Минобрнауки РФ

ФГБОУ ВО Тверской Государственный Технический Университет

Кафедра  “Программное Обеспечение”

Лабораторная работа N3

По дисциплине “Структуры и алгоритмы обработки данных”

Выполнил: студент группы Б.ПИН.РИС-22.06

Иванов Алексей Михайлович

Проверил: Мальков Александр Анатольевич

г. Тверь, 2023

Оглавление

[**Часть 1. Постановка задачи.** 3](#_Toc145406338)

[**Часть 2. Алгоритм решения.** 3](#_Toc145406339)

[**Часть 3. Оценка временной эффективности в терминах О-функций.** 4](#_Toc145406340)

[**Часть 4. Программный код.** 5](#_Toc145406341)

[**Часть 5. Тестирование и результаты работы.** 9](#_Toc145406342)

# **Часть 1. Постановка задачи.**

Работа со строками

- поиск подстроки в тексте

- алгоритм Рабина

- алгоритм Кнута - Морриса - Пратта

- стемминг.

# **Часть 2. Алгоритм решения.**

Наиболее очевиден алгоритм наивного поиска подстроки в строке:

Сравнение каждой подстроки в исходной строке с данной подстрокой. Он наиболее прост в реализации, но требует n\*m операций, так как необходимо перебирать каждую подстроку в тексте.

Алгоритм Рабина также имеет высокую алгоритмическую сложность, но вместе с тем имеет и ряд преимуществ. Его суть – полиномиальная хэш-функция. Для произвольной строки она задается как сумма произведений неких чисел, по которым однозначно можно определить символ строки (в случае моей реализации это ASCII-код) на простое число в степени индекса строки. Тогда хэш-функция будет однозначно определять строку, так как числа простые и коэфициенты перед множителями не пересекаются между собой. Проблема переполнения решается взятием остатка от деления на int.MaxValue. Тогда необходимо подсчитать ее значение для искомой подстроки, и далее последовательно считать для каждой подстроки необходимой длины в исходной строке. Здесь и заключается ключ к эффективности алгоритма: для такой хэш функции нет необходимости ее значение полностью пересчитывать, достаточно лишь вычесть код первого удаляемого символа(так как он больше не учитывается), разделить всю хэш-функцию на простой множитель, лежащий в ее основе(таким образом, хэш-функция будет отображать текущую подстроку без первого элемента), и добавить произведение первого нового символа на простой множитель в степени длины строки(теперь хэш-функция однозначно отображает значение новой проверяемой подстроки), а потом сравить значения целевое и полученное. В случае их совпадения провести явное сравнение подстроки и искомой подстроки – так как для переполнения брался остаток от деления, равенство хэш-функций более равносильно равенству строк, а является лишь следствием из этого утверждения.

И самый совершенный алгоритм поиска подстроки – алгоритм Кнута-Морриса-Прата. Его особенность в том, что он учитывает, что строки могут отличаться лишь немного, и это отображается в префикс-фукции.

Она показывает для каждого символа строки длину максимального префикса подстроки, который будет ее суффиксом, префикс это начало строки, суффикс – ее конец. Для строки “abcdabcabc” это будет массив [0, 0, 0, 0, 1, 2, 3, 1, 2, 3]. Так как ее начало совпадает в определенной степени с ее окончанием, при сравнении можно будет использовать эту информацию, чтобы пропускать многие подстроки в исходной, если быть точнее, то пропустить можно будет столько символов, каково значение префикс функции для последнего совпавшего символа. Ведь уже проверенные символы можно не проверять снова – при смещении на это значение, у нас уже будет определенное количество совпавших символов.

Алгоритм стэмминга – поиска основы слова – это один из алгоритмов лексического анализа, одного из самых наукоемких разделов компьютерной науки. В рамках своей работы я его называю примитивный стэмминг. Сделан он самым очевидным образом: существует массив всех приставок, суффиксов и окончаний слов в русском языке, и в случае совпадения мы отсекаем совпавшую часть слова.

# **Часть 3. Оценка временной эффективности в терминах О-функций.**

Составим сводную таблицу эффективности для матрицы в обоих типах представлений.

|  |  |
| --- | --- |
|  | Эффективность (n и m длины исходной и искомой строк) |
| Наивный | O(n\*m) |
| Рабина | О(n\*m) |
| КМП | O(n+m) |

Наивный поиск подстроки будет требовать сравнения n подстрок длиной m, а значит его сложность будет им прямо пропорциональна.

Алгоритм Рабина также требует для каждой из (n-m) подстрок сделать сделать перерасчет текущей хэш-функции, а для этого требуется лишь одно вычитание, деление и сложение с возведением в степень (оно может быть выполнено за log n). Таким образом, он будет линейно зависить от n и m. И тем не менее, из-за неоднозначности хэш-функции он может при точном поиске деградировать до nm при больших размерах данных.

