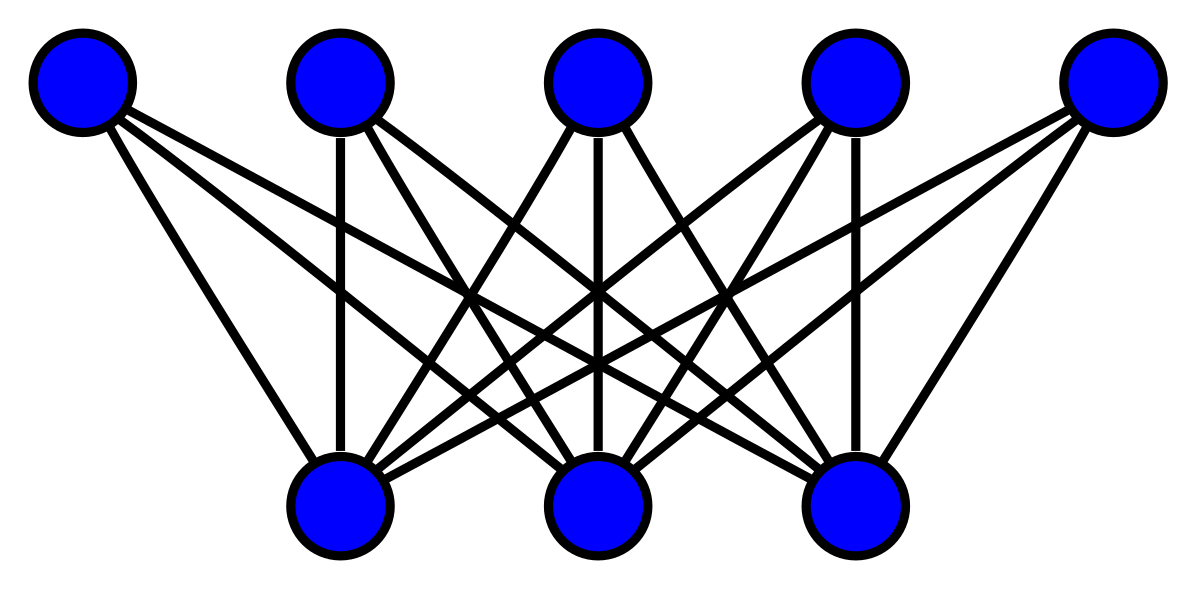
количество вершин для генерации матрицы смежности двудольного графа. После ввода количества вершин, пользователь может выбрать как заполнить граф – вручную или автоматически. После заполнения матрицы он может вызвать функцию алгоритма Куна, которая выведет на экран наибольшее паросочетание. Необходимо предусмотреть различные исходы поиска, чтобы программа не выдавала ошибок и работала правильно.

Устройство ввода – клавиатура и мышь.

Задания выполняются в соответствии с вариантом №27.

**2. Теоретическая часть задания**

Граф G = (V,X) (рисунок 1) называется *двудольным*, если множество V его вершин допускает разбиение на два непересекающихся подмножества V1 и V2 (две доли), причем каждое ребро графа соединяет вершины из разных долей.



**Рисунок 1 – двудольный граф**

Обозначим G = (V1,V2, X) двудольный граф G с долями V1 и V2. *Паросочетанием Р*для двудольного графа G=(V1,V2,X) называется любое множество попарно несмежных ребер в G.

**Цепью** длины k назовём некоторый простой путь (т.е. не содержащий повторяющихся вершин или рёбер), содержащий k рёбер.

**Чередующейся цепью** (в двудольном графе, относительно некоторого паросочетания) назовём цепь, в которой рёбра поочередно принадлежат/не принадлежат паросочетанию.

**Увеличивающей цепью** (в двудольном графе, относительно некоторого паросочетания) назовём чередующуюся цепь, у которой начальная и конечная вершины не принадлежат паросочетанию.

На основании предыдущих терминов можно сформулировать теорему Бержа.

**Теорема Бержа:**

Паросочетание является максимальным тогда и только тогда, когда не существует увеличивающих относительно него цепей.

**Алгоритм Куна** — непосредственное применение теоремы Бержа. Его можно кратко описать так: сначала возьмём пустое паросочетание, а потом — пока в графе удаётся найти увеличивающую цепь, — будем выполнять чередование паросочетания вдоль этой цепи, и повторять процесс поиска увеличивающей цепи. Как только такую цепь найти не удалось — процесс останавливаем, — текущее паросочетание и есть максимальное.

**Алгоритм Куна** — просто ищет любую из таких цепей с помощью **обхода в глубину** или **в ширину**. Он просматривает все вершины графа по очереди, запуская из каждой обход, пытающийся найти увеличивающую цепь, начинающуюся в этой вершине. Удобнее описывать этот алгоритм, считая, что граф уже разбит на две доли.

Алгоритм просматривает все вершины v первой доли графа: v = 1 \ldots n_1 Если текущая вершина v уже насыщена текущим паросочетанием (т.е. уже выбрано какое-то смежное ей ребро), то эту вершину алгоритм пропускает. Иначе — алгоритм пытается насытить эту вершину, для чего запускается поиск увеличивающей цепи, начинающейся с этой вершины.

Поиск увеличивающей цепи осуществляется с помощью обхода в глубину. Изначально обход в глубину стоит в текущей ненасыщенной вершине v первой доли. Просматривая все рёбра из этой вершины, пусть текущее ребро — это ребро (v,to). Если вершина to ещё не насыщена паросочетанием, то, значит, алгоритм смог найти увеличивающую цепь: она состоит из единственного ребра (v,to); в таком случае он включает это ребро в паросочетание и прекращает поиск увеличивающей цепи из вершины v. Иначе, — если to уже насыщена каким-то ребром (p,to), то алгоритм попытается пройти вдоль этого ребра: тем самым попробует найти увеличивающую цепь, проходящую через рёбра (v,to), (to,p). Для этого просто перейдёт в обходе в вершину p — и теперь уже пробует найти увеличивающую цепь из этой вершины.

**3. Описание алгоритма программы**

Для программной реализации алгоритма понадобиться три двумерных массива: двумерный массив n – в нём хранится исходная матрица смежности двудольного графа; вектор mt – содержит в себе информацию о текущем паросочетании (в начале работы программы вектор заполнен числами -1); вектор used–массив посещённости вершин в обходе в глубину.

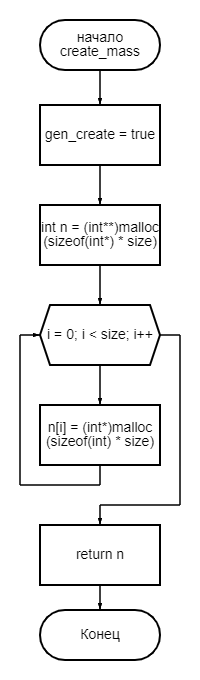
Функция *khun* — есть главная функция программы. Она возвращает *true*, если ей удалось найти увеличивающую цепь из вершины *v*, при этом считается, что эта функция уже произвела чередование паросочетания вдоль найденной цепи.

Внутри функции просматриваются все рёбра, исходящие из вершины *v* первой доли, и затем проверяется: если это ребро ведёт в ненасыщенную вершину *to*, либо если эта вершина *to* насыщена, но удаётся найти увеличивающую цепь рекурсивным запуском из *mt[to]*, то функция нашла увеличивающуюся цепь, и перед возвратом из функции с результатом *true* произвела чередование в текущем ребре: перенаправила ребро, смежное с *to*, в вершину *v*.

