Филиал федерального государственного бюджетного образовательного учреждения
высшего образования
«Национальный исследовательский университет «МЭИ»
в городе Смоленске

Кафедра вычислительной техники

Отчет
по лабораторной работе №3
Тема: «Решение дискретных задач оптимизации генетическими алгоритмами»
Предмет: «Интеллектуальный анализ данных и знаний»

Студент: Старостенков А.А.
Группа: ВМ-22 (маг)
Вариант: 22
Преподаватели: Зернов М. М.

Смоленск
2023

1. Задание

С помощью генетического алгоритма решить задачу дискретной оптимизации.

Вариант задания №2: Нахождение гамильтонова обхода в неориентированном графе.

2. Ход работы

Постановка задачи

Дан какой-либо неориентированный граф, по которому необходимо проложить путь. Соответственно, на решение накладываются следующие ограничения:

- Маршрут должен быть таким, чтобы пройти через каждую вершину ровно один раз
- Поскольку в данном случае рассматривается путь, то исходная вершина и конечная должны быть разными
- Количество ребер должно быть на 1 меньше, чем вершин.
- Так как граф неориентированный, то должен быть не только путь от исходной точки к конечной, но и обратный

В данном случае граф берется из файла или же задаются вручную через матрицу смежности.

3. Описание генетического алгоритма решения задачи

1. Читаем матрицу смежности из текстового файла либо задаем вручную. Главная диагональ должна быть пустой, проверяем на дубликаты.
2. В качестве особи будем рассматривать какой-нибудь маршрут в графе.
3. Функция оценки приспособленности:
    а. Принимаем ребра что есть в выбранном пути.
    б. Происходит штраф, максимальный если последовательность пустая
    в. Ищем вершины, у которых степень больше 2
    г. Узнаем, через все ли вершины проходит путь. Берем первое ребро из списка, сжигаем вершины и идем дальше по списку, сравнивая через уже горящие вершины.
    д. Считаем длину пути

е. После чего штрафуем на разницу между количеством вершин и количеством ребер.

4. Оператор селекции выбирает случайную пару родителей из определённого числа лучших особей популяции.
5. Оператор селекции выбирает разделитель в хромосоме, до разделителя записываются:
    а. Для первого потомка: последовательно элементы первого родителя.
    б. Для второго потомка: последовательно элементы второго родителя.

После разделителя записываются:

    в. Для первого потомка: последовательно элементы второго родителя
    г. Для второго потомка: последовательно элементы первого родителя.

6. Из двух этих пар выбирается самая удачная особь
7. Оператор мутации выбирает случайный разделитель в хромосоме, и производит случайную инверсию, после чего производится перерасчет приспособленности
8. Ход генетического алгоритма:
    а. Сгенерировать заданное количество особей.
    б. Отсортировать особи по приспособленности.
    в. Провести размножение, для новых особей выполнить мутацию с заданной вероятностью и посчитать приспособленность.
    г. Отсортировать популяцию по приспособленности.
    д. Затем происходит отбор лучших особей до исходного заданного количества особей.
    е. Второе поколение возвращается к шагу «в» и дальше по циклу, пока не дойдет до последнего, которое также задается пользователем.

