СОДЕРЖАНИЕ

[ВВЕДЕНИЕ 4](#_Toc151051156)

[1 ПОИСК И ОСНОВНЫЕ ОПЕРАЦИИ С НИМ 5](#_Toc151051157)

[2 ПЕРЕЧЕНЬ ТРЕБОВАНИЙ К ПРИЛОЖЕНИЮ 8](#_Toc151051158)

[3 ОСНОВНОЙ ИНТЕРФЕЙС ПРИЛОЛОЖЕНИЯ 10](#_Toc151051159)

[4 РУКОВОДСТВО ПОЛЬЗОВАНИЯ 16](#_Toc151051160)

[5 ЛИСТИНГ ПРОГРАММЫ 19](#_Toc151051161)

[ЗАКЛЮЧЕНИЕ 24](#_Toc151051162)

[СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ](#_Toc151051163)

# ВВЕДЕНИЕ

Целью проекта является реализация алгоритма последовательного поиска с применением графической иллюстрации. Это позволит пользователям не только освоить алгоритм, но и визуально отслеживать каждый его шаг в наборе данных.

В рамках данной курсовой работы рассматривается создание приложения, нацеленного на осуществление алгоритма последовательного поиска, с использованием языка программирования C# и графического интерфейса Windows Forms. Этот проект не только предоставляет возможность практического применения навыков программирования на C#, но и обеспечивает визуализацию работы алгоритма через графическое представление в среде Windows Forms.

В современной среде разработки программ с широким использованием графических интерфейсов, создание подобного проекта становится ключевым элементом обучения. Этот опыт не только способствует применению теоретических знаний в области алгоритмизации, но и активно развивает навыки создания интуитивно понятных программных продуктов. В данной курсовой работе подробно рассмотриваются технические аспекты реализации приложения на языке C# с использованием Windows Forms и графической иллюстрации алгоритма последовательного поиска.

# ПОИСК И ОСНОВНЫЕ ОПЕРАЦИИ С НИМ

Алгоритмы поиска представляют собой важную часть вычислительных наук, их применение охватывает множество областей, начиная от базового поиска в массивах и заканчивая более сложными системами информационного поиска. В рамках курсовой работы, посвященной разработке приложения с использованием алгоритма последовательного поиска, следует рассмотреть классификацию алгоритмов поиска:

1. Последовательный поиск:

Алгоритм, при котором элементы последовательности проверяются поочередно до тех пор, пока не будет найден искомый элемент или не будет пройдена вся последовательность. Используется для небольших объемов данных, когда последовательность не упорядочена или не предоставляет возможности для применения более эффективных алгоритмов.

1. Бинарный поиск:

Работает с отсортированным массивом, разделяя его пополам на каждом шаге и сравнивая искомый элемент с серединой. Процесс повторяется до тех пор, пока элемент не будет найден. Эффективен при работе с упорядоченными данными, позволяет уменьшить количество сравнений.

1. Хэширование:

Использует хэш-функции для быстрого поиска элемента в коллекции данных. Результат хэш-функции используется в качестве индекса для доступа к элементу. Подходит для поиска в больших объемах данных, где требуется высокая производительность.

1. Интерполяционный поиск:

Предполагает равномерное распределение элементов в отсортированной последовательности и использует интерполяцию для приближенного определения местоположения искомого элемента. Эффективен при равномерном распределении данных.

В приложении, созданном в среде Windows Forms на языке C#, используется последовательный поиск для обеспечения простоты и понятности визуализации. Подходит для небольших объемов данных и обеспечивает интуитивно понятный процесс поиска, что является ключевым элементом обучения основам алгоритмизации и программирования.

Алгоритм последовательного поиска является одним из базовых методов поиска элемента в наборе данных. Он проходит последовательно через каждый элемент, начиная с первого, и сравнивает его с целевым значением. Если элемент найден, возвращается его позиция (индекс), в противном случае - программа сообщает об отсутствии элемента.

**Принцип работы алгоритма**

1. Исходные данные:

- Набор данных, чаще всего массив или список.

- Искомый элемент.

2. Шаги алгоритма:

- Начиная с первого элемента, каждый элемент последовательно сравнивается с искомым.

- Если элемент найден, возвращается его индекс.

- Если весь набор данных просмотрен и искомый элемент не обнаружен, возвращается информация об отсутствии элемента.

