# 4. Метод динамического программирования

## 4.1. Реализация текстовой анимации

Мы продолжаем работу с нашей программой MathPanel. Модифицируем test\_graph.htm, чтобы можно было управлять содержимым графического окна.

<!DOCTYPE html>

<html>

<head>

<meta http-equiv="Content-Type" content="text/html; charset=utf-8"/>

<title>Canvas drawing</title>

<!-- добавить новые стили -->

<style type="text/css">

body { background-color:#ffe; border:2px solid #cfc; }

.red { background-color:#fcc; }

.green { background-color:#cfc; }

#scoretable td { width:24px; }

</style>

<!-- добавить Jquery библиотеку для упрощения кодирования -->

<script type="text/javascript" src="jquery-1.11.1.min.js"></script>

</head>

<body>

<!—добавить новый тег divX -->

<div id="divX">divXCanvas drawing</div>

<!-- скрипт -->

<script type="text/javascript">

//загрузить текст "divX" тег

function ext\_div(w) {

$("#divX").html(w);

return false;

}

</script>

</body>

</html>

Добавим новый метод в MainWindow.xaml.cs

/// <summary>

/// передать html код в веб-компонент

/// </summary>

/// <param name="data">html код</param>

public static void SetHtml(string data)

{

if (!bReady || dispObj.HasShutdownStarted) return;

//мы запускаем код в UI потоке

dispObj.Invoke(delegate

{

webConsole.InvokeScript("ext\_div", data);

});

}

Пишем скрипт test48\_ext\_div.cs.

Dynamo.Console("test48\_ext\_div");

//сообщения

string[] fnames = { "<h5>Wau!</h5>", "<h4>Wau!</h4>", "<h3>Wau!</h3>", "<h2>Wau!</h2>", "<h1>Wau!</h1>" };

//в цикле по всем сообщениям

for (int i = 0; i < fnames.Length; i++)

{

var fn = fnames[i];

//передать html код в веб-компонент

Dynamo.SetHtml(fn);

//заснуть немного

System.Threading.Thread.Sleep(500);

}

Запускаем MathPanel, открываем скрипт, переключаемся на графическое окно, нажимаем «Выполнить», получаем анимацию. На рис.4.1 представлен последний шаг анимации.

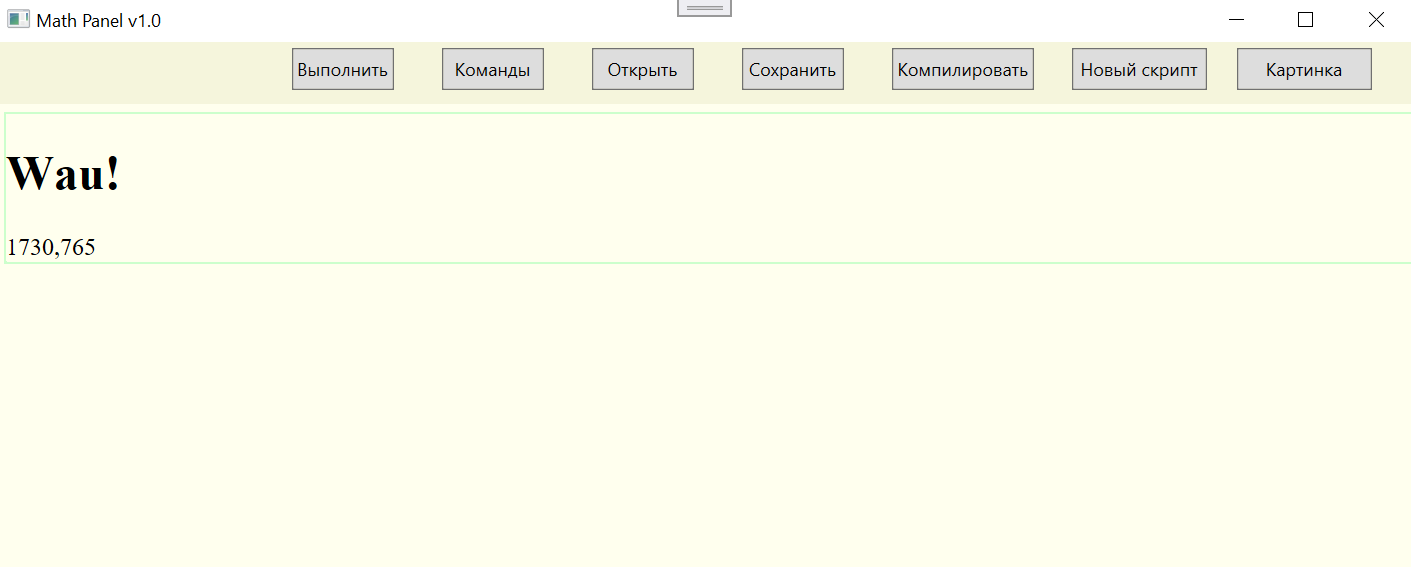


Рис.4.1 Последний шаг анимации.

## 4.2. Задача оптимального выравнивания строк или объектов

Переходим к задаче сравнению (выравнивания) строк. Допускаются несовпадения символов и пропуски. Задача - выровнять 2 строки для наилучшего сопоставления. Эту задачу можно решать методом динамического программирования. Здесь программирование является синонимом слова оптимизация.

Например, есть 2 строки: "optimal construction" и "optimization const".

Оптимальное выравнивание представлено на рис.4.2.

optim al construction

optimization const

Рис.4.2. Оптимальное выравнивание двух строк.

На красном фоне – ошибка сопоставления, на голубом – пропуск. Как критерий оптимальности - сумма поощрений за правильное сопоставление минус сумма штрафов за неверное сопоставление и пропуски. Поощрения и штрафы могут меняться от задачи к задаче.

## 4.3 Как работает метод динамического программирования?

Мы формируем пространство вариантов на пути к цели. Затем начинаем двигаться к цели, на каждом шаге мы решаем локальную оптимизационную задачу, пока не дойдем до окончательного решения. Мы запоминаем маршрут движения к цели. Затем возвращаемся на исходную позицию, восстанавливая промежуточные решения.

