МИНИСТЕРСТВО НАУКИ и высшего ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное автономное образовательное учреждение  
 высшего образования   
**«Национальный исследовательский   
Нижегородский государственный университет им. Н.И. Лобачевского»**

**(ННГУ)**

**Институт информационных технологий, математики и механики**

**Кафедра информатики и автоматизации научных исследований**

Направление подготовки: «Прикладная информатика»

Профиль подготовки: «Прикладная информатика в области принятия решений»

**ОТЧЕТ**

по практике по получению профессиональных умений  
 и опыта профессиональной деятельности

на тему:

**«Реализация итерационного подхода к решению задачи теории расписаний»**

**Выполнила:** студентка группы 381707-2

Кузнецова Анастасия Олеговна\_\_\_\_\_\_\_\_\_\_\_

*подпись*

**Научный руководитель:**

Доцент кафедры ИАНИ

Кумагина Елена Александровна\_\_\_\_\_\_\_\_\_\_

*подпись*

Нижний Новгород  
2021

Оглавление

[Введение 3](#_Toc66172355)

[1. Задача о назначениях на минимум с аддитивным критерием 4](#_Toc66172356)

[1.1. Формулировка задачи 4](#_Toc66172357)

[1.2. Математическая модель 4](#_Toc66172358)

[2. Венгерский алгоритм для решения задачи о назначениях 5](#_Toc66172359)

[2.1. Алгоритм решения простейшей задачи о назначениях 5](#_Toc66172360)

[2.2. Венгерский алгоритм 5](#_Toc66172361)

[3. Задача теорий расписаний с директивными сроками 7](#_Toc66172362)

[3.1. Формулировка задачи упорядочения 7](#_Toc66172363)

[3.2. Мат модель 7](#_Toc66172364)

[3.3. Постановка задачи 7](#_Toc66172365)

[4. Итерационный алгоритм решения задачи теории расписаний 8](#_Toc66172366)

[4.1. Разработка итерационного алгоритма решения задачи теории расписаний 8](#_Toc66172367)

[4.2. Программная реализация 9](#_Toc66172368)

[Заключение 14](#_Toc66172369)

[Список литературы 14](#_Toc66172370)

[Приложение 15](#_Toc66172371)

# **Введение**

Целью моей работы является:

1. Разработка итерационного подхода для решения задачи упорядочения работ для 1 станка с целью минимизации суммарного взвешенного запаздывания;

2. реализация Венгерского алгоритма для решения задачи о назначениях с аддитивным критерием;

3. Реализация итерационного подхода, основанного на решении ряда задач о назначении.

При выполнении задания по преддипломной практике будет реализован предложенный эвристический алгоритм и разработан интерфейс программы. Язык программирования: C#. Среда разработки: Visual Studio 2019.

# Задача о назначениях на минимум с аддитивным критерием

## Формулировка задачи

Задано n работ, каждую из которых может выполнить любой из n исполнителей. Затрата на выполнение j исполнителем i работы равна cij. Требуется распределить исполнителей по работам таким образом, чтобы суммарные затраты на производство были минимальны. При этом каждый исполнитель должен выполнить ровно одну работу, а каждая работа должна быть поручена ровно одному исполнителю.

Исходными данными в задаче о назначениях является матрица затрат. Если коэффициенты cij. этой матрицы принимают значения 0 или 1, то такая задача называется простейшей задачей о назначении.

## Математическая модель

Математическая модель этой задачи следующая:

. (1)

, j=. (2)

, i=. (3)

xij ∈{0,1} i,j=. (4)

Переменные xij имеют следующую содержательную интерпретацию: 

Если коэффициенты cij интерпретировать как затраты i-го исполнителя по работе j, то в этом случае требуется назначить исполнителей на работы так, чтобы их суммарная затрата была минимальна. За это как раз и отвечает критерий (1).

Условие (2) гарантирует, что каждая работа будет закреплена за каким-то исполнителем. Условие (3) гарантирует, что каждый исполнитель будет назначен на какую-то работу.

В дальнейшем будем называть эту задачу Nn. Здесь С=⎟⎜сij⎟⎜n×n - матрица стоимостей или производительностей. Из условий (2)-(4) следует, что решением задачи Nn является бистохастическая булева квадратная матрица.

Итак, в математической постановке задача о назначениях заключается в минимизации функции (1), при ограничениях (2-4).

В данной постановке задача о назначениях не является задачей линейного программирования. Этому препятствуют условия (4). Однако можно показать, что в данном случае условия (4) можно заменить условиями

*x*ij ≥ 0 ( *i* = 1, …, *n; j* = 1, … , *n*),

сведя задачу о назначениях к частному случаю транспортной задачи.

# Венгерский алгоритм для решения задачи о назначениях

## Алгоритм решения простейшей задачи о назначениях

Входные данные: квадратная матрица любого размера из 0 и 1, размер матрицы.

1. Назначение

Идём по строчкам и по столбцам циклом. Как только натыкаемся на первую единицу в строчке, назначаем данного работника на эту (первую попавшуюся) работу. Если данная работа уже занята, то ищем назначение дальше по строчке.

Тут возможны два случая:

- Если нам удалось назначить каждого работника на свою, неповторяющуюся работу, то решение найдено, алгоритм завершён. Переход на шаг 3.

- Если нам не удалось назначить каждого работника на свою, неповторяющуюся работу, то записываем работников, не нашедших работу, в стек. Переход на шаг 2.

1. Цепочки переназначений

Теперь пытаемся назначить тех работников, которым работа не досталась, они лежат у нас в стеке. Для этого строим, так называемые, «цепочки переназначений» по следующей схеме:

- Ищем в строке не назначенного работника работу, которую он может делать(она обозначена в строке единицей) и назначаем данного работника на неё.

- Теперь работу ищет другой работник, ведь из-за нашего переназначения он остался без работы. Записываем его в стек и теперь пытаемся назначить уже ему работу.

Тут возможны несколько случаев:

1. Мы смогли назначить его на такую работу, которая не занята. Переход на шаг 3.
2. Данный работник больше ничего не может. Удаляем его из стека, переходим на шаг назад, запоминая, чтобы его больше не выбрать.
3. Мы смогли назначить данного работника на работу, которая уже занята. Тогда повторяем действия шага 2 уже для нового работника, который потерял свою работу.

По итогу работы цепочек, мы либо найдём назначение пар в нашей матрице, либо наш стек опустеет, так и не найдя решения.

1. Выход из алгоритма

- Если решение найдено, то мы отображаем его на экране.

- Если решение не найдено, то выводим соответствующее сообщение.

## Венгерский алгоритм

Мы не зря разобрали алгоритм решения простейшей задачи о назначениях. Теперь рассмотрим алгоритм решения задачи о назначениях на максимум в каноническом виде с помощью алгоритма Куна (венгерского алгоритма).

Входные данные: квадратная матрица и её размер.

1. Поиск максимума в каждой строке

Нам понадобятся вспомогательные одномерные массивы U и V, с помощью которых в дальнейшем будем формировать простейшую матрицу. Для начала нам нужно их заполнить:

- значения U в начале равны 0, этот массив будет отвечать за работы.

- в значения V записываем максимум по каждой строчке, этот массив будет отвечать за работников.

1. Формирование простейшей матрицы

Когда найдены значения U и V, формируем простейшую матрицу. Размер этой матрицы будет соответствовать размеру входной матрицы. Для начала заполняем её нулями, а 1 ставим там, где:

,

где ui – значение одномерного массива U на конкретном i-ом столбце.

vj – значение одномерного массива V на конкретной j-ой строке.

cij – элемент матрицы на пересечении j-ой строки и i-ого столбца.

1. Решение простейшей матрицы

Далее эта простейшая матрица назначается с помощью описанного выше алгоритма решения простейшей задачи о назначениях, причём

- каждый раз, когда мы в стек добавляем работника, соответствующее значение vj уменьшается на 1.

- каждый раз, когда мы в «цепочке переназначений» выбираем некоторую работу, её значение ui увеличивается на 1.

Если алгоритм решения простейшей задачи о назначениях нашёл пары работник-работа, то переход на шаг 5. Иначе переход на шаг 4.

1. Повтор шага 2 и 3, пока не будет найдено решение.
2. Подсчет 

Для подсчёта Сmax необходимо просуммировать те значения исходной матрицы, которые стоят на пересечении выбранного работника и выбранной работы в найденных парах.

1. Выход из программы.

Ответом будут являться найденные пары и значение Cmax

# **Задача теорий расписаний с директивными сроками**

## Формулировка задачи упорядочения

Пусть N = {1, 2, …, n} заявок выполняются одним прибором. Каждая заявка 𝑖, 𝑖𝜖N характеризуется длительностью обслуживания 𝑡𝑖, директивным сроком 𝑑𝑖 и коэффициентом 𝑤𝑖 за нарушение директивного срока на единицу времени. В каждый момент времени прибор выполняет не более одной заявки. Порядок обслуживания заявок произвольный. Прерывания в обслуживании отдельной заявки не допускаются. Требуется найти такой порядок выполнения заявок, при котором минимизируется суммарное взвешенное запаздывание относительно директивных сроков.

