Министерство науки и высшего образования РФ

Федеральное государственное автономное образовательное

учреждения высшего образования

Университет науки и технологий МИСИС

«МИСиС»

Отчет по практической работе №1 на тему:

«Апертурное сжатие данных методом линейной интерполяции»

по дисциплине:

«Методы и средства защиты компьютерной информации»

Направление подготовки: 09.03.03

Семестр 5

|  |  |
| --- | --- |
| Выполнила:    Постнова Дария Сергеевна  БПИ-20-3  (группа)  10.10.2022  (дата)  Подпись: \_\_\_\_\_\_\_\_\_\_\_\_\_\_ | Проверил:  доцент, к. т. н.  Костин Виталий Николаевич  \_\_\_\_\_\_\_\_\_\_  (дата)  \_\_\_\_\_\_\_\_\_\_  (оценка)  Подпись: \_\_\_\_\_\_\_\_\_\_\_\_\_\_ |

Москва 2022

ОГЛАВЛЕНИЕ

[ВВЕДЕНИЕ 3](#_Toc115834883)

[Определение входных и выходных данных 4](#_Toc115834884)

[1.1 Входные данные 4](#_Toc115834885)

[1.2 Выходные данные 4](#_Toc115834886)

[Алгоритм сжатия данных 5](#_Toc115834887)

[2.1 Принцип работы алгоритма сжатия 5](#_Toc115834888)

[2.2 Алгоритм восстановления 6](#_Toc115834889)

[Блок-схемы алгоритма 7](#_Toc115834890)

[3.1 Блок-схема общего алгоритма 7](#_Toc115834891)

[3.2 Функция сжатия 8](#_Toc115834892)

[3.3 Функция восстановления 9](#_Toc115834893)

[Листинг программы 10](#_Toc115834894)

[Тестирование программы 13](#_Toc115834895)

ВВЕДЕНИЕ

Отчёт составлен по результатам выполнения практической работы на тему «Апертурное сжатие данных методом линейной интерполяции» по дисциплине «Методы и средства защиты компьютерной информации».

Целью работы является создание алгоритма сжатия и восстановления данных методом линейной интерполяции, соответствующего поставленным требованиям, а также его реализация на языке программирования C#.

Достижение указанной цели предполагает решение следующих задач:

1. Изучение метода линейной интерполяции;
2. Рассмотрение возможности его применения для данного случая;
3. Составление блок-схем, описывающих работу алгоритма;
4. Проверка корректности работы алгоритма путем его тестирования.

Определение входных и выходных данных

1.1 Входные данные

Формат входных данных, согласно требованиям, установленным преподавателем, определяется следующим образом.

Ввод точек осуществляется с помощью клавиатуры через консоль. В первой строке задается число точек, в последующих строках через пробел вводится индекс точки и значение функции в ней. Далее задается значение апертуры и длина дискреционного шага (длина интервала между индексами). Все входные данные являются значениями целого типа int.

1.2 Выходные данные

Вывод выходных данных будет реализован двумя способами: посредством консоли и записью в текстовый файл. Считается, что такой файл заранее присутствует по указанному пути и у программы есть право на чтение и запись для него.

Выходными данными, согласно требованиям, будут являться два массива:

1. Сжатый массив – измененный исходный массив, полученный посредством алгоритма сжатия с заданной погрешностью (т. е. значением апертуры);
2. Восстановленный массив – ранее сжатый массив, обработанный посредством алгоритма восстановления.

Все выходные данные являются целочисленными, за исключением значений функций в восстановленном массиве (вещественный тип double). При этом значения функции в восстановленном массиве не должны отличаться от исходного набора данных в той же точке более, чем на значение апертуры.

Алгоритм сжатия данных

2.1 Принцип работы алгоритма сжатия

Рассмотрим алгоритм сжатия подробнее.

* Начальный шаг.

На данном этапе происходит добавление в выходной массив начальной точки функции.

* Промежуточный шаг – выполняется N-2 раз.

Здесь рассматривается интерполированный отрезок между двумя точками – условно обозначим их точками с *i* и *j+1* индексами, где *i* – точка начала отрезка, *j+1* – точка конца, *j* – рассматриваемая точка.