В нашем случае сформируем таблицу размерность по ширине=длина 1-го слова + 1,

по высоте=длина 2-го слова + 1.

Мы находимся в самом левой верхней ячейке таблицы. Присваиваем ей значение функционала ноль. Мы можем двигаться по диагонали, влево и вниз.

Движение по диагонали означает, что мы решили выравнивать следующие элементы по горизонтали и вертикали. Движение по горизонтали означает, мы берем следующий элемент по горизонтали, по вертикали остаемся на месте (пропуск). Движение по вертикали означает, мы берем следующий элемент по вертикали, по горизонтали остаемся на месте (пропуск).

Мы можем вычислить веса всех ячеек (оптимальное значение функционала) в первом ряду. Как уже сказано, первая ячейка имеет вес ноль. Вторая – штраф за пропуск, третья – вес второй ячейки минус штраф. И так далее.

Переходим к следующему ряду. Первая ячейка – вес ячейки по вертикали из предыдущего ряда минус штраф. Для каждой последующей – минимум из трех вариантов:

**W[I, J] = W[I-1, J-1] + (Match(I, J) ? MatchFee : MismatchFee)** -

вес предыдущей ячейки по диагонали плюс поощрение при правильном сопоставлении либо штраф при неправильном.

**W[I, J] = W[I, J-1] + SkipFee** -

вес предыдущей ячейки по горизонтали минус штраф за пропуск.

**W[I, J] = W[I-1, J] + SkipFee** -

вес предыдущей ячейки по вертикали минус штраф за пропуск.

Для каждой ячейки запоминаем оптимальный маршрут к ней.

Повторяем эти действия для всех рядов. Вес правой нижней ячейки равен функционалу при оптимальном выравнивании. Так как мы сохраняли переходы в ячейки, то можем восстановить оптимальный путь (из последней ячейки в предыдущую и так далее пока не дойдем до начальной ячейки) (рис. 4.3).

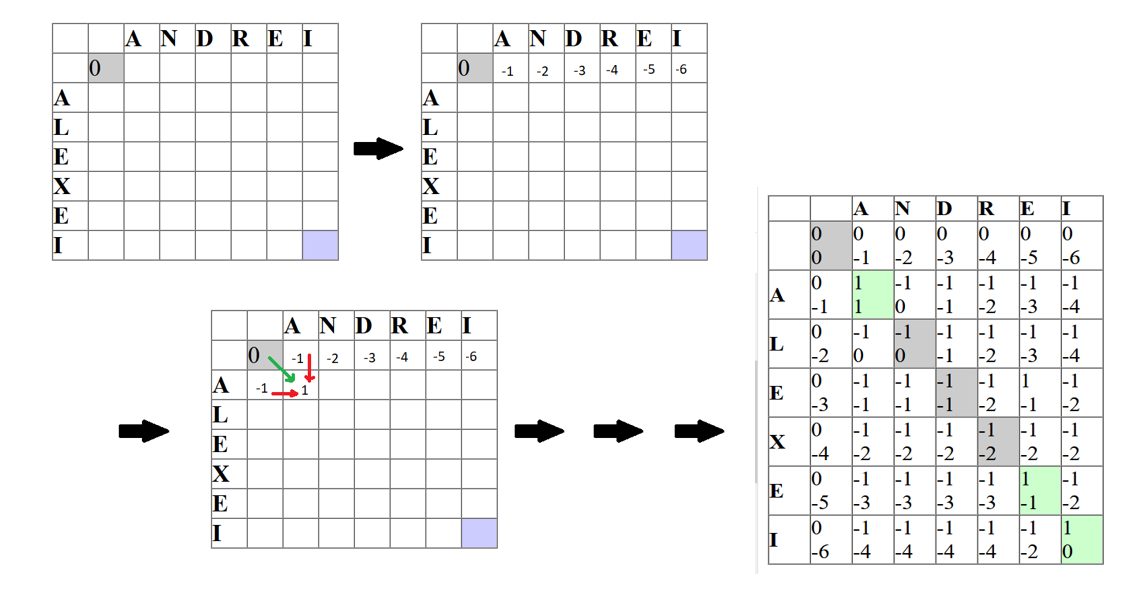


Рис.4.3. Вычисления по методу динамического программирования.

## 4.4. Обзор кода

В соответствии с данным алгоритмом напишем класс Similarica (для поиска подобия).

using System;

using System.Collections.Generic;

using System.Text;

namespace MathPanelExt

{

//простой класс для выравнивания строк и объектов, вычисления похожести

public class Similarica

{

//шаблон делегата для вычисления разности между объектами

public delegate double CalcDifference(object x, object y);

//члены класса

int m\_width, m\_height; //ширина и высота таблицы

double[] m\_weight; //веса или промежуточные результаты ячеек

int[] m\_from; //наилучшие родители

int m\_iType; //режим: 0-сравнить строки, 1-массивы строк, 2-массивы объектов

string m\_sWid, m\_sHei; //строки для сравнения

string[] m\_sliWid; //массивы для сравнения

string[] m\_sliHei;

object[] m\_objWid; //объекты для сравнения

object[] m\_objHei;

double m\_dMax; //итоговый результат

//параметры для изменения на ходу

public double m\_dSkipFee = -1.0; //штраф за пропуск

public double m\_dMisFee = -1.0; //штраф за несопадение

public double m\_dMatchFee = 1.0; //штраф за сопадение

public bool bNoSpace = false; //удалять пробелы из строк

public CalcDifference pc\_hand = null; //обработчик для сравнения объектов

//простой конструктор

public Similarica()

{

m\_dMax = 0;

m\_iType = -1;

}

//сравнить 2 строки

public double Calc(string sWid, string sHei)

{

//копировать данные

m\_sWid = " " + sWid; //' ' для начальной строки

m\_sHei = " " + sHei;

m\_iType = 0;

