Министерство науки и высшего образования Российской Федерации

Пензенский государственный университет

Кафедра «Вычислительная техника»

**ПОЯСНИТЕЛЬНАЯ ЗАПИСКА**

к курсовому проектированию

по курсу «Логика и основы алгоритмизации в инженерных задачах»

на тему «Реализация алгоритма поиска максимальных паросочетаний»

Выполнил:

студент группы 22ВВВ2

Расторгуева К.В.

Принял:

д.т.н. профессор Митрохин М.А.

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 Расторгуевой Ксении Владимирове Группа 22ВВВ2

Тема проекта Реализация алгоритма поиска максимальных паросочетаний.

Исходные данные (технические требования) на проектирование

Разработка алгоритмов и программного обеспечения в соответствии с данным заданием курсового проекта.

Обязательные требования к проекту:

1. Текстовое или графическое меню
2. Возможность задания пользователем размера графа (множества)
3. Возможность выбора автоматического (случайного) или ручного (с клавиатуры или из файла) задания графа (элементов множества)
4. Возможность сохранения результатов работы программы

Пояснительная записка должна содержать следующие разделы:

1. Постановка задачи
2. Теоретические сведения
3. Описание разработки программы
4. Описание программы
5. Отладка и тестирование
6. Ручной расчёт задачи

**Объем работы по курсу**

1. Расчетная часть

Ручной расчет работы алгоритма.

1. Графическая часть

Схема работы программы.

1. Экспериментальная часть

Отладка и тестирование программы.

Срок выполнения проекта по разделам

В соответствии с графиком выполнения курсового проекта

Дата выдачи задания “ 07 ” 09.2023г.

Дата защиты проекта “ ”

**Руководитель\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Задание получил** “ 07 ” 09.2023г.

**Студент\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

СОДЕРЖАНИЕ

[РЕФЕРАТ 5](#_Toc153911284)

[ВВЕДЕНИЕ 6](#_Toc153911285)

[1. ПОСТАНОВКА ЗАДАЧИ 7](#_Toc153911286)

[2. ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ 8](#_Toc153911287)

[3. ОПИСАНИЕ РАЗРАБОТКИ ПРОГРАММЫ 10](#_Toc153911288)

[5. ОПИСАНИЕ ПРОГРАММЫ 12](#_Toc153911289)

[4. ОТЛАДКА И ТЕСТИРОВАНИЕ 17](#_Toc153911290)

[6. РУЧНОЙ РАСЧЁТ ЗАДАЧИ 24](#_Toc153911291)

[ЗАКЛЮЧЕНИЕ 26](#_Toc153911292)

[СПИСОК ИСПОЛЬЗУЕМЫХ ИСТОЧНИКОВ 27](#_Toc153911293)

[ПРИЛОЖЕНИЕ 28](#_Toc153911294)

# РЕФЕРАТ

Пояснительная записка 36 с., 16 рис., 1 таблица, 4 источника,1 приложение

ГРАФ, ТЕОРИЯ ГРАФОВ. МАКСИМАЛЬНОЕ ПАРОСОЧЕНТАНИЕ, ПОИСК В ГЛУБИНЫ

Целью проекта является создание программы, способной находить максимальное паросочетание в двудольном графе.

В ходе работы был проведен анализ предметной области, установлены задачи проекта для достижения поставленной цели.

Результатом проведенной работы является создание функционирующего приложения для поиска максимального паросочетания в двудольном графе, которое выполняет требуемый круг задач.

# ВВЕДЕНИЕ

Среди дисциплин и методов дискретной математики теория графов и особенно алгоритмы на графах находят наиболее широкое применение в программировании. Теория графов предоставляет очень удобный язык для описания программных и многих других моделей.

Первой работой теории графов как математической дисциплины считают статью Эйлера (1736 г.), в которой рассматривалась задача о Кенигсбергских мостах. Эйлер показал, что нельзя обойти семь городских мостов и вернуться в исходную точку, пройдя по каждому мосту ровно один раз. Следующий импульс теория графов получила спустя почти 100 лет с развитием исследований по электрическим сетям, кристаллографии, органической химии и другим наукам, эти работы связаны с именами Г. Кирхгофа, А. Кэли, К. Жордана.

Графы служат удобным средством описания связей между объектами. Например, рассматривая граф, изображающий сеть дорог между населёнными пунктами, можно определить маршрут проезда от пункта А до пункта Б. Если таких маршрутов окажется несколько, хотелось бы выбрать в определённом смысле оптимальный, например, самый короткий или самый безопасный. Для решения задачи выбора требуется проводить определённые вычисления над графами.

Методы теории графов широко применяются в программировании. Без них невозможно обойтись при анализе и синтезе различных дискретных преобразователей: функциональных блоков компьютеров, комплексов программ и т. д. В настоящее время теория графов охватывает большой материал и активно развивается.

Одна из классических задач в теории графов состоит в поиске паросочетания максимальной мощности. Решение этой задачи имеет прикладное значение при формировании комитетов, распределении работников по должностям и пр. При этом вершинам рассматриваемого двудольного графа в некоторых приложениях соответствуют игроки – люди или организации.

# 1. ПОСТАНОВКА ЗАДАЧИ

Целью работы является реализация алгоритма поиска максимальных паросочетаний.

На вход программы подаётся размер графа. Программа должна обработать данные, сформировать граф, осуществить алгоритм поиска максимальных паросочетаний, вывести результат.

Разрабатываемая программа так же должна удовлетворять следующим требованиям:

— удобство пользователя при работе;

— возможность задания пользователем размера графа (множества);

— возможность выбора автоматического (случайного) или ручного (с клавиатуры) задания графа в виде матрицы смежности;

— возможность сохранения результата в файл.

# 2. ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ

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. Максимальное паросочетание можно найти простым жадным алгоритмом. Жадный алгоритм (англ. *Greedy algorithm*) — алгоритм, заключающийся в принятии локально оптимальных решений на каждом этапе, допуская, что конечное решение также окажется оптимальным.

Рассмотрим теорему о максимальном паросочетании и дополняющих цепях. Для этого введём понятия чередующейся цепи, увеличивающей цепи и двудольного графа:

— Чередующаяся цепь – это путь, в котором рёбра поочерёдно принадлежат паросочетанию и не принадлежат ему.

— Увеличивающая цепь (дополняющая цепь) – чередующаяся цепь, у которой начальная и конечная вершины не принадлежат паросочетанию.

— Двудольный граф – граф, вершины которого можно разбить на два множества (две доли), так что каждое ребро графа соединяет вершины из разных долей.

Теорема: Паросочетание M в двудольном графе G является максимальным тогда и только тогда, когда в графе G нет дополняющей цепи.

Доказательство:

Пусть в двудольном графе G с максимальным паросочетанием M существует дополняющая цепь. Тогда пройдя по ней и заменив вдоль неё все рёбра, входящие в паросочетание, на не входящие и наоборот, мы получим большее паросочетание. То есть M не являлось максимальным. Противоречие.

Рассмотрим паросочетание M в графе G и предположим, что M — не наибольшее. Докажем, что тогда имеется увеличивающая цепь относительно M. Пусть Mmax — другое паросочетание и |Mmax|>|M|. Рассмотрим подграф H графа G, образованный теми рёбрами, которые входят в одно и только в одно из паросочетаний M, Mmax. Иначе говоря, множеством рёбер графа H является симметрическая разность M⊕Mmax. Иллюстрация симметрической разности M⊕Mmax представлена на рисунке 1.

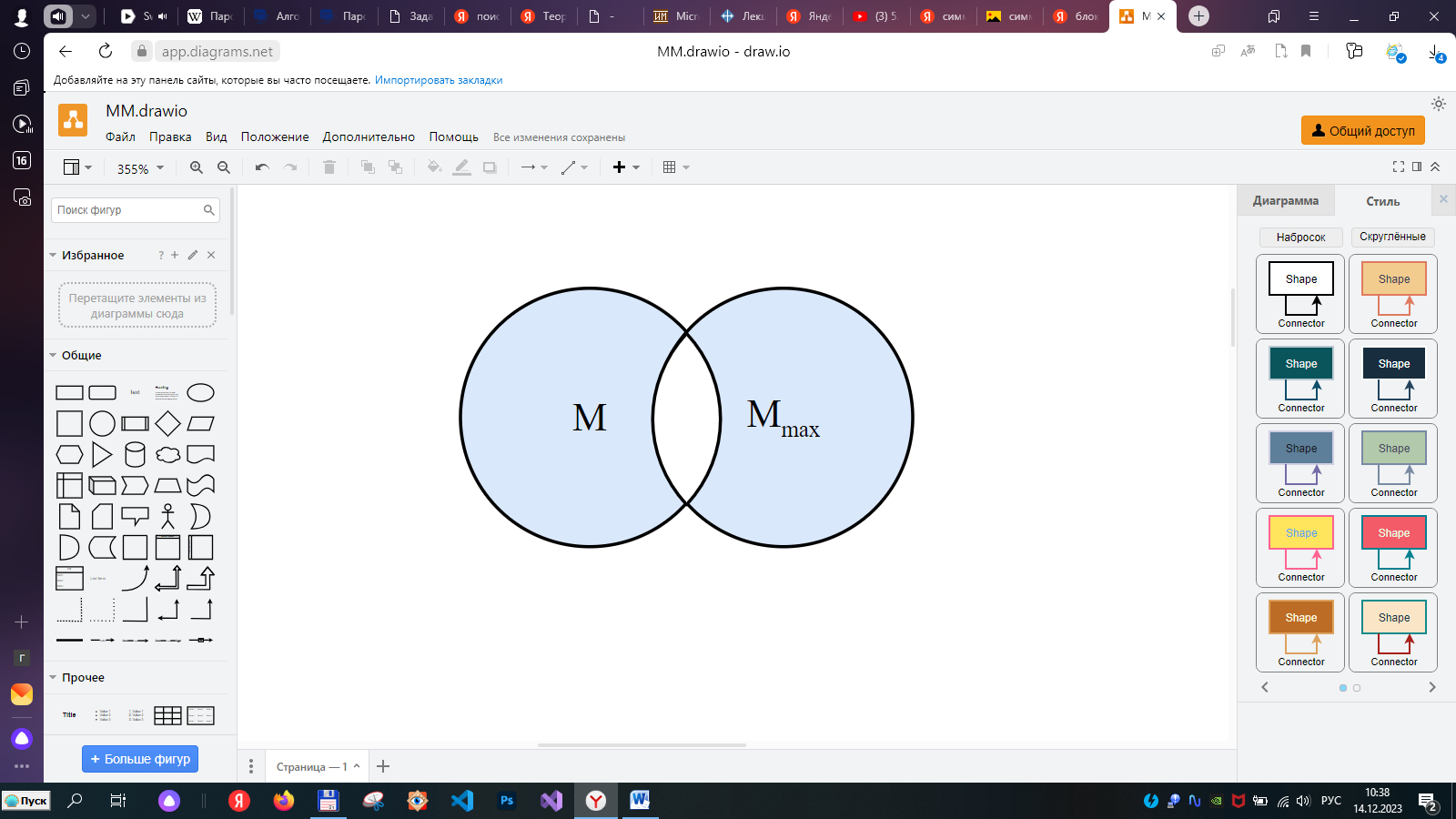


Рисунок 1 – M⊕Mmax

В графе H каждая вершина инцидентна не более чем двум рёбрам (одному из M и одному из Mmax), т.е. имеет степень не более двух. В таком графе каждая компонента связности — путь или цикл. В каждом из этих путей и циклов чередуются рёбра из M и Mmax. Так как |Mmax|>|M|, имеется компонента, в которой рёбер из Mmax содержится больше, чем рёбер из M. Это может быть только путь, у которого оба концевых ребра принадлежат Mmax. Заметим, что относительно M этот путь является увеличивающей (дополняющей) цепью. Дополняющая цепь относительно М представлена на рисунке 2.