3. Сложность алгоритма:

- В наихудшем случае алгоритм может потребовать проверки всех элементов, что ведет к временной сложности O(n), где n - количество элементов в коллекции данных.

**Особенности алгоритма**

Простота реализации делает его хорошим выбором для небольших коллекций данных. При большом объеме данных алгоритм может потребовать значительного времени из-за линейного поиска каждого элемента.

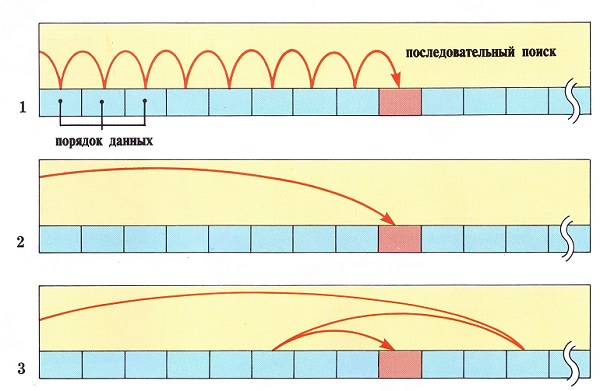
Итак, совмещение теории алгоритма последовательного поиска с графической иллюстрацией позволит не только понять принципы работы алгоритма, но и визуально продемонстрировать его шаги, облегчая процесс усвоения информации о методе поиска.

Рисунок 1.1 – Последовательный поиск

# 2 ПЕРЕЧЕНЬ ТРЕБОВАНИЙ К ПРИЛОЖЕНИЮ

1. Инициализация данных:

Загрузка набора данных, который будет представлен в графическом интерфейсе программы.

1. Отображение данных

Использование графических компонентов для отображения входных данных.

1. Интеграция блок-схемы:

Вставка шагов алгоритма последовательного поиска, как описано в теоретической части, в программный код.

1. Анимация поиска:

Создание динамической анимации, которая визуально представляет процесс последовательного поиска согласно блок-схеме.

1. Отображение результатов:

Подсветка найденного элемента или вывод сообщения об отсутствии элемента в случае завершения поиска.

На рисунке 2.1 изображена блок-схема последовательного поиска

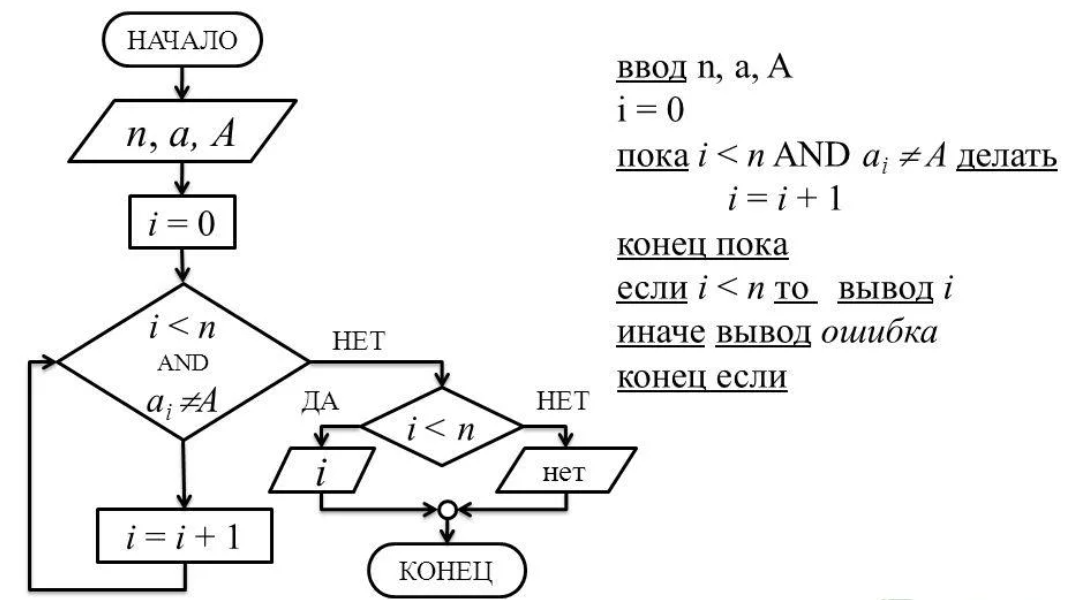
****

Рисунок 2.1 – Схема работы последовательного поиска

Из схемы видно, что если искомый элемент найден, то поиск может закончиться до окончания просмотра всего набора данных. Если же элемент не обнаружен, то поиск закончится только после просмотра всего набора данных.