//выделить место для весов и путей

m\_height = m\_sHei.Length;

m\_width = m\_sWid.Length;

m\_weight = new double[m\_height \* m\_width];

m\_from = new int[m\_height \* m\_width];

//для красивого вывода

m\_sliWid = new string[m\_sWid.Length];

for (int i = 0; i < m\_sWid.Length; i++)

m\_sliWid[i] = m\_sWid.Substring(i, 1);

m\_sliHei = new string[m\_sHei.Length];

for (int i = 0; i < m\_sHei.Length; i++)

m\_sliHei[i] = m\_sHei.Substring(i, 1);

//найти веса и наилучший путь

return DoCalc();

}

//сравнить 2 массива строк

public double Calc(string[] sliWid, string[] sliHei)

{

//копировать данные

m\_sliWid = new string[sliWid.Length + 1];

m\_sliWid[0] = "";

for (int i = 0; i < sliWid.Length; i++)

m\_sliWid[i + 1] = sliWid[i];

m\_sliHei = new string[sliHei.Length + 1];

m\_sliHei[0] = "";

for (int i = 0; i < sliHei.Length; i++)

m\_sliHei[i + 1] = sliHei[i];

m\_iType = 1;

//выделить место для весов и путей

m\_height = m\_sliHei.Length;

m\_width = m\_sliWid.Length;

m\_weight = new double[m\_height \* m\_width];

m\_from = new int[m\_height \* m\_width];

//найти веса и наилучший путь

return DoCalc();

}

//сравнить 2 массива объектов

public double Calc(object[] objWid, object[] objHei, CalcDifference hand)

{

//копировать данные

m\_objWid = new object[objWid.Length + 1];

m\_objWid[0] = "";

for (int i = 0; i < objWid.Length; i++)

m\_objWid[i + 1] = objWid[i];

m\_objHei = new object[objHei.Length + 1];

m\_objHei[0] = "";

for (int i = 0; i < objHei.Length; i++)

m\_objHei[i + 1] = objHei[i];

m\_iType = 2;

//установить обработчик

pc\_hand = hand;

//выделить место для весов и путей

m\_height = m\_objHei.Length;

m\_width = m\_objWid.Length;

m\_weight = new double[m\_height \* m\_width];

m\_from = new int[m\_height \* m\_width];

//для красивого вывода

m\_sliWid = new string[m\_objWid.Length];

m\_sliWid[0] = "";

for (int i = 1; i < m\_objWid.Length; i++)

m\_sliWid[i] = string.Format("{0}", m\_objWid[i].ToString());

m\_sliHei = new string[m\_objHei.Length];

m\_sliHei[0] = "";

for (int i = 1; i < m\_objHei.Length; i++)

m\_sliHei[i] = string.Format("{0}", m\_objHei[i].ToString());

//найти веса и наилучший путь

return DoCalc();

}

//возвращает максимальный вес ячейки

double GetWeight(int row0, int col0)

{

double dMax, weight;

int i0 = row0 - 1; //предыдущая строка

int j0 = col0 - 1; //предыдущая колонка

int ia = row0 \* m\_width + col0;

//переход из предыдущей колонки, эта строка

weight = m\_weight[row0 \* m\_width + j0] + m\_dSkipFee; //добавляем штраф за пропуск

dMax = weight;

m\_from[ia] = -1;

//переход из предыдущей строки, эта колонка

weight = m\_weight[i0 \* m\_width + col0] + m\_dSkipFee; //добавляем штраф за пропуск

if (weight > dMax)

{ //если лучше, запоминаем

dMax = weight;

m\_from[ia] = 1;

}

//переход по диагонали, из предыдущих строки и колонки

weight = m\_weight[i0 \* m\_width + j0] + GetMatch(row0, col0);

if (weight > dMax)

{ //если лучше, запоминаем

dMax = weight;

m\_from[ia] = 0;

}

return dMax;//возвращаем лучший результат

}

//штраф за несопадение или сопадение в строке i и колонке j

double GetMatch(int i, int j)

{

if (m\_iType == 0)

{ //сравнить строки

string ch1 = m\_sHei.Substring(i, 1);

string ch2 = m\_sWid.Substring(j, 1);

//возвращаем результат

return (ch1 == ch2) ? m\_dMatchFee : m\_dMisFee;

}

double d = 0;

if (m\_iType == 1)

{ //сравнить массивы строк

string sl1 = m\_sliHei[i];

string sl2 = m\_sliWid[j];

if (bNoSpace)

{ //удалить пробелы и табуляторы

sl1 = sl1.Replace(" ", "").Replace("\t", "");

sl2 = sl2.Replace(" ", "").Replace("\t", "");

}

//возвращаем результат

d = (sl1 == sl2) ? m\_dMatchFee : m\_dMisFee;

}

else if(pc\_hand != null)

{ //сравнить массивы объектов используя обработчик

var diff = pc\_hand((m\_objHei[i]), (m\_objWid[j]));

d = (diff == 0) ? m\_dMatchFee : m\_dMisFee;

}

return d;

}

//найти веса и наилучший путь

double DoCalc()

{

int i, j;

//инициализирем первую строку весов

for (j = 0; j < m\_width; j++)

{

m\_weight[0 \* m\_width + j] = j \* m\_dSkipFee;

}

//внутри

for (i = 1; i < m\_height; i++)

{ //строка за строкой

m\_weight[i \* m\_width + 0] = i \* m\_dSkipFee;

for (j = 1; j < m\_width; j++)

{

m\_weight[i \* m\_width + j] = GetWeight(i, j);

}

}

//результат в последней строке, в последней колонке

m\_dMax = m\_weight[((m\_height - 1) \* m\_width) + m\_width - 1];

return m\_dMax;

}

//отладочная функция

//генерировать html-таблицу с весами и наилучшим путем

public string Printweights(string style = "font-size:9pt;")

{

int i, j;