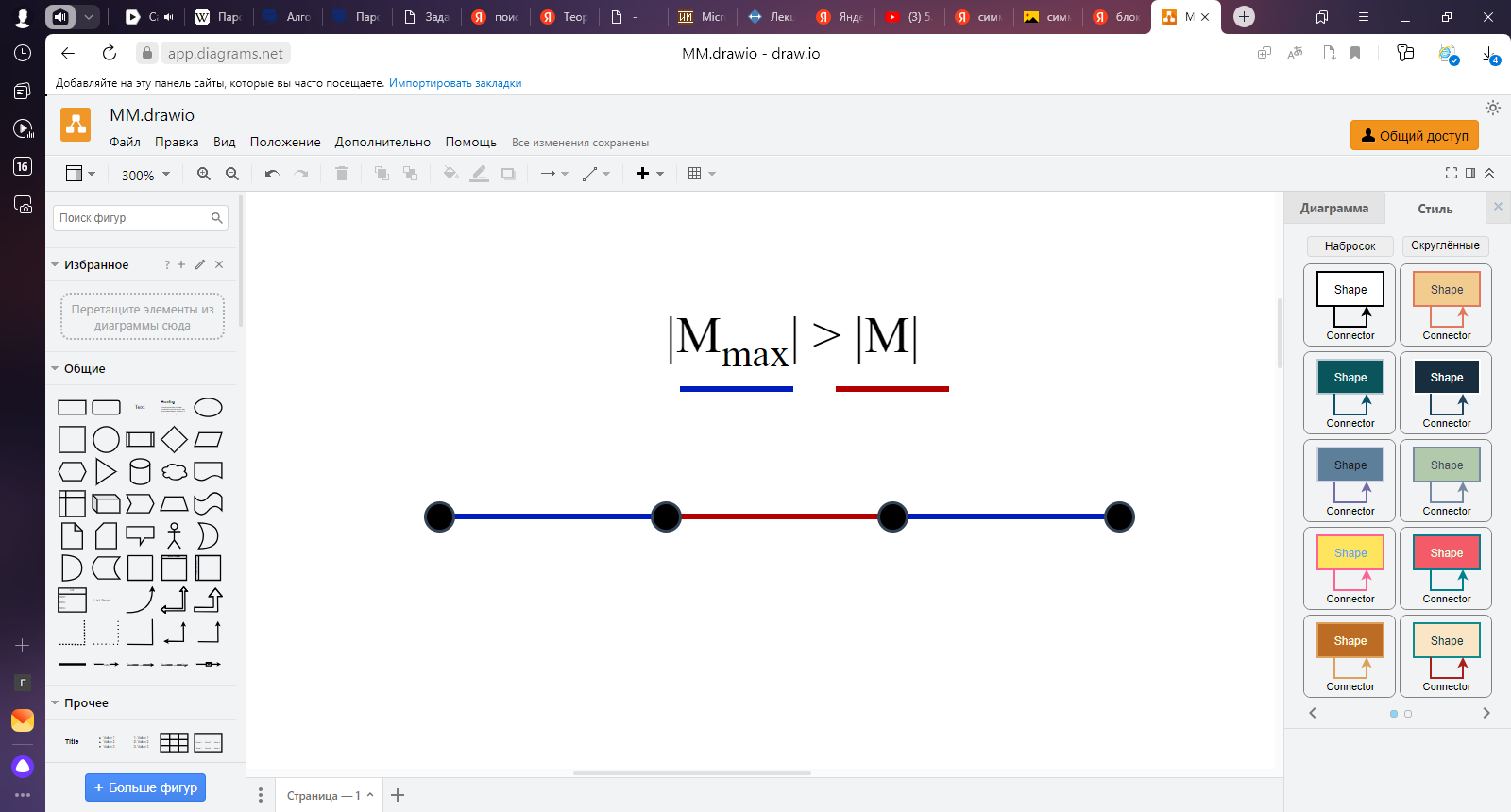


Рисунок 2 – Компонента, в которой |Mmax|>|M|

# 3. ОПИСАНИЕ РАЗРАБОТКИ ПРОГРАММЫ

Пусть задан двудольный граф G(v, e). 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.

Для программной реализации алгоритма понадобятся массивы *matching[]* и *tops\_visited[]*. Массив *matching[]* предназначен для хранения паросочетаний. Предварительно все ячейки массива заполняются числами –1. Если паросочетания с вершиной *v* не существует, то *matching[v]* = –1. Массив *tops\_visited[]* – массив для учёта посещённых вершин в обходе в глубину. Он нужен, чтобы обход в глубину не заходил в одну вершину дважды.

Внутри функции *DFS()*просматриваются все рёбра, исходящие из вершины *v*, и затем проверяется: если это ребро ведёт в ненасыщенную вершину *i* (ненасыщенная вершина в двудольном графе – это вершина, которая не соединена ни с одной вершиной из противоположной доли), либо если эта вершина *i* насыщена, но удаётся найти увеличивающую цепь рекурсивным запуском из *matching[i]*, то мы говорим, что мы нашли увеличивающую цепь, и перед возвратом из функции с результатом *true* производим чередование в текущем ребре: перенаправляем ребро, смежное с *i*, в вершину *v*. Таким образом, функция возвращает *true*, если ей удалось найти увеличивающую цепь из вершины v, при этом считается, что эта функция уже произвела чередование паросочетания вдоль найденной цепи.

В основной программе сначала указывается, что текущее паросочетание — пустое (массив *matching[]* заполняется числами –1). Затем перебирается вершина *v*, и из неё запускается обход в глубину *DFS()*, предварительно обнулив массив *tops\_visited[]*.

Алгоритм ровно *n* раз ищет увеличивающий путь, каждый раз просматривая не более чем *m* рёбер, а значит, работает за *O(nm)*.

Ниже представлен псевдокод функции *DFS()*, и её вызов в основной программе.

**DFS()**

1 ЕСЛИ (tops\_visited [v] == 1)

2 ВОЗВРАТИТЬ false

3 tops\_visited [v] = 1

4 ДЛЯ всех i ИЗ G[i][j]:

5 ЕСЛИ (matching[i] == -1 или DFS(matching[i]))

6 matching[i] = v

7 ВОЗВРАТИТЬ true

8 ВОЗВРАТИТЬ false

**main()**

1 ДЛЯ всех v matching[v] = -1;

2 ДЛЯ всех i ИЗ G[i][j]

3 tops\_visited [i] = 0

4 **DFS()**

Полный код программы можно увидеть в Приложении.

# 5. ОПИСАНИЕ ПРОГРАММЫ

При запуске программы происходит вывод на экран *Windows Forms* формы – визуальной поверхности, на которой выводится информация для пользователя.

Для начала работы алгоритма необходимо задать размер графа, который выводится в виде матрицы смежности.

Если пользователь выбрал кнопку «Ввести элементы графа самостоятельно», то выводится матрица смежности, заполненная нулями. В первом цикле *for* также формируются заголовки для столбцов и строк, путём прибавления к переменной *i* (начальное значение которой равно 0) единицы.

for (int i = 0; i < n; i++) { //формирование матрицы смежности М

for (int j = 0; j < n; j++) { //для самостоятельного заполнения

dataGridView1->Rows[i]->Cells[j]->Value = 0;

}

dataGridView1->Rows[i]->HeaderCell->Value = System::Convert::ToString(i + 1);

dataGridView1->Columns[i]->HeaderCell->Value = System::Convert::ToString(i + 1);

}

Пользователю необходимо задать в матрице смежности нужные связи между вершинами.

При нажатии кнопки «Сгенерировать граф с помощью случайных чисел» используется функция *srand()*. Вводится переменная *n\_last\_first\_part,* которая равняется половине размера матрицы смежности. В двудольном графе вершины одной доли связи иметь не могут, поэтому в матрице смежности две четверти заполняются нулями (*j*< *n\_last\_first\_part* и *i*< *n\_last\_first\_part* или *j* >=*n\_last\_first\_part* и *i*>= *n\_last\_first\_part*). Также исключены петли, поэтому главная диагональ матрицы также заполняются нулями (*i*== *j*). Оставшаяся часть матрицы заполняется симметрично для задания неориентированного графа. В первом цикле *for* также формируются заголовки для столбцов и строк, путём прибавления к переменной *i* (начальное значение которой равно 0) единицы.

srand(time(0));

for (int i = 0; i < n; i++) { //формирование матрицы смежности М

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

if (j < n\_last\_first\_part && i < n\_last\_first\_part) {

dataGridView1->Rows[i]->Cells[j]->Value = 0;

}

else {

if (j >= n\_last\_first\_part && i >= n\_last\_first\_part) {

dataGridView1->Rows[i]->Cells[j]->Value = 0;

}

else {

if (i == j) dataGridView1->Rows[i]->Cells[j]->Value = 0;

else {

dataGridView1->Rows[i]->Cells[j]->Value = rand() % 2;

dataGridView1->Rows[j]->Cells[i]->Value =

dataGridView1->Rows[i]->Cells[j]->Value;

}

}

}

}

dataGridView1->Rows[i]->HeaderCell->Value = System::Convert::ToString(i + 1);

dataGridView1->Columns[i]->HeaderCell->Value = System::Convert::ToString(i + 1);

}

Перед поиском максимального паросочетания происходит определение типа графа: ориентированный или неориентированный. Для этого проверяется симметричность матрицы смежности. Если найдены различные симметричные элементы, устанавливается флаг *flag\_orient*, на экран выводится предупреждение.

int flag\_orient = 0;

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

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

if (i != j) {

Object^ tmp1 = dataGridView1->Rows[j]->Cells[i]->Value;

int tmp1\_int = System::Convert::ToInt64(tmp1);

Object^ tmp2 = dataGridView1->Rows[i]->Cells[j]->Value;

int tmp2\_int = System::Convert::ToInt64(tmp2);

if (tmp1\_int != tmp2\_int) {

flag\_orient++;

}

}

}

}

if (flag\_orient > 0) MessageBox::Show("Предупреждение: задан ориентированный граф");

Поиск максимальных паросочетаний осуществляется с помощью алгоритма Куна. Для этого создаётся массив *matching[]*, предназначенный для сохранения результата. А также массив посещенных вершин *tops\_visited[]*. Из каждой вершины матрицы смежности происходит вызов функции *DFS()*.

int\* tops\_visited = (int\*)malloc(n \* sizeof(int\*));

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

memset(matching, -1, n \* sizeof(int\*));

for (int i = 0; i < n; i++) { //алг Куна

for (int i\_i = 0; i\_i < n; i\_i ++) {

tops\_visited[i\_i] = 0;

}

DFS(i, n, tops\_visited, matching);

}

bool DFS(int v, int n, int\* tops\_visited, int\* matching) {

if (tops\_visited[v] == 1) return false;

tops\_visited[v] = 1;

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

Object^ tmp = dataGridView1->Rows[v]->Cells[i]->Value;

int tmp\_int = System::Convert::ToInt64(tmp);

if (tmp\_int > 0) {

if ((matching[i] == -1) || (DFS(matching[i], n, tops\_visited, matching))) {

matching[i] = v;

return true;

}

}

}

return false;

}

Для ориентированного графа предусмотрена дополнительная проверка для корректного вывода результата. Создается массив *mas\_already\_output[].* Каждый элемент массива *matching[]* сравнивается с элементами массива *mas\_already\_output[]* и добавляется в его конец. При добавлении нового элемента увеличивается на 1 переменная *size\_dataGridView2*.