Результатом работы программы должна быть интуитивно понятная графическая среда, в которой пользователи смогут не только увидеть каждый шаг алгоритма, но и взаимодействовать с программой, улучшая свои навыки в области алгоритмизации и программирования.

# 3 ОСНОВНОЙ ИНТЕРФЕЙС ПРИЛОЛОЖЕНИЯ

Процесс разработки начался с тщательного анализа требований, поставленных перед программой. Этот этап включал в себя изучение особенностей блок-схемы алгоритма последовательного поиска и определение ключевых характеристик, которые должны были быть реализованы в программе.

Следующим этапом стало проектирование графического интерфейса, где был разработан дизайн с учетом интуитивности и привлекательности. В этот момент были также выбраны технологии и язык программирования, которые наилучшим образом соответствовали поставленным задачам. В результате анализа требований и особенностей задачи, было принято решение использовать язык программирования C# с использованием платформы Windows Forms для создания графического интерфейса приложения.

На рисунке 3.1 изображен дизайн программы:

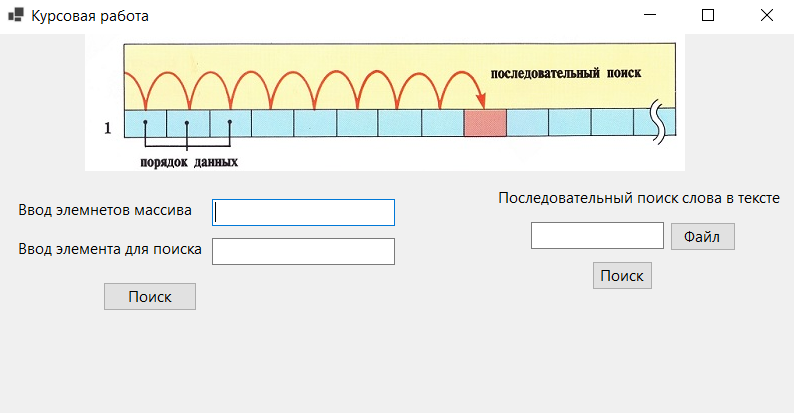


Рисунок 3.1 – Окно приложения при запуске

Далее шел этап написания программного кода, в ходе которого интегрировалась блок-схема алгоритма. Были реализованы функции отображения данных, динамической анимации поиска и вывода результатов.

**Описание работы функций**

Функция InitializePictureBoxes():

private void InitializePictureBoxes()

{

int elementWidth = 40;

int elementHeight = 40;

arrayElementPictureBoxes = new PictureBox[arrPic];

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

{

arrayElementPictureBoxes[i] = new PictureBox

{

Location = new Point(50 + i \* (elementWidth + 10), 300),

Size = new Size(elementWidth, elementHeight),

BackColor = Color.White,

BorderStyle = BorderStyle.FixedSingle,

};

Label label1 = new Label()

{

Text = array[i].ToString(),

TextAlign = ContentAlignment.MiddleCenter,

Dock = DockStyle.Fill,

};

arrayElementPictureBoxes[i].Controls.Add(label1);

this.Controls.Add(arrayElementPictureBoxes[i]);

}

}

Эта функция, вероятно, используется для инициализации и настройки массива объектов типа PictureBox в вашем приложении. Разберем код по шагам:

1. int elementWidth = 40; и int elementHeight = 40; определяют ширину и высоту каждого элемента PictureBox в массиве.
2. arrayElementPictureBoxes = new PictureBox[arrPic]; создает массив объектов PictureBox с размером arrPic.
3. Затем выполняется цикл for, который инициализирует каждый элемент массива arrayElementPictureBoxes:

* arrayElementPictureBoxes[i] = new PictureBox { ... }; создает новый объект PictureBox и настраивает его параметры, такие как положение (Location), размер (Size), цвет фона (BackColor), и стиль границы (BorderStyle).
* Label label1 = new Label() { ... }; создает новый объект Label и настраивает его параметры, такие как текст (Text), выравнивание текста (TextAlign), и док-стиль (Dock).
* arrayElementPictureBoxes[i].Controls.Add(label1); добавляет созданный Label в Controls PictureBox, что позволяет разместить текст поверх изображения в PictureBox.
* this.Controls.Add(arrayElementPictureBoxes[i]); добавляет каждый созданный PictureBox в Controls текущей формы.