//найти наилучшие колонки в обратной последовательности

int[] iBestCol = new int[m\_height];

iBestCol[m\_height - 1] = m\_width - 1;

for (i = m\_height - 1; i > 0; i--)

{

j = iBestCol[i];

while (j >= 0)

{ //как сюда попали

int from = m\_from[i \* m\_width + j];

if (from == 1)

{ //вверх

iBestCol[i - 1] = j;

break;

}

else if (from == 0)

{ //по диагонали

iBestCol[i - 1] = j > 0 ? j - 1 : 0;

break;

}

else

{ //по горизонтали

if (j == 0)

{

iBestCol[i - 1] = 0;

break;

}

j--;

}

}

}

//генерировать html-таблицу

StringBuilder sb = new StringBuilder();

sb.AppendFormat("<table id='scoretable' border='1' cellpadding='0' cellspacing='0' style='{0}'>\n", style);

//первая строка

sb.Append("<tr><td></td>\n");

for (j = 0; j < m\_sliWid.Length; j++)

{

var q = m\_sliWid[j].Trim();

sb.Append(string.Format("<td><b>{0}</b></td>\n", q.Length > 0 ? q.Substring(0, 1) : ""));

}

sb.Append("</tr>\n");

//следущие строки

for (i = 0; i < m\_height; i++)

{

sb.Append("<tr>\n");

var q2 = m\_sliHei[i].Trim();

sb.Append(string.Format("<td><b>{0}</b></td>\n", q2.Length > 0 ? q2.Substring(0, 1) : ""));

//выделяем оптимальный путь

for (j = 0; j < m\_width; j++)

{

double ma = 0.0;

if (i > 0 && j > 0) ma = GetMatch(i, j);

sb.Append(string.Format("<td{0}>{1}<br>{2}</td>\n",

(j == iBestCol[i]) ? " style='background:#cccccc'" : "",

ma, m\_weight[i \* m\_width + j]));

}

sb.Append("</tr>\n");

}

sb.Append("</table>\n");

return sb.ToString();

}

//отладочная функция

//генерировать цветное выравнивание в html

//белый - одинаковые, красный (class='red') - данные из первого массива, зеленый (class='green') - данные из второго массива

public string PrintStrings(string style = "font-size:9pt;")

{

int i, j;

//найти наилучшие колонки в обратной последовательности

int[] iBestCol = new int[m\_height];

iBestCol[m\_height - 1] = m\_width - 1;

for (i = m\_height - 1; i > 0; i--)

{

j = iBestCol[i];

while (j >= 0)

{ //как сюда попали

int from = m\_from[i \* m\_width + j];

if (from == 1)

{ //вверх

iBestCol[i - 1] = j;

break;

}

else if (from == 0)

{ //по диагонали

iBestCol[i - 1] = j > 0 ? j - 1 : 0;

break;

}

else

{ //по горизонтали

if (j == 0)

{

iBestCol[i - 1] = 0;

break;

}

j--;

}

}

}

//генерировать div'ы

StringBuilder sb = new StringBuilder();

sb.AppendFormat("<div style='{0}'>\n", style);

int prevCol = 0;

for (i = 1; i < m\_height; i++)

{

j = iBestCol[i];

if (prevCol + 1 == j)

{ //движение по диагонали

double ma = GetMatch(i, j);

if (ma > 0)

{ //совпали

sb.Append(string.Format("<div class='{0}'>{1}</div>\n", "",

m\_sliHei[i].Replace("<", "&lt;").Replace(">", "&gt;")));

}

else

{ //не совпали, показываем строки из двух массивов

sb.Append(string.Format("<div class='{0}'>{1}</div>\n", "red",

m\_sliWid[j].Replace("<", "&lt;").Replace(">", "&gt;")));

sb.Append(string.Format("<div class='{0}'>{1}</div>\n", "green",

m\_sliHei[i].Replace("<", "&lt;").Replace(">", "&gt;")));

}

}

else if (prevCol == j)

{ //движение вниз, показываем строку из второго массива

sb.Append(string.Format("<div class='{0}'>{1}</div>\n", "green",

m\_sliHei[i].Replace("<", "&lt;").Replace(">", "&gt;")));

}

else

{ //движение вправо, проверяем диагональ

prevCol++;

double ma = GetMatch(i, prevCol);

if (ma > 0)

{ //совпали, одна строка

sb.Append(string.Format("<div class='{0}'>{1}</div>\n", "",

m\_sliHei[i].Replace("<", "&lt;").Replace(">", "&gt;")));

}

else

{ //не совпали, показать 2 строки

sb.Append(string.Format("<div class='{0}'>{1}</div>\n", "red",

m\_sliWid[prevCol].Replace("<", "&lt;").Replace(">", "&gt;")));

sb.Append(string.Format("<div class='{0}'>{1}</div>\n", "green",

m\_sliHei[i].Replace("<", "&lt;").Replace(">", "&gt;")));

}

prevCol++;

while (prevCol <= j)

{ //пропуск, показываем строку из первого массива

sb.Append(string.Format("<div class='{0}'>{1}</div>\n", "red",

m\_sliWid[prevCol].Replace("<", "&lt;").Replace(">", "&gt;")));

prevCol++;

}

}

prevCol = j;

}

sb.Append("</div>");

return sb.ToString();

}

}

}

Вначале идет описания членов класса. Далее конструктор и 3 метода вычислений

public double Calc(string sWid, string sHei);

public double Calc(string[] sliWid, string[] sliHei);

public double Calc(object[] objWid, object[] objHei, CalcDifference hand);

И 2 функции для визуализации результата

public string Printweights(string style = "font-size:9pt;");

public string PrintStrings(string style = "font-size:9pt;");

## 4.5 Сравнение двух строк.

Теперь напишем скрипт test48\_compare\_strings.