Далее определяется, попадет ли *j*-точка в выходной массив. Это произойдет в случае, если значение функции в этой точке исходного массива будет отличаться на значение *y`* апертуры от значения функции на интерполированном отрезке. Значение *y`* находится по следующей формуле:

Значение в этой точке нельзя будет в последствии восстановить с исходной погрешностью. Если же точка попадает в промежуток апертуры, следует пропустить её, оставив точку *i* такой же, а точку *j* установить следующей.

* Конечный шаг.

Добавление в выходной массив конечной точки набора входных данных.

2.2 Алгоритм восстановления

Рассмотрим аналогично предыдущему пункту основные шаги.

* Начальный шаг.

Добавление в выходной массив начальной точки исходного набора (ранее сжатого массива дискретных значений функции).

* Промежуточный шаг – выполняется столько раз, сколько точек в сжатом массиве -1.

Аналогично алгоритму сжатия условно определяется отрезок, где *i*-ая точка – точка начала отрезка, *j*-ая – точка конца, причем *j = i + 1*. Соответственно отрезком будет являться прямая между двумя последовательными точками входного массива. На данном шаге определяется, существует ли ранее сжатые точки между *i*-ой и *j*-ой: производится сравнение длины отрезка по оси Х и значения дискреционного шага.

В случае, если длина отрезка больше длины шага, производится восстановление пропущенных точек путем нахождения для каждой из них значения интерполированного отрезка (см. формулу из п. 2.1).

Иначе предполагается, что между точками *i* и *j* сжатие не производилось.

В обоих случаях необходимо добавить точку *j* в выходной массив.