4. Схема алгоритма

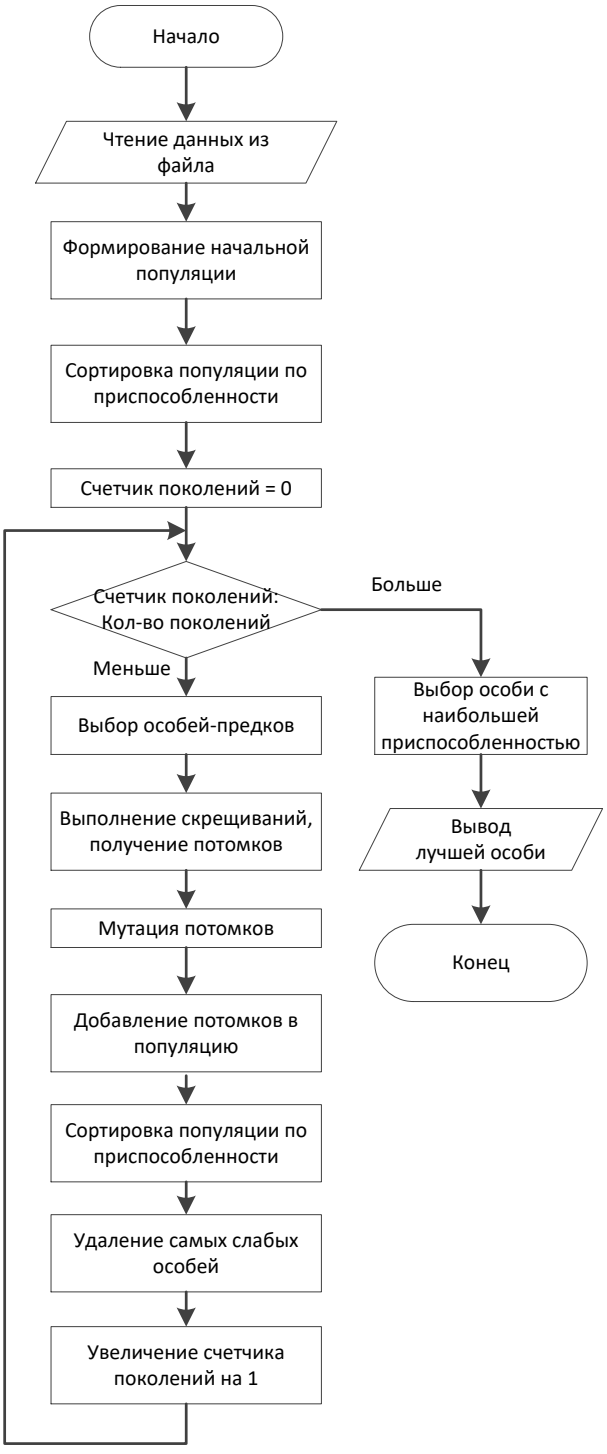

Рисунок 1 – схема генетического алгоритма

5. Результаты работы программы:

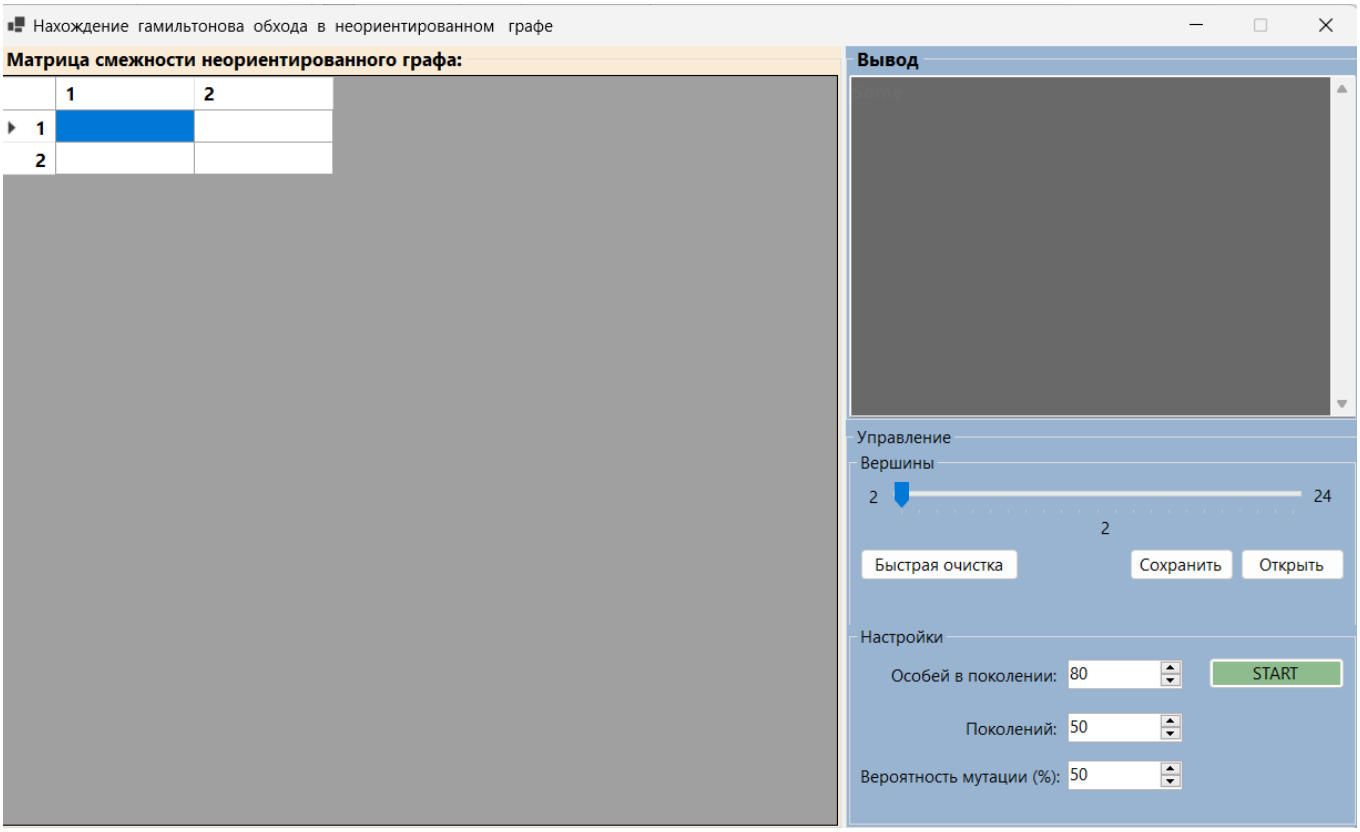

Рисунок 2 – окно программы

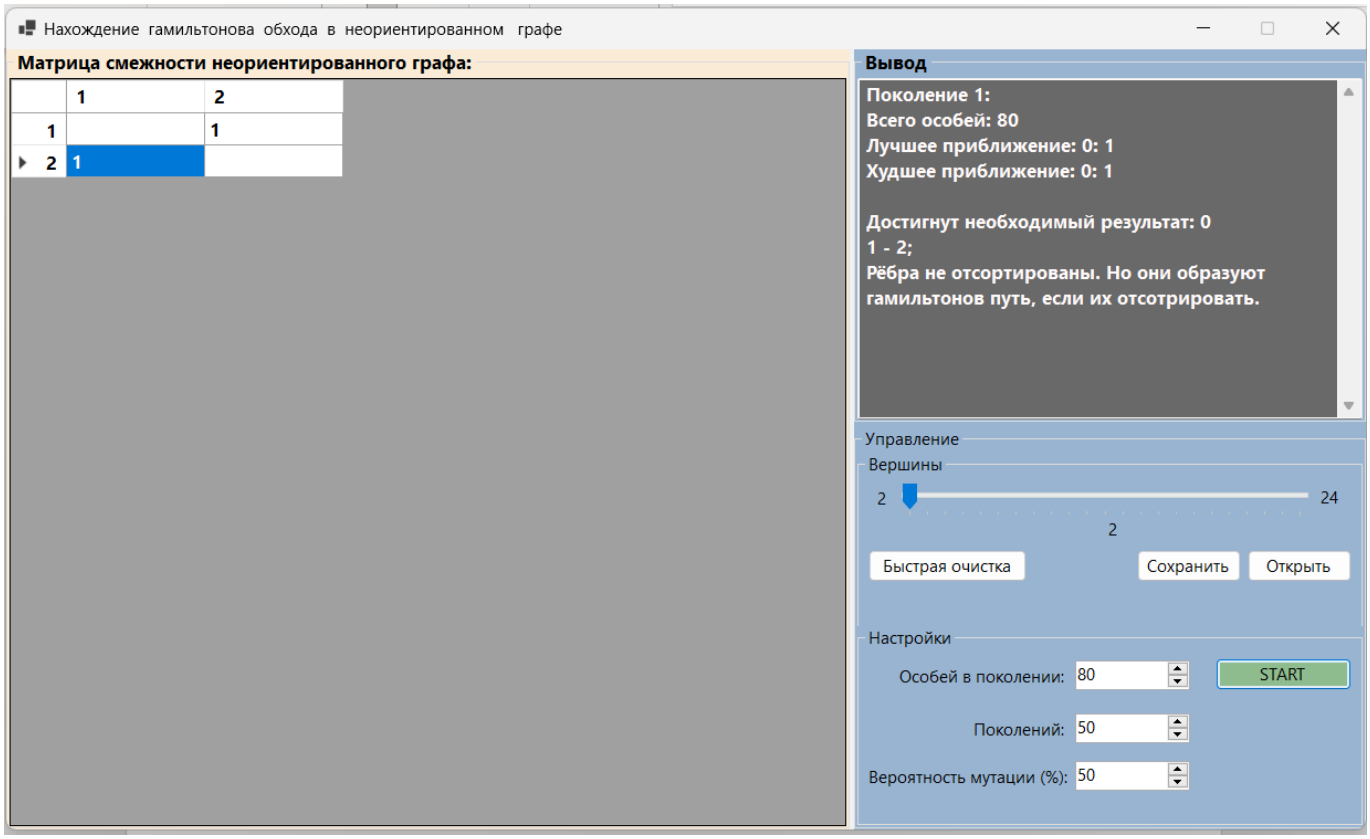

Рисунок 4 – результат работы программы с ручным вводом данных

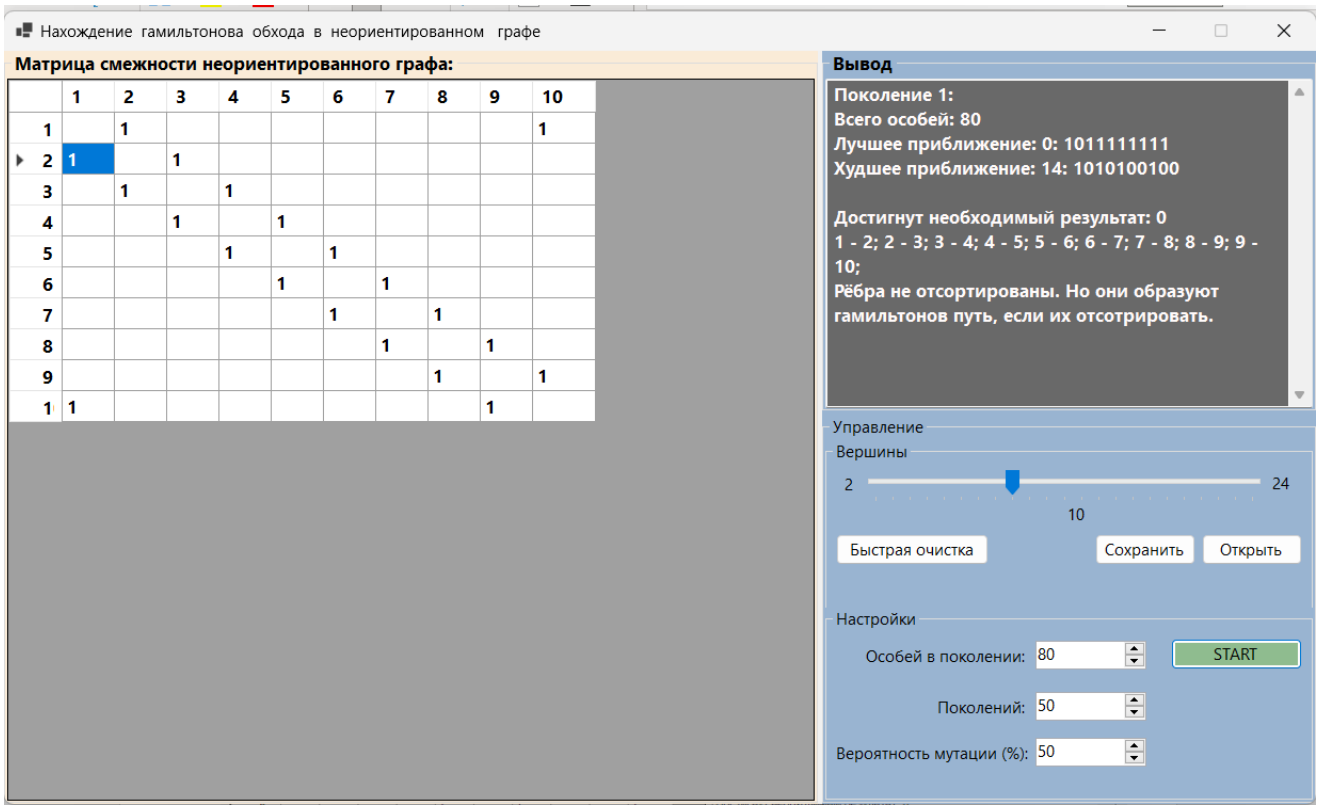

Рисунок 4 – результат работы программы с загруженными из файла данными

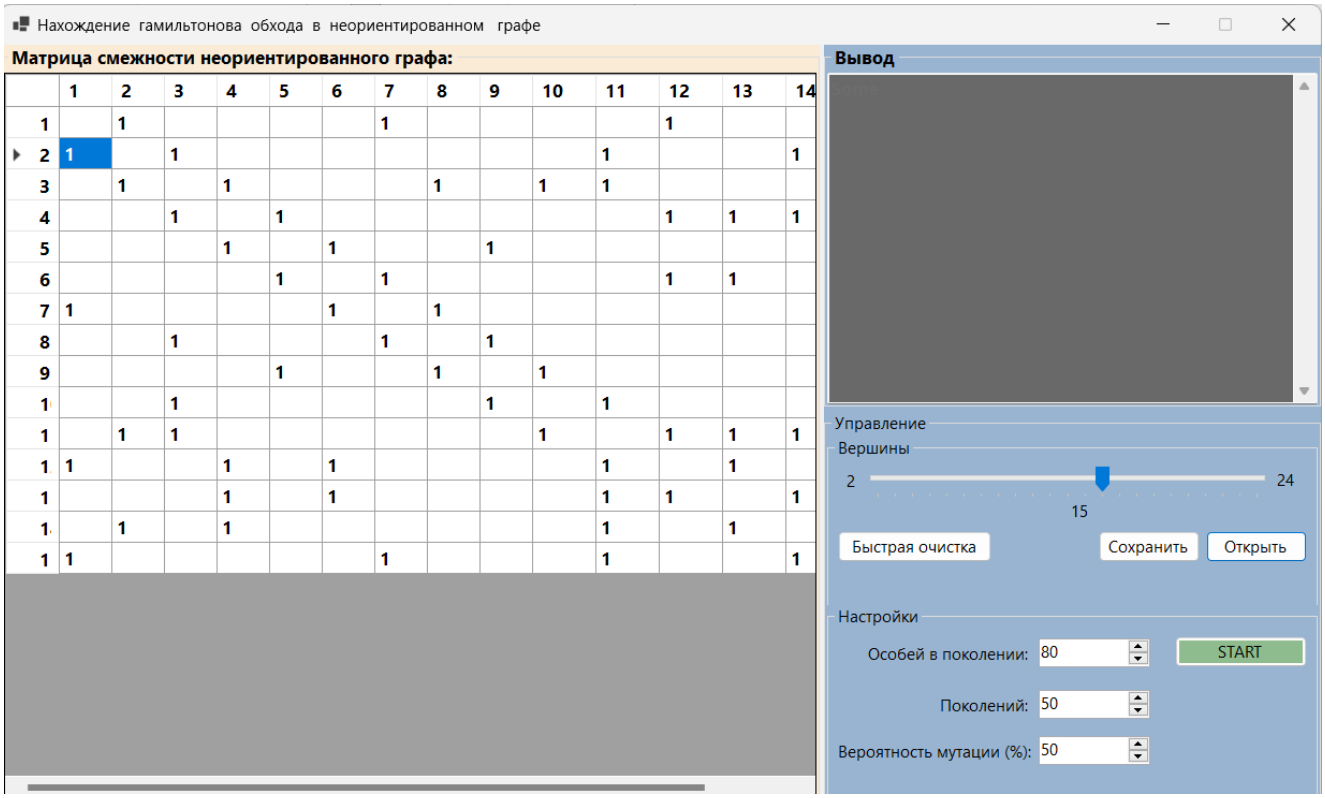

Рисунок 4 – результат работы программы с загруженными из файла данными дополненными вручную

Приложение А. Текст программы

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace HamiltonUndirectedGraph.Models;

public class GraphModel
{
    public int VertexCount { get; private set; }