Функция VisualizeSearchh():

private async void VisualizeSearchh()

{

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

{

if (array[i] == target)

{

arrayElementPictureBoxes[i].BackColor = Color.Green;

((Label)arrayElementPictureBoxes[i].Controls[0]).Text = array[i].ToString();

await Task.Delay(300);

MessageBox.Show($"Элемент {target} найден в позиции {i}.");

return;

}

else

{

arrayElementPictureBoxes[i].BackColor = Color.Red;

((Label)arrayElementPictureBoxes[i].Controls[0]).Text = array[i].ToString();

await Task.Delay(300);

}

}

MessageBox.Show($"Элемент {target} не найден.");

}

Функция VisualizeSearchh использует объекты PictureBox для отображения каждого элемента массива и изменяет их цвет в зависимости от совпадения с целевым элементом.

1. for (int i = 0; i < array.Length; i++): Цикл перебирает каждый элемент массива.
2. if (array[i] == target) { ... } else { ... }: Проверяет, совпадает ли текущий элемент массива с целевым.
3. В блоке if:

* arrayElementPictureBoxes[i].BackColor = Color.Green;: Устанавливает цвет фона PictureBox в зеленый для отображения совпадения.
* ((Label)arrayElementPictureBoxes[i].Controls[0]).Text = array[i].ToString();: Обновляет текст Label в PictureBox текущим значением элемента.
* await Task.Delay(300);: Добавляет задержку для визуального эффекта.
* MessageBox.Show($"Элемент {target} найден в позиции {i}.");: Выводит сообщение о нахождении элемента и завершает функцию.

1. В блоке else:

* arrayElementPictureBoxes[i].BackColor = Color.Red;: Устанавливает цвет фона PictureBox в красный для отображения несовпадения.
* ((Label)arrayElementPictureBoxes[i].Controls[0]).Text = array[i].ToString();: Обновляет текст Label в PictureBox текущим значением элемента.
* await Task.Delay(300);: Добавляет задержку для визуального эффекта.

1. После цикла:

* MessageBox.Show($"Элемент {target} не найден.");: Выводит сообщение о том, что элемент не был найден.

На рисунке 3.2 представлена работа функции VisualizeSearchh():

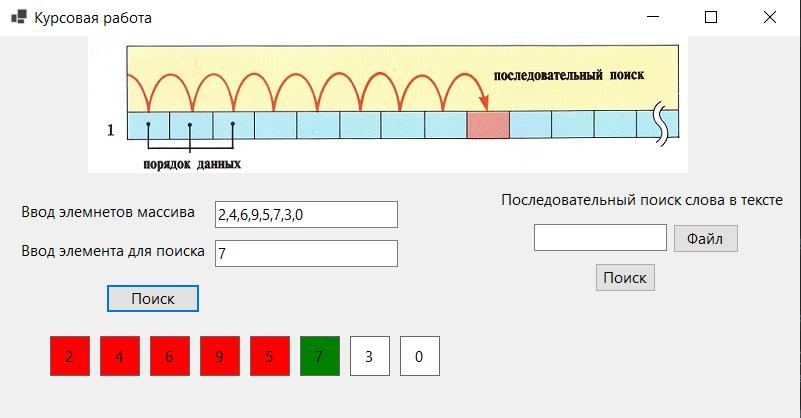


Рисунок 3.2 – Работа функции VisualizeSearchh()

Функция button2\_Click(object sender, EventArgs e):

private void button2\_Click(object sender, EventArgs e)

{

string wordToFind = textBox3.Text.ToString();

if (string.IsNullOrEmpty(textBox3.Text))

{

label3.Text = "Введите слово для поиска";

return;

}

int pos = 0;

bool wordFound = false;

while ((pos = line.IndexOf(wordToFind, pos)) != -1)

{

label3.Text = $"Слово найдено на позиции {pos}";

wordFound = true; // слово было найдено

pos += wordToFind.Length;

}

if (!wordFound)

{

label3.Text = "Слово не найдено";

}

}

Эта функция осуществляет поиск введенного слова в строке текстового поля и обновляет текст метки (`label3`) с указанием позиции первого вхождения слова. Если слово не найдено, выводится сообщение "Слово не найдено". Раннее описана функция для считывания теста из файла(button3\_Click(object sender, EventArgs e)).