int\* mas\_already\_output = (int\*)malloc(n \* sizeof(int\*));

memset(mas\_already\_output, 0, n \* sizeof(int\*));

int flag\_already\_out = 0;

int size\_dataGridView2 = 0;

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

if (matching[i] + 1 != 0) {

flag\_already\_out = 0;

for (int i\_already = 0; i\_already < n; i\_already++) {

if (matching[i] + 1==mas\_already\_output[i\_already])flag\_already\_out=1;

}

if (flag\_already\_out == 1) continue;

mas\_already\_output[i] = i + 1;

size\_dataGridView2++;

}

}

Количество строк в результирующей таблице задаётся переменной *size\_dataGridView2*. Это позволяет избежать пустых ячеек.

this->dataGridView2->ColumnCount = 2;

this->dataGridView2->RowCount = size\_dataGridView2;

Для исключения вывода повторяющихся элементов организован следующий цикл, в котором выводятся только элементы, записанные в массив *mas\_already\_output[]*.

int i\_row = 0;

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

if (matching[i] + 1 != 0) {

flag\_already\_out = 0;

for (int i\_already = 0; i\_already < n; i\_already++) {

if (matching[i] + 1 == mas\_already\_output[i\_already]) flag\_already\_out = 1;

}

if (flag\_already\_out == 1) continue;

this->dataGridView2->Rows[i\_row]->Cells[0]->Value = matching[i]+1;

this->dataGridView2->Rows[i\_row]->Cells[1]->Value = i + 1;

i\_row++;

}

}

При нажатии на кнопку «Сохранить результат» происходит запись всех элементов результирующей таблицы в файл *max\_ matching.txt* с помощью функции *fprintf()*.

int n\_last\_first\_part = n / 2;

FILE\* file;

file = fopen("max\_matching.txt", "w");

int size\_dataGridView2 = dataGridView2->RowCount;

for (int i = 0; i < size\_dataGridView2; i++) {

for (int j = 0; j < 2; j++) {

Object^ tmp = dataGridView2->Rows[i]->Cells[j]->Value;

int tmp\_el = System::Convert::ToInt64(tmp);

fprintf(file, "%d", tmp\_el);

if (j == 0) fprintf(file, " - ");

}

fprintf(file, "\n");

}

fclose(file);

Схема данных представлена на рисунке 10.



Рисунок 3 - Схема данных

# 4. ОТЛАДКА И ТЕСТИРОВАНИЕ

В качестве среды разработки была выбрана программа Microsoft Visual Studio 2022. Для отладки использовались несколько возможностей Visual Studio: точка останова, трассировка, анализ содержимого переменных. Тестирование проводилось во время разработки и также после завершения разработки.

Точка останова - это преднамеренная остановка или приостановка в программе, установленная в целях отладки. Точка останова - это средство получения знаний о программе во время ее выполнения. Во время прерывания есть возможность проверить тестовую среду (регистры общего назначения, память, журналы, файлы и т.д.), чтобы выяснить, функционирует ли программа должным образом.

Трассировка — процесс пошагового выполнения программы. В режиме трассировки программист видит последовательность выполнения команд и значения переменных на данном шаге выполнения программы, что позволяет легче обнаруживать ошибки. Трассировка может быть начата и окончена в любом месте программы, выполнение программы может останавливаться на каждой команде или на точках останова, трассировка может выполняться с заходом в процедуры и без заходов, а также осуществляться в обратном порядке.

Точка останова для проверки корректного заполнения матрицы смежности графа случайными числами представлена на рисунке 4.

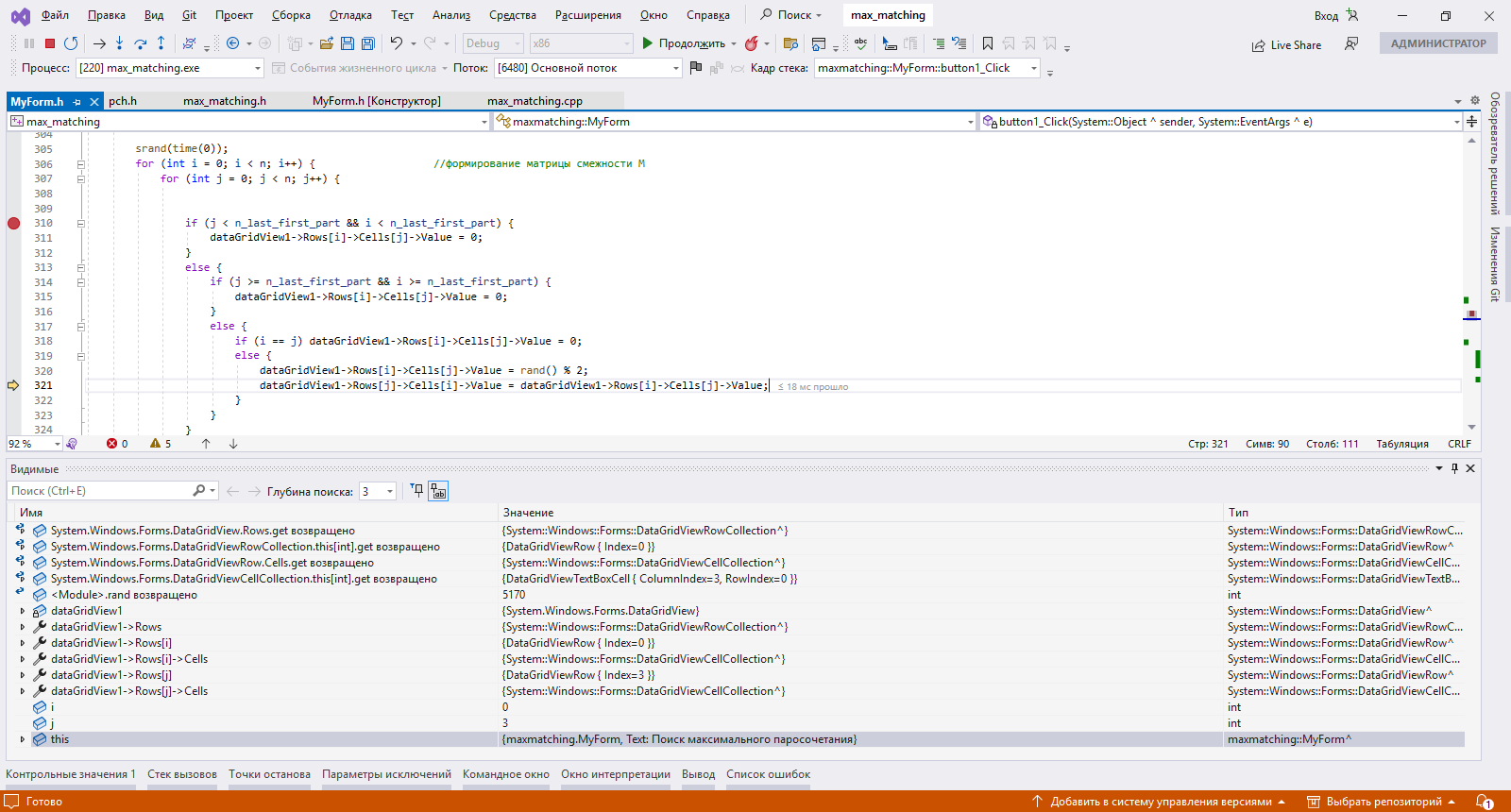


Рисунок 4 - Проверка корректного заполнения матрицы смежности графа

случайными числами

Точка останова для проверки корректности работы функции *DFS()* представлена на рисунке 5.

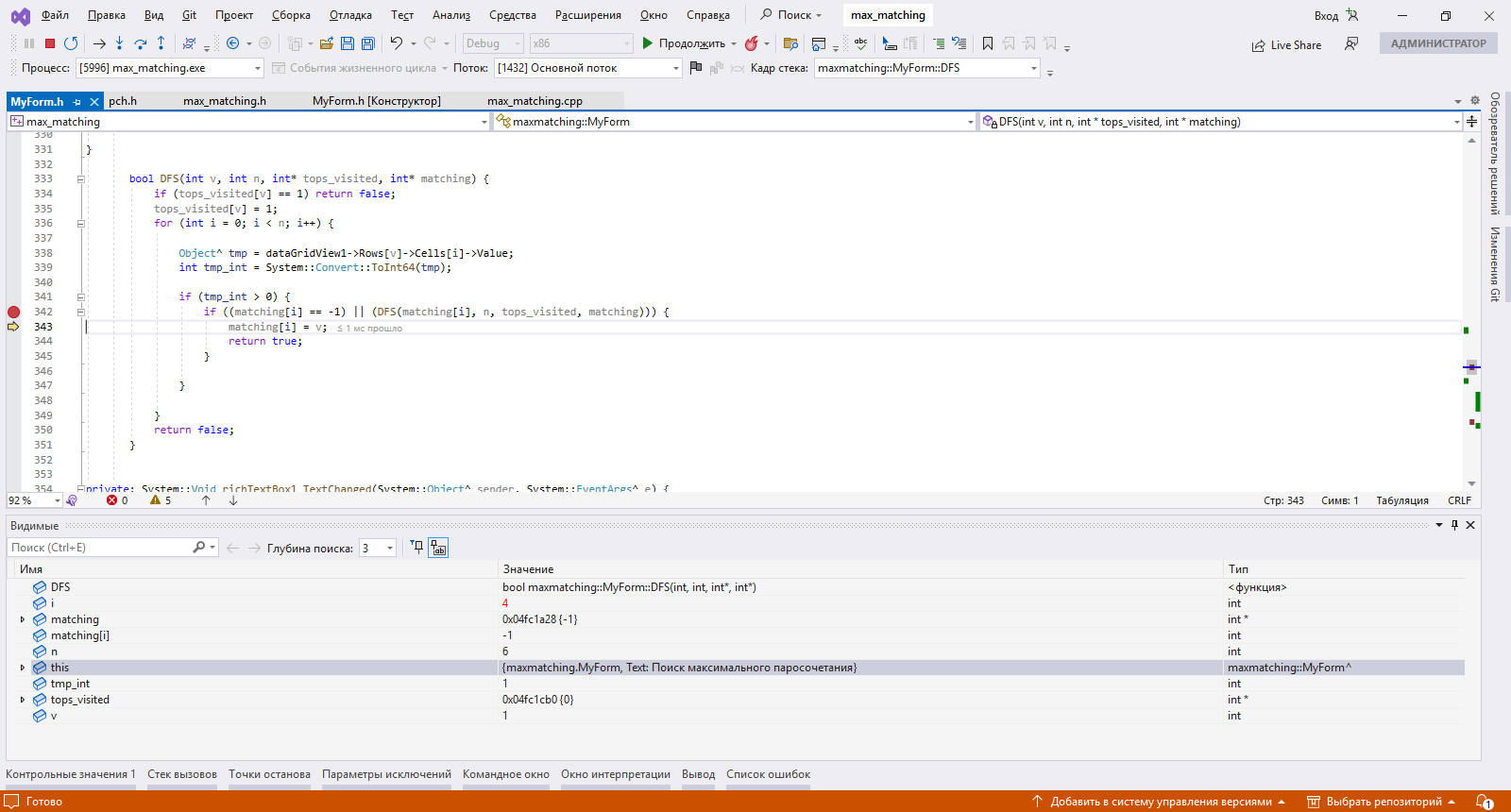


Рисунок 5 - Проверка корректности работы функции *DFS()*

Точка останова для проверки корректного вывода результата представлена на рисунке 6.