Данная задача известна как ТР5 и является NP-полной [1].

## Мат модель

**Исходные данные**

N – количество заданий,

– вектор длительностей выполнений заданий,

– вектор директивных сроков,

– вектор коэффициентов штрафа за обслуживание после директивного срока.

Предполагаем, что все данные целочисленные.

**Варьируемые переменные**

– вектор моментов начала выполнений заданий,

– вектор моментов окончания выполнения заданий.

**Ограничения**

1. , i∈N – выполнение работы на машине производится без прерываний;
2. j>i, i,j∈N – на машине одновременно может выполняться только одна работа;

## Постановка задачи

**Критерий**

Минимизируется суммарное взвешенное запаздывание:

*,* где является запаздыванием.

# Итерационный алгоритм решения задачи теории расписаний

## Разработка итерационного алгоритма решения задачи теории расписаний

Для решения задачи о построении расписания можно использовать уже разобранную мной ранее задачу о назначениях и Венгерский алгоритм. Основная идея – на основании данных задачи теории расписаний строится матрица производительностей для задачи о назначениях. На основании решения задачи о назначениях строится вектор расписаний и коэффициенты матрицы уточняются (изменяются). Задача о назначениях решается еще раз и т.д., пока есть изменения матрицы.

Приведу подробное описание шагов алгоритма для n работ.

Исходные данные: вектор штрафов, директивных сроков и длительностей исполнения работ. Причём каждой работе должно соответствовать одно значение штрафа, одно значение длительности и директивного срока, то есть вектора по размерности совпадают с числом работ.

1. Шаг №1: формирование матрицы очередей.

Необходимо сформировать n последовательностей с номерами работ. Причем в каждом столбце и строчке каждый номер должен повторяться ровно 1 раз. Это можно сделать разными способами, но самый простой – сдвиг последовательности на 1 вправо(влево).

| 0 1 2 3 … n |

| n 0 1 2 … n-1 |

| … |

| 1 2 3 4… 0 |

1. Шаг №2: расчёт начальной матрицы производительностей Pi для сформированных n последовательностей очередей.

На втором шаге целью является формирование матрицы, с которой мы будем работать в алгоритме на следующих шагах – матрицы производительностей. Для её формирования нужно рассчитать нарушение для каждой работы в каждой из n последовательностей. Приведу формулу для расчета работы под номером i, которая выполняется j-ой по порядку.

Нарушение(i, j) = (время окончания работы(i) – дир. срок(i)) \* штраф(i) (\*)

1. Шаг 3: работа Венгерского алгоритма.

Матрица Pi на этом шаге является исходными данными для работы венгерского алгоритма с аддитивным критерием на минимум. На выходе получаем вектор порядка исполнения выбранных работ и суммарное взвешенное запаздывание.

1. Шаг 4: Перерасчет Pi для полученного из Венгерского алгоритма порядка работ.

На этом шаге происходит перерасчет нарушений по формуле (\*). В матрице производительностей Pi уточняются элементы, соответствующие полученному решению;

1. Шаг 5: повтор шага 3-4.

Шаги 3 и 4 будут исполняться до тех пор, пока матрица Pi не перестанет меняться. Как только это произойдет, искомое решение задачи о расписаниях будет найдено.

Выходные данные: вектор порядка выполнения работ и суммарное взвешенное запаздывание.