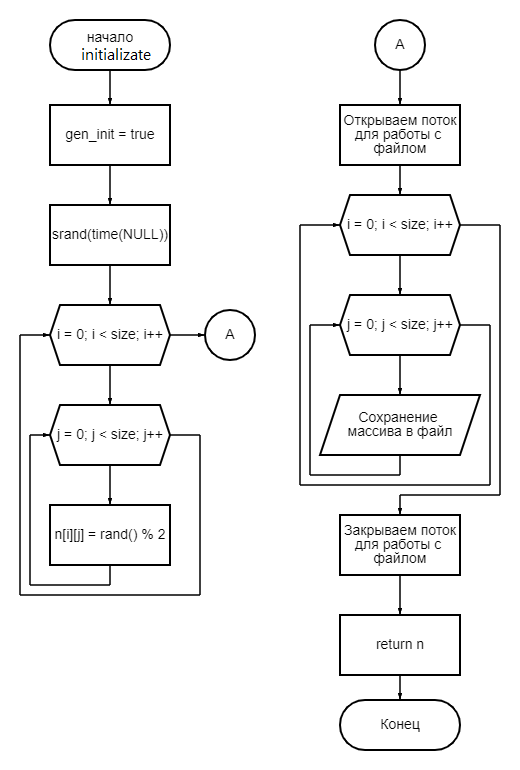
В основной программе сначала указывается, что текущее паросочетание — пустое (массив *mt* заполняется числами -1). Затем перебирается вершина v первой доли, и из неё запускается обход в глубину *khun*, предварительно обнулив массив *used*.

Само искомое паросочетание содержится в массиве *mt*.

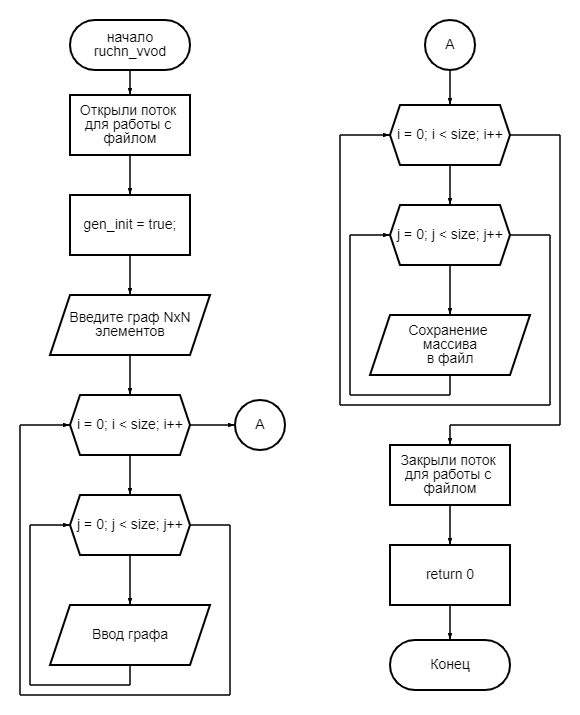
Ниже представлены блок-схемы функций программы.

****Схема программы. Функция create\_mass.

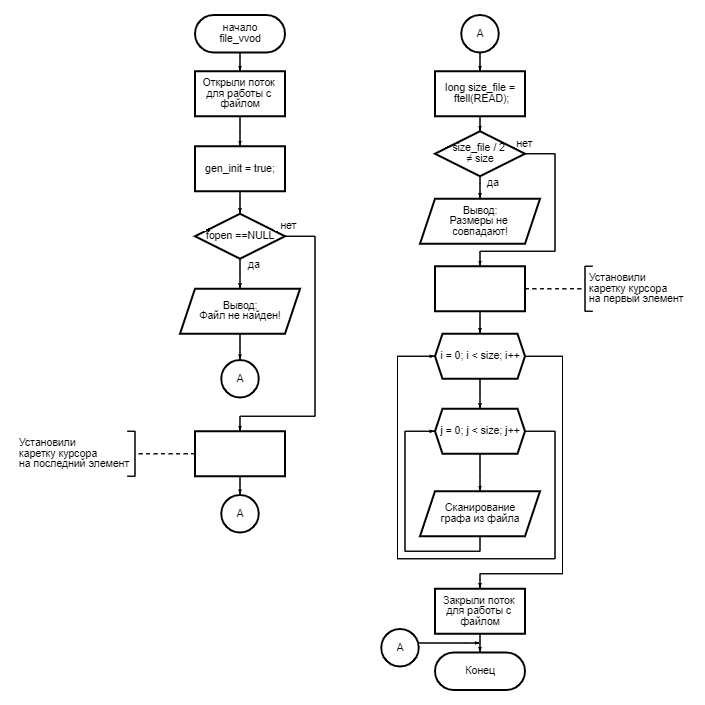
**Рисунок 2 – Функция create\_mass**

Схема программы. Функция initializate.

**Рисунок 3 – Функция initializate**

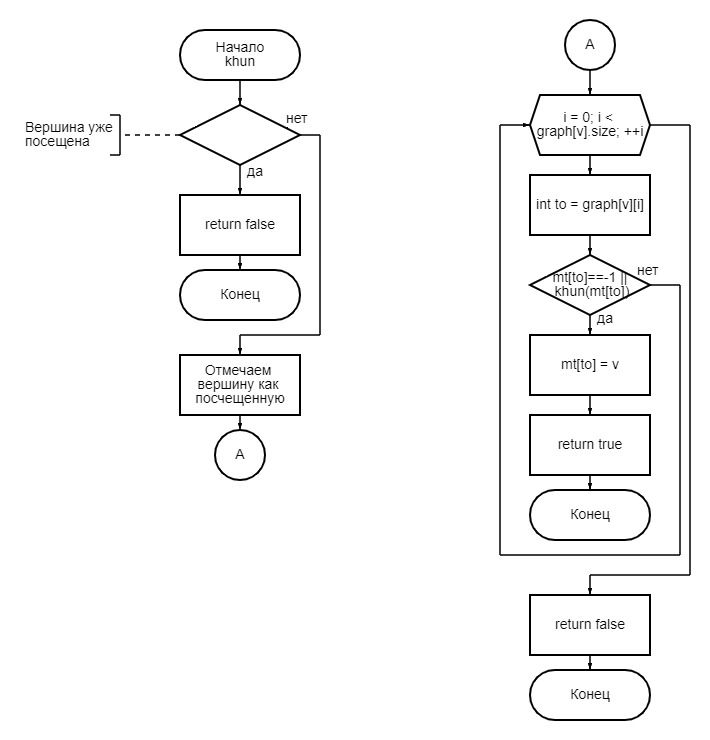
****Схема программы. Функция ruchn\_vvod.

**Рисунок 4 – Функция ruchn\_vvod**

Схема программы. Функция file\_vvod.

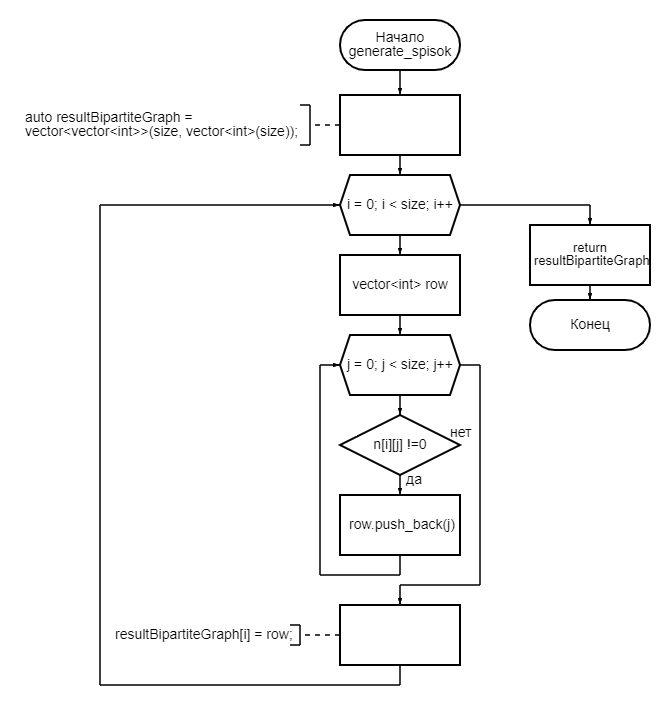
**Рисунок 5 – Функция file\_vvod**

Схема программы. Функция khun.