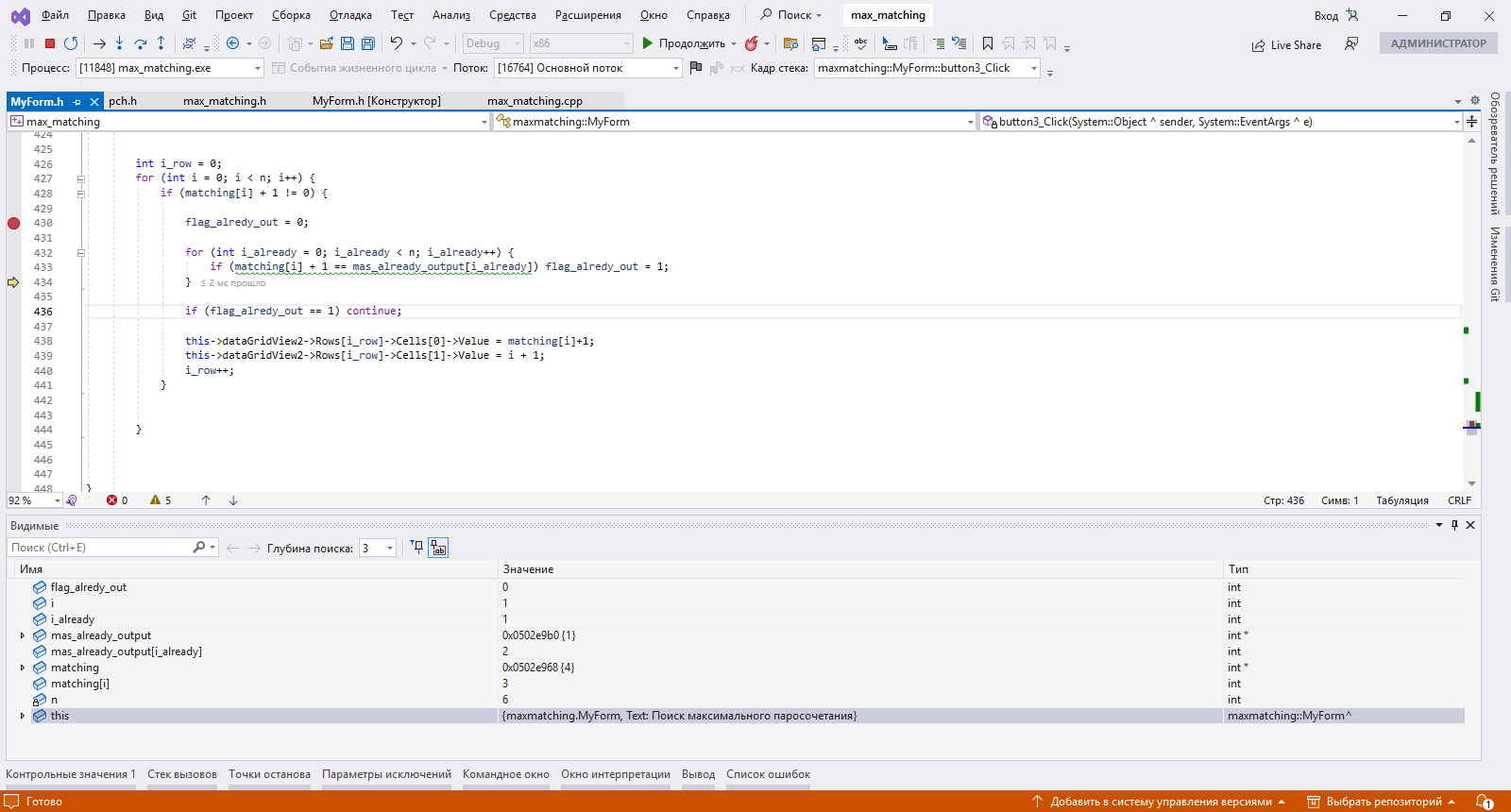


Рисунок 6 - Проверка корректного вывода результата

На рисунке 7 представлена работа программы при заданном пустом множестве.

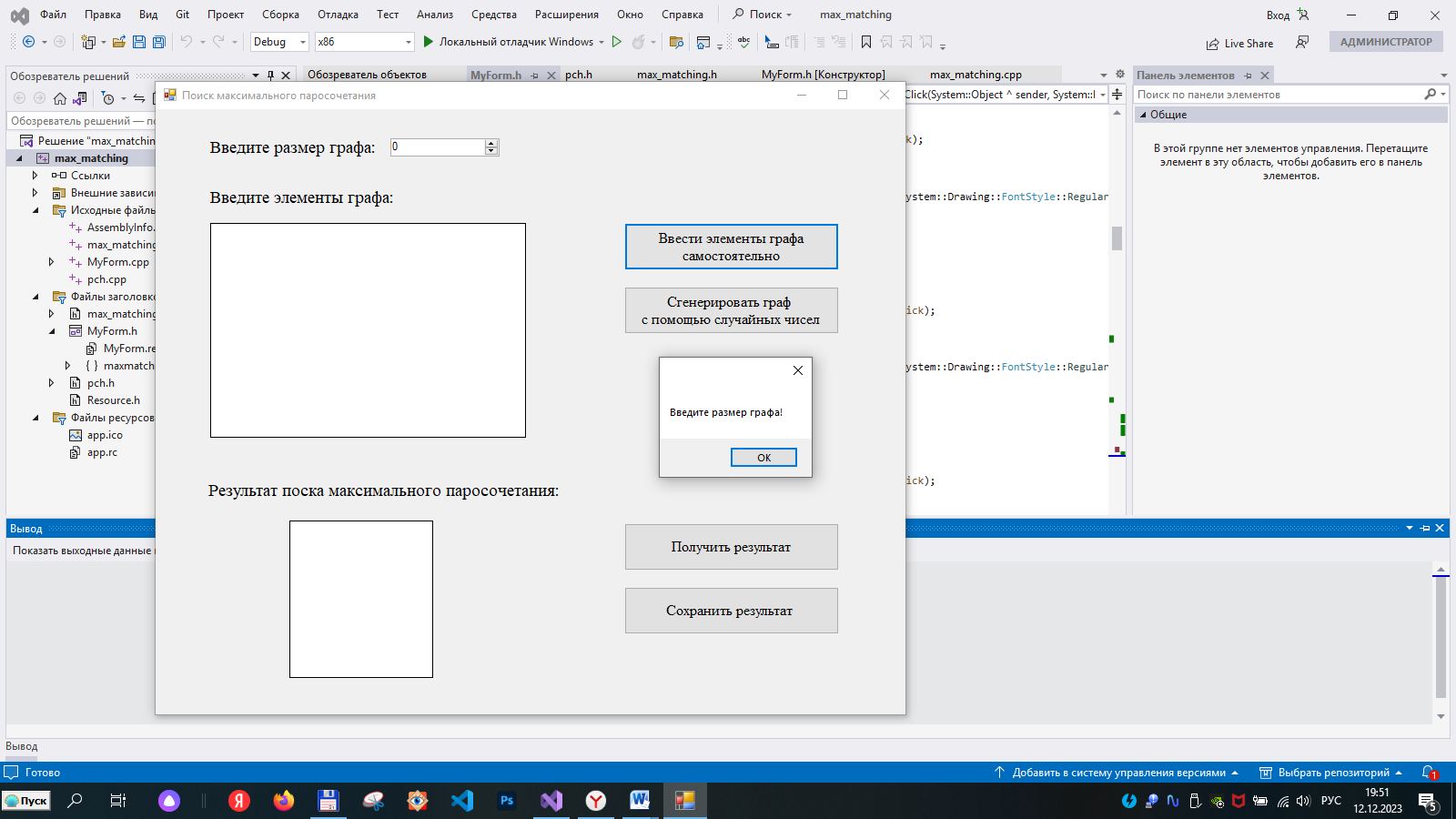


Рисунок 7 - Работа программы при заданном пустом множестве

На рисунке 7 представлена работа программы при заданном множестве, превышающем 50 элементов. Программа выдаёт предупреждение и задаёт граф с максимально возможным количеством вершин, равным 50.

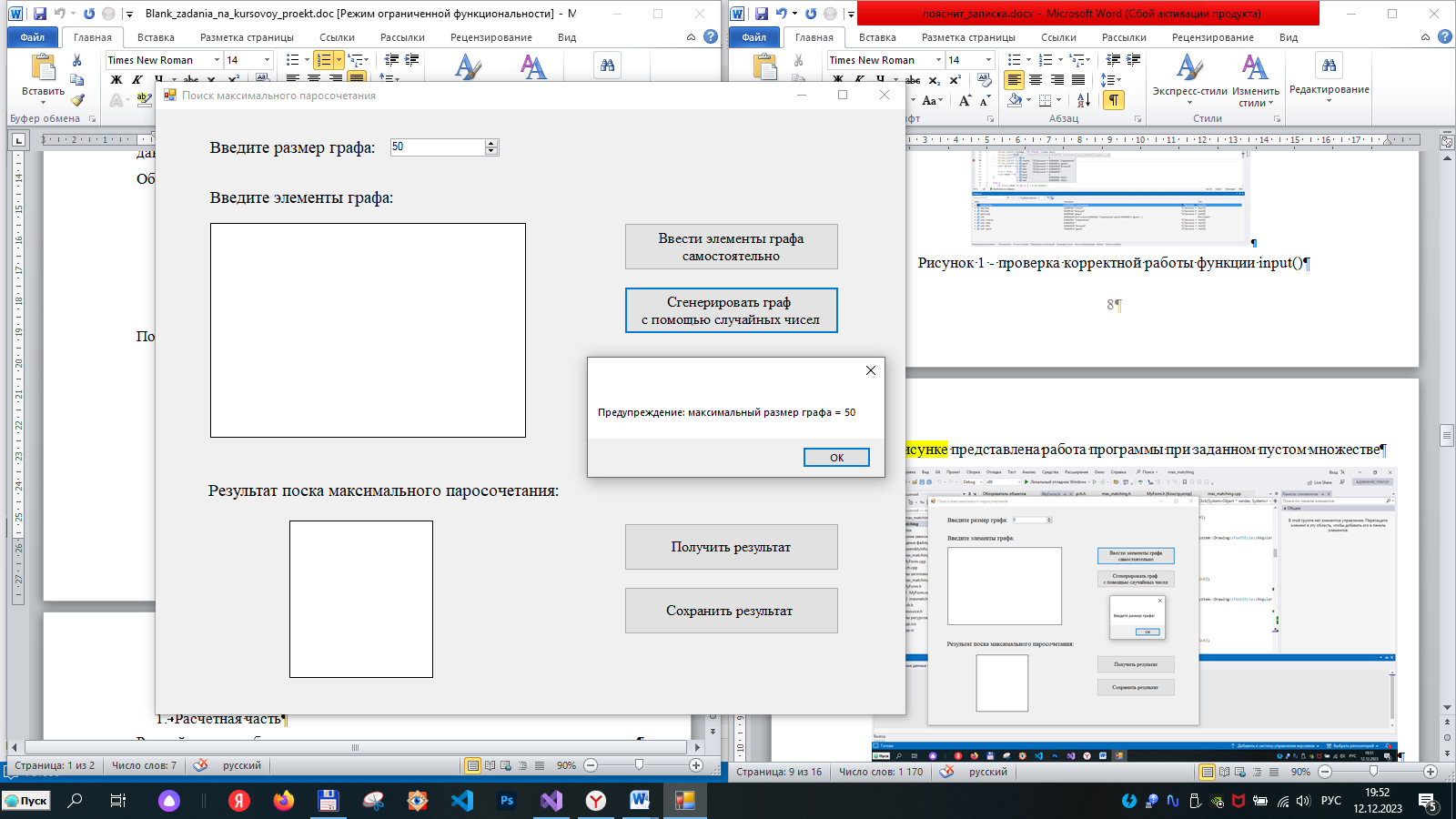


Рисунок 8 - Работа программы при заданном множестве, превышающем максимально возможное количество

На рисунке 9 представлено формирование матрицы смежности для самостоятельного ввода элементов.