Блок-схемы алгоритма

3.1 Блок-схема общего алгоритма

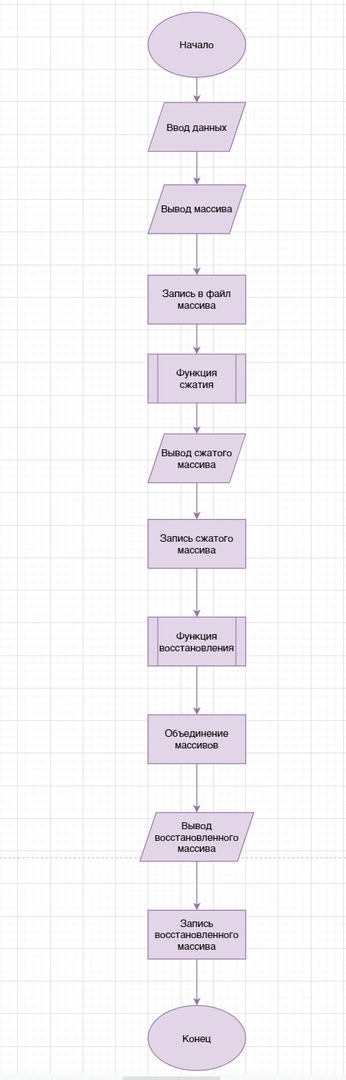


Рисунок 3.1 – Общая блок-схема алгоритма

3.2 Функция сжатия

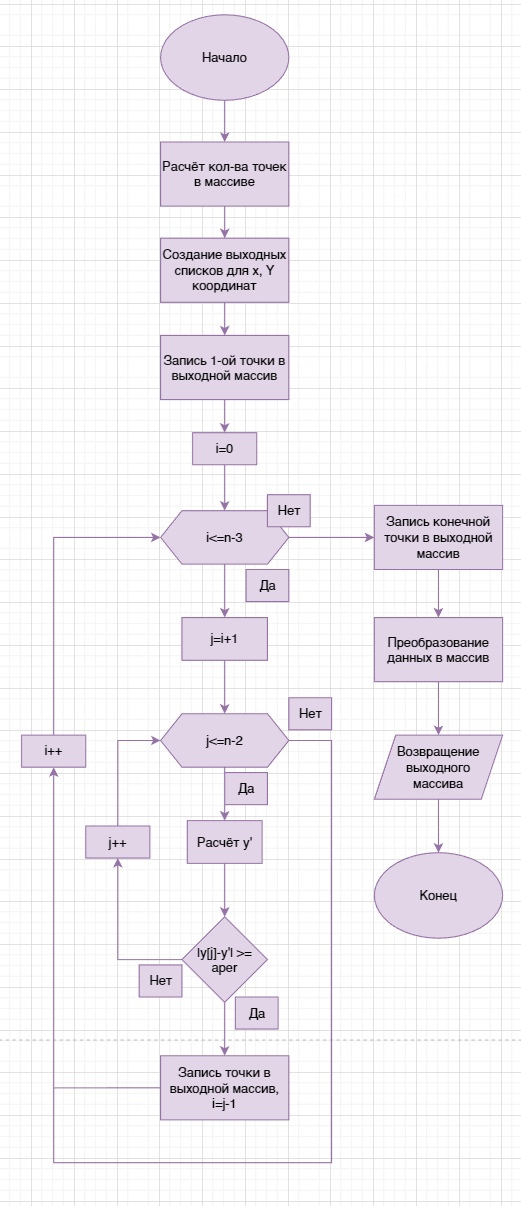


Рисунок 3.2 – Функция сжатия

3.3 Функция восстановления

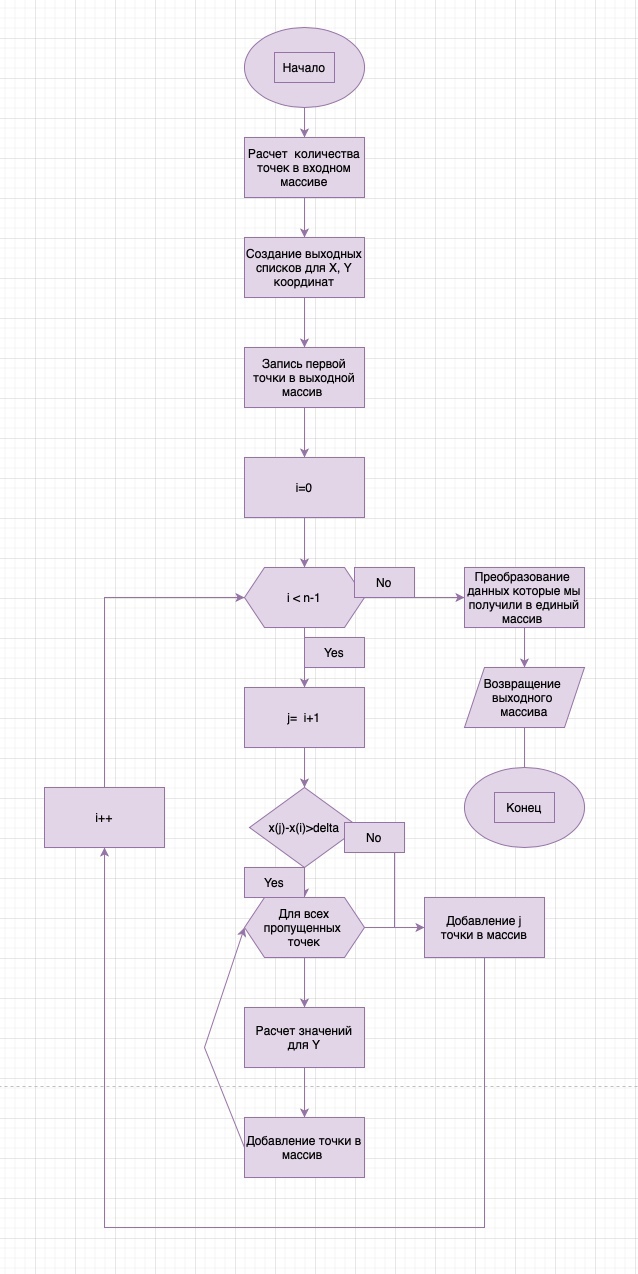


Рисунок 3.3 – Функция восстановления

Листинг программы

Программа разработана на языке программирования C#. Ниже приведен листинг программы.

using System;

using System.Collections.Generic;

using System.IO;

namespace InfBez

{

class Program

{

static void Main(string[] args)

{

int[,] Compression(int[,] input, int aperture)

{

int n = input.GetUpperBound(0) + 1;

List<int> output\_X = new List<int>();

List<int> output\_Y = new List<int>();

output\_X.Add(input[0, 0]);

output\_Y.Add(input[0, 1]);

for (int i = 0; i <= n - 3; i++)

{

for (int j = i + 1; j <= n - 2; j++)

{

double y = (input[j, 0] - input[i, 0]) \* (input[j + 1, 1] - input[i, 1]) / (input[j + 1, 0] - input[i, 0]) + input[i, 1];

if (Math.Abs(input[j, 1] - y) >= aperture)