Тестирование программы предоставило возможность проверить ее работоспособность на различных наборах данных и сценариях поиска. Выявленные ошибки и недочеты были внимательно исправлены.

Оптимизация кода и улучшения в графическом интерфейсе были проведены с целью повышения производительности и удобства использования программы. Завершающим этапом процесса стала подготовка документации, обеспечивающей полное понимание структуры и функциональности программы для будущих пользователей и разработчиков.

# 4 РУКОВОДСТВО ПОЛЬЗОВАНИЯ

При запуске приложения мы видим окно, изображенное на картинке 5.1:

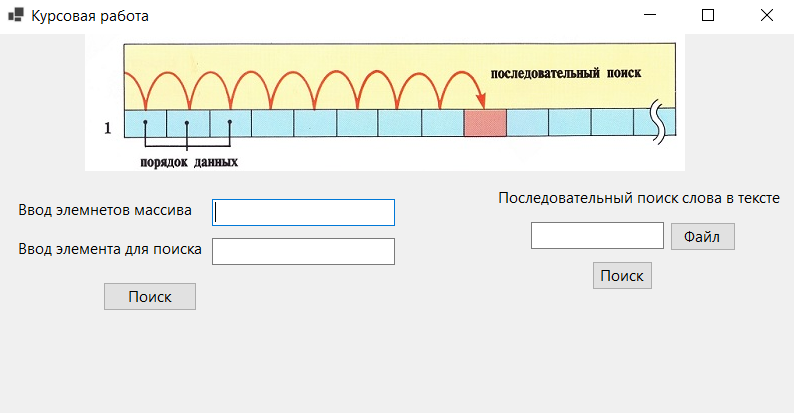


Рисунок 4.1 – Окно приложения при запуске

Необходимо ввести массив в поле «Ввод элементов массива» и ввести элемент, который необходимо ввести в «Ввод элемента для поиска»

Далее нажать кнопку поиск, после чего появится визуальная иллюстрация процесса поиска.

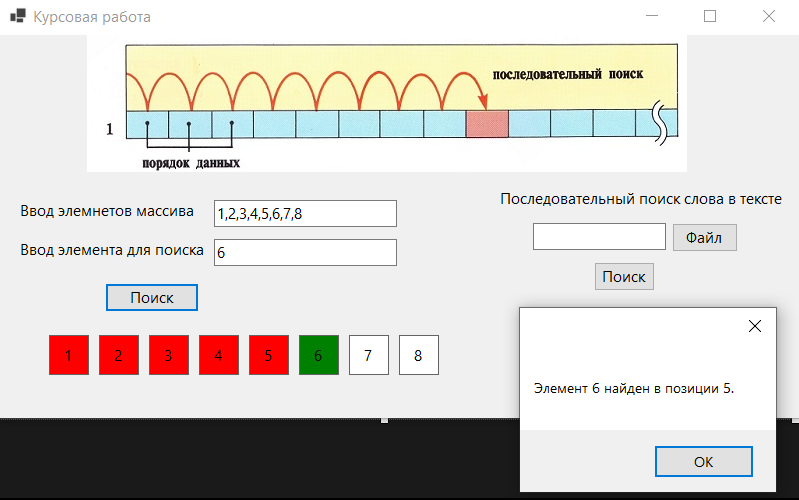


Рисунок 4.2 – Визуальная иллюстрация поиска

Как видно на изображении, появилось всплывающее окно указывающее позицию элемента в массиве.

Если поиск необходимо проводить в текстовом файле необходимо нажать кнопку «Файл» и в всплывающем окне выбрать файл с текстом.

