**Министерство науки и высшего образования Российской Федерации**

**ФГАОУ ВО «Севастопольский государственный университет»**

Институт информационных технологий

Кафедра «Информационные системы»

**Отчет по лабораторной работе №9**

по дисциплине «Технологии обработки информации»

Выполнил: студент группы

ИС/б-21-2-о

Мельничук В.В.

Принял:

г. Севастополь

**Лабораторная работа №9**

Тема: Реализация алгоритмов поиска по тексту

Цель работы: научиться реализовывать на выбранном языке программирования алгоритмы поиска по тексту: прямой поиск; алгоритм Кнута, Морриса и Пратта; алгоритм Бойера-Мура.

Ход работы:

1. Алгоритм прямого поиска

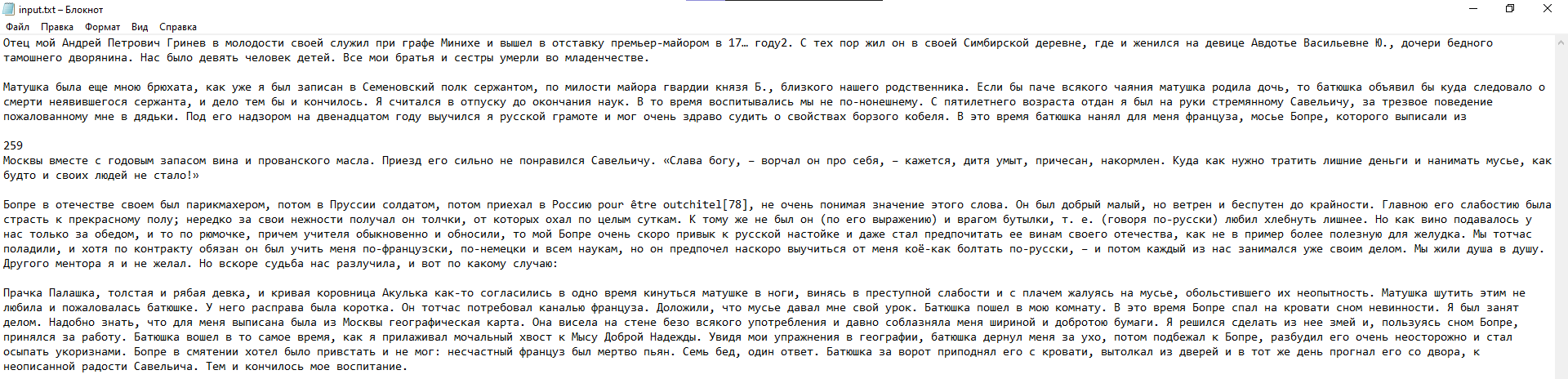


Рисунок 1 – Входные данные

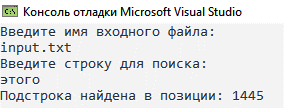


Рисунок 2 – Результат работы программы

Листинг:

using System;

using System.IO;

class Program

{

static void Main()

{

Console.WriteLine("Введите имя входного файла: ");

string fileName = Console.ReadLine();

Console.WriteLine("Введите строку для поиска: ");

string searchString = Console.ReadLine();

string fileContent;

try

{

fileContent = File.ReadAllText(fileName);

}

catch (Exception ex)

{

Console.WriteLine("Ошибка при чтении файла: " + ex.Message);

return;

}

int index = DirectSearch(fileContent, searchString);

if (index != -1)

{

Console.WriteLine("Подстрока найдена в позиции: " + index);

}

else

{

Console.WriteLine("Подстрока не найдена");

}

}

static int DirectSearch(string text, string pattern)

{

int n = text.Length;

int m = pattern.Length;

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

{

int j;

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

{

if (text[i + j] != pattern[j])

{

break;

}

}

if (j == m)

{

return i;

}

}

return -1;

}

}

1. Алгоритм Кнута

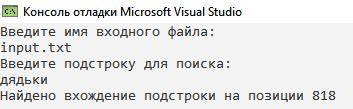


Рисунок 3 – Результат работы программы

Листинг:

using System;

using System.IO;

class KnuthMorrisPratt

{

static int[] ComputeLPSArray(string pattern)