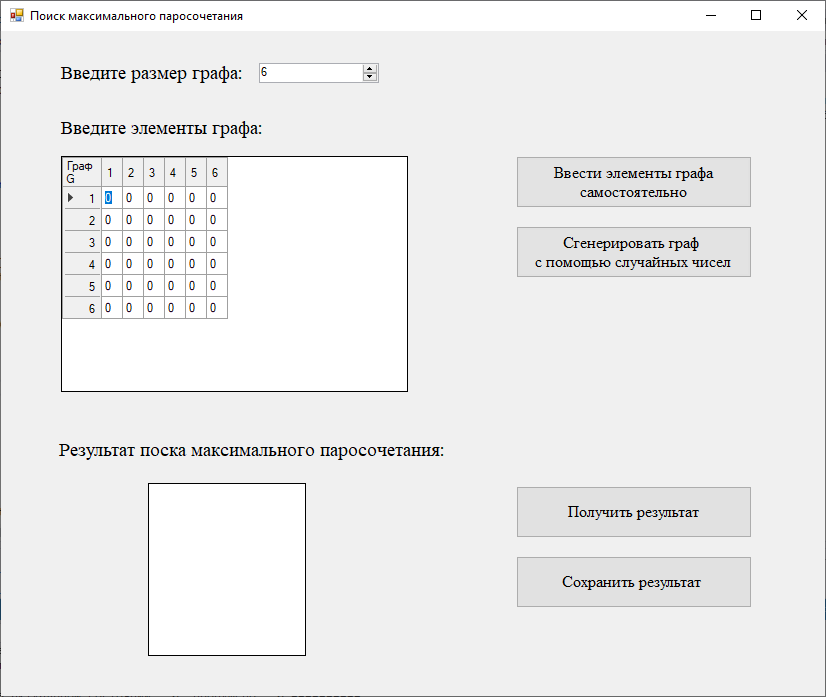


Рисунок 9 - Формирование матрицы для самостоятельного ввода элементов

На рисунке 10 представлена работа программы при заданном ориентированном графе.

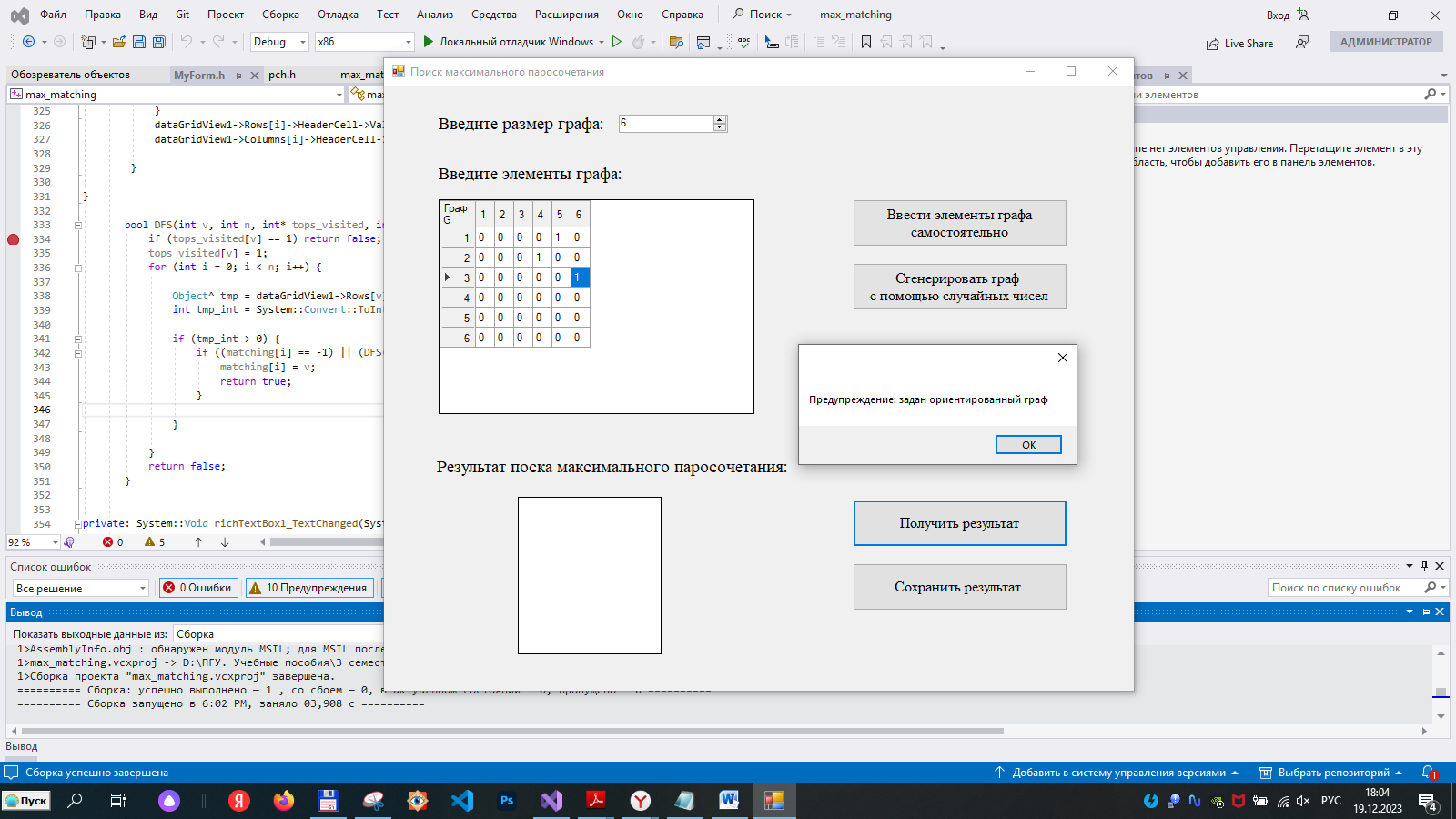


Рисунок 10 - Матрица смежности для ориентированного графа

На рисунке 11 представлен результат поиска максимального паросочетания для ориентированного графа.