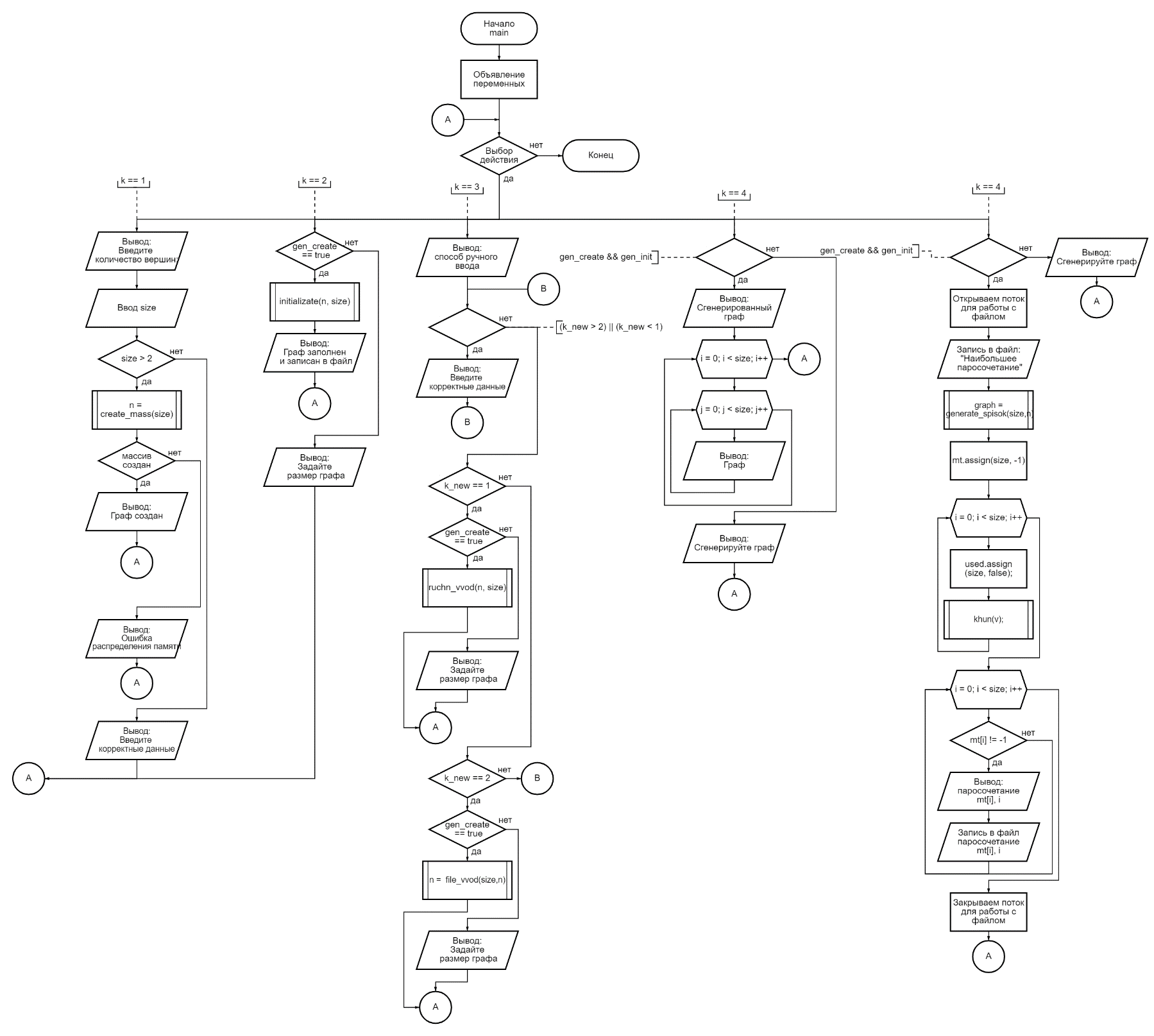
****

**Рисунок 6 – Функция khun**

Схема программы. Функция generate\_spisok.

****

**Рисунок7 – Функция generate\_spisok**

****Схема программы. Функция main.

**Рисунок8 – Функция main**

**4. Описание программы**

Для написания данной программы использован язык программирования С++. Язык программирования С++ - 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++).

Данная программа является многомодульной, поскольку состоит из нескольких функций: create\_mass, initializate, ruchn\_vvod, file\_vvod, khun, generate\_spisok.

Работа программы начинается с запроса действий пользователя. Если пользователь выбрал “Задать двудольный граф размером N×N”, то ему предложат ввести количество вершин в графе. Затем он может выбрать: заполнить массив вручную (с клавиатуры или из файла) или заполнить его автоматически (при помощи функции rand()). После заполнения графа пользователь может сразу вызвать алгоритм Куна, который выведет ему наибольшее паросочетание или вывести исходную матрицу смежности на экран.

Часть кода, ответственная за выбор и вызов функций:

if (k == 1) {

cout <<"Введите количество вершин (больше 2): ";

cin >> size;

if (size > 2) {

n = create\_mass(size);

if (n)

cout <<"Графсоздан!\n";

else

cout <<"Ошибка распределения памяти!\n";

}

else

cout <<"Введите корректные данные!\n";

}

//---------------------------------------------------------Заполнениеграфа

if (k == 2) {

initializate(n, size);

cout <<"Граф заполнен и записан в файл!\n";

}

//--------------------------------------------------------------Ручной ввод

if (k == 3) {

cout <<"\n1 - Задать граф с клавиатуры\n2 - Задать граф из файла в папке проекта\n";

cin >> k\_new;

if ((k\_new > 2) || (k\_new < 1))

cout <<"Введите корректные данные!\n";

elseif (k\_new == 1) {

if(gen\_create == true){

ruchn\_vvod(n, size);

}

else

cout<<"Задайтеразмерграфа!\n";

}

elseif (k\_new == 2){

if(gen\_create == true)

n = file\_vvod(size,n);

else

cout <<"Задайте размер графа!\n";

}

}

Часть кода, генерирующая динамический массив, заполняющая его автоматически и записывающая в файл.

int\*\* create\_mass(int size)

{

gen\_create = true;

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

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

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

}

return n;

}

int\*\* initializate(int\*\* n, int size)

{

gen\_init = true;

srand(time(NULL));

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

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

n[i][j] = rand() % 2;

}

}

WRITE = fopen("Write\_graph.txt", "w");

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

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

fprintf(WRITE, "%3d ", n[i][j]);

}

fprintf(WRITE, "\n");

}

fclose(WRITE);

return n;

}

Если пользователь захотел ввести матрицу с клавиатуры или из файла, то будут задействованы следующие функции:

Ввод с клавиатуры:

int ruchn\_vvod(int\*\* n, int size)

{

gen\_init = true;

cout <<"Введите граф размером "<< size \* size <<" символов (только 0 и 1): \n";