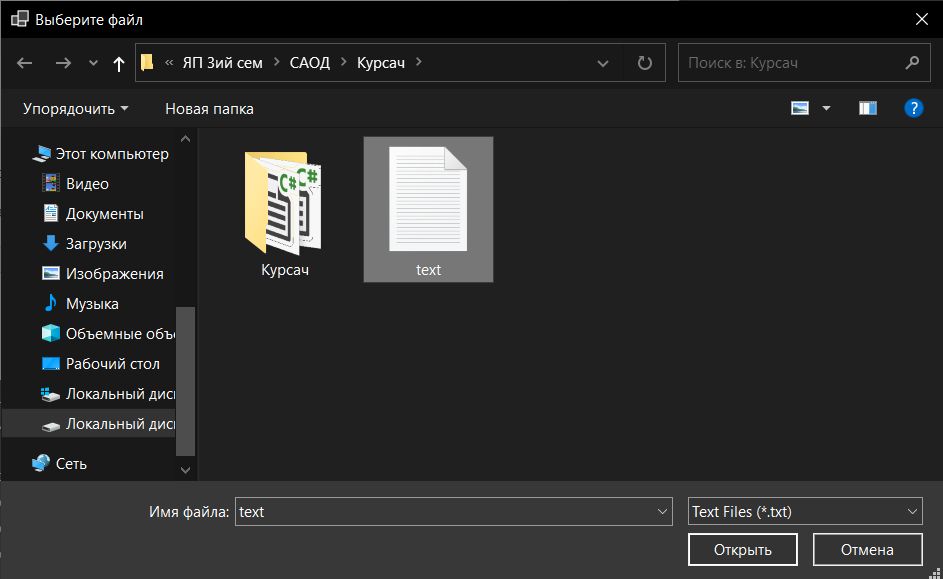


Рисунок 4.3 – Выбор файла

Для демонстрации функциональности программы выбран файл с названием "text.txt". Внутри данного файла содержится набор данных, который будет использоваться в качестве входных данных для поиска при выполнении программы.

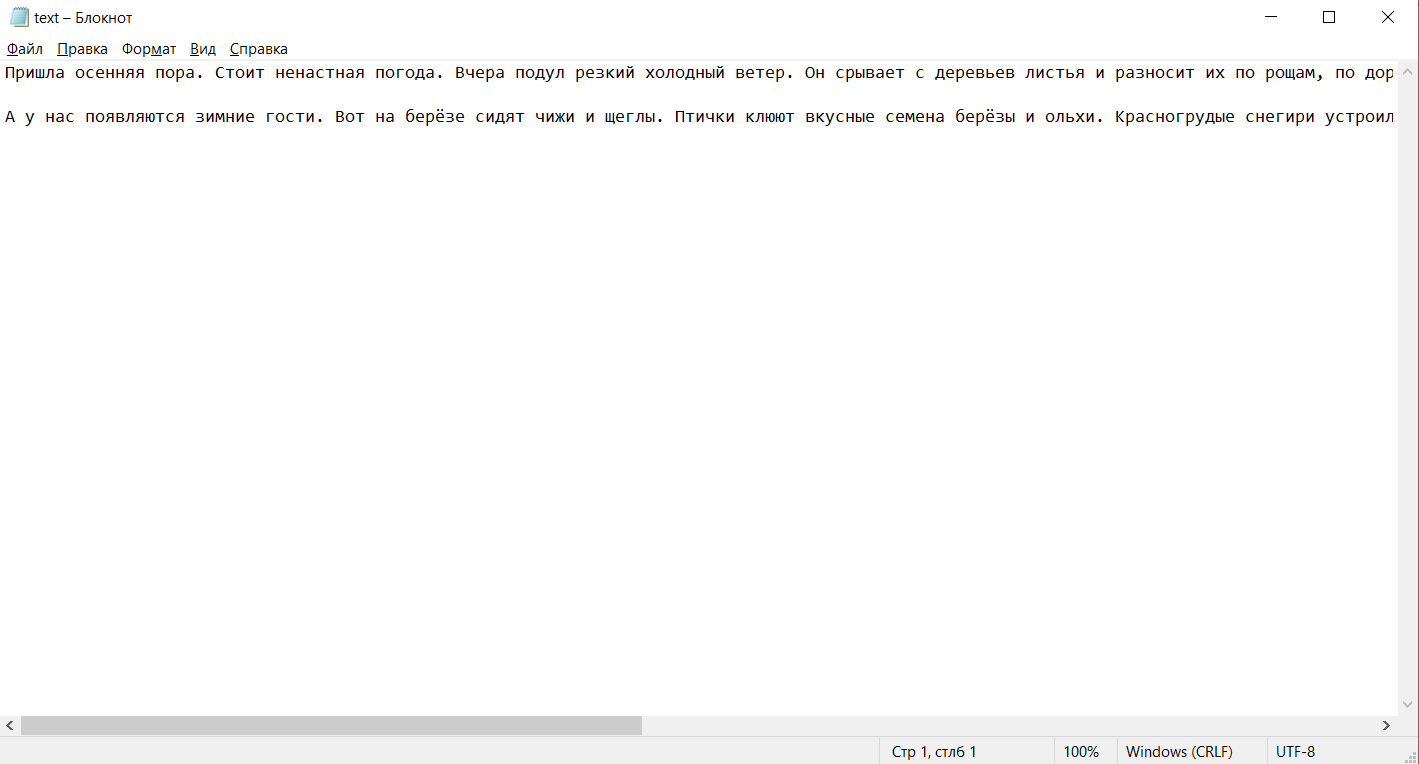


Рисунок 4.4 – Содержание файла «text.txt»