    public List<int[]> Edges { get; private set; }

    public GraphModel(int vertexCount)
    {
        if (vertexCount >= 2)
        {
            VertexCount = vertexCount;
            Edges = new List<int[]>();
        }
    }

    public int AddEdge(int topA, int topB)
    {
        //Вершины в графе нумеруются с 0, номер вершины должен быть меньше количества
вершин в графе. Также запретим создавать петли из вершины в саму себя
        bool checkInput = (topA < VertexCount) && (topB < VertexCount) && (topA >= 0) &&
(topB >= 0) && (topA != topB);
        if (!checkInput) return -1;

        //Пусть в рёбрах, для порядка, вершины идут от меньшей к большей
        if (topA > topB)
        {
            var temp = topA;
            topA = topB;
            topB = temp;
        }

        //Проверим на наличие дубликатов
        var duplicate = false;
        foreach (var edge in Edges)
        {
            //Мы знаем, что у нас вершины в рёбрах упорядочены, так что, просто проверяем
их на соответствие
            if (topA == edge[0] && topB == edge[1]) duplicate = true;
        }

        //Если дубликатов нет, то закинем ребро в список. Если есть, то ничего не делаем.
        if (!duplicate)
        {
            Edges.Add(new int[] { topA, topB });
            return 1;
        }