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

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

cin >> n[i][j];

}

}

return 0;

}

Вводизфайла:

int\*\* file\_vvod(int size,int\*\* n)

{

gen\_init = true;

READ = fopen("Read\_graph.txt", "r");

if (fopen("Read\_graph.txt", "r") == NULL){

cout <<"Файл не найден!";

}

else {

fseek(READ, 0, SEEK\_END);

long size\_file = ftell(READ);

if ((size\_file+1) / 2 != size\*size) {

cout <<"Размеры не совпадают!";

cout << size\_file;

}

else {

fseek(READ, 0, SEEK\_SET);

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

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

fscanf(READ, "%d", &n[i][j]);

}

}

}

}

fclose(READ);

return n;

}

После ввода двумерного графа пользователь может:

Вывести его на экран:

if (k == 4) {

if((gen\_create && gen\_init) == true){

cout <<"Сгенерированныйграф: \n";

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

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

printf("%3d", n[i][j]);

}

cout <<"\n";

}

}

else

cout <<"Сгенерируйтеграф!\n";

}

Вызвать алгоритм Куна для поиска наибольшего паросочетания:

Часть кода, отвечающая за вызов:

if (k == 5) {

if((gen\_create && gen\_init) == true){

WRITE = fopen("Write\_graph.txt", "a+");

fprintf(WRITE, "Максимальное паросочетание\n");

cout <<"Максимальноепаросочетание:\n";

graph = generate\_spisok(size,n);

//cout << "\n";

mt.assign(size, -1);

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

used.assign(size, false);

khun(v);

}

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

if (mt[i] != -1){

printf("%d %d\n", mt[i], i);

fprintf(WRITE, "%d %d\n", mt[i], i);

}

}

}

else

cout <<"Сгенерируйтеграф!\n";

fclose(WRITE);

}

Часть кода, отвечающая за алгоритм.

boolkhun(intv)

{

if (used[v])

returnfalse;

used[v] = true;

for (int i = 0; i < graph[v].size(); ++i) {

int to = graph[v][i];

if (mt[to] == -1 || khun(mt[to])) {

mt[to] = v;

returntrue;

}

}

returnfalse;

}

Для корректной работы алгоритма, перед вызовом функции *khun(),* вызывается функция *generate\_spisok()*, которая записывает в массив *graph*двудольный граф в виде списка смежности исходной матрицы.

vector<vector<int>> generate\_spisok(int size,int\*\* n){

auto resultBipartiteGraph = vector<vector<int>>(size, vector<int>(size));

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

vector<int> row;

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

if (n[i][j] !=0)

row.push\_back(j);

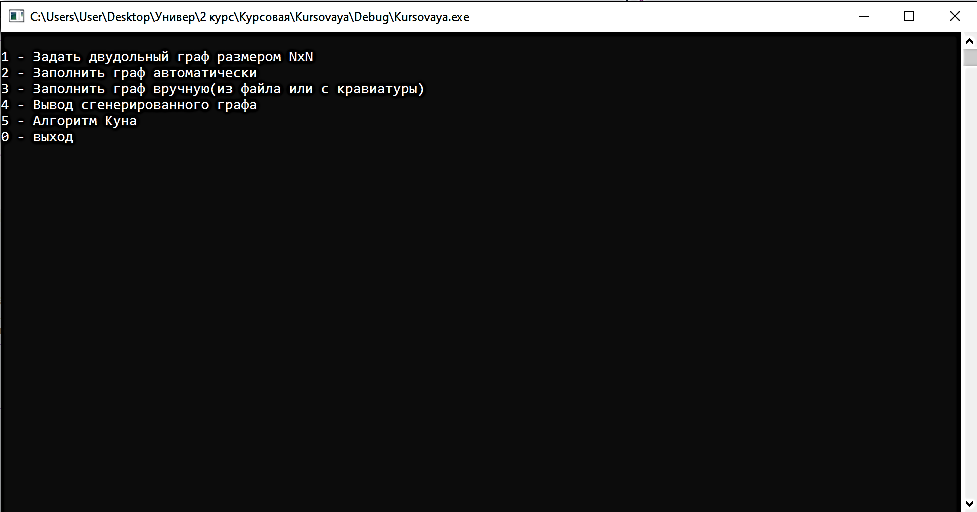
}

resultBipartiteGraph[i] = row;

}

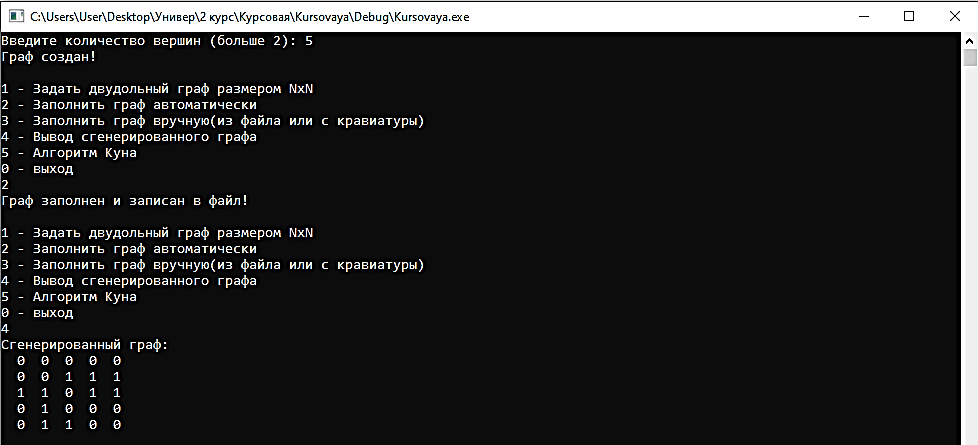
return resultBipartiteGraph;

}

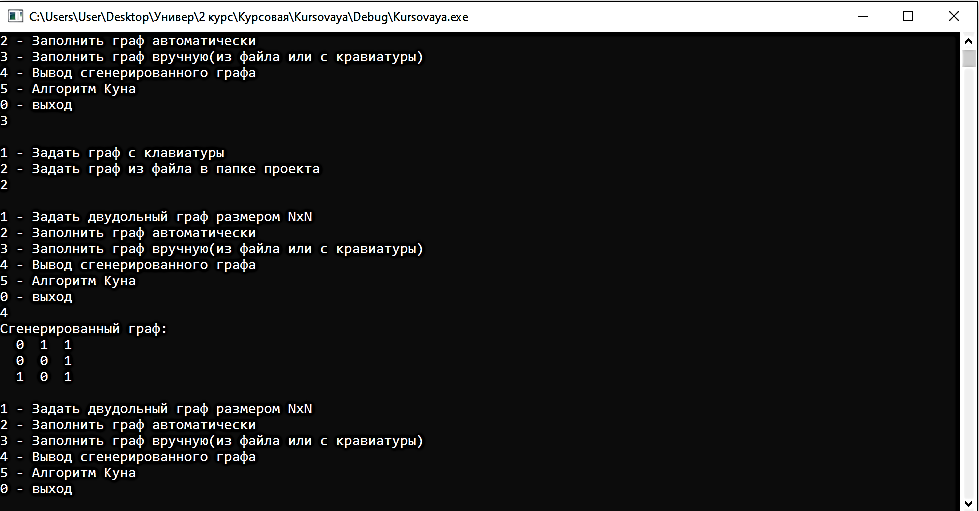
Ниже можно увидеть оформление начального запроса и дальнейшие действия с ним.

**Рисунок 9 – Стартовое окно программы**

Генерация графа размером 5 вершин, его автоматическое заполнение и вывод на экран.