//строки для сравнения

string[] fnames = { "optimal construction", "optimization const" };

//создать объект класса Similarica

var solv = new Similarica();

//найти веса и наилучший путь

double dScore = solv.Calc(fnames[0], fnames[1]);

Dynamo.Console("Score=" + dScore);

//генерировать html-таблицу с весами и наилучшим путем

var sWg = solv.Printweights("font-size:14pt;");

//генерировать цветное выравнивание в html

var sRs = solv.PrintStrings("font-size:14pt;");

for (int i = 0; i < 10; i++)

{

//вывести результат по очереди

Dynamo.SetHtml(i % 2 == 0 ? sWg : sRs);

//спать

System.Threading.Thread.Sleep(2000);

}

Запускаем скрипт, переходим в «График», получаем 2 изображения. Первое - таблица с весами и оптимальным путем (серые ячейки) (рис. 4.4). Второе - построчная визуализация результата выравнивания (рис. 4.5).

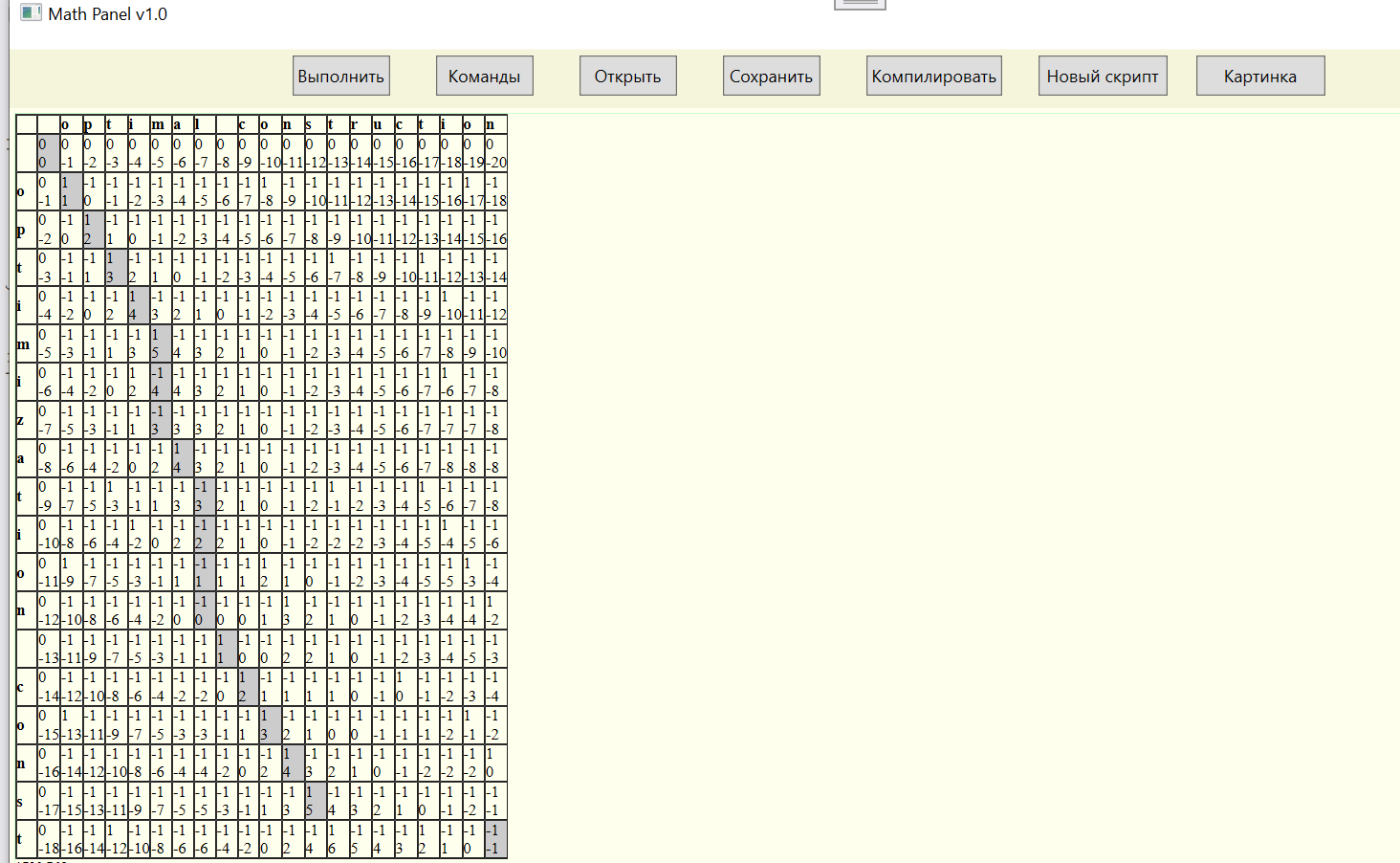


Рис. 4.4. Таблица с весами и оптимальным путем (серые ячейки).