{

int patternLength = pattern.Length;

int[] lps = new int[patternLength];

int len = 0;

int i = 1;

while (i < patternLength)

{

if (pattern[i] == pattern[len])

{

len++;

lps[i] = len;

i++;

}

else

{

if (len != 0)

{

len = lps[len - 1];

}

else

{

lps[i] = 0;

i++;

}

}

}

return lps;

}

static void KMPSearch(string pattern, string text)

{

int patternLength = pattern.Length;

int textLength = text.Length;

int[] lps = ComputeLPSArray(pattern);

int i = 0;

int j = 0;

while (i < textLength)

{

if (pattern[j] == text[i])

{

j++;

i++;

}

if (j == patternLength)

{

Console.WriteLine("Найдено вхождение подстроки на позиции " + (i - j));

j = lps[j - 1];

}

else if (i < textLength && pattern[j] != text[i])

{

if (j != 0)

{

j = lps[j - 1];

}

else

{

i++;

}

}

}

}

static void Main()

{

Console.WriteLine("Введите имя входного файла:");

string fileName = Console.ReadLine();

try

{

string text = File.ReadAllText(fileName);

Console.WriteLine("Введите подстроку для поиска:");

string pattern = Console.ReadLine();

KMPSearch(pattern, text);

}

catch (FileNotFoundException)

{

Console.WriteLine("Файл не найден.");

}

}

}

1. Алгоритм Морриса и Пратта

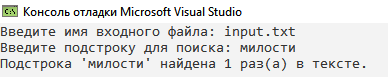


Рисунок 4 – Результат работы программы

Листинг:

using System;

using System.IO;

public class MorrisPrattSubstringSearch

{

public static void Main()

{

Console.Write("Введите имя входного файла: ");

string fileName = Console.ReadLine();

Console.Write("Введите подстроку для поиска: ");

string substring = Console.ReadLine();

try

{

string text = File.ReadAllText(fileName);

int count = SearchSubstring(text, substring);

Console.WriteLine($"Подстрока '{substring}' найдена {count} раз(а) в тексте.");

}

catch (FileNotFoundException)

{

Console.WriteLine("Файл не найден.");

}

catch (Exception ex)

{

Console.WriteLine($"Ошибка при чтении файла: {ex.Message}");

}

}

private static int SearchSubstring(string text, string substring)

{

int count = 0;

int[] prefixTable = BuildPrefixTable(substring);

int i = 0; // указатель на текущий символ в тексте

int j = 0; // указатель на текущий символ в подстроке

while (i < text.Length)

{

if (text[i] == substring[j])

{

i++;

j++;

if (j == substring.Length)

{

count++;

j = prefixTable[j];

}

}

else if (j > 0)

{

j = prefixTable[j];

}

else

{

i++;

}

}

return count;

}

private static int[] BuildPrefixTable(string substring)

{

int[] prefixTable = new int[substring.Length + 1];

int i = 0;

int j = -1;

prefixTable[i] = j;

while (i < substring.Length)

{

while (j >= 0 && substring[i] != substring[j])

{

j = prefixTable[j];

}

i++;

j++;

prefixTable[i] = j;

}

return prefixTable;

}

}

1. Алгоритм Бойера-Мура

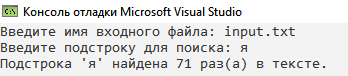


Рисунок 5 – Результат работы программы

Листинг:

using System;

using System.IO;

using System.Collections.Generic;

public class BoyerMooreSubstringSearch

{

public static void Main()

{

Console.Write("Введите имя входного файла: ");

string fileName = Console.ReadLine();

Console.Write("Введите подстроку для поиска: ");

string substring = Console.ReadLine();

try

{

string text = File.ReadAllText(fileName);

int count = SearchSubstring(text, substring);

Console.WriteLine($"Подстрока '{substring}' найдена {count} раз(а) в тексте.");

}

catch (FileNotFoundException)

{

Console.WriteLine("Файл не найден.");

}

catch (Exception ex)

{

Console.WriteLine($"Ошибка при чтении файла: {ex.Message}");

}

}

private static int SearchSubstring(string text, string substring)