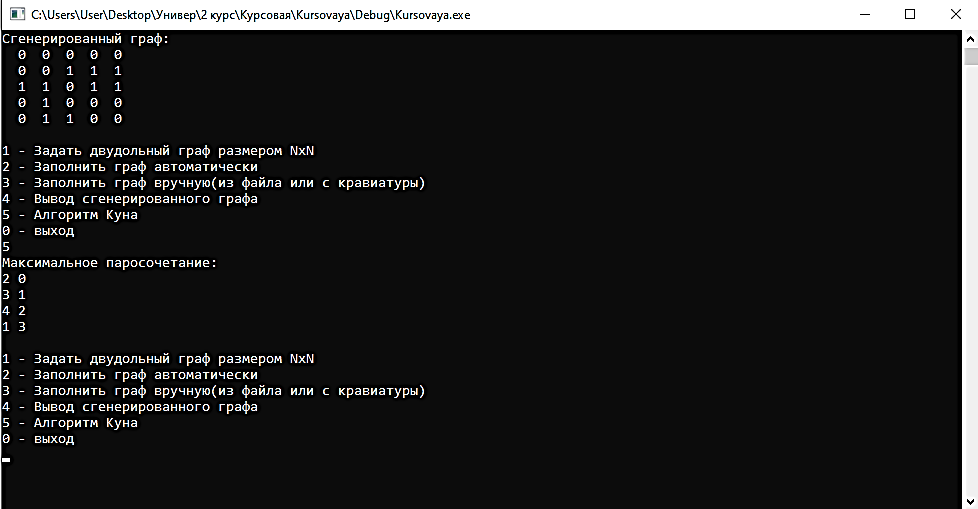


**Рисунок 10 – Случайная генерация графа**

Генерация графа размером 3 вершины, считывание его из файла и вывод на экран.

**Рисунок 11 – Считывание из файла**

Вызов функции поиска наибольшего паросочетания и вывод его на экран:



**Рисунок 12 – Алгоритм Куна**

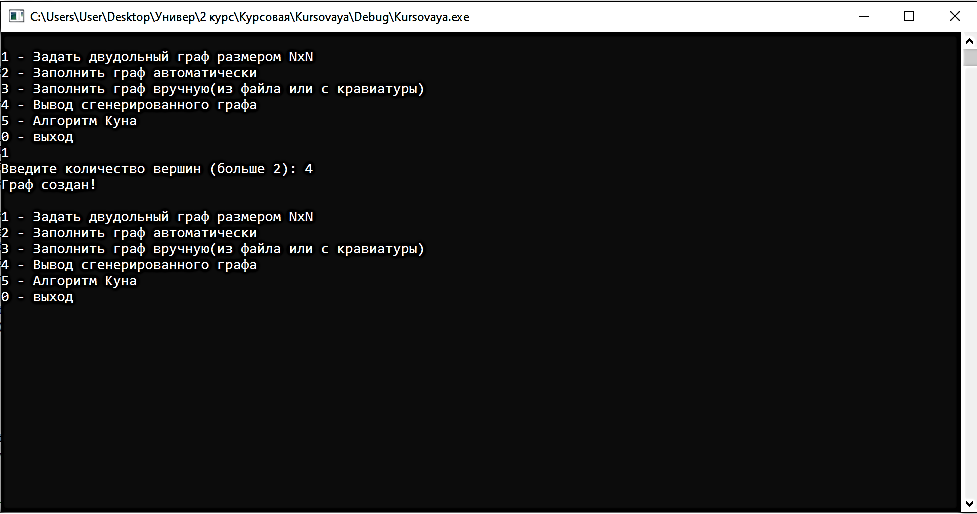
**5. Тестирование**

Среда разработки Microsoft Visual Studio 2010 предоставляет все средства, необходимые при разработке и отладке многомодульной программы.

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

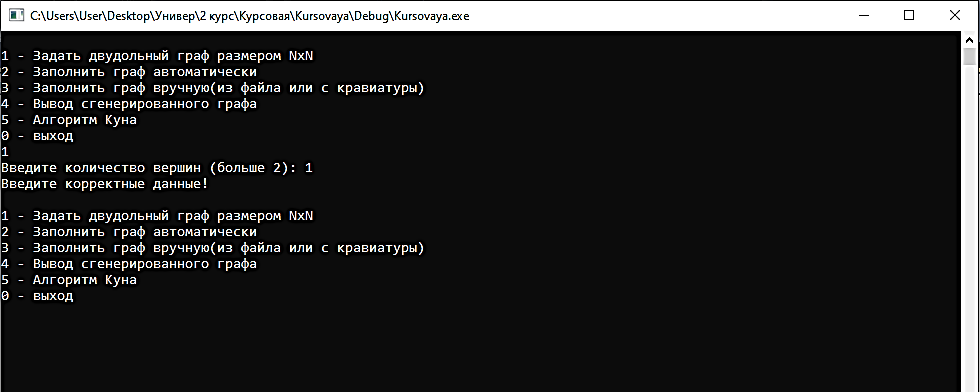
Ниже продемонстрирован результат тестирования программы при вводе пользователем различных количеств вершин и вызове функций без предварительного создания графа.

Тестирование на ввод вершин:



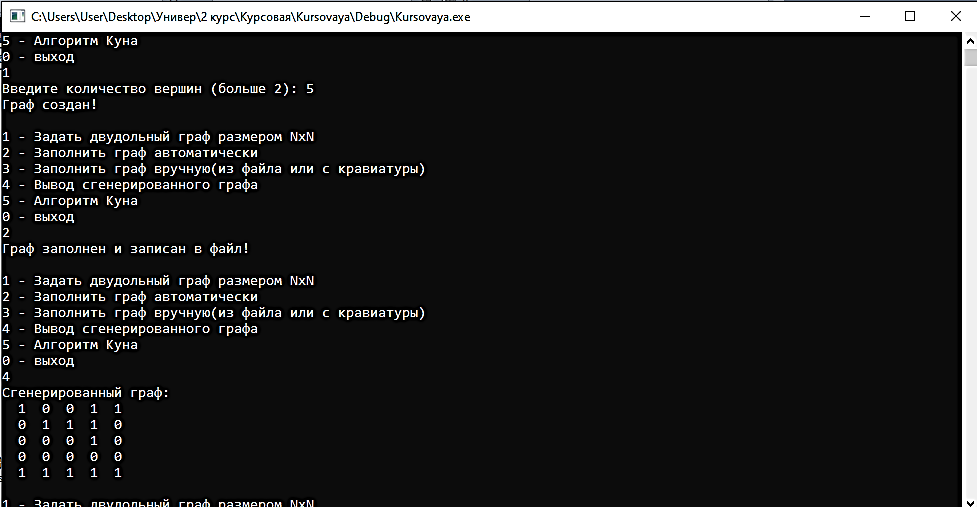
**Рисунок 13 – Тестирование при вводе количества вершин – 4.**

Тестирование на ввод одной вершины:



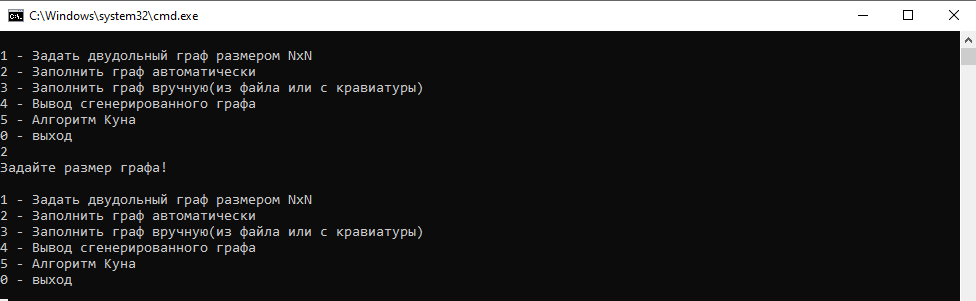
**Рисунок 14 – Тестирование при вводе количества вершин – 1.**