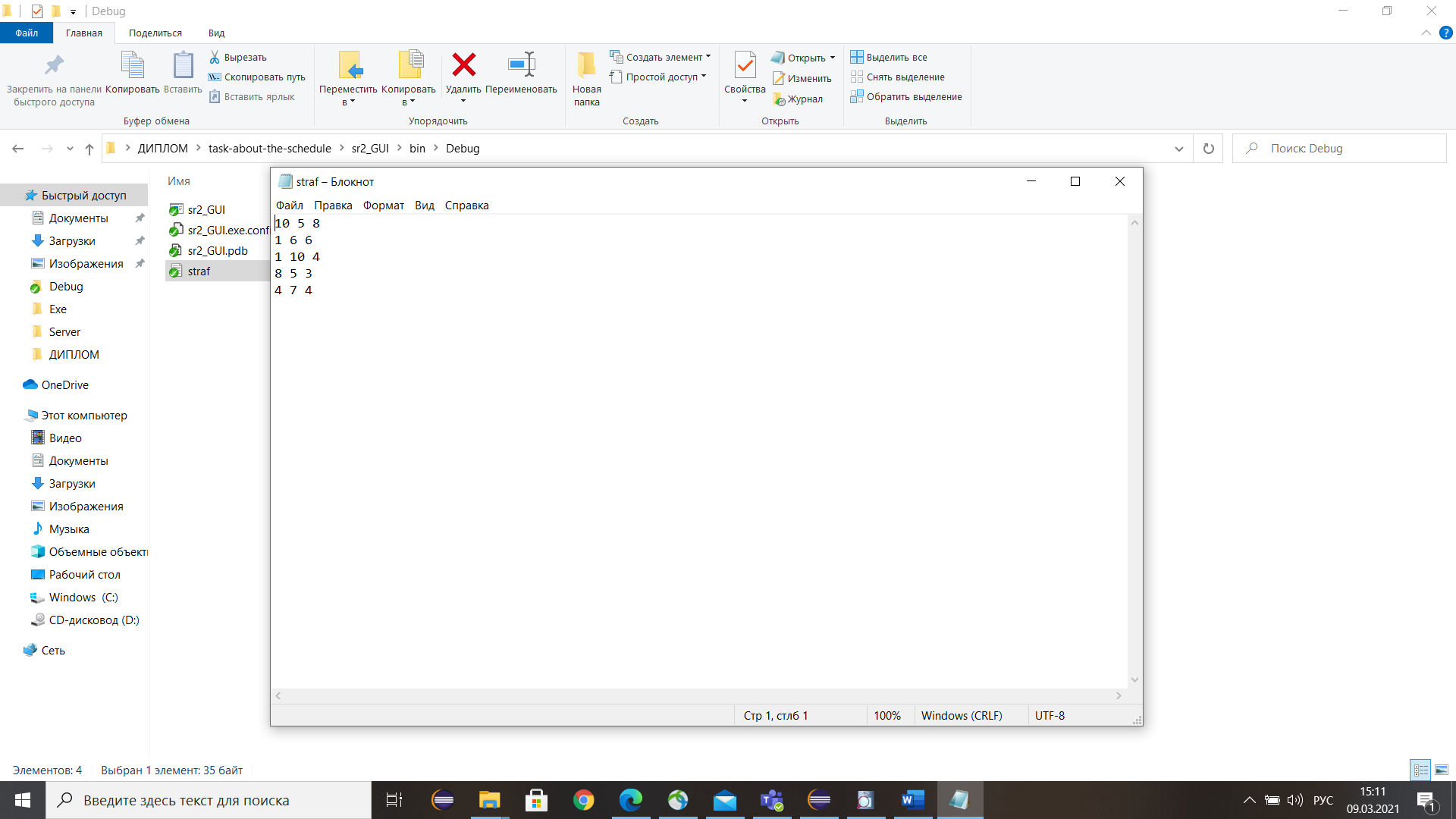
## Программная реализация

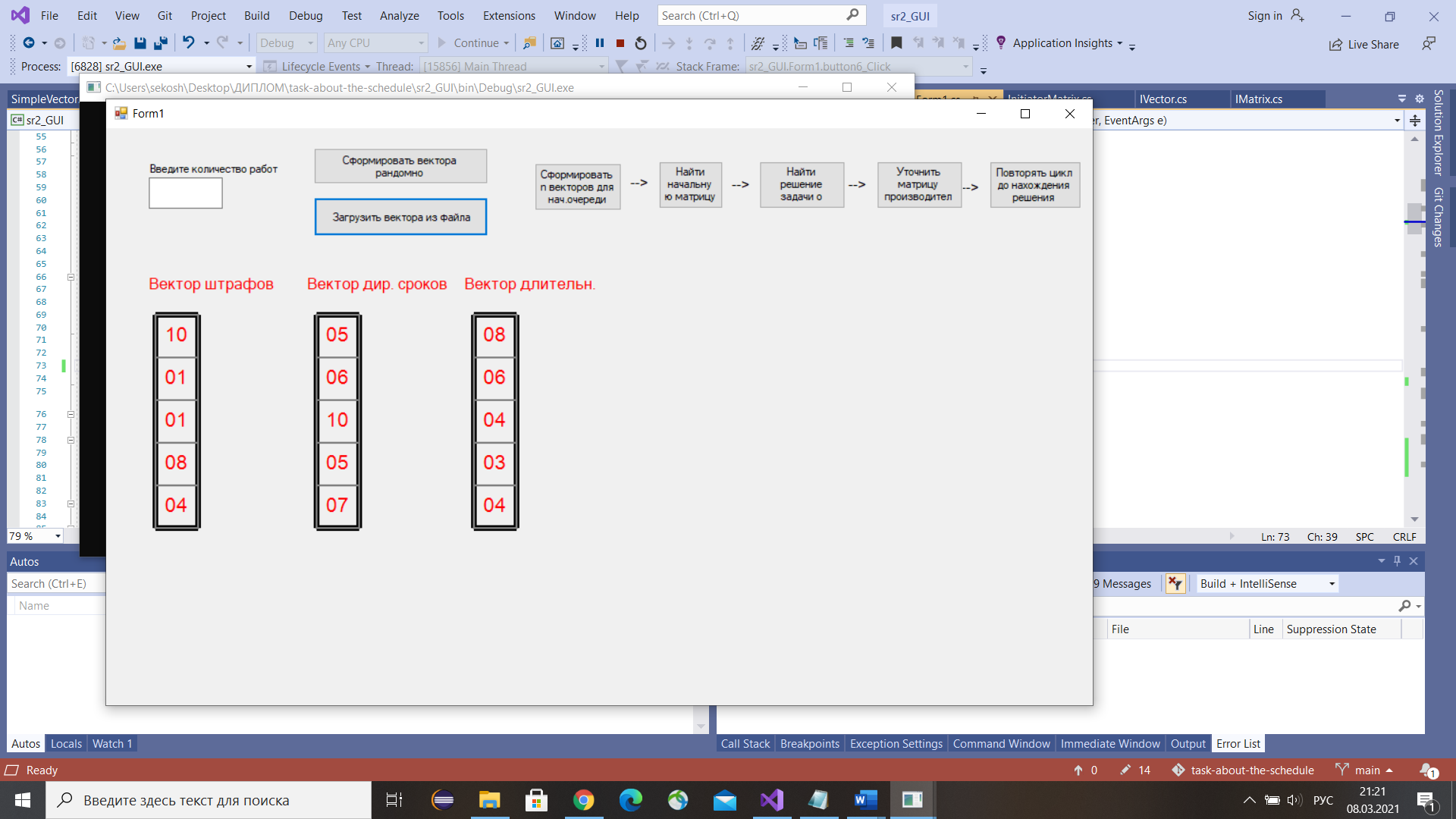
Для программной реализации итерационного алгоритма я воспользовалась Windows Forms C#. Код основных модулей приведен в Приложении.

Привожу скриншоты работы программы по шагам.

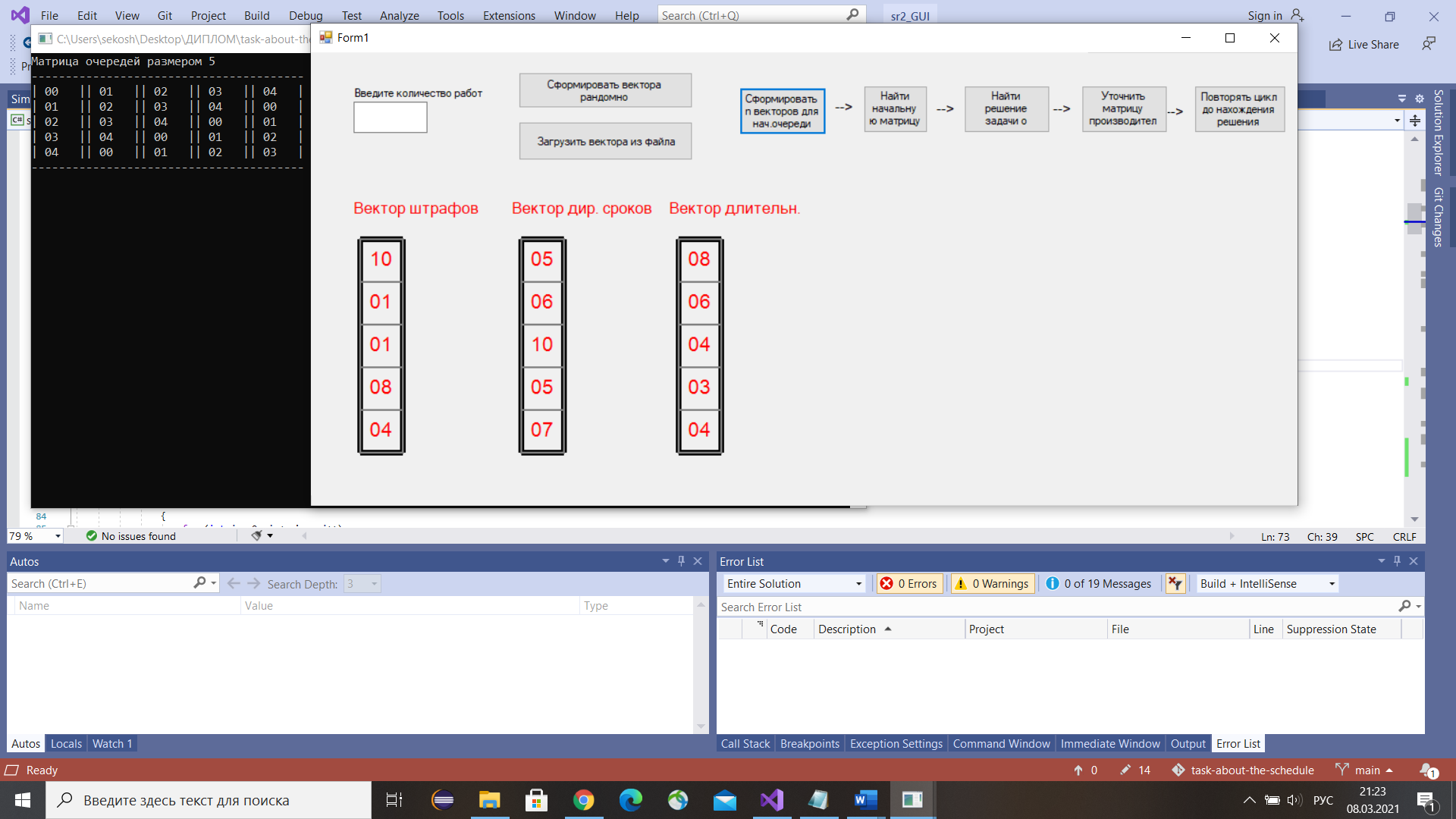
*Пример 1.*

Исходные данные – из файла:

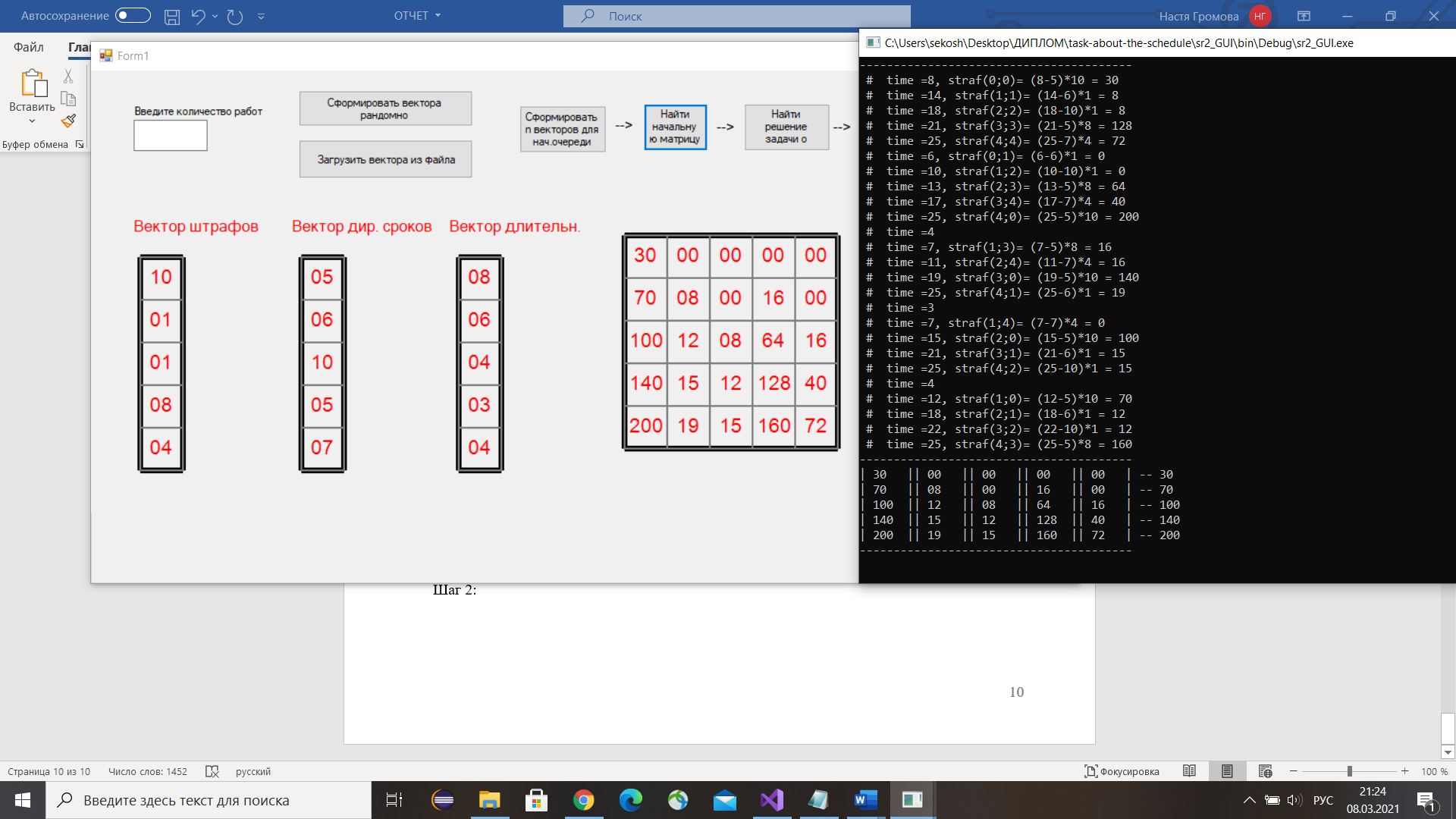




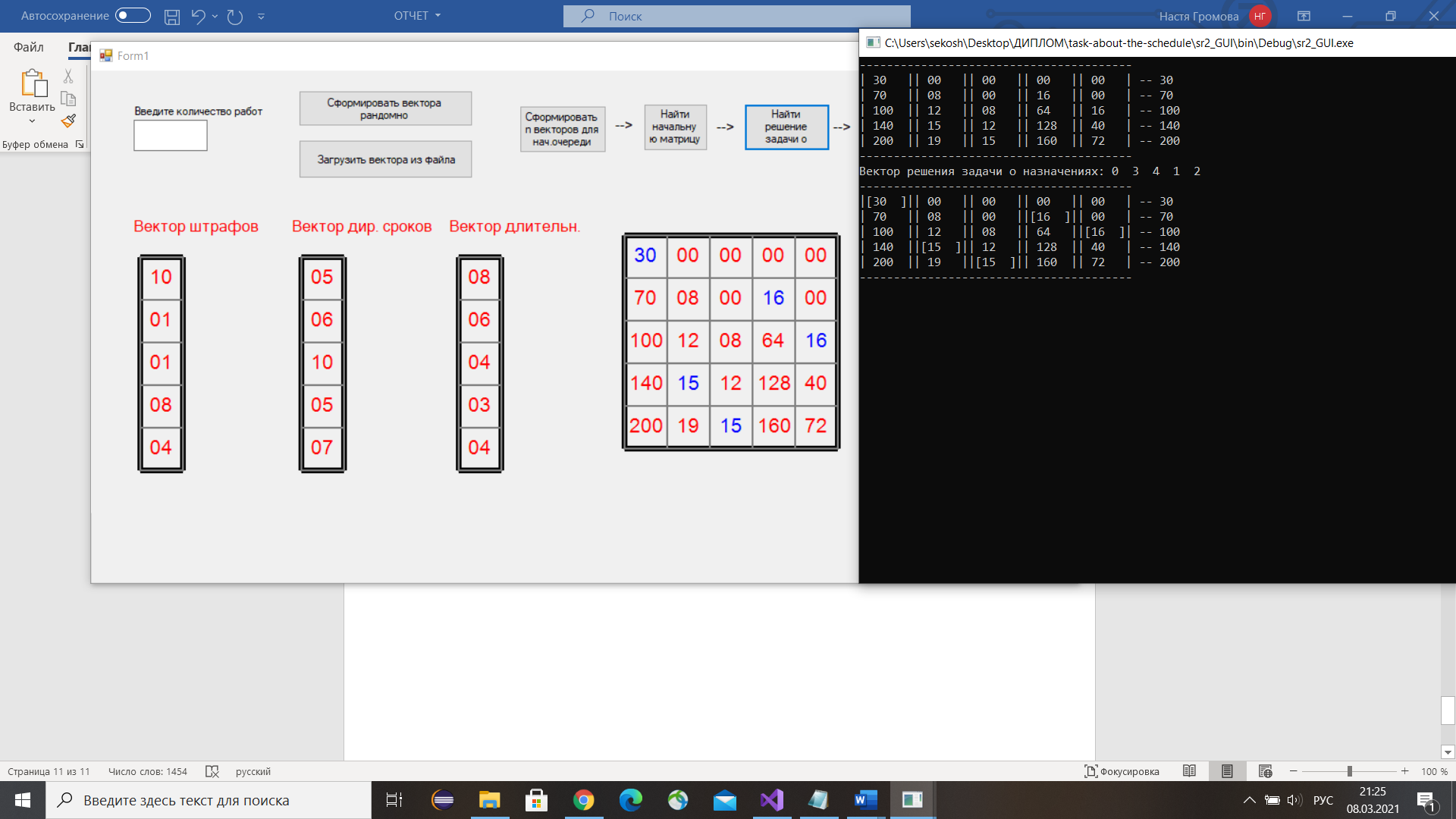
Шаг 1:



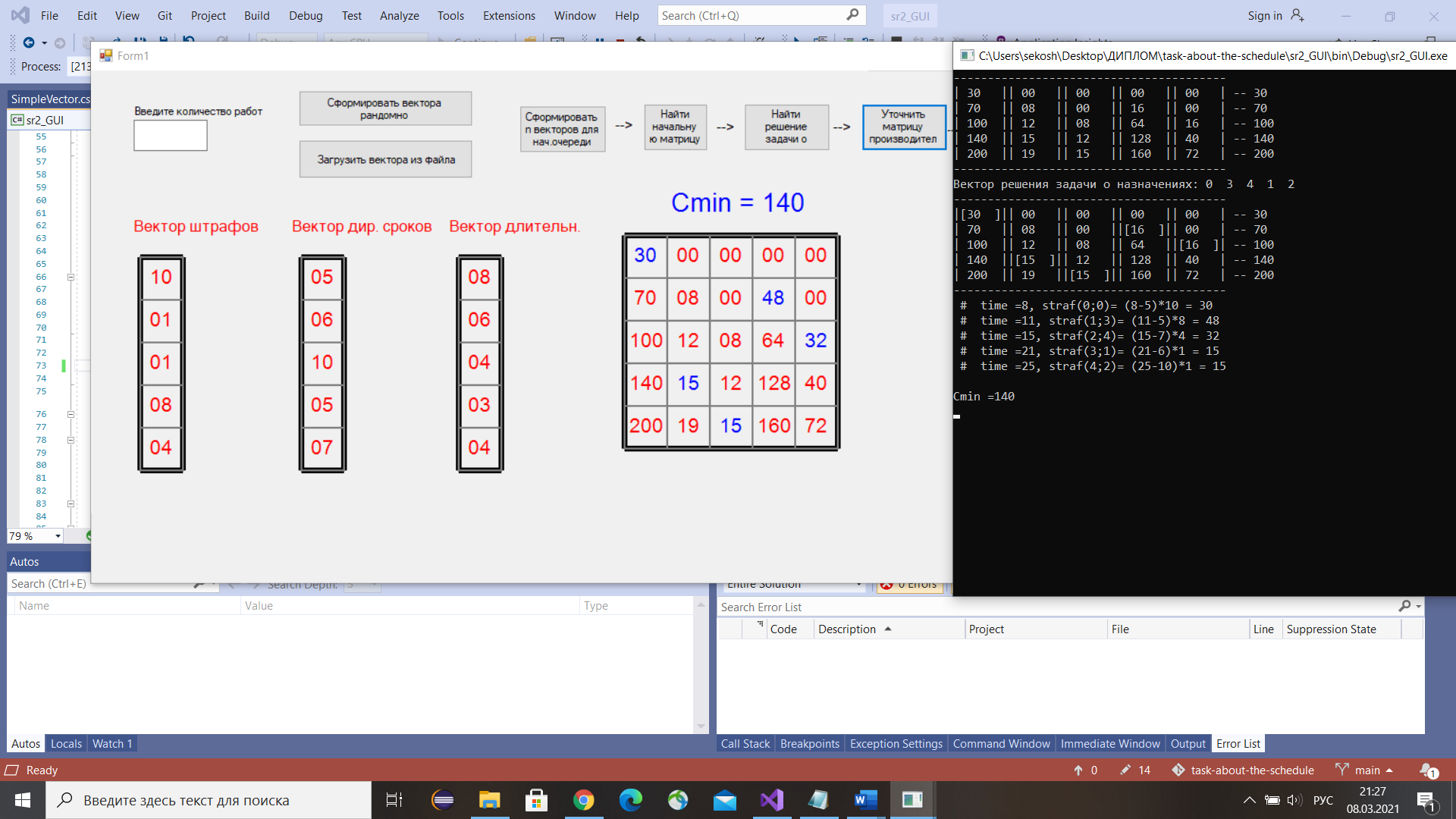
Шаг 2:



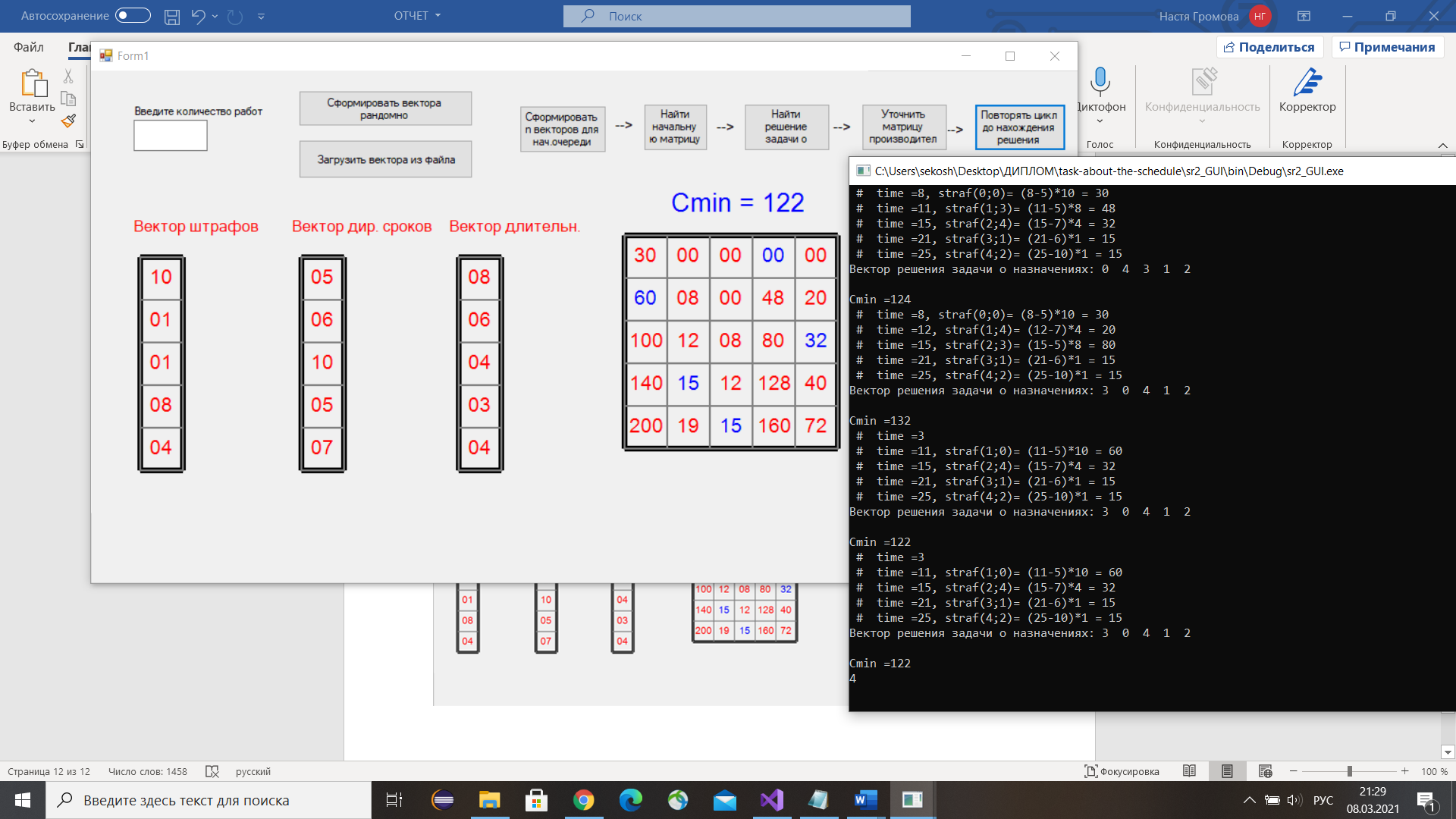
Шаг 3:



Шаг 4:

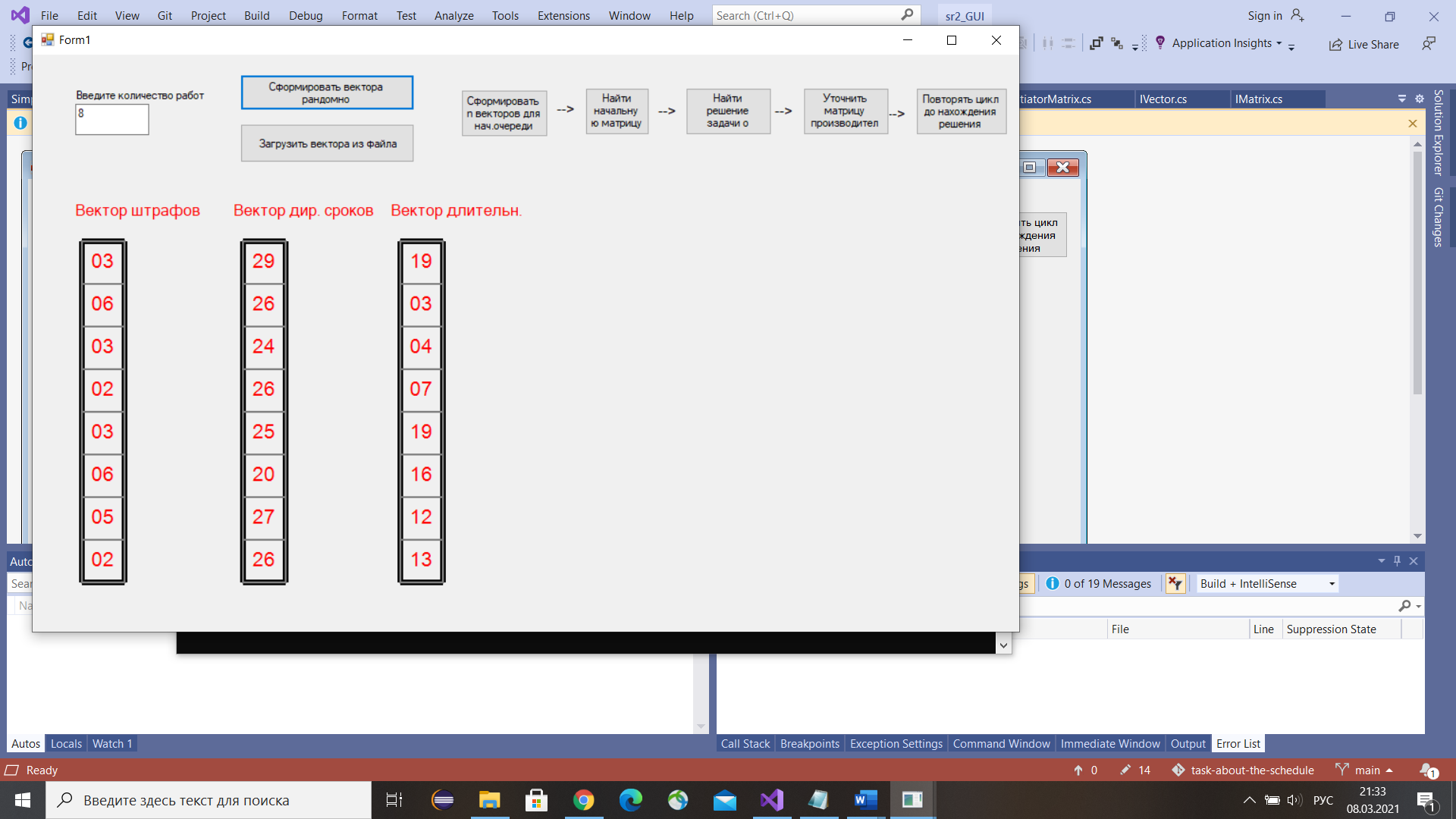


Шаг 5:

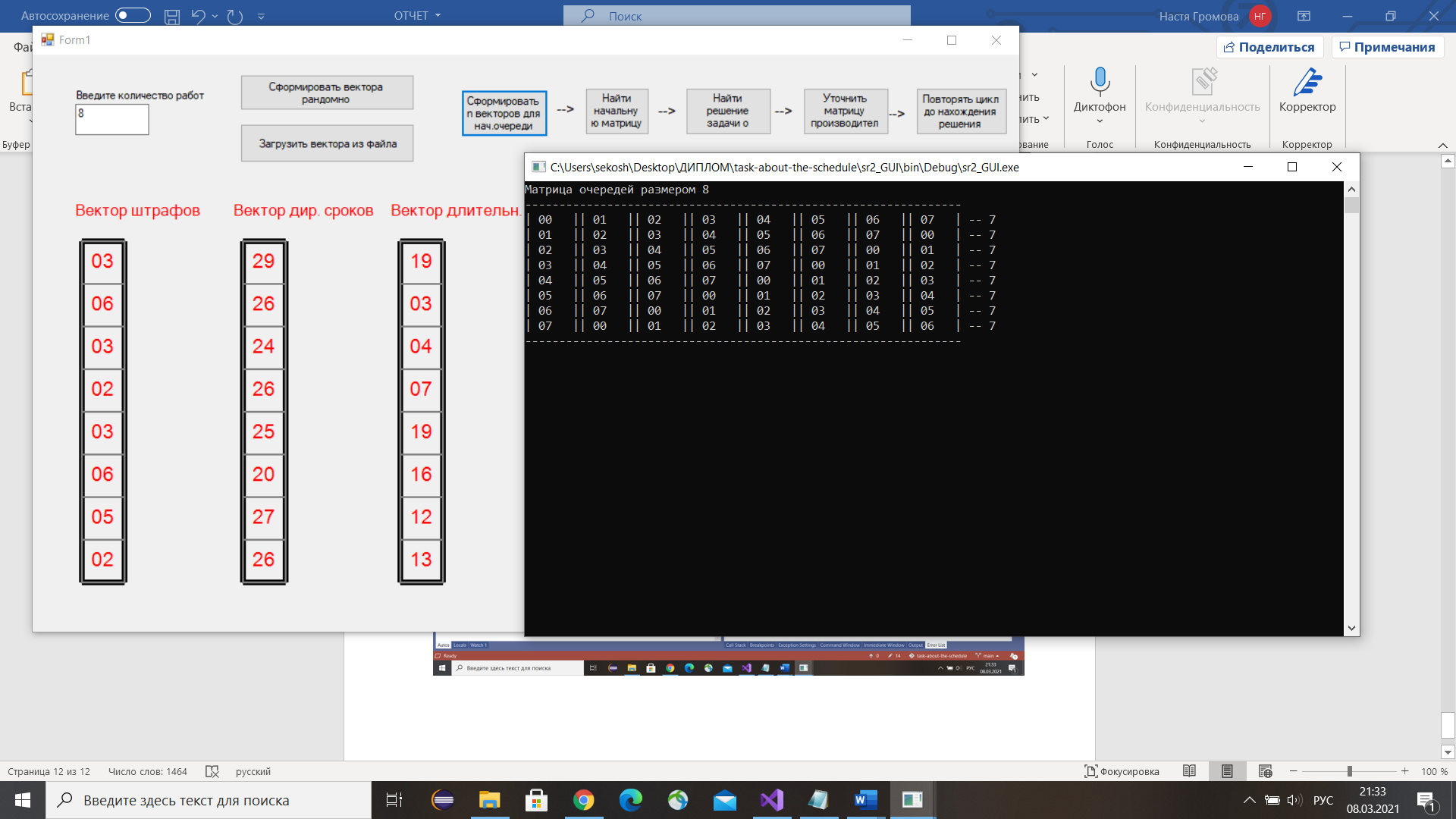


*Пример 2.*

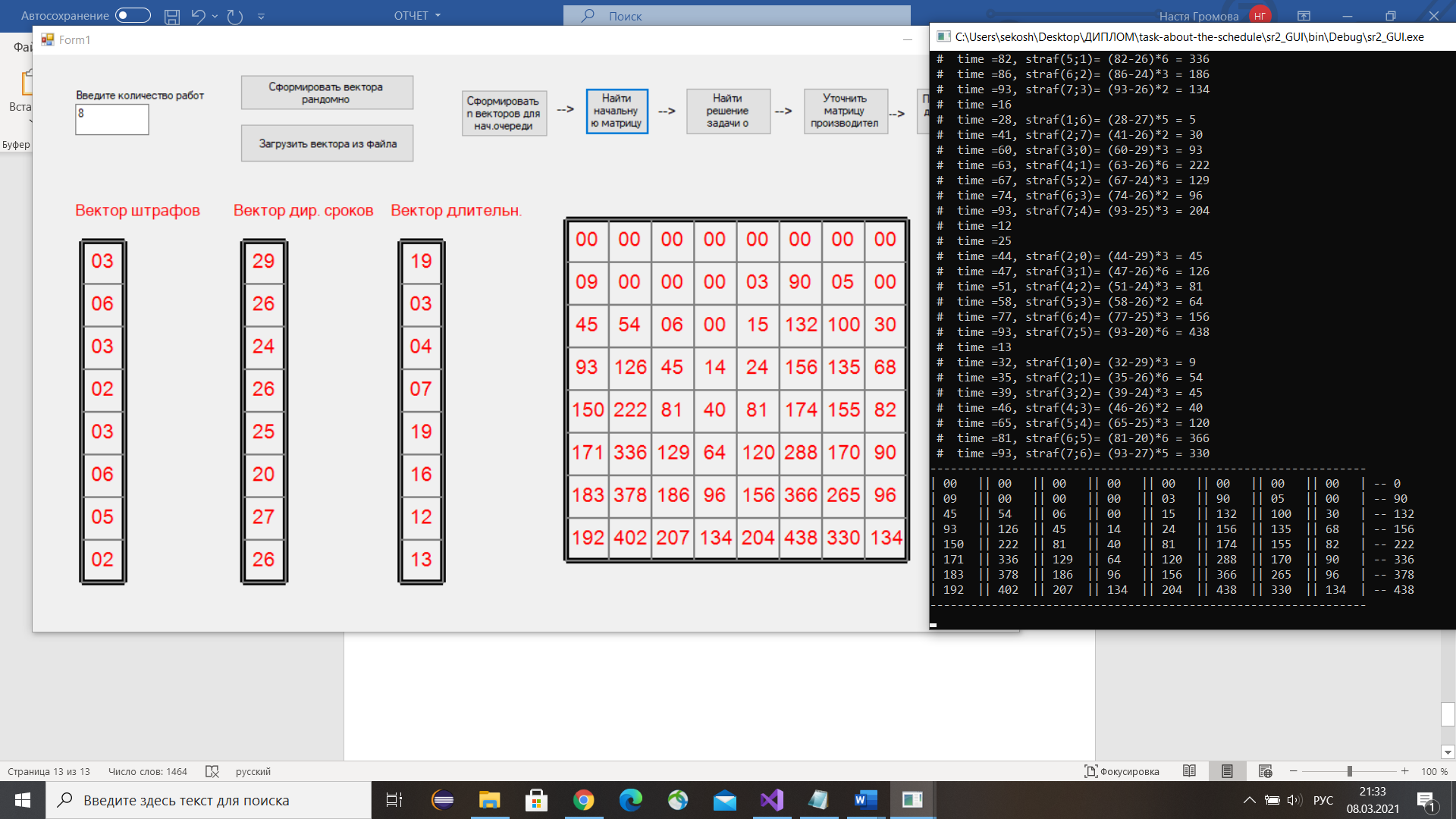
Исходные данные – сформированы случайно:



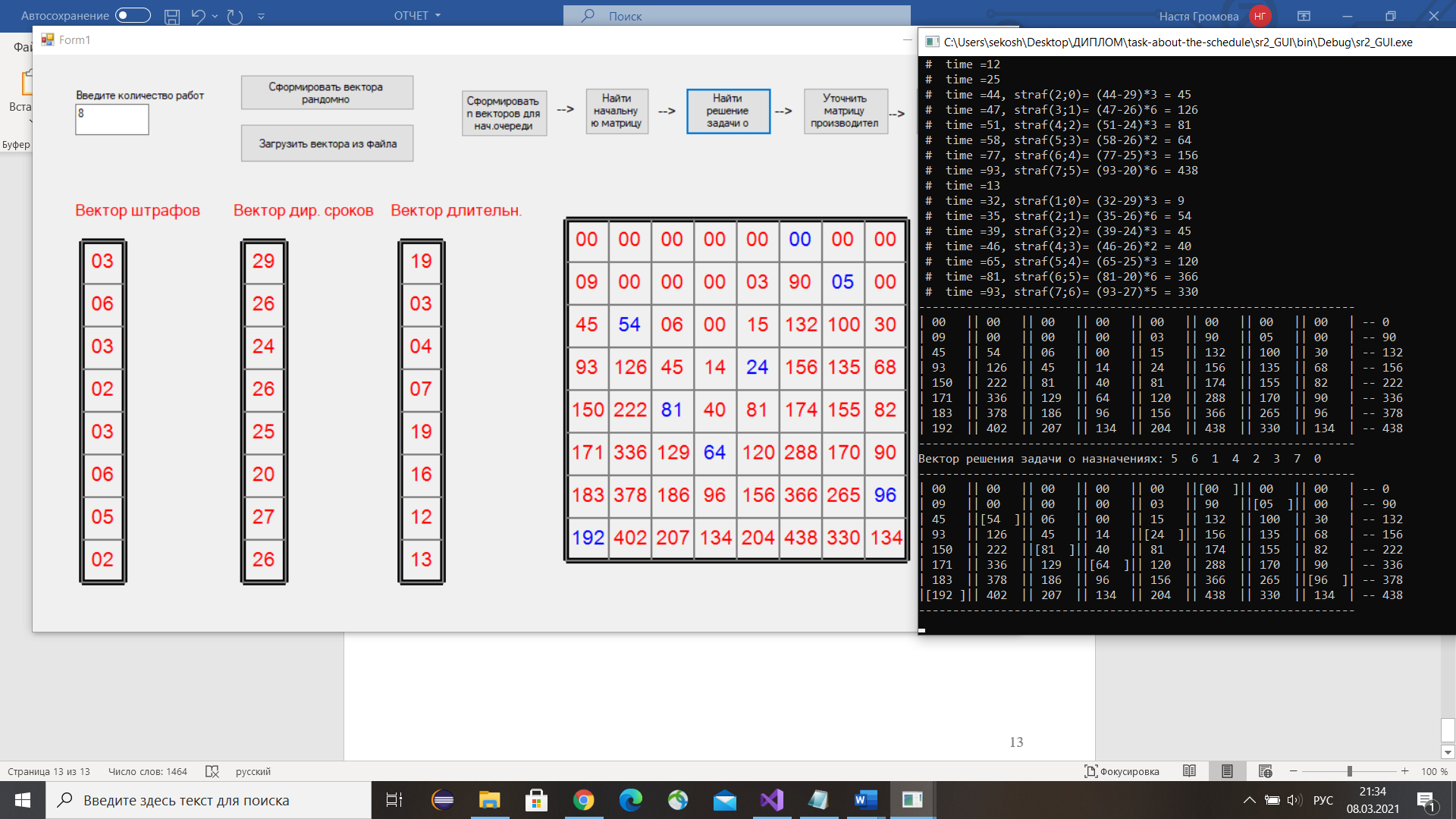
Шаг 1:



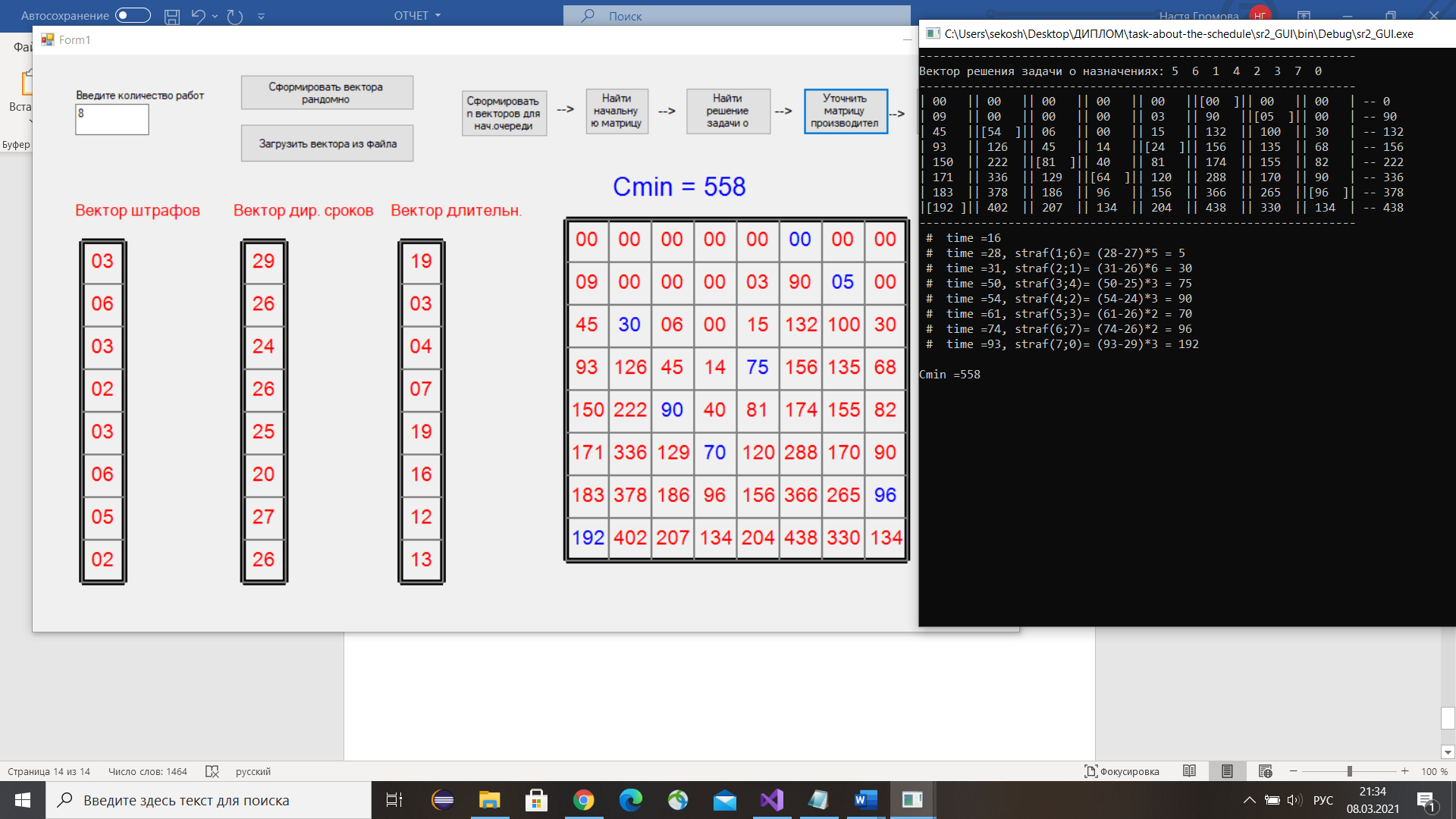
Шаг 2:



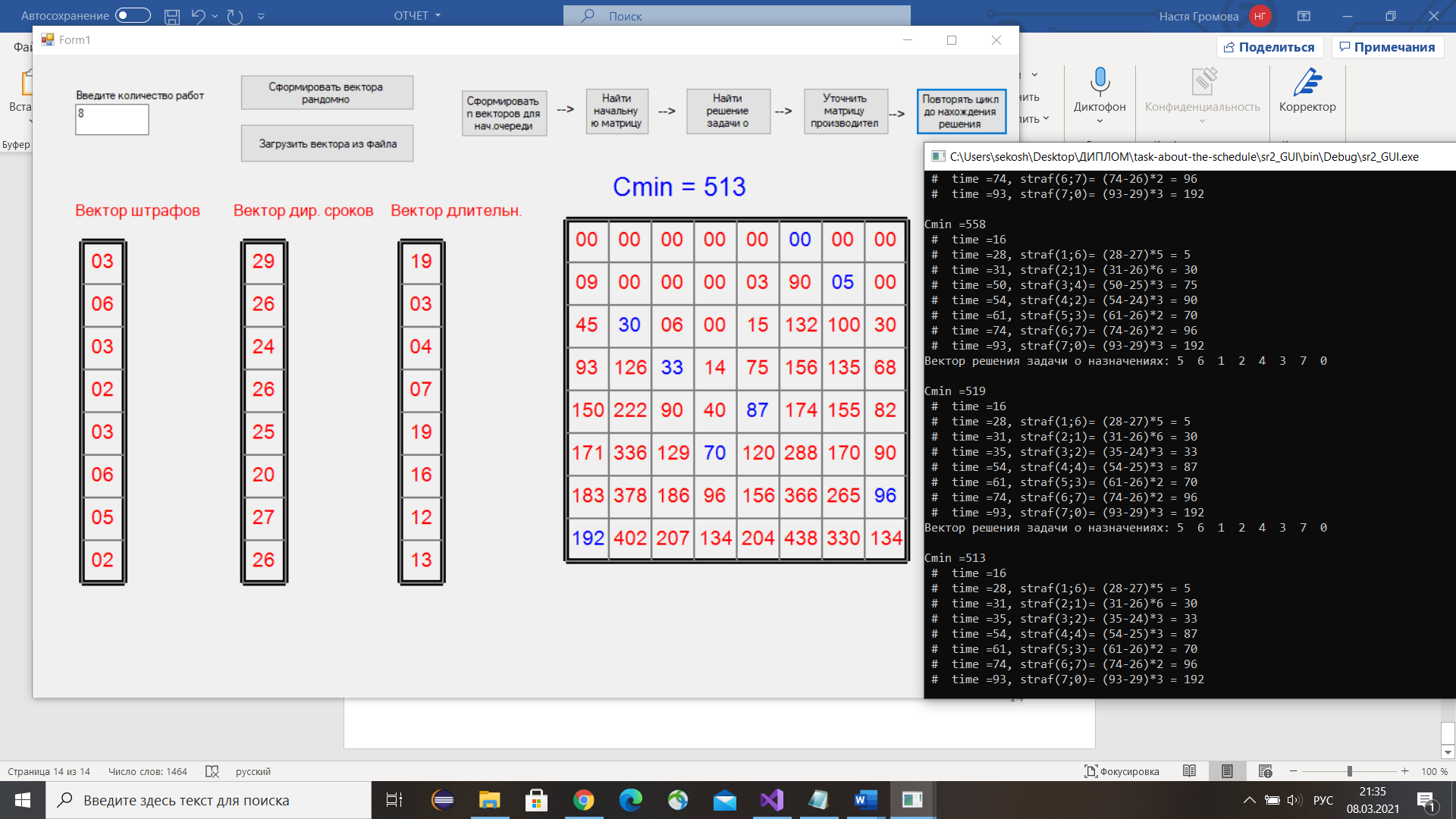
Шаг 3:



Шаг 4:



Шаг 5:



# Заключение

Во время прохождении практики я познакомилась с областью задач теории расписаний. Подробно рассмотрена задача минимизации суммарного штрафа за нарушение директивных сроков в системе с 1 прибором. Так же рассмотрена задача о назначениях с аддитивным критерием и метод ее решения. Предложен и реализован на языке С# итерационный алгоритм решения задачи теории расписаний, который основан на построении и решении задач о назначении.

# Список литературы

1. Гэри М., Джонсон Д. Вычислительные машины и труднорешаемые задачи. Изд.: Мир, 1982. 416 с.
2. Лазарев А.А., Гафаров Е.Р. Теория расписаний. Задачи и алгоритмы. Изд.: Московский государственный университет им. М.В. Ломоносова, 2011, 222 стр.
3. Прилуцкий М.Х., Кумагина Е.А. Задача упорядочения работ как задача о назначениях // Вестник Нижегородского государственного университета. Математическое моделирование и оптимальное управление. Нижний Новгород: Изд-во ННГУ, 1999. Вып. 21. C. 18–24.
4. Танаев В.С., Теория расписаний. – М.: Знание, 1988.

# Приложение

1. Расчет матрицы производительностей

class ScheduleTask

{

IVector straf;

IVector dir\_srok;

IVector srok\_vip;

IMatrix straf\_matr;

public int size { get; set; }

public ScheduleTask(IVector str, IVector dir\_sro, IVector srok\_vi)

{

this.straf = str;

this.dir\_srok = dir\_sro;

this.srok\_vip = srok\_vi;

size = str.Size;

straf\_matr = new SomeMatrix(size, size);

}

public IMatrix MakeStrafMatrix(int[,] queue)

{

int time; //для сравнения прошедшего времени с директивными сроками.

int size1 = queue.Length / size;

for (int i=0; i<size1; i++)

{

time = 0;

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

{

time += (int)srok\_vip[queue[i,j]];

Console.Write(" # time =" + time);

if (time >= dir\_srok[queue[i,j]])

{

//какая по порядку выпб номер заявки

straf\_matr[j, queue[i, j]] = (time - dir\_srok[queue[i, j]]) \* straf[queue[i, j]];

Console.Write(", straf(" + j + ";" + queue[i, j] + ")" + "= (" + time + "-" + dir\_srok[queue[i, j]] + ")\*" + straf[queue[i, j]] + " = " + straf\_matr[j, queue[i, j]]);

}

else

{

straf\_matr.SetValue(0, j, queue[i, j]);

}

Console.WriteLine();

}

}

return straf\_matr;

}

}

1. Венгерский алгоритм

public sealed class HungarianAlgorithm

{

private readonly int[,] \_costMatrix; //матрица производительностей

private int \_inf;

private int \_n; //количество работ-работников

private int[] \_V; //"поблажки" для работнков

private int[] \_U; //

private bool[] \_s;

private bool[] \_t;

private int[] \_matchX;

private int[] \_matchY;

private int \_maxMatch;

private int[] \_slack;

private int[] \_slackx;

private int[] \_prev; //запоминание путей

public HungarianAlgorithm(int[,] costMatrix)

{

\_costMatrix = costMatrix;

}

public int[] Run()

{

\_n = \_costMatrix.GetLength(0);

\_V = new int[\_n];

\_U = new int[\_n];

\_s = new bool[\_n];

\_t = new bool[\_n];

\_matchX = new int[\_n];

\_matchY = new int[\_n];

\_slack = new int[\_n];

\_slackx = new int[\_n];

\_prev = new int[\_n];

\_inf = int.MaxValue;

//заполнение \_matchX \_matchY

InitMatches();

//если матрица не ровная возвращаем 0

if (\_n != \_costMatrix.GetLength(1))

return null;

InitLbls();

\_maxMatch = 0;

InitialMatching();

var q = new Queue<int>();

while (\_maxMatch != \_n)

{

q.Clear();

for (var i = 0; i < \_n; i++)

{

\_s[i] = false;

\_t[i] = false;

}

var root = 0;

int x;

var y = 0;

//нахождение неназначенного корня(работника)

for (x = 0; x < \_n; x++)

{

if (\_matchX[x] != -1) continue; //i++

q.Enqueue(x);

root = x;

\_prev[x] = -2;

\_s[x] = true;

break;

}

for (var i = 0; i < \_n; i++)

{

\_slack[i] = \_costMatrix[root, i] - \_V[root] - \_U[i];

\_slackx[i] = root;

}

while (true)

{

while (q.Count != 0)

{

x = q.Dequeue();

var lxx = \_V[x];

for (y = 0; y < \_n; y++)

{

if (\_costMatrix[x, y] != lxx + \_U[y] || \_t[y]) continue;

if (\_matchY[y] == -1) break;

\_t[y] = true;

q.Enqueue(\_matchY[y]);

AddToTree(\_matchY[y], x);

}

if (y < \_n) break;

}

if (y < \_n) break;

UpdateLabels();

for (y = 0; y < \_n; y++)

{

if (\_t[y] || \_slack[y] != 0) continue;

if (\_matchY[y] == -1)

{

x = \_slackx[y];

break;

}

\_t[y] = true;

if (\_s[\_matchY[y]]) continue;

q.Enqueue(\_matchY[y]);

AddToTree(\_matchY[y], \_slackx[y]);

}

if (y < \_n) break;

}

\_maxMatch++;

int ty;

for (int cx = x, cy = y; cx != -2; cx = \_prev[cx], cy = ty)

{

ty = \_matchX[cx];

\_matchY[cy] = cx;

\_matchX[cx] = cy;

//Console.Write("\nработник" + cx + " выбрал " + cy + "работу");

}

}

Console.Write("Вектор решения задачи о назначениях: ");

for (int i = 0; i < \_n; i++)

Console.Write(\_matchX[i] + " ");

Console.WriteLine();

return \_matchX;

}

private void InitMatches()

{

for (var i = 0; i < \_n; i++)

{

\_matchX[i] = -1;

\_matchY[i] = -1;

}

}

private void InitLbls()

{

for (var i = 0; i < \_n; i++)

{

var minRow = \_costMatrix[i, 0];

for (var j = 0; j < \_n; j++)

{

if (\_costMatrix[i, j] < minRow) minRow = \_costMatrix[i, j];

if (minRow == 0) break;

}

\_V[i] = minRow;

}

for (var j = 0; j < \_n; j++)

{

var minColumn = \_costMatrix[0, j] - \_V[0];

for (var i = 0; i < \_n; i++)

{

if (\_costMatrix[i, j] - \_U[i] < minColumn) minColumn = \_costMatrix[i, j] - \_V[i];

if (minColumn == 0) break;

}

\_U[j] = minColumn;

}

}

private void UpdateLabels()

{

var delta = \_inf;

for (var i = 0; i < \_n; i++)

if (!\_t[i])

if (delta > \_slack[i])

delta = \_slack[i];

for (var i = 0; i < \_n; i++)

{

if (\_s[i])

\_V[i] = \_V[i] + delta;

if (\_t[i])

\_U[i] = \_U[i] - delta;

else \_slack[i] = \_slack[i] - delta;

}

}

private void AddToTree(int x, int prevx)

{

\_s[x] = true;

\_prev[x] = prevx;

var lxx = \_V[x];

for (var y = 0; y < \_n; y++)

{

if (\_costMatrix[x, y] - lxx - \_U[y] >= \_slack[y]) continue;

\_slack[y] = \_costMatrix[x, y] - lxx - \_U[y];

\_slackx[y] = x;

}

}

private void InitialMatching()

{

for (var x = 0; x < \_n; x++)

{

for (var y = 0; y < \_n; y++)

{

if (\_costMatrix[x, y] != \_V[x] + \_U[y] || \_matchY[y] != -1) continue;

\_matchX[x] = y;

\_matchY[y] = x;

\_maxMatch++;

break;

}

}

}

}