{

output\_X.Add(input[j, 0]);

output\_Y.Add(input[j, 1]);

i = j - 1;

break;

}

}

}

output\_X.Add(input[n - 1, 0]);

output\_Y.Add(input[n - 1, 1]);

int[,] output = new int[output\_X.Count, 2];

for (int i = 0; i < output\_X.Count; i++)

{

output[i, 0] = output\_X[i];

output[i, 1] = output\_Y[i];

}

return output;

}

double[,] Recover(int[,] input, int delta)

{

int n = input.GetUpperBound(0) + 1;

List<int> output\_X = new List<int>();

List<double> output\_Y = new List<double>();

output\_X.Add(input[0, 0]);

output\_Y.Add(input[0, 1]);

for (int i = 0; i < n - 1; i++)

{

int j = i + 1;

if (input[j, 0] - input[i, 0] > delta)

{

for (int k = delta; k < input[j, 0] - input[i, 0]; k=k+delta)

{

double y = (double)(input[i, 0] + k - input[i, 0]) \* (double)(input[j, 1] - input[i, 1]) / (double)(input[j, 0] - input[i, 0]) + (double)input[i, 1];

output\_X.Add(input[i, 0] + k);

output\_Y.Add(Math.Round(y, 1));

}

}

output\_X.Add(input[j, 0]);

output\_Y.Add(input[j, 1]);

}

double[,] output = new double[output\_X.Count, 2];

for (int i = 0; i < output\_X.Count; i++)

{

output[i, 0] = output\_X[i];

output[i, 1] = output\_Y[i];

}

return output;

}

Console.WriteLine("Введите число точек:");

int N = int.Parse(Console.ReadLine());

int[,] input = new int[N, 2];

Console.WriteLine("Через пробел введите индекс точки и значение функции (целый тип):");

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

{

string[] line = Console.ReadLine().Split(' ');

input[i, 0] = int.Parse(line[0]);

input[i, 1] = int.Parse(line[1]);

}

Console.WriteLine("Введите значение апертуры (целое):");

int e = int.Parse(Console.ReadLine());

Console.WriteLine("Введите длину дискреционного шага:");

int delta = int.Parse(Console.ReadLine());

string path = "C:/Users/Xiaomi/Downloads/Compress.txt";

Console.WriteLine("Исходный массив:");

int n = input.GetUpperBound(0) + 1;

string text = "";

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

{

text += "[" + input[i, 0] + "; " + input[i, 1] + "] ";

Console.Write("[" + input[i, 0] + "; " + input[i, 1] + "] ");

}

using (StreamWriter writer = new StreamWriter(path, true))

{

writer.WriteLineAsync("Исходный массив:");

writer.WriteLineAsync(text);

}

Console.WriteLine();

Console.WriteLine();

Console.WriteLine("Сжатый массив:");

int[,] compressed = Compression(input, e);

n = compressed.GetUpperBound(0) + 1;

string ctext = "";

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

{

ctext += "[" + compressed[i, 0] + "; " + compressed[i, 1] + "] ";

Console.Write("[" + compressed[i, 0] + "; " + compressed[i, 1] + "] ");

}

using (StreamWriter writer = new StreamWriter(path, true))

{

writer.WriteLineAsync("\nСжатый массив:");

writer.WriteLineAsync(ctext);

}

Console.WriteLine();

Console.WriteLine();

Console.WriteLine("Восстановленный массив:");

double[,] recovered = Recover(compressed, delta);

n = recovered.GetUpperBound(0) + 1;

string rtext = "";

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

{

rtext += "[" + recovered[i, 0] + "; " + recovered[i, 1] + "] ";

Console.Write("[" + recovered[i, 0] + "; " + recovered[i, 1] + "] ");

}

using (StreamWriter writer = new StreamWriter(path, true))

{

writer.WriteLineAsync("\nВосстановленный массив:");

writer.WriteLineAsync(rtext);

}

Console.ReadKey();

}

}

}

Тестирование программы

Для тестирования программы выберем следующий набор дискретных значений некоторой функции:

Значение апертуры равно 1. Значение длины шага также равно 1. Запустим программу:

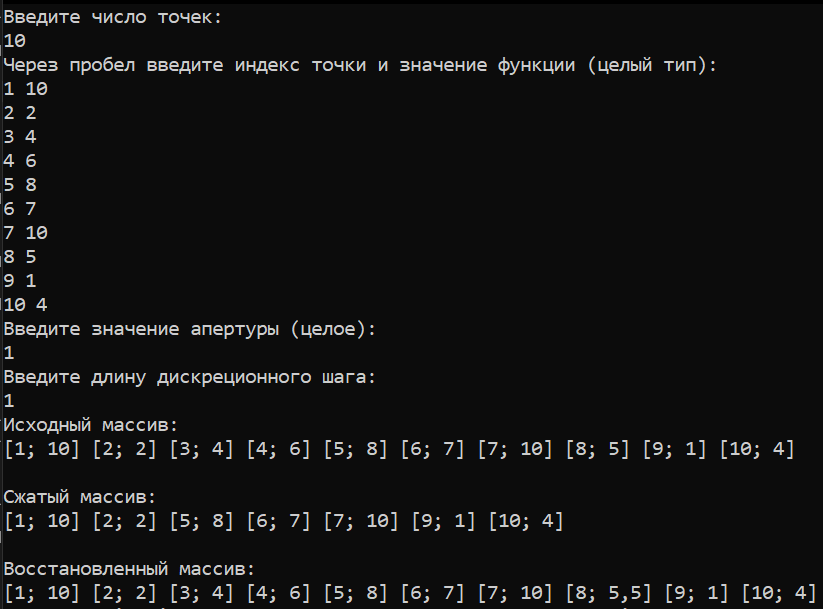


Рисунок 5.1 – Вводим данные

Визуализируем полученный результат.