Тестирование на автоматическую генерацию графа и его вывод на экран:



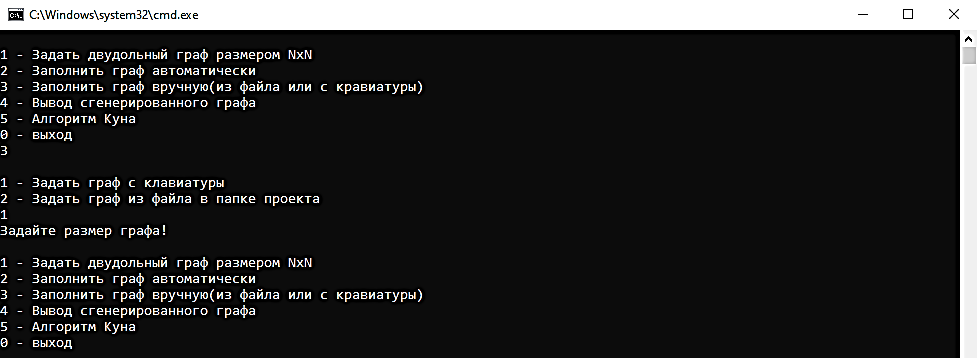
**Рисунок 15 – Тестирование на вывод матрицы.**

Тестирование на автоматическое заполнение графа без его создания:



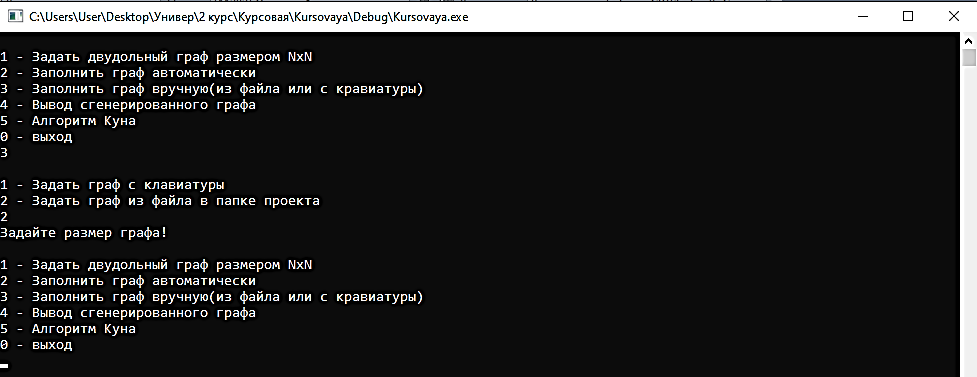
**Рисунок 16 – Тестирование на автомат. заполнение без создания.**

Тестирование на ручное заполнение графа с клавиатуры без его создания:



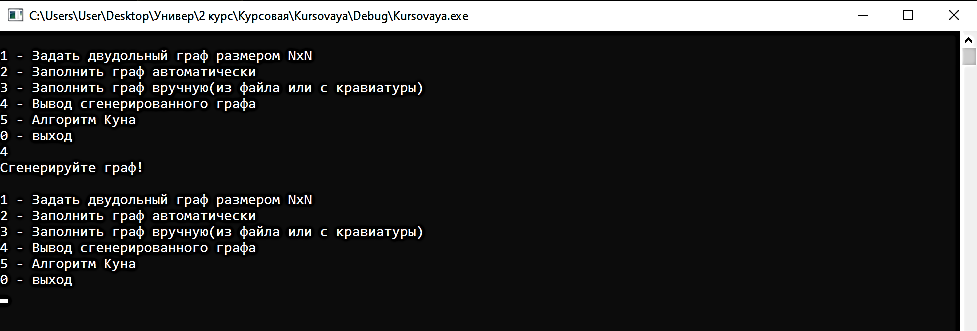
**Рисунок 17 – Тестирование на ручное заполнение без создания.**

Тестирование на ручное заполнение графа из файла без его создания:

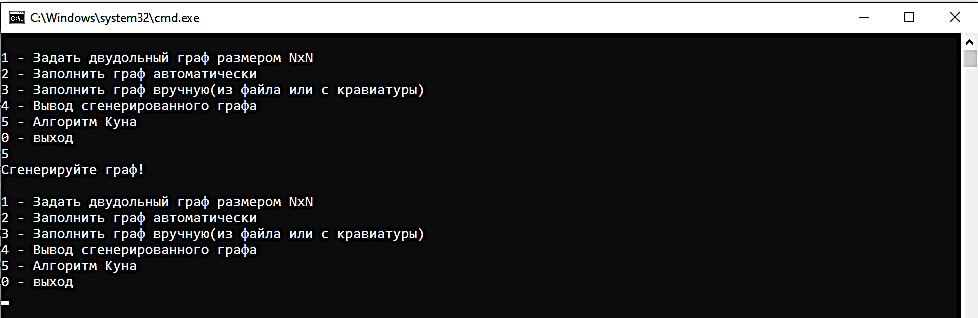
****

**Рисунок 18 – Тестирование на ручное заполнение без создания.**

Тестирование на вывод графа без его создания:

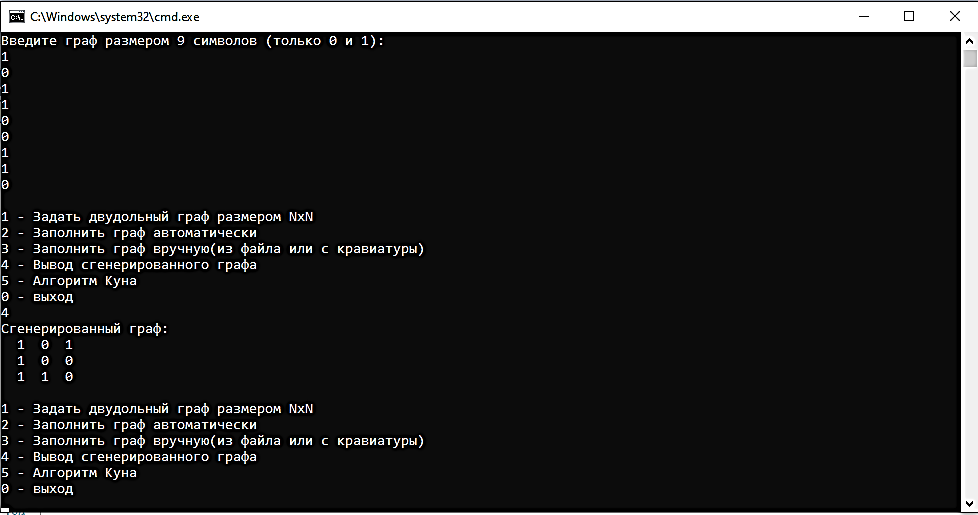
****

**Рисунок 19 – Тестирование на вывод без создания.**

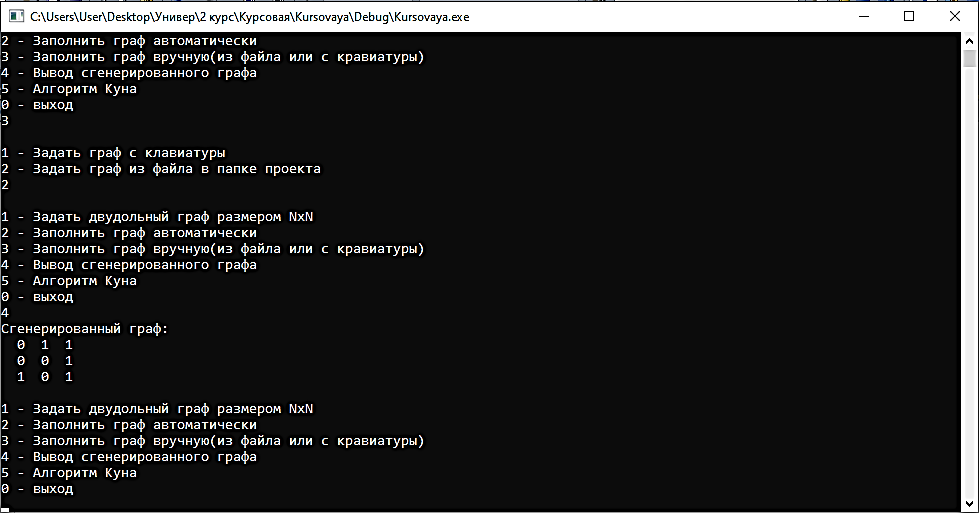
Тестирование на вызов алгоритма Куна без создания графа:

**Рисунок 20 – Тестирование на вывод алгоритма без создания.**

Тестирование на ручной ввод графа с клавиатуры:

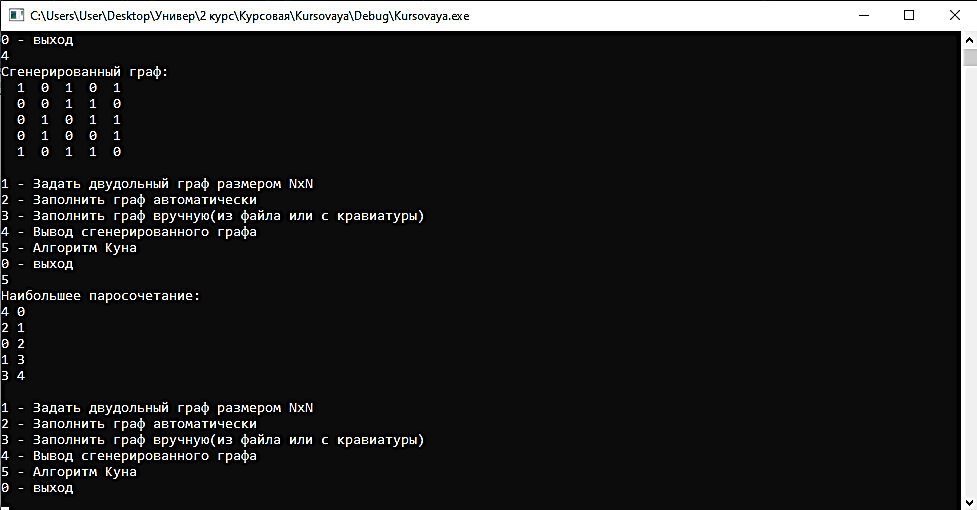


**Рисунок 21 – Тестирование на ввод графа с клавиатуры.**

Тестирование на ручной ввод графа из файла:

**Рисунок 22 – Тестирование на ввод графа из файла.**

Тестирование на работу алгоритма Куна:



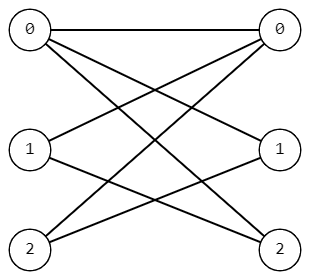
**Рисунок 23 – Тестирование на работу алгоритма Куна.**

Таблица 1 – Описание поведения программы при тестировании.

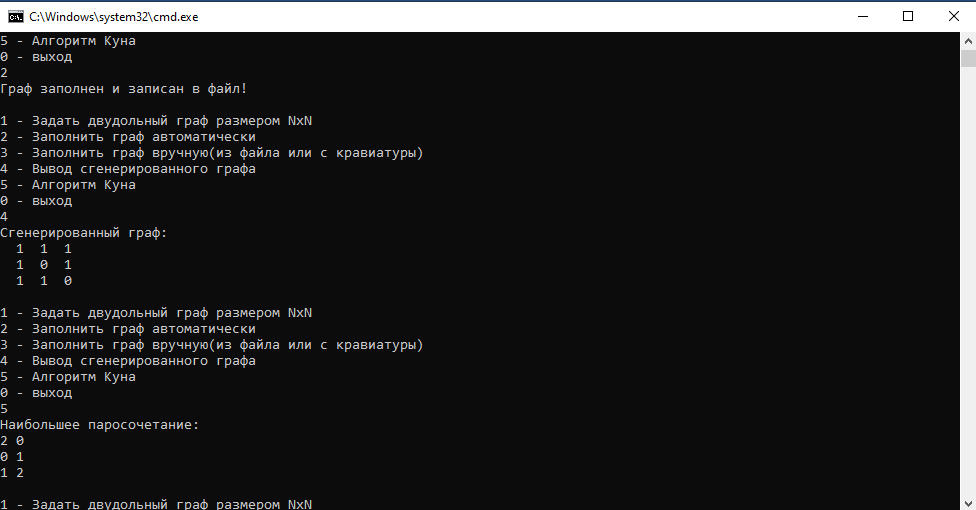
|  |  |  |
| --- | --- | --- |
| Описание теста | Ожидаемый результат | Полученный результат |
| Запуск программы | Вывод пользователю списка с функциями программы | Верно |
| Задание размера матрицы | Вывод пользователю предложения задать размер матрицы числом | Верно |
| Задание матрицы единичной вершиной | Вывод пользователю предупреждения о неправильности вводимых данных | Верно |
| Тестирование на вывод матрицы | Генерация и вывод матрицы на экран | Верно |
| Вызов функции автоматического заполнения без задания размера графа. | Вывод пользователю предупреждения о необходимости создания графа | Верно |
| Вызов функции ручного заполнения с клавиатуры без задания размера графа. | Вывод пользователю предупреждения о необходимости задания размера графа | Верно |
| Вызов функции заполнения из файла без задания размера графа. | Вывод пользователю предупреждения о необходимости задания размера графа | Верно |
| Вызов функции вывода графа без его создания | Вывод пользователю предупреждения о необходимости создания графа | Верно |
| Вызов функции алгоритма Куна без создания графа | Вывод пользователю предупреждения о необходимости создания графа | Верно |
| Тестирование на ручной ввод графа с клавиатуры | Запись графа в массив | Верно |
| Тестирование на ручной ввод графа из файла | Запись графа в массив | Верно |
| Тестирование на работу алгоритма Куна. | Вывод результата работы алгоритма на экран | Верно |

**6. Ручной расчет задачи**

Проведем проверку программы посредством ручных вычислений на примере графа с 3 вершинами(рисунок 17), сгенерированного автоматически в программе (рисунок 18).



**Рисунок 24 – Двудольный граф.**



**Рисунок 25 –Результат генерации.**

Рассмотрим левое множество вершин двудольного графа, а именно вершину **0**. Обозначим её как *v.* Возьмём ненасыщенную паросочетанием вершину из правого множества вершин графа, пусть это будет вершина **0**, обозначим её как *to*. Мы нашли увеличивающуюся цепь, она состоит из 1 ребра (0,0) —(*v,to*). Включим это ребро в паросочетание и прекратим поиск увеличивающей цепи из вершины *v*.