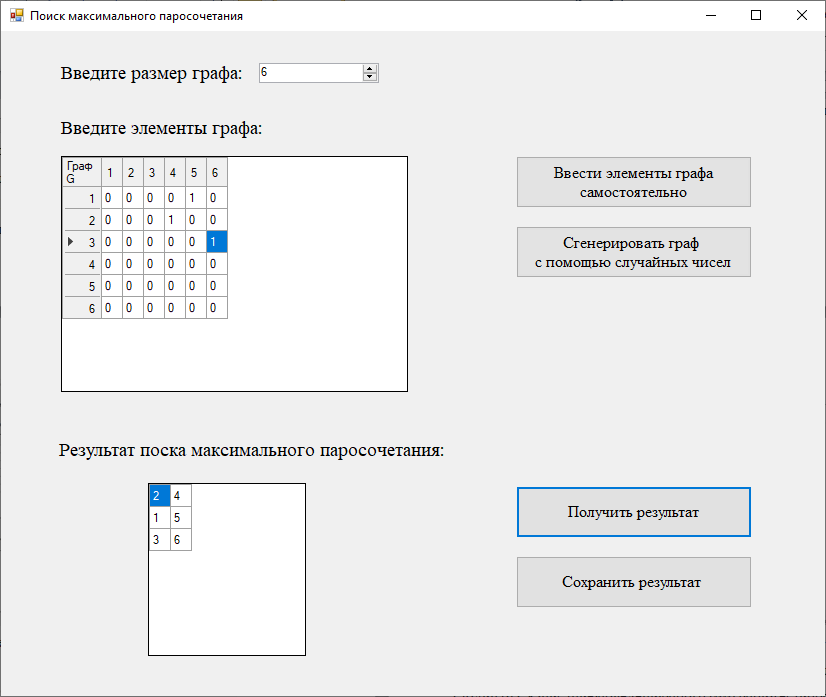


Рисунок 11 - результат поиска максимального паросочетания

для ориентированного графа

На рисунке 12 представлена работа программы при сохранении результата в файл

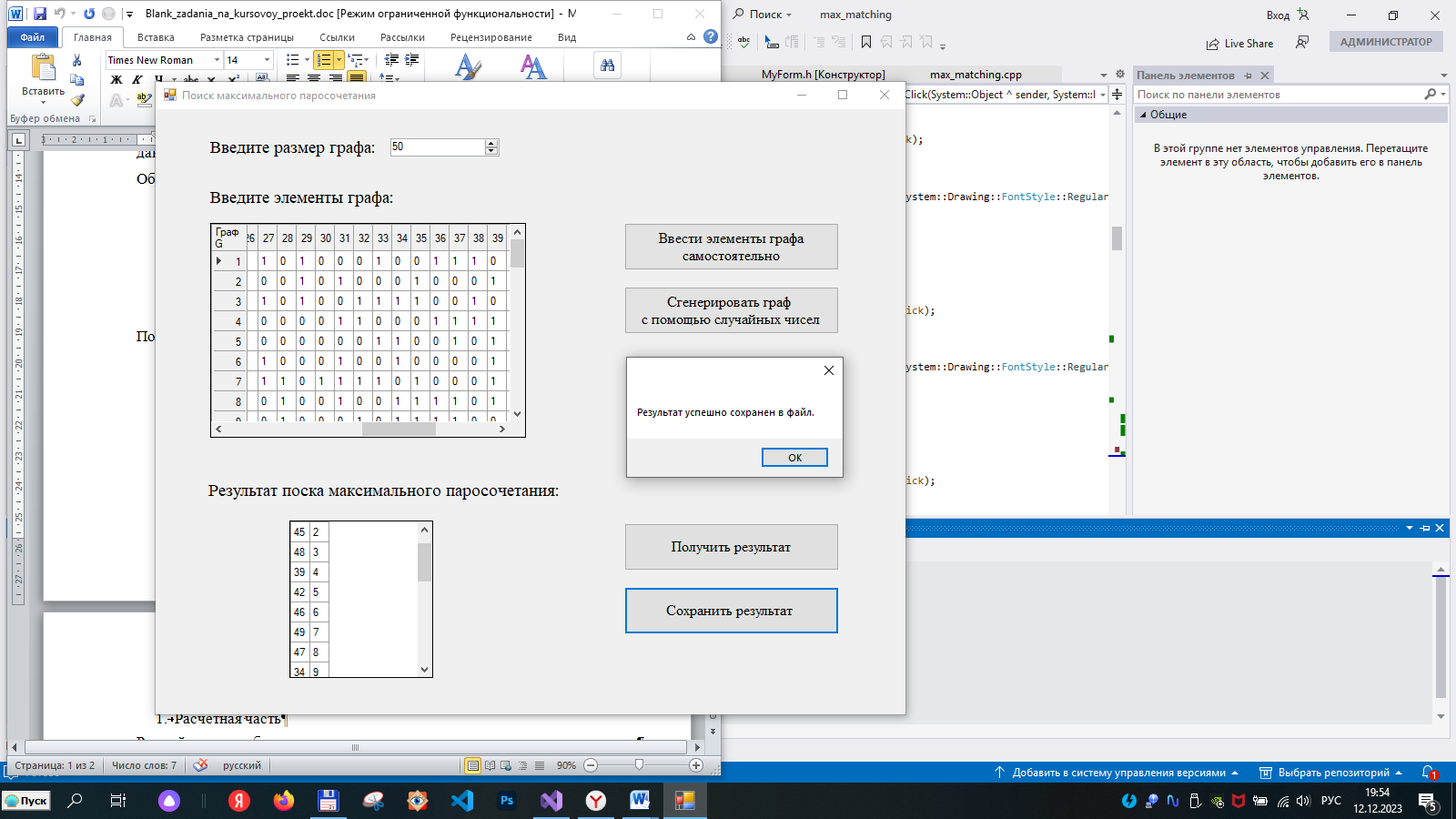


Рисунок 12 - Сохранение результата в файл

На рисунке 13 представлен результат, сохранённый в файл max\_matching.txt

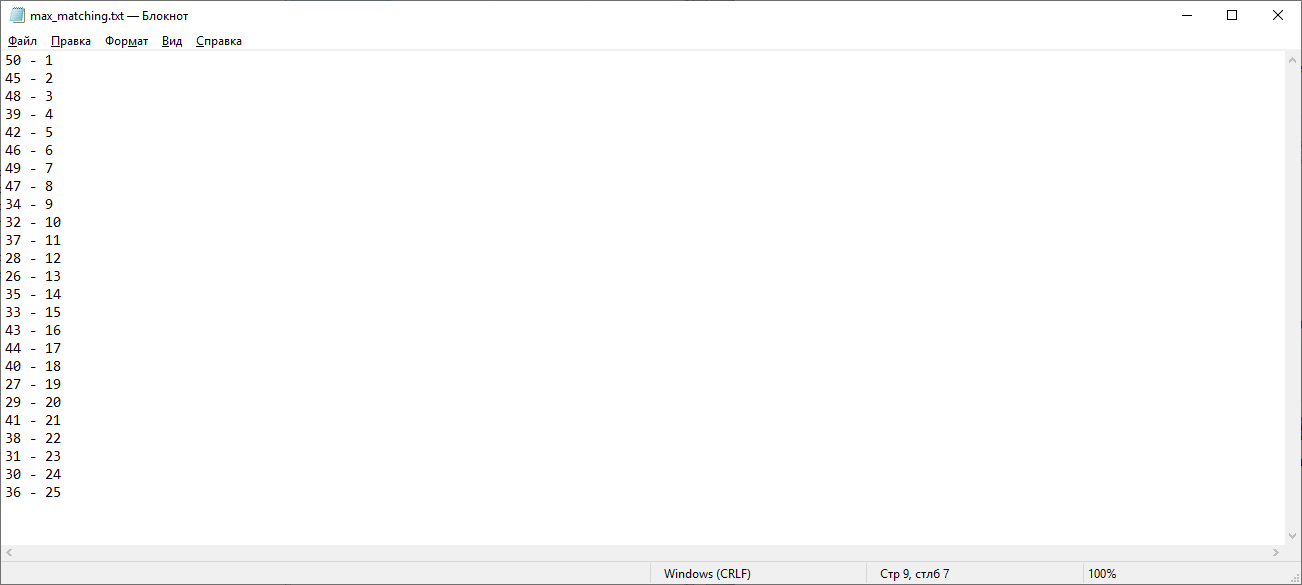


Рисунок 13 - Результат, сохранённый в файл

Описание поведения программы при тестировании выполнено в таблице 1.

Таблица 1 – Описание поведения программы при тестировании

|  |  |  |
| --- | --- | --- |
| Описание теста | Ожидаемый результат | Полученный результат |
| Получение результата при заданном пустом множестве | Вывод напоминания о введении размера графа | Верно  (рисунок 7) |
| Ввод максимально возможного размера графа | Вывод предупреждения и дальнейшая работа приложения | Верно  (рисунок 7) |
| Самостоятельное формирование матрицы смежности графа | Вывод матрицы, заполненной нулями, заданного размера с возможностью редактирования элементов | Верно  (рисунок 8) |
| Ввод ориентированного графа | Вывод предупреждения о заданном ориентированном графе. Вывод результирующей таблицы. | Верно  (рисунок 10) |
| Сохранение результата | Создание файла, содержащего элементы результирующей таблицы | Верно  (рисунок 12) |

В результате тестирования программы ошибок не было выявлено.

# 6. РУЧНОЙ РАСЧЁТ ЗАДАЧИ

Проведем проверку программы посредством ручных вычислений на примере графа *G* с количеством вершин, равным 8. На рисунке14 представлен граф *G*.

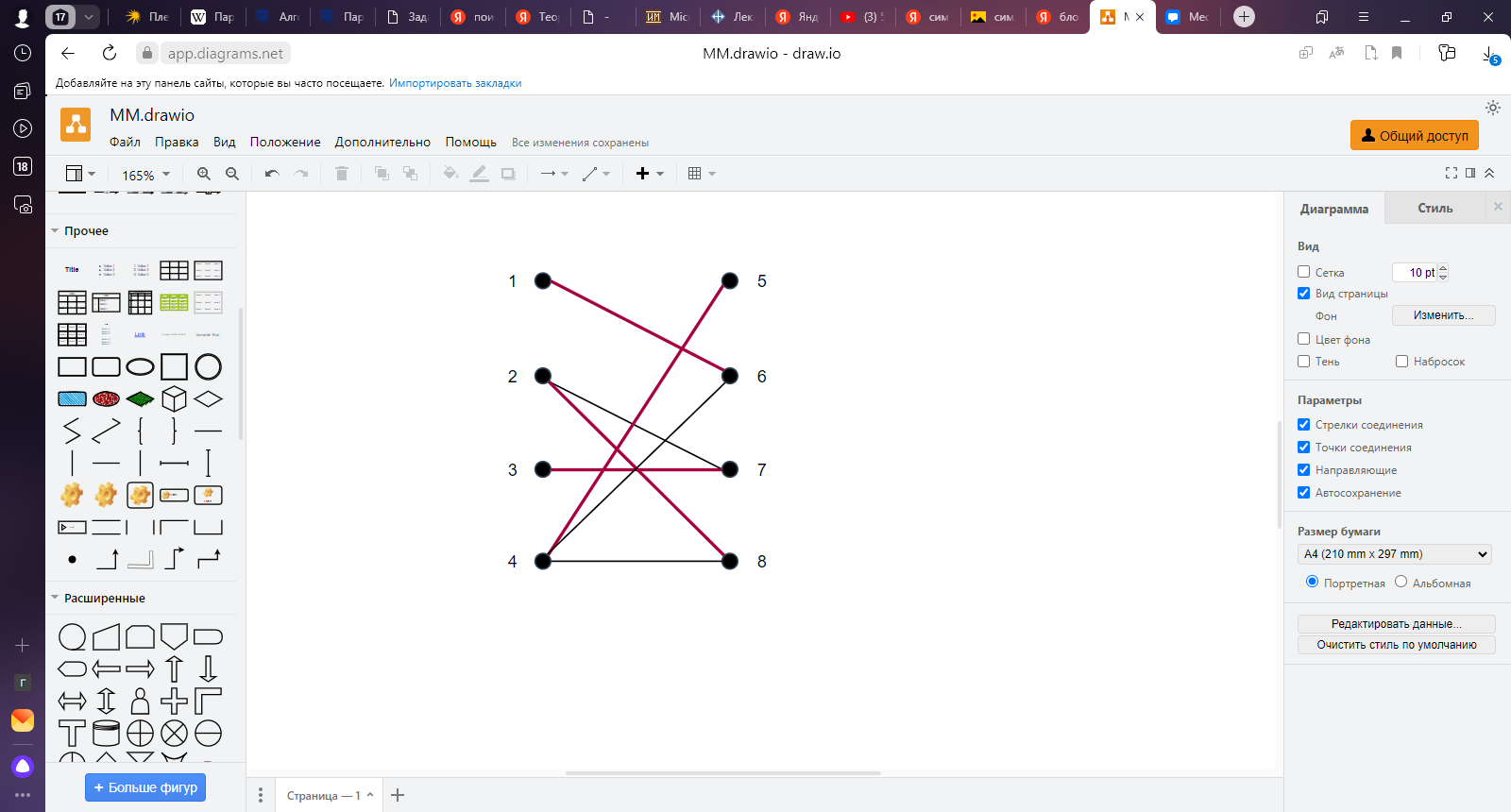


Рисунок 14 - Граф G

Матрица смежности графа G представлена на рисунке 15

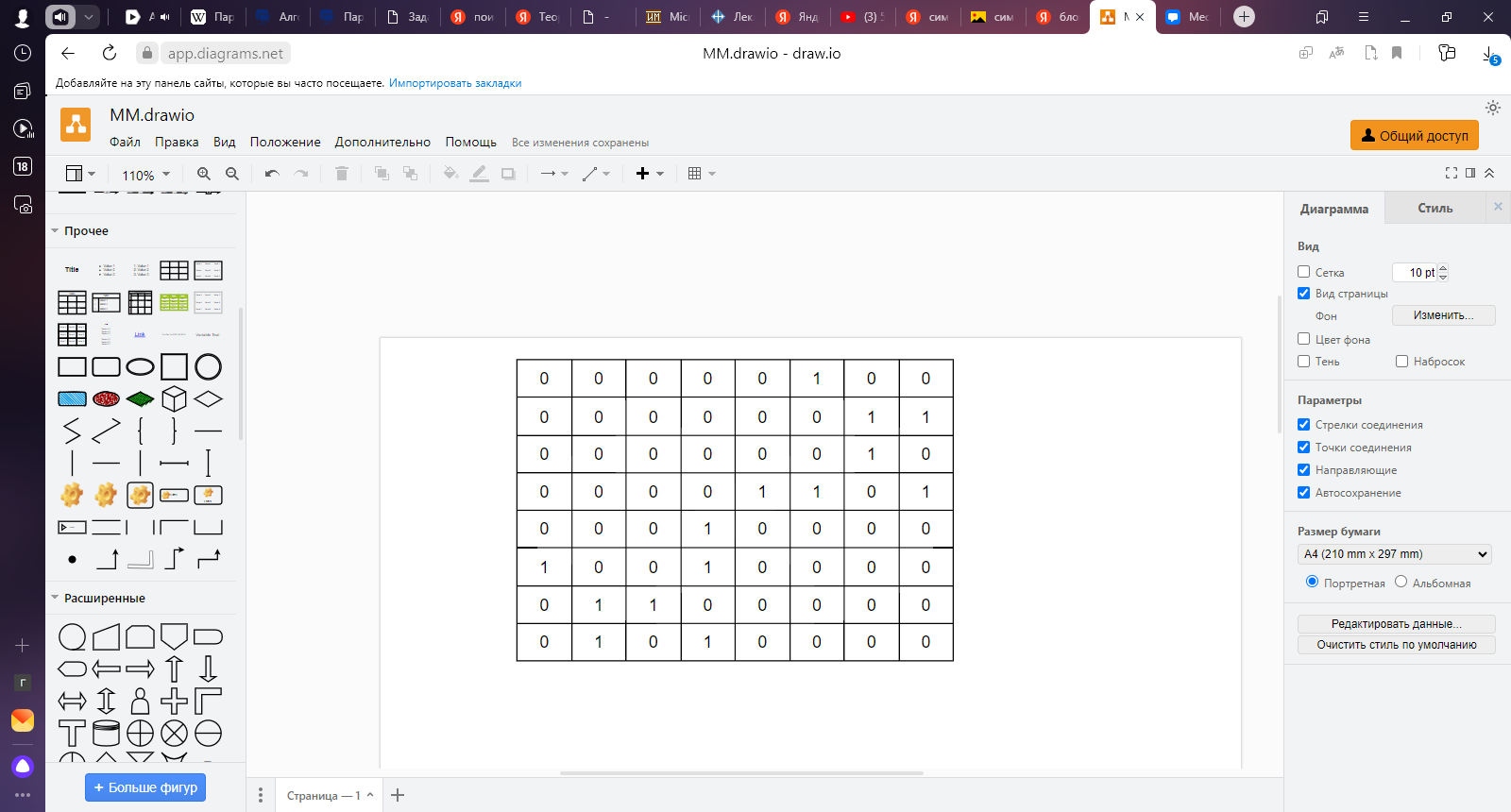


Рисунок 15 - Матрица смежности графа G

Начинаем с вершины №1. Отмечаем её как посещённую: *tops\_visited*[0] = 1 (порядковый номер вершины в программе = 0). Проходим по матрице смежности и смотрим, с какими вершинами связана первая вершина: только вершина №6. В массиве *matching*[5] = 0.