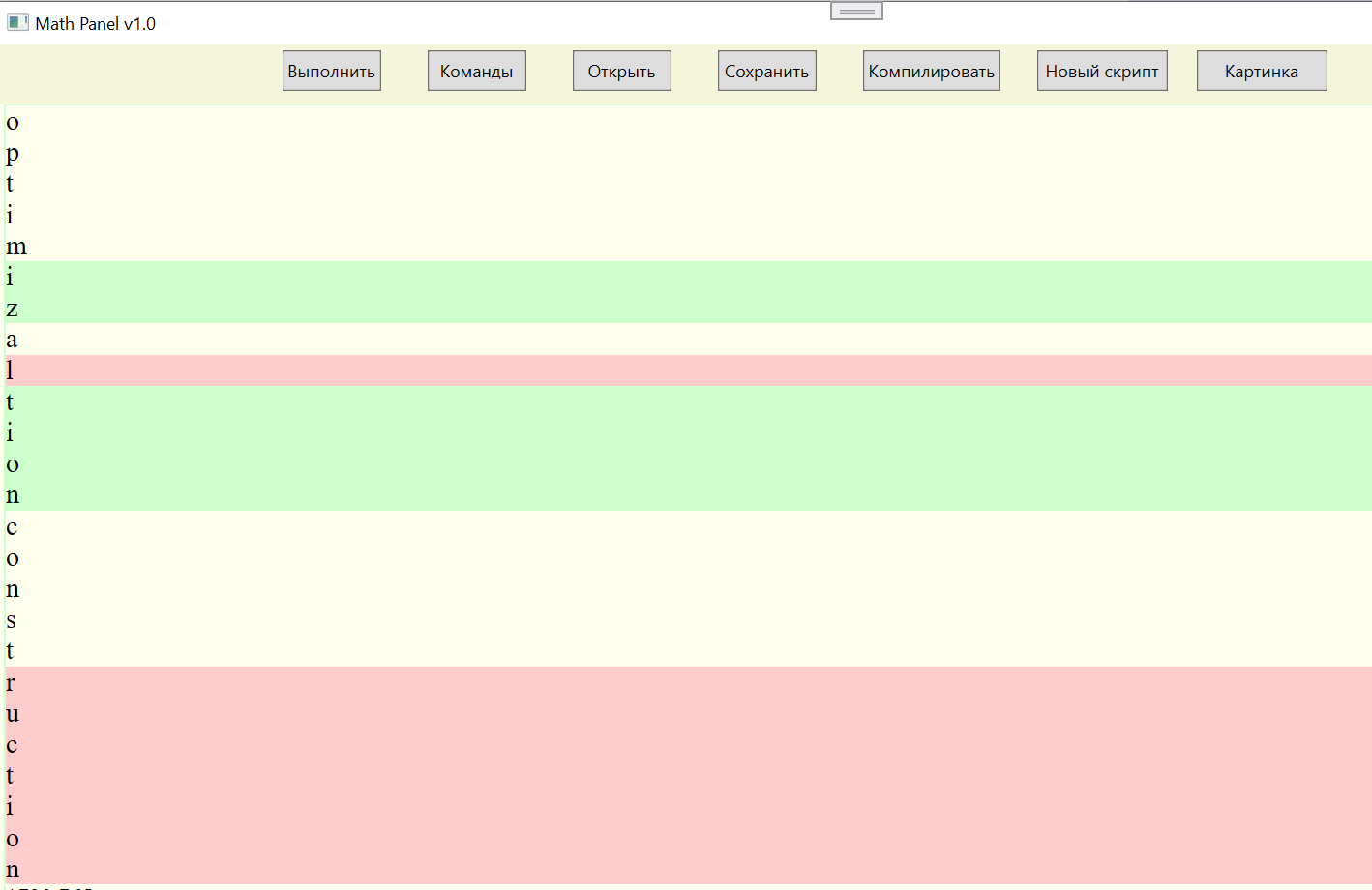


Рис. 4.5. Построчная визуализация результата выравнивания.

## 4.6. Сравнение двух текстовых файлов

Теперь напишем скрипт test48\_compare\_text\_files для сравнения двух текстовых файлов.

//путь к папке с файлами

string sDir = @"C:\c\_devel\data\";

//файлы для сравнения

string[] fnames = { "test.htm", "test2.htm" };

//оздать объект класса Similarica

var solv = new Similarica();

//чтение строк из 1-го файла

var dat0 = System.IO.File.ReadAllLines(sDir + fnames[0], System.Text.Encoding.UTF8);

//чтение строк из 2-го файла

var dat1 = System.IO.File.ReadAllLines(sDir + fnames[1], System.Text.Encoding.UTF8);

//найти веса и наилучший путь

double dScore = solv.Calc(dat0, dat1);

Dynamo.Console("Score=" + dScore);

//генерировать цветное выравнивание в html

var sRs = solv.PrintStrings("font-size:14pt;");

Dynamo.SetHtml(sRs);

В скрипте задаем путь к папке с файлами и названия файлов. Создаем объект класса Similarica, производим чтение строк из 1-го файла и из 2-го файла, находим веса и наилучший путь, генерируем цветное выравнивание в html.

Запустим скрипт, видим результат выравнивания (рис.4.6). Красным строки уникальные в первом файле, зеленым строки уникальные в втором файле, на светлом фоне строки, присутствующие в обоих файлах.