Обратимся к вершине **1** левого множества графа, обозначим её как *v.* Возьмём вершину **0** из левого множества графа, обозначив её за *to*, она насыщена паросочетанием, значит попробуем найти увеличивающуюся цепь, проходящую через рёбра (1,0) и (0,0). Такая цепь есть, это вершина **1** из правого множества графа. Заменили паросочетание (0,0) на (0,1)*.* Мы нашли увеличивающуюся цепь, она состоит из 1 ребра(0,1). Включим это ребро в паросочетание и прекратим поиск увеличивающей цепи.

Обратимся к вершине 2 левого множества графа, обозначим её как *v.* Возьмём вершину **0** из левого множества графа, обозначив её за *to,* она насыщена паросочетанием, значит, попробуем найти увеличивающуюся цепь, проходящую через рёбра (2,0) и (0,1). Такая цепь есть, это вершина 2 из правого множества графа. Заменили паросочетание (1,0) на (1,2). Мы нашли увеличивающуюся цепь, она состоит из 1 ребра (1,2). Включим это ребро в паросочетание и прекратим поиск увеличивающей цепи.

Таким образом у нас получилось множество наибольшего паросочетания, состоящего из рёбер: (2,0), (0,1) и (1,2).

**Заключение**

Таким образом, в процессе создания данного проекта разработана

программа, реализующая алгоритм Куна для поиска наибольшего паросочетания в Microsoft Visual Studio 2010.

При выполнении данной курсовой работы были получены навыки

разработки программ и освоены приемы создания двудольных графов,

и динамических структур данных. Приобретены навыки по осуществлению

алгоритма Куна. Углублены знания языка программирования C++.

Недостатком разработанной программы является примитивный

пользовательский интерфейс. Потому что программа работает в консольном

режиме, не добавляющем к сложности языка сложность программного

оконного интерфейса.

Код написан максимально компактно и интерфейс удобен для использования. Полученные знания можно применять в дальнейшей деятельности.

**Список литературы**

1. Герберт Шилдт. C++. Базовый курс. 2014 г.
2. В. Л. Дольников О. П. Якимова. Основные алгоритмы на графах. 2011 г.
3. Уилсон Р. Введение в теорию графов. Пер. с анг. 1977. 208 с.

**Приложение А.**

**Листинг программы.**

#include"StdAfx.h"

#include<stdlib.h>

#include"stdafx.h"

#include"locale.h"

#include"iostream"

#include<vector>

#include"time.h"

bool gen\_create = false;

bool gen\_init = false;

int metka = 0;

int size;

usingnamespace std;

vector<int> mt;

vector<vector<int>> graph;

vector<char> used(size);

FILE\* READ;

FILE\* WRITE;

int\*\* create\_mass(int size)

{

gen\_create = true;

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

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

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

}

return n;

}

int\*\* initializate(int\*\* n, int size)

{

gen\_init = true;

srand(time(NULL));

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

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

n[i][j] = rand() % 2;

}

}

WRITE = fopen("Write\_graph.txt", "w");

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

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

fprintf(WRITE, "%3d ", n[i][j]);

}

fprintf(WRITE, "\n");

}

fclose(WRITE);

return n;

}

int ruchn\_vvod(int\*\* n, int size)

{

gen\_init = true;

WRITE = fopen("Write\_graph.txt", "w");

cout <<"Введите граф размером "<< size \* size <<" символов (только 0 и 1): \n";

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

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

cin >> n[i][j];

}

}

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

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

fprintf(WRITE, "%3d ", n[i][j]);

}

fprintf(WRITE, "\n");

}

fclose(WRITE);

return 0;

}

int\*\* file\_vvod(int size,int\*\* n)

{

gen\_init = true;

READ = fopen("Read\_graph.txt", "r");

if (fopen("Read\_graph.txt", "r") == NULL){

cout <<"Файл не найден!";

return 0;

}

else {

fseek(READ, 0, SEEK\_END);

long size\_file = ftell(READ);

if ((size\_file+1) / 2 != size\*size) {

cout <<"Размеры не совпадают!";

}

else {

fseek(READ, 0, SEEK\_SET);

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

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

fscanf(READ, "%d", &n[i][j]);

}

}

}

}

fclose(READ);

return n;

}

bool khun(int v)

{

if (used[v])

returnfalse;

used[v] = true;

for (int i = 0; i < graph[v].size(); ++i) {

int to = graph[v][i];

if (mt[to] == -1 || khun(mt[to])) {

mt[to] = v;

returntrue;

}

}

returnfalse;

}

vector<vector<int>> generate\_spisok(int size,int\*\* n){

auto resultBipartiteGraph = vector<vector<int>>(size, vector<int>(size));

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

vector<int> row;

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

if (n[i][j] !=0)

row.push\_back(j);

}

resultBipartiteGraph[i] = row;

}

return resultBipartiteGraph;

}

int main()

{

int\*\* n;

setlocale(LC\_ALL, "");

int k = 1;

int k\_new = 1;

while (k) {

cout <<"\n1 - Задать двудольный граф размером NхN\n2 - Заполнить граф автоматически\n3 - Заполнить граф вручную(из файла или с кравиатуры)\n4 - Вывод сгенерированного графа\n5 - Алгоритм Куна\n"

<<"0 - выход"<< endl;

cin >> k;

//----------------------------------------------------------------Создание (ввод количества вершин)

if (k == 1) {

cout <<"Введите количество вершин (больше 2): ";

cin >> size;

if (size > 2) {

n = create\_mass(size);

if (n)

cout <<"Графсоздан!\n";

else

cout <<"Ошибка распределения памяти!\n";

}

else

cout <<"Введите корректные данные!\n";

}

//---------------------------------------------------------Заполнениеграфа

if (k == 2) {

if(gen\_create == true){

initializate(n, size);

cout <<"Граф заполнен и записан в файл!\n";

}

else

cout <<"Задайте размер графа!\n";

}

//--------------------------------------------------------------Ручной ввод

if (k == 3) {

cout <<"\n1 - Задать граф с клавиатуры\n2 - Задать граф из файла в папке проекта\n";

cin >> k\_new;

if ((k\_new > 2) || (k\_new < 1))

cout <<"Введите корректные данные!\n";

elseif (k\_new == 1) {

if(gen\_create == true){

ruchn\_vvod(n, size);

}

else

cout <<"Задайтеразмерграфа!\n";

}

elseif (k\_new == 2){

if(gen\_create == true)

n = file\_vvod(size,n);

else

cout <<"Задайте размер графа!\n";

}

}

//-------------------------------------------------------------Выводграфа

if (k == 4) {

if((gen\_create && gen\_init) == true){

cout <<"Сгенерированныйграф: \n";

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

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

printf("%3d", n[i][j]);

}

cout <<"\n";

}

}

else

cout <<"Сгенерируйтеграф!\n";

}

//------------------------------------------------------------Проверкаграфа

if (k == 5) {

if((gen\_create && gen\_init) == true){

WRITE = fopen("Write\_graph.txt", "a+");

fprintf(WRITE, "Наибольшее паросочетание\n");

cout <<"Наибольшеепаросочетание:\n";

graph = generate\_spisok(size,n);

//cout << "\n";

mt.assign(size, -1);

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

used.assign(size, false);

khun(v);

}

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

if (mt[i] != -1){

printf("%d %d\n", mt[i], i);

fprintf(WRITE, "%d %d\n", mt[i], i);

}

}

fclose(WRITE);

}

else

cout <<"Сгенерируйтеграф!\n";

}

//------------------------------------------------------—

if (k == 0)

return 0;

}

system("pause");

}