{

int count = 0;

Dictionary<char, int> lastOccurrence = BuildLastOccurrenceTable(substring);

int i = substring.Length - 1; // указатель на текущий символ в тексте

int j = substring.Length - 1; // указатель на текущий символ в подстроке

while (i < text.Length)

{

if (text[i] == substring[j])

{

if (j == 0)

{

count++;

i += substring.Length;

j = substring.Length - 1;

}

else

{

i--;

j--;

}

}

else

{

if (lastOccurrence.ContainsKey(text[i]))

{

int shift = j - lastOccurrence[text[i]];

i += Math.Max(1, shift);

}

else

{

i += substring.Length;

}

j = substring.Length - 1;

}

}

return count;

}

private static Dictionary<char, int> BuildLastOccurrenceTable(string substring)

{

Dictionary<char, int> lastOccurrence = new Dictionary<char, int>();

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

{

lastOccurrence[substring[i]] = i;

}

return lastOccurrence;

}

}

1. Оценка трудоемкости алгоритмов поиска подстрок в строках:

Алгоритм прямого поиска (Brute-Force):

Худший случай: O(n \* m), где n - длина текста, m - длина подстроки. В худшем случае выполняется сравнение всех символов текста со всеми символами подстроки.

Алгоритм Кнута, Морриса и Пратта (KMP):

Построение таблицы сдвигов: O(m), где m - длина подстроки.

Поиск: O(n), где n - длина текста. В худшем случае выполняется один проход по всем символам текста.

Алгоритм Бойера-Мура:

Построение таблицы последнего вхождения: O(m), где m - длина подстроки.

Поиск: O(n/m), где n - длина текста, m - длина подстроки. В худшем случае выполняется поиск с пропусками по тексту.

Оба алгоритма Кнута, Морриса и Пратта, и Бойера-Мура имеют лучшую трудоемкость по сравнению с прямым поиском.

Вывод: в ходе выполнения лабораторной работы была рассмотрена оценка трудоемкости алгоритмов поиска подстрок в строках, а также реализованы все возможные алгоритм на практике.

Контрольные вопросы:

1. Что такое поиск подстроки в строке?

Поиск подстроки в строке - это процесс нахождения всех вхождений заданной подстроки внутри исходной строки.

1. Как работает алгоритм прямого поиска?

Алгоритм прямого поиска (также известный как "Naive" или "Brute-Force") осуществляет поиск подстроки путем последовательного сравнения всех возможных смещений подстроки от начала строки. Если найдено совпадение всех символов, то считается, что подстрока найдена.

1. Как работает алгоритм Кнута, Морриса и Пратта?

Алгоритм Кнута, Морриса и Пратта (KMP) использует предварительную обработку подстроки, чтобы определить таблицу сдвигов, основываясь на повторяющихся префиксах и суффиксах. Затем алгоритм осуществляет эффективный поиск, используя эту таблицу для определения правильных сдвигов.

1. Как работает алгоритм Бойера и Мура?

Алгоритм Бойера и Мура использует таблицу сдвигов, основанную на последнем вхождении каждого символа в подстроке. Это позволяет сразу перескакивать через большие участки текста, если символ не совпадает, и осуществлять более эффективный поиск.

1. Как оценить трудоемкость алгоритма поиска по тексту?

Трудоемкость алгоритма поиска по тексту обычно оценивается по времени выполнения. Она может быть измерена в количестве операций сравнения символов или количестве сравнений и сдвигов, которые выполняет алгоритм.

1. Как задается величина сдвига в алгоритме Кнута, Морриса и Пратта?

В алгоритме Кнута, Морриса и Пратта величина сдвига определяется с помощью таблицы сдвигов, которая указывает, насколько можно сдвинуть подстроку относительно исходного текста при возникновении несоответствия символов.

1. Как задается величина сдвига в алгоритме Бойера и Мура?

В алгоритме Бойера и Мура величина сдвига определяется на основе таблицы сдвигов, которая указывает, насколько можно сдвинуть поиск вправо, когда возникает несоответствие символов. Основная идея состоит в том, чтобы использовать информацию о последнем вхождении каждого символа в подстроке для определения оптимального сдвига.