        return 0;
    }
}
```

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using static System.Windows.Forms.VisualStyles.VisualStyleElement.Rebar;

namespace HamiltonUndirectedGraph.Models;

//В качестве особи будем рассматривать какой-нибудь маршрут в графе
public class CreatureModel
{
    private GraphModel _liveArea; //Граф, для которого построен маршрут
    private int _fitness; //Её будем стараться минимизировать. Но при этом, это должно быть
неотрицательное значение.

    private Random _rand = new Random();

    //Для каждого ребра в графе будем ставить 0, если он не включён в маршрут и 1, если он
в него включен.
    public string GeneticCode { get; private set; }

    public CreatureModel(GraphModel lifeArea)
    {
        _liveArea = lifeArea;
        GeneticCode = "";

        StringBuilder genCode = new StringBuilder();

        //Генерируем случайную последовательность 0 и 1, заданной длины
        for(int i = 0; i < _liveArea.Edges.Count; i++)
        {
            if (_rand.Next(2) == 0) genCode.Append('0');
            else genCode.Append('1');

            GeneticCode = genCode.ToString();
            _fitness = -1;
        }
    }

    //Функция приспособленности
    public int GetFitness()
    {
        //В гамильтоновом цикле (именно цикле) столько рёбер, сколько вершин в исходном
графе

        //Если ранее не считали приспособленность, считаем её
        if (_fitness == -1) return this.CalcFitness();

        return _fitness;
    }

    private int CalcFitness()
    {
        _fitness = 0;

        //Добавим сюда только те рёбра, что есть в выбранном пути
        List<int[]> usedEdges = new List<int[]>();

        for (int i = 0; i < _liveArea.Edges.Count; i++)
        {
            if (GeneticCode[i] == '1')
            {
                int[] temp = new int[2];
```

```
                _liveArea.Edges[i].CopyTo(temp, 0);
                usedEdges.Add(temp);
            }
        }

        //Если сгенерировалась пустая последовательность, всё ломается.
        //Поэтому рассмотрим этот случай отдельно и дадим ему сразу большой штраф
        if (usedEdges.Count == 0)
        {
            _fitness = 9999;
            return _fitness;
        }

        //В гамильтоновом пути не должно быть вершин, имеющих степень больше 2.
        //Найдём такие вершины, если они есть.
        int overload = 0;
        for (int i = 0; i < _liveArea.VertexCount; i++)
        {
            int counter = 0;
            foreach (int[] tempEdge in usedEdges)
            {
                if (tempEdge[0] == i || tempEdge[1] == i) counter++;
            }

            //Чем выше степень у вершины, тем хуже.
            //В идеале, степени всех вершин либо равны 2,
            //либо среди них есть 2 вершины со степенями 1 (начало и конец пути)
            if (counter > 2) overload += counter - 2;
        }

        //Узнаем, через все ли вершины проходит наш путь.
        //Для этого, будем "сжигать" все вершины, до которых можно по нему добраться

        List<int> burnedVertex = new List<int>(); //Те вершины, что уже сгорели

        List<int> burningVertex = new List<int>(); //Те вершины, что горят сейчас

        //Берём первое ребро в пути и запихиваем его вершины в список горящих вершин
        burningVertex.Add(usedEdges[0][0]);
        burningVertex.Add(usedEdges[0][1]);

        //Само ребро сразу выпиливаем, ибо оно сгорело.
        usedEdges.RemoveAt(0);

        //Заодно посчитаем длину пути
        int pathLength = 1;
        while (burningVertex.Count > 0)
        {
            //Это будут вершины, которые мы подожжём на текущей итерации
            List<int> newBurningVertex = new List<int>();

            //В каждом оставшемся ребре ищем "горящие" вершины...
            foreach (int[] tempEdge in usedEdges)
            {
                //Сравниваем каждую горящую вершину с каждой вершиной рассматриваемого
ребра
                foreach (int tempVertex in burningVertex)
                {
                    if (tempVertex == tempEdge[0])
                    {
                        //Поджигаем противоположную вершину ребра
                        newBurningVertex.Add(tempEdge[1]);

                        //Сжигаем ребро
                        tempEdge[0] = -1;
```

```
                        //Т.к. во время прохода по списку, нельзя удалять его элементы,
                        //будем их просто менять, чтобы они больше не мешали
                        tempEdge[1] = -1;

                        pathLength++;
                        break; //И больше его не рассматриваем
                    }

                    //То же самое для противоположного ребра, если оно не сгорело
                    if (tempVertex == tempEdge[1])
                    {
                        //Поджигаем противоположную вершину ребра
                        newBurningVertex.Add(tempEdge[0]);

                        //Сжигаем ребро
                        tempEdge[0] = -1;

                        //Т.к. во время прохода по списку, нельзя удалять его элементы,
                        //будем их просто менять, чтобы они больше не мешали
                        tempEdge[1] = -1;
                        pathLength++;
                        break;//И больше его не рассматриваем
                    }
                }
            }

            //То, что горело на этой итерации, считаем сгоревшим.
            burnedVertex.AddRange(burningVertex);
            burningVertex.Clear();

            //Добавляем то, что подожгли...
            burningVertex.AddRange(newBurningVertex);
        }

        //Штрафуем особь на разницу между количеством вершин в графе и количеством
сожжённых вершин...
        _fitness += Math.Abs(burnedVertex.Count - _liveArea.VertexCount);

        //Гамильтонов путь (не являющийся циклом) имеет длину, равную количеству вершин
графа минус 1.
        //Если это цикл, то просто количеству вершин графа.