Результаты выполнения алгоритма последовательного поиска в тексте представлены на рисунке 5.5.

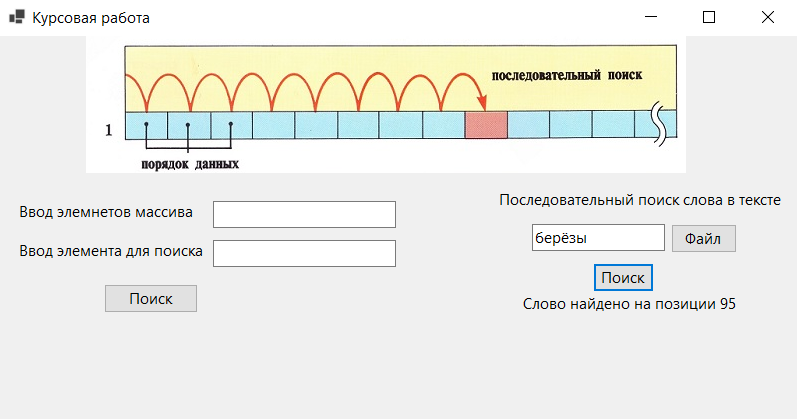


Рисунок 4.5 – Результат работы поиска

# 5 ЛИСТИНГ ПРОГРАММЫ

Листинг файла Form1.cs:

using System.Windows.Forms;

using static System.Net.Mime.MediaTypeNames;

using static System.Windows.Forms.VisualStyles.VisualStyleElement;

namespace Курсач

{

public partial class Form1 : Form

{

private int[] array;

private int target;

private int currentIndex = -1;

private PictureBox[] arrayElementPictureBoxes;

string line;

public Form1()

{

InitializeComponent();

}

int arrPic = 0;

private void InitializePictureBoxes()

{

int elementWidth = 40;

int elementHeight = 40;

arrayElementPictureBoxes = new PictureBox[arrPic];

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

{

arrayElementPictureBoxes[i] = new PictureBox

{

Location = new Point(50 + i \* (elementWidth + 10), 300),

Size = new Size(elementWidth, elementHeight),

BackColor = Color.White,

BorderStyle = BorderStyle.FixedSingle,

};

Label label1 = new Label()

{

Text = array[i].ToString(),

TextAlign = ContentAlignment.MiddleCenter,

Dock = DockStyle.Fill,

};

arrayElementPictureBoxes[i].Controls.Add(label1);

this.Controls.Add(arrayElementPictureBoxes[i]);

}

}

private void button1\_Click(object sender, EventArgs e)

{

if (ParseInput())

{

currentIndex = 0;

InitializePictureBoxes();

UpdatePictureBoxes();

VisualizeSearchh();

}

}

private bool ParseInput()

{

string[] inputArray = textBox1.Text.Split(',');

arrPic= inputArray.Length;

if (inputArray.Length == 0)

{

MessageBox.Show("Введите числа через запятую.");

return false;

}

array = new int[inputArray.Length];

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

{

if (!int.TryParse(inputArray[i], out array[i]))

{

MessageBox.Show("Ошибка ввода чисел. Пожалуйста, введите числа через запятую.");

return false;

}

}

if (int.TryParse(textBox2.Text, out var parsedTarget))

{

target = parsedTarget;

return true;

}

else

{

MessageBox.Show("Ошибка ввода данных для поиска. Пожалуйста, введите корректное число.");

return false;

}

}

private void VisualizeSearch()

{

if (currentIndex >= 0 && currentIndex < array.Length)

{

if (array[currentIndex] == target)

{

arrayElementPictureBoxes[currentIndex].BackColor = Color.Red;

MessageBox.Show($"Элемент {target} найден в позиции {currentIndex}.");

}

else

{

arrayElementPictureBoxes[currentIndex].BackColor = Color.White;

currentIndex++;