Вершина №2. Отмечаем её как посещённую: *tops\_visited*[1] = 1. Вершина №2 связана с вершинами №7 и №8. У вершины №7 паросочетание не определено, поэтому *matching*[6] = 1.

Вершина №3. Отмечаем её как посещённую: *tops\_visited*[2] = 1. Вершина №3 связана только с вершиной №7. Но вершина №7 была определена в пару с вершиной №2, поэтому вызывается функция *DFS()*. Отмечаем её как посещённую: *tops\_visited*[1] = 1. Вершина №2 связана с вершинами №7 и №8. Вызывается *DFS(),* которая возвращает *false*, так как вершина №7 уже была посещена. Продолжается цикл для вершины №3, переходим на вершину №8. Вершина №8 не была посещена, *matching*[7] = 1. Возврат из функции, *matching*[6] = 2.

Вершина №4. Отмечаем её как посещённую: *tops\_visited*[3] = 1. Вершина №4 связана с вершинами №5, №6, №8. Вершина №5 не посещенная, поэтому *matching*[4] = 3.

На рисунке 14 выделено максимальное паросочетание графа *G*.

На рисунке 16 представлена работа программы для данного графа *G*.

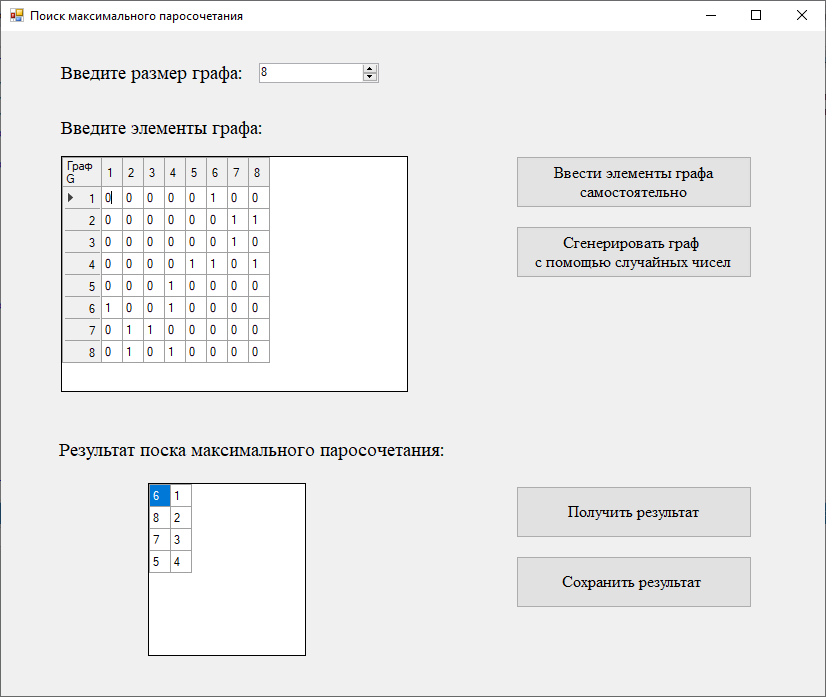


Рисунок 16 - Тестирование программы

Ручные расчеты совпали с результатами программы.

# ЗАКЛЮЧЕНИЕ

Главным результатом проведенной работы является создание функционирующей программы, которая выполняет требуемый от неё круг задач. Она предоставляет возможность задать желаемый двудольный граф, найти максимальные паросочетания и сохранить результат в файл.

В дальнейшем программу можно улучшить путём учёта возможности выбора вида графа: двудольного или произвольного. Для этого необходимо создать функцию поиска максимального паросочетания для произвольного графа.

В ходе выполнения курсового проекта были получены навыки разработки программ с использованием платформы *Windows Forms*. Также были закреплены навыки по осуществлению алгоритма поиска в глубину и углублены знания языка программирования Cи.

# СПИСОК ИСПОЛЬЗУЕМЫХ ИСТОЧНИКОВ

1. Омельченко А.В., Теория графов. М.: МЦНМО, 2018. 416
2. Подбельский В.В., Фомин С.С., Курс программирования на языке Си: учебник. — М.: ДМК Пресс, 2012. — 384с.
3. Рафгарден Т., Совершенный алгоритм. Графовые алгоритмы и структуры данных. — СПб.: Питер, 2019. — 256 с.
4. Керниган Б.В., Ритчи Д.М., Язык программирования Си. 3-е изд. – URL: https://www.r5.org/files/books/computers/languages/c/kr/Brian\_Kernighan\_Dennis\_Ritchie-The\_C\_Programming\_Language-RU.pdf

# ПРИЛОЖЕНИЕ

Листинг программы

#pragma once

#include <stdio.h>

#include <locale.h>

#include <stdlib.h>

#include <time.h>

#include <iostream>

using namespace std;

namespace maxmatching {

using namespace System;

using namespace System::ComponentModel;

using namespace System::Collections;

using namespace System::Windows::Forms;

using namespace System::Data;

using namespace System::Drawing;

/// <summary>

/// Сводка для MyForm

/// </summary>

public ref class MyForm : public System::Windows::Forms::Form

{

public:

MyForm(void)

{

InitializeComponent();

//

//TODO: добавьте код конструктора

//

}

protected:

/// <summary>

/// Освободить все используемые ресурсы.

/// </summary>

~MyForm()

{

if (components)

{

delete components;

}

}

private: System::Windows::Forms::Label^ label1;

private: System::Windows::Forms::Label^ label2;

private: System::Windows::Forms::Button^ button1;

private: System::Windows::Forms::Label^ label3;

private: System::Windows::Forms::Button^ button2;

private: System::Windows::Forms::Button^ button3;

private: System::Windows::Forms::DataGridView^ dataGridView1;

private: System::Windows::Forms::DataGridView^ dataGridView2;

private: System::Windows::Forms::Button^ button4;

private: System::Windows::Forms::NumericUpDown^ numericUpDown1;

protected:

protected:

private:

/// <summary>

/// Обязательная переменная конструктора.

/// </summary>

System::ComponentModel::Container ^components;

#pragma region Windows Form Designer generated code

/// <summary>

/// Требуемый метод для поддержки конструктора — не изменяйте

/// содержимое этого метода с помощью редактора кода.

/// </summary>

void InitializeComponent(void)

{

this->label1 = (gcnew System::Windows::Forms::Label());

this->label2 = (gcnew System::Windows::Forms::Label());

this->button1 = (gcnew System::Windows::Forms::Button());

this->label3 = (gcnew System::Windows::Forms::Label());

this->button2 = (gcnew System::Windows::Forms::Button());

this->button3 = (gcnew System::Windows::Forms::Button());

this->dataGridView1 = (gcnew System::Windows::Forms::DataGridView());

this->dataGridView2 = (gcnew System::Windows::Forms::DataGridView());

this->button4 = (gcnew System::Windows::Forms::Button());

this->numericUpDown1 = (gcnew System::Windows::Forms::NumericUpDown());

(cli::safe\_cast<System::ComponentModel::ISupportInitialize^>(this->dataGridView1))->BeginInit();

(cli::safe\_cast<System::ComponentModel::ISupportInitialize^>(this->dataGridView2))->BeginInit();

(cli::safe\_cast<System::ComponentModel::ISupportInitialize^>(this->numericUpDown1))->BeginInit();

this->SuspendLayout();

//

// label1

//

this->label1->AutoSize = true;

this->label1->Font = (gcnew System::Drawing::Font(L"Times New Roman", 14.25F, System::Drawing::FontStyle::Regular, System::Drawing::GraphicsUnit::Point,

static\_cast<System::Byte>(204)));

this->label1->Location = System::Drawing::Point(56, 32);

this->label1->Name = L"label1";

this->label1->Size = System::Drawing::Size(196, 21);

this->label1->TabIndex = 0;

this->label1->Text = L"Введите размер графа: ";

this->label1->Click += gcnew System::EventHandler(this, &MyForm::label1\_Click\_1);

//

// label2

//

this->label2->AutoSize = true;

this->label2->Font = (gcnew System::Drawing::Font(L"Times New Roman", 14.25F, System::Drawing::FontStyle::Regular, System::Drawing::GraphicsUnit::Point,

static\_cast<System::Byte>(204)));

this->label2->Location = System::Drawing::Point(56, 87);

this->label2->Name = L"label2";

this->label2->Size = System::Drawing::Size(211, 21);

this->label2->TabIndex = 2;

this->label2->Text = L"Введите элементы графа:";

this->label2->Click += gcnew System::EventHandler(this, &MyForm::label2\_Click);

//

// button1

//

this->button1->BackColor = System::Drawing::SystemColors::ControlLight;

this->button1->Font = (gcnew System::Drawing::Font(L"Times New Roman", 12, System::Drawing::FontStyle::Regular, System::Drawing::GraphicsUnit::Point,

static\_cast<System::Byte>(204)));

this->button1->Location = System::Drawing::Point(515, 195);

this->button1->Name = L"button1";

this->button1->Size = System::Drawing::Size(236, 52);

this->button1->TabIndex = 3;

this->button1->Text = L"Сгенерировать граф \r\nс помощью случайных чисел";

this->button1->UseVisualStyleBackColor = false;

this->button1->Click += gcnew System::EventHandler(this, &MyForm::button1\_Click);

//

// label3

//

this->label3->AutoSize = true;

this->label3->Font = (gcnew System::Drawing::Font(L"Times New Roman", 14.25F, System::Drawing::FontStyle::Regular, System::Drawing::GraphicsUnit::Point,

static\_cast<System::Byte>(204)));

this->label3->Location = System::Drawing::Point(54, 409);

this->label3->Name = L"label3";

this->label3->Size = System::Drawing::Size(395, 21);

this->label3->TabIndex = 4;

this->label3->Text = L"Результат поска максимального паросочетания:";

this->label3->Click += gcnew System::EventHandler(this, &MyForm::label3\_Click);

//

// button2

//

this->button2->Font = (gcnew System::Drawing::Font(L"Times New Roman", 12, System::Drawing::FontStyle::Regular, System::Drawing::GraphicsUnit::Point,

static\_cast<System::Byte>(204)));

this->button2->Location = System::Drawing::Point(515, 525);

this->button2->Name = L"button2";

this->button2->Size = System::Drawing::Size(236, 52);

this->button2->TabIndex = 6;

this->button2->Text = L"Сохранить результат \r\n";

this->button2->UseVisualStyleBackColor = true;

this->button2->Click += gcnew System::EventHandler(this, &MyForm::button2\_Click);

//

// button3

//

this->button3->Font = (gcnew System::Drawing::Font(L"Times New Roman", 12, System::Drawing::FontStyle::Regular, System::Drawing::GraphicsUnit::Point,

static\_cast<System::Byte>(204)));

this->button3->Location = System::Drawing::Point(515, 455);

this->button3->Name = L"button3";

this->button3->Size = System::Drawing::Size(236, 52);

this->button3->TabIndex = 8;

this->button3->Text = L"Получить результат";

this->button3->UseVisualStyleBackColor = true;

this->button3->Click += gcnew System::EventHandler(this, &MyForm::button3\_Click);