        //В принципе, если убрать из следующей строки это "-1",
        //то оно будет искать гамильтонов цикл, а не путь.
        _fitness += Math.Abs(_liveArea.VertexCount - 1 - pathLength);

        _fitness += overload; //Добавим штрафы за слишком большие степени вершин, если
таковые есть

        return _fitness;
    }

    //Расшифровываем геном особи и выводим его в удобоваримом виде
    public string DecodeGenome()
    {
        string res = "";
        for (int i = 0; i < GeneticCode.Length; i++)
        {
            //Добавляем единички, т.к. в таблице нумерация идёт не с нуля
            if (GeneticCode[i] == '1') res += (_liveArea.Edges[i][0] + 1).ToString() + " -
" + (_liveArea.Edges[i][1] + 1).ToString() + "; ";
        }
        return res;
    }
```

```
    //Первая важная фишка генетических алгоритмов. Тут существа должны плодиться и
размножаться
    public CreatureModel Crossover(CreatureModel partner)
    {
        CreatureModel child0 = new CreatureModel(_liveArea);
        CreatureModel child1 = new CreatureModel(_liveArea);

        int slice = _rand.Next(_liveArea.Edges.Count);

        //Выбираем точку разреза и комбинируем гены предков
        child0.GeneticCode = GeneticCode.Substring(0, slice) +
partner.GeneticCode.Substring(slice);

        //Данный кроссовер допускает 2 варианта комбинации генов... Будем выбирать лучший
из них
        child1.GeneticCode = partner.GeneticCode.Substring(0, slice) +
GeneticCode.Substring(slice);

        //Отдаём наиболее удачную особь
        if (child0.GetFitness() < child1.GetFitness()) return child0;
        else return child1;
    }

    //Вторая фишка генетических алгоритмов - случайные мутации.
    //Бывают разными. Здесь ограничимся только инверсией
    public void MutateMe()
    {
        //Так как генокод меняется, нужно обновить приспособленность
        _fitness = -1;

        //Просто инвертируем случайный бит в генокоде
        int point = _rand.Next(_liveArea.Edges.Count);

        if (GeneticCode[point] == '1') GeneticCode = GeneticCode.Remove(point,
1).Insert(point, "0");
        else GeneticCode = GeneticCode.Remove(point, 1).Insert(point, "1");
    }
}


using HamiltonUndirectedGraph.Models;
using HamiltonUndirectedGraph.Services;
using Newtonsoft.Json;
using System.Text.Json;

namespace HamiltonUndirectedGraph
{
    public partial class Form1 : Form
    {
        private IMessageService _srvMsg;

        public Form1(IMessageService srvMsg)
        {
            _srvMsg = srvMsg;
            InitializeComponent();

            label4.Text = Convert.ToString(trackBar1.Value + 2);//Показываем текущее
значение
            dataGridView1.RowCount = trackBar1.Value + 2;//Меняем параметры таблицы
            dataGridView1.ColumnCount = trackBar1.Value + 2;
            for (int i = 0; i < dataGridView1.RowCount; i++)
            {//Подписываем хэдеры
                dataGridView1.Rows[i].HeaderCell.Value = Convert.ToString(i + 1);
                dataGridView1.Columns[i].HeaderText = Convert.ToString(i + 1);
```

```
            }
            dataGridView1.AutoResizeColumns();
        }

        //Меняем размеры таблицы смежности и, соответственно, количество вершин графа
        private void trackBar1_Scroll(object sender, EventArgs e)
        {
            label4.Text = Convert.ToString(trackBar1.Value + 2);//Показываем текущее
значение
            dataGridView1.RowCount = trackBar1.Value + 2;//Меняем параметры таблицы
            dataGridView1.ColumnCount = trackBar1.Value + 2;
            for (int i = 0; i < dataGridView1.RowCount; i++)
            {//Подписываем хэдеры
                dataGridView1.Rows[i].HeaderCell.Value = Convert.ToString(i + 1);
                dataGridView1.Columns[i].HeaderText = Convert.ToString(i + 1);
            }
            dataGridView1.AutoResizeColumns();
        }

        private void button1_Click(object sender, EventArgs e)
        {
            //Нормальные методы очистки не завезли, делаем напрямую через циклы
            for (int i = 0; i < dataGridView1.ColumnCount; i++)
            {
                for (int j = 0; j < dataGridView1.RowCount; j++)
                {
                    dataGridView1[i, j].Value = null;
                }
            }
            textBox1.Clear();
        }

        private void button3_Click(object sender, EventArgs e)
        {
            try
            {
                saveFileDialog1.Title = "Сохранить данные в файл";
                if (saveFileDialog1.ShowDialog() == System.Windows.Forms.DialogResult.OK)
                {//Выбран файл

                    bool[,] temp = new bool[dataGridView1.ColumnCount,
dataGridView1.RowCount];
                    for (int i = 0; i < dataGridView1.ColumnCount; i++)
                    {
                        for (int j = 0; j < dataGridView1.RowCount; j++)
                            if (Convert.ToString(dataGridView1[i, j].Value) != "")//Если в
ячейке не пусто, то есть связь
                            {
                                dataGridView1[i, j].Value = "1";//На всякий случай,
перезапишем туда 1
                                temp[i, j] = true;
                            }
                            else { temp[i, j] = false; }
                    }

                    //Сериализуем в строку
                    string json = JsonConvert.SerializeObject(temp, Formatting.Indented);
                    StreamWriter sw = new
StreamWriter(saveFileDialog1.FileName);//Сохраняем в виде текстового документа
                    sw.WriteLine(json);//Сохраним
                    sw.Close();
                }
            }
            catch (Exception ex)
            {
```

```
                _srvMsg.ShowError(ex.Message);
            }
        }

        private void button4_Click(object sender, EventArgs e)
        {
            openFileDialog1.Title = "Открыть файл";
            openFileDialog1.InitialDirectory = Environment.CurrentDirectory;