VisualizeSearch();

}

}

else

{

MessageBox.Show($"Элемент {target} не найден.");

}

}

private async void VisualizeSearchh()

{

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

{

if (array[i] == target)

{

arrayElementPictureBoxes[i].BackColor = Color.Green;

((Label)arrayElementPictureBoxes[i].Controls[0]).Text = array[i].ToString();

await Task.Delay(300);

MessageBox.Show($"Элемент {target} найден в позиции {i}.");

return;

}

else

{

arrayElementPictureBoxes[i].BackColor = Color.Red;

((Label)arrayElementPictureBoxes[i].Controls[0]).Text = array[i].ToString();

await Task.Delay(300);

}

}

MessageBox.Show($"Элемент {target} не найден.");

}

private void UpdatePictureBoxes()

{

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

{

arrayElementPictureBoxes[i].BackColor = Color.White;

}

}

private void button3\_Click(object sender, EventArgs e)

{

openFileDialog1.Filter = "Text Files (\*.txt)|\*.txt|All Files (\*.\*)|\*.\*";

openFileDialog1.InitialDirectory = "D:\\ЯП 3ий сем\\САОД\\Курсач";

openFileDialog1.Title = "Выберите файл";

if (openFileDialog1.ShowDialog() == DialogResult.OK)

{

string selectedFilePath = openFileDialog1.FileName;

//"D:\\ЯП 3ий сем\\Разработка приложений в визуальных средах\\лаб 6\\file.txt"

using (System.IO.StreamReader sr = new System.IO.StreamReader(selectedFilePath))

{

while (!sr.EndOfStream)

{

line = sr.ReadLine();

}

}

}

}

private void button2\_Click(object sender, EventArgs e)

{

string wordToFind = textBox3.Text.ToString();

if (string.IsNullOrEmpty(textBox3.Text))

{

label3.Text = "Введите слово для поиска";

return;

}

int pos = 0;

bool wordFound = false;

while ((pos = line.IndexOf(wordToFind, pos)) != -1)

{

label3.Text = $"Слово найдено на позиции {pos}";

wordFound = true; // Устанавливаем флаг, что слово было найдено

pos += wordToFind.Length;

}

if (!wordFound)

{

label3.Text = "Слово не найдено";

}

}

private void textBox3\_TextChanged(object sender, EventArgs e)

{

}

}

}

# ЗАКЛЮЧЕНИЕ

В заключение процесса разработки приложения, реализующего алгоритм последовательного поиска с графической иллюстрацией на платформе Windows Forms с использованием языка программирования C#, можно подчеркнуть успешную реализацию поставленных задач и достижение поставленных целей.

Выбор языка программирования C# обоснован его преимуществами в части читаемости кода, высокого уровня абстракции, а также богатства инструментов и библиотек, что содействует эффективной разработке программного продукта.

Использование Windows Forms в качестве инструмента для создания графического интерфейса обеспечило простоту в разработке и интеграцию с языком C#. Это решение позволило легко визуализировать алгоритм последовательного поиска, обеспечивая пользователей наглядным и понятным способом.

Программа успешно прошла тестирование на различных сценариях и наборах данных, что подтверждает её функциональность и стабильность работы. Выявленные в процессе тестирования ошибки были успешно устранены

Оптимизация кода и улучшения в графическом интерфейсе были внедрены для повышения производительности и удобства использования программы.

В итоге, разработанное приложение не только соответствует поставленным задачам и требованиям, но и предоставляет пользователю интуитивно понятный и визуально привлекательный инструмент для изучения алгоритма последовательного поиска.

# СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Официальная документация Microsoft по C# и Windows Forms [Электронный ресурс]. URL:  [https://learn.microsoft.com/ru-ru/visualstudio/ide/create-csharp-winform-visual-studio?view=vs-2022](http://yann.lecun.com/exdb/mnist/) (дата обращения: 15.10.2023).
2. Файловый архив студентов StudFile [Электронный ресурс]. URL: [https://studfile.net/preview/5553821/page:3/](https://www.tensorflow.org/guide/keras/train_and_evaluate#training_with_datasets) (дата обращения: 27.10.2023).
3. Stack Overflow [Электронный ресурс]. URL:  [<https://stackoverflow.com/questions/tagged/winforms>](http://yann.lecun.com/exdb/mnist/) (дата обращения: 15.10.2023).