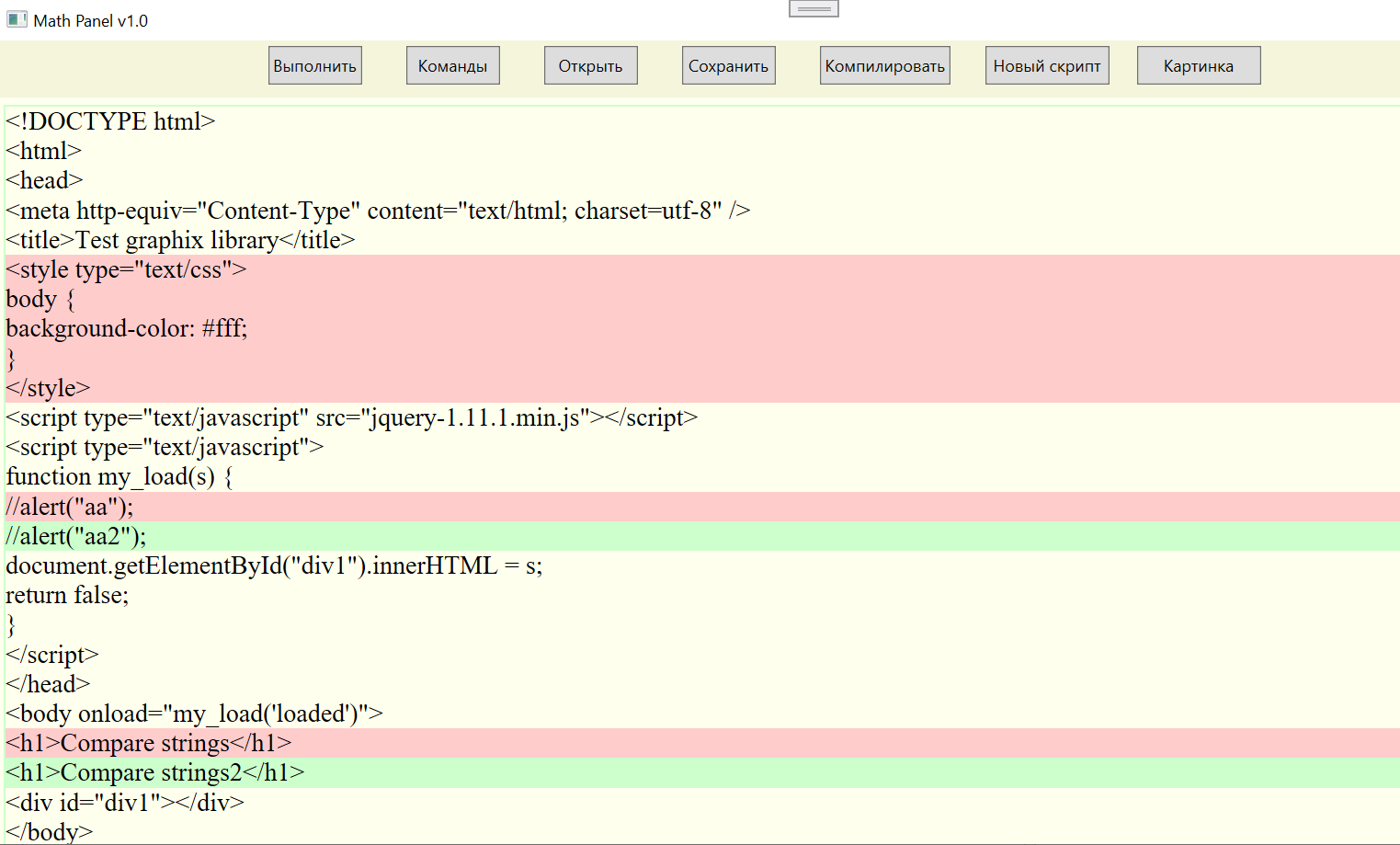


Рис.4.6. Результат выравнивания двух текстовых файлов.

## 4.7 Сравнение массивов объектов.

Рассмотрим сравнение (выравнивание) объектов.

//функция для вычисления разницы между объектами типа double

double my\_compare(object x, object y)

{

//Dynamo.Console("x " + x.ToString().Replace(",", "."));

//Dynamo.Console("y " + y.ToString().Replace(",", "."));

//создать double из первого объекта

double x1 = double.Parse(x.ToString().Replace(",", "."), System.Globalization.CultureInfo.InvariantCulture.NumberFormat);

//создать double из второго объекта

double y1 = double.Parse(y.ToString().Replace(",", "."), System.Globalization.CultureInfo.InvariantCulture.NumberFormat);

//найти абсолютную разницу

double dMax = (Math.Abs(x1) > Math.Abs(y1)) ? Math.Abs(x1) : Math.Abs(y1);

if (dMax == 0) return 0.0; //equal

//найти относительную разницу

var q = Math.Abs(x1 - y1) / dMax;

//если разница мала, вернуть 0, иначе -1

return q <= 0.01 ? 0.0 : -1.0;

}

public void Execute()

{

Dynamo.Console("test48\_compare\_float");

//первый массив из double

object[] f0 = { 1, 2, 3, 4, 7 };

//второй массив из double

object[] f1 = { 1.1, 2.01, 3.1, 4.01, 5.1, 6.1, 7 };

//создать объект класса Similarica

var solv = new MathPanelExt.Similarica();

//найти веса и наилучший путь, третий параметр наш delegate

double dScore = solv.Calc(f0, f1, my\_compare);

//генерировать цветное выравнивание в html

var sRs = solv.PrintStrings("font-size:14pt;");

Dynamo.SetHtml(sRs);

}

В скрипте задаем нашу функцию для вычисления разницы между объектами типа double. Определяем два массива типа double. Их длина может не совпадать. Создаем объект класса Similarica, находим веса и наилучший путь, генерируем цветное выравнивание в html.

Запустим скрипт, видим результат выравнивания (рис.4.7). Красным числа уникальные из первого массива, зеленым числа уникальные из второго массива, на светлом фоне строки, присутствующие в обоих массивах и приблизительно равные.

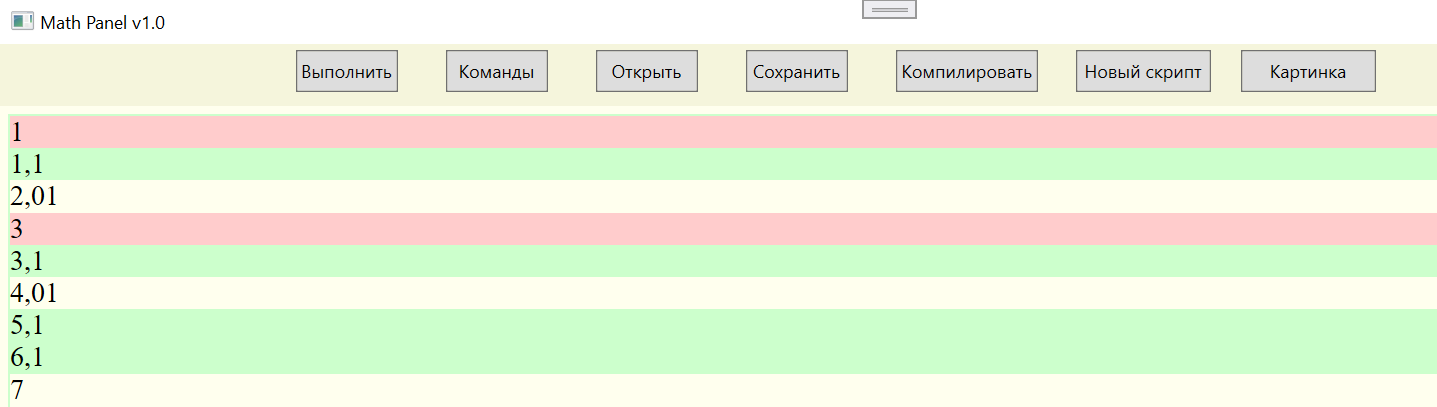


Рис.4.7. Результат выравнивания двух вещественных массивов.