            if (openFileDialog1.ShowDialog() == System.Windows.Forms.DialogResult.OK)
            {
                button1_Click(null, null);//Очистим рабочее поле
                StreamReader sr = new StreamReader(openFileDialog1.FileName);//Прочитаем
файл
                string temp = sr.ReadLine();//Строка для десериализации. Ожидаем одну
строку
                sr.Close();

                bool[,] testtest = JsonConvert.DeserializeObject<bool[,]>(temp);//Получаем
матрицу из строки
                trackBar1.Value = testtest.GetLength(0) - 2;//Устанавливаем ползунки в
соответствии с тем, что загрузили
                trackBar1_Scroll(null, null);//Эта функция подгонит габартиы таблицы под
то, что загрузили
                for (int i = 0; i < testtest.GetLength(0); i++)
                {
                    for (int j = 0; j < testtest.GetLength(1); j++)
                    {
                        if (testtest[i, j])
                        {
                            dataGridView1[i, j].Value = 1;//заполняем.
                        }
                    }
                }

            }
        }

        //Сделаем более удобный ввод в таблицу
        private void dataGridView1_CellClick(object sender, DataGridViewCellEventArgs e)
        {
            if (e.RowIndex >= 0 && e.ColumnIndex >= 0) //Если это убрать, прога будет
вылетать при попытке тыкать по полям таблицы
            {
                DataGridViewCell selected_cell =
dataGridView1.Rows[e.RowIndex].Cells[e.ColumnIndex];
                if (e.RowIndex != e.ColumnIndex) //Если строка равна столбцу, то это ребро
является петлёй, в цикле его точно не будет, поэтому, запретим его вводить.
                {
                    DataGridViewCell mirror_cell =
dataGridView1.Rows[e.ColumnIndex].Cells[e.RowIndex]; // Так как граф у нас не
ориентированный, если есть путь из A в B, то должен быть и из B в A
                    if (selected_cell.Value == null)
                    {
                        selected_cell.Value = "1";
                        mirror_cell.Value = "1";
                    }
                    else
                    {
                        selected_cell.Value = null;
                        mirror_cell.Value = null;
                    }

                }
                else //На всякий случай, очистим ячейку, лежащую на главной диагонали. А
то, мало ли, что у нас может быть в сохранённом файле...
```

```
                {
                    selected_cell.Value = null;
                }

            }
        }

        private void button2_Click(object sender, EventArgs e)
        {
            textBox1.Clear();
            GraphModel field = new GraphModel(trackBar1.Value + 2); //Число вершин задаётся
в интерфейсе.
            for (int i = 0; i < field.VertexCount; i++)
            {
                for (int j = i; j < field.VertexCount; j++)
                {
                    if (dataGridView1[i, j].Value != null)
                    {
                        field.AddEdge(i, j);
                    }
                }
            }
            List<CreatureModel> population = new List<CreatureModel>();
            int populationCount = Convert.ToInt32(PopulationSize.Value);
            int gens = Convert.ToInt32(GenerationsCount.Value);
            int chance = Convert.ToInt32(MutateChance.Value);
            Random rand = new Random();

            for (int i = 0; i < populationCount; i++)//Генерируем начальную популяцию.
            {
                population.Add(new CreatureModel(field));
                System.Threading.Thread.Sleep(1); //Из-за говнокода, постоянно создаются
одинаковые экземпляры класса random и, как следствие, одинаковые особи, которые ничего не
могут породить нормального. Надо делать задержку или использовать глобальный рандом
            }
            for (int i = 0; i < gens; i++)
            {
                population.Sort((a, b) =>
a.GetFitness().CompareTo(b.GetFitness()));//Сортируем список по возрастанию
приспособленности.

                textBox1.AppendText("Поколение " + (i + 1).ToString() + ": " +
Environment.NewLine
                                    + "Всего особей: " + population.Count.ToString() +
Environment.NewLine
                                    + "Лучшее приближение: " +
population[0].GetFitness().ToString() + ": " + population[0].GeneticCode +
Environment.NewLine
                                    + "Худшее приближение: " + population[populationCount /
2 - 1].GetFitness().ToString() + ": " + population[populationCount / 2 - 1].GeneticCode +
Environment.NewLine + Environment.NewLine);

                population.RemoveRange(populationCount / 2, populationCount / 2);//Обрезаем
худшую половину текущей популяции. Почти как Танос, только умнее

                for (int j = 0; j < populationCount / 2; j++) //Проводим кроссовер и
мутации
                {
                    int partnernumber = j; //Выбираем каждой особи партнёра, отличного от
самой особи
                    while (partnernumber == j)
                    {
                        partnernumber = rand.Next(populationCount / 2);
                    }
```

```
                    CreatureModel nextchild =
population[j].Crossover(population[partnernumber]);
                    if (rand.Next(100) < chance) //Проводим мутацию, если надо
                    {
                        nextchild.MutateMe();
                    }
                    population.Add(nextchild);//Закидываем его в популяцию
                }
                if (population[0].GetFitness() == 0)
                {
                    textBox1.AppendText("Достигнут необходимый результат: " +
population[0].GetFitness().ToString() + Environment.NewLine
                        + population[0].DecodeGenome() + Environment.NewLine
                        + "Рёбра не отсортированы. Но они образуют гамильтонов путь, если
их отсотрировать.");
                    break;
                }
            }
        }
    }
}
```