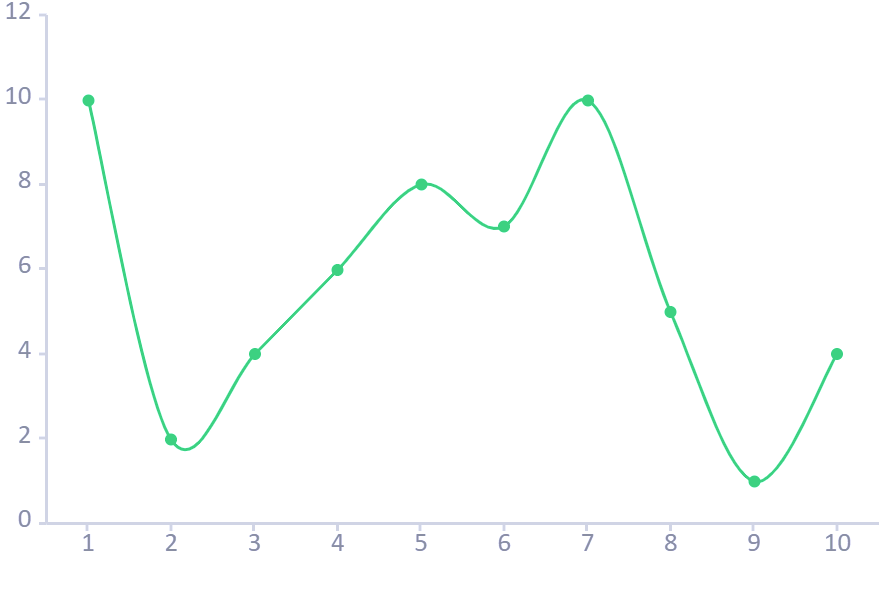


Рисунок 5.2 – Исходный набор данных

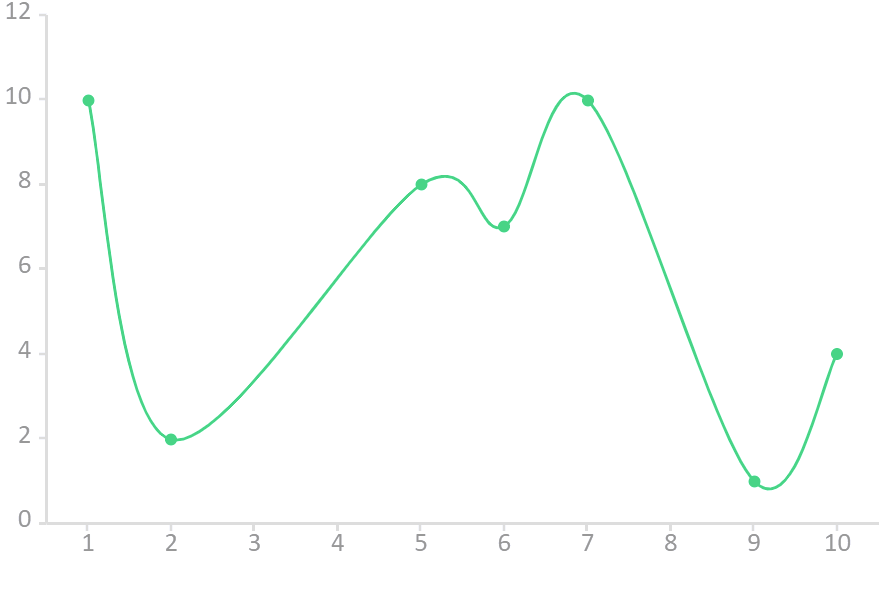


Рисунок 5.3 – Сжатый набор данных

Сжатие было произведено от 10 точек до 7, основные тенденции поведения функции не нарушены.

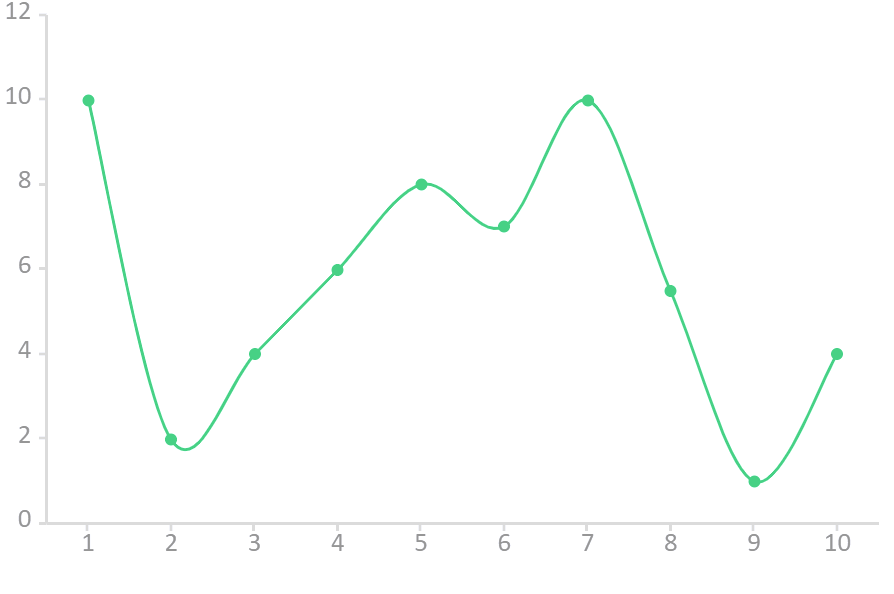


Рисунок 5.4 – Восстановленный набор данных

Значение функции для каждой точки из восстановленного массива не отличается более чем на апертуру (1) от исходного набора, что свидетельствует о корректной работе программы.

Текстовый файл по указанному пути будет иметь следующее содержимое, аналогично тому, что было выведено на консоль:

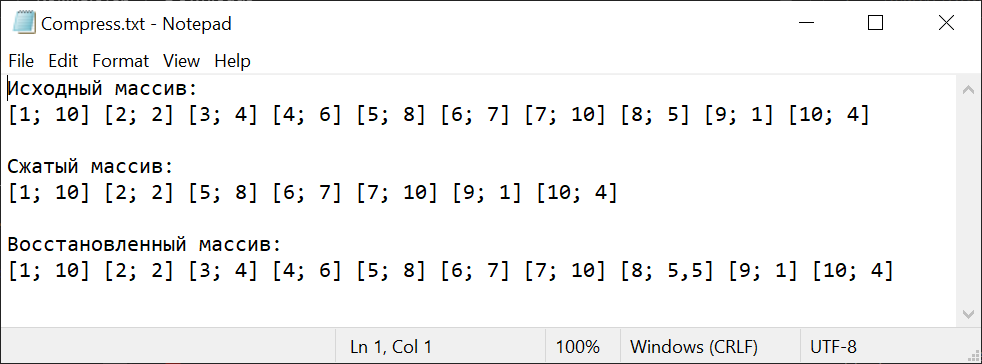


Рисунок 5.5 – Текстовый файл