//

// dataGridView1

//

this->dataGridView1->AllowUserToAddRows = false;

this->dataGridView1->AllowUserToDeleteRows = false;

this->dataGridView1->AllowUserToResizeColumns = false;

this->dataGridView1->AllowUserToResizeRows = false;

this->dataGridView1->AutoSizeColumnsMode = System::Windows::Forms::DataGridViewAutoSizeColumnsMode::AllCellsExceptHeader;

this->dataGridView1->BackgroundColor = System::Drawing::SystemColors::ButtonHighlight;

this->dataGridView1->ColumnHeadersBorderStyle = System::Windows::Forms::DataGridViewHeaderBorderStyle::Single;

this->dataGridView1->ColumnHeadersHeightSizeMode = System::Windows::Forms::DataGridViewColumnHeadersHeightSizeMode::AutoSize;

this->dataGridView1->EditMode = System::Windows::Forms::DataGridViewEditMode::EditOnEnter;

this->dataGridView1->EnableHeadersVisualStyles = false;

this->dataGridView1->Location = System::Drawing::Point(60, 125);

this->dataGridView1->Name = L"dataGridView1";

this->dataGridView1->RightToLeft = System::Windows::Forms::RightToLeft::No;

this->dataGridView1->RowHeadersWidth = 40;

this->dataGridView1->Size = System::Drawing::Size(347, 236);

this->dataGridView1->TabIndex = 9;

//

// dataGridView2

//

this->dataGridView2->AllowUserToAddRows = false;

this->dataGridView2->AllowUserToDeleteRows = false;

this->dataGridView2->AllowUserToResizeColumns = false;

this->dataGridView2->AllowUserToResizeRows = false;

this->dataGridView2->AutoSizeColumnsMode = System::Windows::Forms::DataGridViewAutoSizeColumnsMode::AllCells;

this->dataGridView2->BackgroundColor = System::Drawing::SystemColors::ButtonHighlight;

this->dataGridView2->ColumnHeadersHeightSizeMode = System::Windows::Forms::DataGridViewColumnHeadersHeightSizeMode::AutoSize;

this->dataGridView2->ColumnHeadersVisible = false;

this->dataGridView2->Location = System::Drawing::Point(147, 452);

this->dataGridView2->Name = L"dataGridView2";

this->dataGridView2->ReadOnly = true;

this->dataGridView2->RowHeadersVisible = false;

this->dataGridView2->Size = System::Drawing::Size(158, 173);

this->dataGridView2->TabIndex = 10;

this->dataGridView2->CellContentClick += gcnew System::Windows::Forms::DataGridViewCellEventHandler(this, &MyForm::dataGridView2\_CellContentClick);

//

// button4

//

this->button4->BackColor = System::Drawing::SystemColors::ControlLight;

this->button4->Font = (gcnew System::Drawing::Font(L"Times New Roman", 12, System::Drawing::FontStyle::Regular, System::Drawing::GraphicsUnit::Point,

static\_cast<System::Byte>(204)));

this->button4->Location = System::Drawing::Point(515, 125);

this->button4->Name = L"button4";

this->button4->Size = System::Drawing::Size(236, 52);

this->button4->TabIndex = 11;

this->button4->Text = L"Ввести элементы графа самостоятельно";

this->button4->UseVisualStyleBackColor = false;

this->button4->Click += gcnew System::EventHandler(this, &MyForm::button4\_Click);

//

// numericUpDown1

//

this->numericUpDown1->Location = System::Drawing::Point(258, 32);

this->numericUpDown1->Maximum = System::Decimal(gcnew cli::array< System::Int32 >(4) { 50, 0, 0, 0 });

this->numericUpDown1->Name = L"numericUpDown1";

this->numericUpDown1->Size = System::Drawing::Size(120, 20);

this->numericUpDown1->TabIndex = 12;

this->numericUpDown1->ValueChanged += gcnew System::EventHandler(this, &MyForm::numericUpDown1\_ValueChanged);

//

// MyForm

//

this->AutoScaleDimensions = System::Drawing::SizeF(6, 13);

this->AutoScaleMode = System::Windows::Forms::AutoScaleMode::Font;

this->BackColor = System::Drawing::SystemColors::Control;

this->ClientSize = System::Drawing::Size(824, 665);

this->Controls->Add(this->numericUpDown1);

this->Controls->Add(this->button4);

this->Controls->Add(this->dataGridView2);

this->Controls->Add(this->dataGridView1);

this->Controls->Add(this->button3);

this->Controls->Add(this->button2);

this->Controls->Add(this->label3);

this->Controls->Add(this->button1);

this->Controls->Add(this->label2);

this->Controls->Add(this->label1);

this->Name = L"MyForm";

this->Text = L"Поиск максимального паросочетания";

this->Load += gcnew System::EventHandler(this, &MyForm::MyForm\_Load);

(cli::safe\_cast<System::ComponentModel::ISupportInitialize^>(this->dataGridView1))->EndInit();

(cli::safe\_cast<System::ComponentModel::ISupportInitialize^>(this->dataGridView2))->EndInit();

(cli::safe\_cast<System::ComponentModel::ISupportInitialize^>(this->numericUpDown1))->EndInit();

this->ResumeLayout(false);

this->PerformLayout();

}

#pragma endregion

int n;

private: System::Void MyForm\_Load(System::Object^ sender, System::EventArgs^ e) {

}

private: System::Void label1\_Click(System::Object^ sender, System::EventArgs^ e) {

}

private: System::Void label1\_Click\_1(System::Object^ sender, System::EventArgs^ e) {

}

private: System::Void label2\_Click(System::Object^ sender, System::EventArgs^ e) {

}

private: System::Void label3\_Click(System::Object^ sender, System::EventArgs^ e) {

}

private: System::Void numericUpDown1\_ValueChanged(System::Object^ sender, System::EventArgs^ e) {

}

private: System::Void button4\_Click(System::Object^ sender, System::EventArgs^ e) {

n = (int)numericUpDown1->Value;

int n\_last\_first\_part = n / 2;

if (n == 0) { MessageBox::Show("Введите размер графа!"); }

dataGridView1->RowCount = n;

dataGridView1->ColumnCount = n;

dataGridView1->TopLeftHeaderCell->Value = "Граф G";

//srand(time(0));

for (int i = 0; i < n; i++) { //формирование матрицы смежности М

for (int j = 0; j < n; j++) { //для самостоятельного заполнения

dataGridView1->Rows[i]->Cells[j]->Value = 0;

}

dataGridView1->Rows[i]->HeaderCell->Value = System::Convert::ToString(i + 1);

dataGridView1->Columns[i]->HeaderCell->Value = System::Convert::ToString(i + 1);

}

}

private: System::Void button1\_Click(System::Object^ sender, System::EventArgs^ e) {

n = (int)numericUpDown1->Value;

int n\_last\_first\_part = n / 2;

if (n == 50) { MessageBox::Show("Предупреждение: максимальный размер графа = 50"); }

if (n == 0) { MessageBox::Show("Введите размер графа!"); }

dataGridView1->RowCount = n;

dataGridView1->ColumnCount = n;

dataGridView1->TopLeftHeaderCell->Value = "Граф G";

srand(time(0));

for (int i = 0; i < n; i++) { //формирование матрицы смежности М

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

if (j < n\_last\_first\_part && i < n\_last\_first\_part) {

dataGridView1->Rows[i]->Cells[j]->Value = 0;

}

else {

if (j >= n\_last\_first\_part && i >= n\_last\_first\_part) {

dataGridView1->Rows[i]->Cells[j]->Value = 0;

}

else {

if (i == j) dataGridView1->Rows[i]->Cells[j]->Value = 0;

else {

dataGridView1->Rows[i]->Cells[j]->Value = rand() % 2;

dataGridView1->Rows[j]->Cells[i]->Value = dataGridView1->Rows[i]->Cells[j]->Value;

}

}

}

}

dataGridView1->Rows[i]->HeaderCell->Value = System::Convert::ToString(i + 1);

dataGridView1->Columns[i]->HeaderCell->Value = System::Convert::ToString(i + 1);

}

}

bool DFS(int v, int n, int\* tops\_visited, int\* matching) {

if (tops\_visited[v] == 1) return false;

tops\_visited[v] = 1;

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

Object^ tmp = dataGridView1->Rows[v]->Cells[i]->Value;

int tmp\_int = System::Convert::ToInt64(tmp);

if (tmp\_int > 0) {

if ((matching[i] == -1) || (DFS(matching[i], n, tops\_visited, matching))) {

matching[i] = v;

return true;

}

}

}

return false;

}

private: System::Void richTextBox1\_TextChanged(System::Object^ sender, System::EventArgs^ e) {

}

private: System::Void button3\_Click(System::Object^ sender, System::EventArgs^ e) {

if (n == 0) { MessageBox::Show("Введите размер графа!"); }

int flag\_orient = 0;

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

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

if (i != j) {

Object^ tmp1 = dataGridView1->Rows[j]->Cells[i]->Value;

int tmp1\_int = System::Convert::ToInt64(tmp1);

Object^ tmp2 = dataGridView1->Rows[i]->Cells[j]->Value;

int tmp2\_int = System::Convert::ToInt64(tmp2);

if (tmp1\_int != tmp2\_int) {

flag\_orient++;

}

}

}

}

if (flag\_orient > 0) MessageBox::Show("Предупреждение: задан ориентированный граф");

int\* tops\_visited = (int\*)malloc(n \* sizeof(int\*));

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

memset(matching, -1, n \* sizeof(int\*));

for (int i = 0; i < n; i++) { //алг Куна

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

tops\_visited[i] = 0;

}

DFS(i, n, tops\_visited, matching);

}

int n\_last\_first\_part = n / 2;

int\* mas\_already\_output = (int\*)malloc(n \* sizeof(int\*));

memset(mas\_already\_output, 0, n \* sizeof(int\*));

int flag\_alredy\_out = 0;

int size\_dataGridView2 = 0;

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

if (matching[i] + 1 != 0) {

flag\_alredy\_out = 0;

for (int i\_already = 0; i\_already < n; i\_already++) {

if (matching[i] + 1 == mas\_already\_output[i\_already]) flag\_alredy\_out = 1;

}

if (flag\_alredy\_out == 1) continue;

mas\_already\_output[i] = i + 1;

size\_dataGridView2++;

}

}

this->dataGridView2->ColumnCount = 2;

this->dataGridView2->RowCount = size\_dataGridView2;

int i\_row = 0;

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

if (matching[i] + 1 != 0) {

flag\_alredy\_out = 0;

for (int i\_already = 0; i\_already < n; i\_already++) {

if (matching[i] + 1 == mas\_already\_output[i\_already]) flag\_alredy\_out = 1;

}

if (flag\_alredy\_out == 1) continue;

this->dataGridView2->Rows[i\_row]->Cells[0]->Value = matching[i]+1;

this->dataGridView2->Rows[i\_row]->Cells[1]->Value = i + 1;

i\_row++;

}

}

}

private: System::Void button2\_Click(System::Object^ sender, System::EventArgs^ e) {

if (n == 0) { MessageBox::Show("Введите размер графа!"); }

else {

int n\_last\_first\_part = n / 2;

FILE\* file;

file = fopen("max\_matching.txt", "w");

int size\_dataGridView2 = dataGridView2->RowCount;

for (int i = 0; i < size\_dataGridView2; i++) {

for (int j = 0; j < 2; j++) {

Object^ tmp = dataGridView2->Rows[i]->Cells[j]->Value;

int tmp\_el = System::Convert::ToInt64(tmp);

fprintf(file, "%d", tmp\_el);

if (j == 0) fprintf(file, " - ");

}

fprintf(file, "\n");

}

fclose(file);

MessageBox::Show("Результат успешно сохранен в файл.");

}

}

private: System::Void dataGridView2\_CellContentClick(System::Object^ sender, System::Windows::Forms::DataGridViewCellEventArgs^ e) {

}

};

}