## 4.8. Сравнение файлов изображений

Продолжим сравнение объектов на примере изображений. Для этого в наш класс BitmapSimple добавим новый метод Hash.

/// <summary>

/// генерировать хэш (hash) для объекта BitmapSimple. Хэш функция - некоторая функция,

/// которая может быть использована для соотнесения данных произвольной длины к некоторому фиксированному размеру.

/// </summary>

/// <param name="nx">делить на 'nx' по горизонтали</param>

/// <param name="ny">делить на 'ny' по вертикали</param>

/// <param name="palette">массив цветов для выбора ближайшего</param>

/// <param name="paletteCode">строка для кодировки цветов</param>

/// <param name="bUpdate">если true, преобразовать себя</param>

public string Hash(int nx, int ny, Color[] palette, string paletteCode, bool bUpdate = false)

{

StringBuilder s = new StringBuilder();

int k, n, alpha, red, green, blue, iMin, index;

int d\_x = width / nx; //ширина ячейки

int d\_y = height / ny; //высота ячейки

//разделить объект на маленькие ячейки

//проход по карте

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

{

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

{

//найти средний цвет в каждой ячейке

//сбросить счетчики

alpha = 0;

red = 0;

green = 0;

blue = 0;

n = 0;

//пройти по пикселям ячейки

for (int y = j \* d\_y; y < (j + 1) \* d\_y; y++)

{

for (int x = i \* d\_x; x < (i + 1) \* d\_x; x++)

{

k = y \* width + x;

var cc = System.Drawing.Color.FromArgb(map[k]);

alpha += cc.A;

red += cc.R;

green += cc.G;

blue += cc.B;

n++;

}

}

//найти средний

alpha /= n;

red /= n;

green /= n;

blue /= n;

//найти ближайший цвет в palette

iMin = int.MaxValue;

index = 0;

for (int m = 0; m < palette.Length; m++)

{

var clr = palette[m];

int diff = Math.Abs(clr.A - alpha) + Math.Abs(clr.R - red) +

Math.Abs(clr.G - green) + Math.Abs(clr.B - blue);

if( diff < iMin )

{

iMin = diff;

index = m;

}

}

//добавить к хэшу букву, кодирующую цвет

s.Append(paletteCode.Substring(index, 1));

if (bUpdate)

{ //заполнить ячейки новым ближайшим цветом

var best = palette[index].ToArgb();

for (int y = j \* d\_y; y < (j + 1) \* d\_y; y++)

{

for (int x = i \* d\_x; x < (i + 1) \* d\_x; x++)

{

k = y \* width + x;

map[k] = best;

}

}

}

}

}

return s.ToString();

}

Суть метода в том, что мы делим наше изображение на ряд ячеек, для каждой ячейки вычисляем средний цвет. Для него находим ближайший из нашей палитры и добавляем в хэш букву для этого цвета. Напишем скрипт для сравнения трех изображений.

//путь к папке с изображениями

string sDir = @"C:\c\_devel\images\";

//палитра цветов

Color[] palette = { Color.Red, Color.Orange, Color.Yellow, Color.Green, Color.Blue, Color.DarkBlue,

Color.Magenta, Color.White, Color.Black, Color.Gray };

//кодировка цветов

string paletteCode = "ROYGBDMWKA";

//вычислить первый хэш

var fn = sDir + "test37\_bitmap4\_a.png";

var bm = new BitmapSimple(fn);

var s = bm.Hash(20, 20, palette, paletteCode, true);

Dynamo.Console("s=" + s);

var fn\_hash = (sDir + "test37\_bitmap4\_a\_hash.png");

bm.Save(fn\_hash);

//вычислить второй хэш

var fn\_2 = sDir + "test37\_bitmap5\_с.png";

var bm\_2 = new BitmapSimple(fn\_2);

var s\_2 = bm\_2.Hash(20, 20, palette, paletteCode, true);

Dynamo.Console(" s\_2=" + s\_2);

var fn\_2\_hash = (sDir + "test37\_bitmap5\_с\_hash.png");

bm\_2.Save(fn\_2\_hash);

//вычислить третий хэш

var fn\_3 = sDir + "test37\_bitmap3\_c.png";

var bm\_3 = new BitmapSimple(fn\_3);

var s\_3 = bm\_3.Hash(20, 20, palette, paletteCode, true);

Dynamo.Console(" s\_3=" + s\_3);

var fn\_3\_hash = (sDir + "test37\_bitmap3\_c\_hash.png");

bm\_3.Save(fn\_3\_hash);

//создать объект класса Similarica

var solv = new Similarica();

//сравнить хэши 1 и 2

double dScore = solv.Calc(s, s\_2);

Dynamo.Console(" s, s\_2 score=" + dScore);

//сравнить хэши 1 и 3

dScore = solv.Calc(s, s\_3);

Dynamo.Console(" s, s\_3 score=" + dScore);

//сравнить хэши 3 и 2

dScore = solv.Calc(s\_3, s\_2);

Dynamo.Console(" s\_3, s\_2 score=" + dScore);

Результат сравнения

s, s\_2 score=388

s, s\_3 score=-287

s\_3, s\_2 score=-285

говорит, что первое и второе изображения довольно похожи, а с третьим у них мало общего. Это видно и на рис. 4.8, где совмещены три хэша.

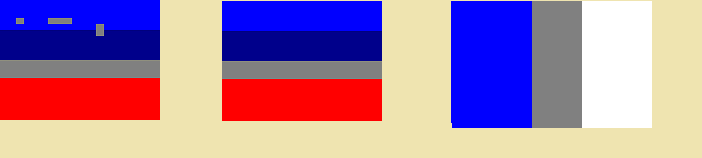


Рис.4.8. Визуальное представление результатов хэширования.