Алгоритм КПМ.

Алгоритм КПМ требует лишь одного прохода по входной и выходной строке, таким образом имея всегда сложность n+m.

# **Часть 4. Программный код.**

Итоговая библиотека классов имеет следующий вид:

/// <summary>

/// Класс, позволяющий работать со строками

/// </summary>

public static class StringMethods

{

/// <summary>

/// Позволяет найти индекс первой встречи подстроки в данной строке

/// </summary>

/// <param name="srcStr">Строка, в которой осуществляется поиск</param>

/// <param name="targetStr">Искомая строка</param>

/// <returns>Индекс первой встречи подстроки в данной строке

/// либо -1 в случае если подстроки нет в данной строке</returns>

public static int NaiveSubstringFind(string srcStr, string targetStr)

{

int len = targetStr.Length;

for (int i = 0; i < srcStr.Length - len; i++)

{

bool find = true;

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

{

if (srcStr[i + j] != targetStr[j])

{

find = false;

break;

}

}

if (find)

return i;

}

return -1;

}

/// <summary>

/// Класс, позволяющий считать хэш-функцию

/// для алгоритма Рабина

/// </summary>

private static class RabineHashF

{

public const int p = 17;

public const int r = int.MaxValue;

/// <summary>

/// Считает значение хэш-функции для данной строки

/// </summary>

/// <param name="str">Данная строка</param>

/// <returns>Значение хэш-функции для данной строки</returns>

public static long RabineHash(string str)

{

long res = 0;

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

{

res += ((int)(char)str[i] \* ((int)Math.Pow(p, i))) % r;

}

return res % r;

}

/// <summary>

/// Считает значение хэш функции для символа

/// </summary>

public static int RabineHash(char c)

{

return (int)c;

}

/// <summary>

/// Позволяет пересчитать значение хэш-функции

/// </summary>

public static long Recalc(long starting, string scrStr, int i, int len)

{

long res = starting;

res -= ((int)(char)scrStr[i]);

res /= p;

res += ((int)(char)scrStr[i + len] \* ((int)Math.Pow(p, len - 1))) % r;

RabineHash(scrStr.Substring(i, len));

return res % r;

}

}

/// <summary>

/// Осуществляет поиск подстроки в данной алгоритмом Рабина

/// </summary>

/// <param name="srcStr">Строка, в которой осуществляется поиск</param>

/// <param name="targetStr">Искомая строка</param>

/// <returns>Индекс первой встречи подстроки в данной строке

/// либо -1 в случае если подстроки нет в данной строке</returns>

public static int RabineFind(string srcStr, string targetStr)

{

long targetHash = RabineHashF.RabineHash(targetStr);

int len = targetStr.Length;

long startingHash = RabineHashF.RabineHash(srcStr.Substring(0, len));

for (int i = 0; i < srcStr.Length - len; i++)

{

if (startingHash == targetHash)

{

bool correct = true;

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

{

if (srcStr[i + j] != targetStr[j])

{

correct = false;

break;

}

}

if (correct)

return i;

}

startingHash = RabineHashF.Recalc(startingHash, srcStr, i, len);

}

return -1;

}

/// <summary>

/// Префикс функция для алгоритма КМП

/// </summary>

private static int[] PrefixFunction(string s)

{

int[] result = new int[s.Length];

int i = 1;

int j = 0;

while (i < s.Length)

{

if (s[j] == s[i])

{

result[i] = j + 1;

i += 1;

j += 1;

}

else

{

if (j == 0)

{

result[i] = 0;

i += 1;

}

else

{

j = result[j - 1];

}

}

}

return result;

}

/// <summary>

/// Осуществляет поиск подстроки с помощью алгоритма КМП

/// </summary>

/// <param name="srcStr">Строка, в которой осуществляется поиск</param>

/// <param name="targetStr">Искомая строка</param>

/// <returns>Индексы встречи подстроки в данной строке

/// либо -1 в случае если подстроки нет в данной строке</returns>

public static int[] KMP(string srcStr, string targetStr)

{

int[] P = PrefixFunction(targetStr);

List<int> result = new List<int>();

int n = srcStr.Length;

int m = targetStr.Length;

int i = 0;

int j = 0;

while (i < n)

{

if (srcStr[i] == targetStr[j])

{

i += 1;

j += 1;

if (j == m)

{

result.Add(i - j);

j -= 1;

}

}

else

{

if (j > 0) { j = P[j - 1]; }

else { i += 1; }

}

}

if (result.Count == 0)

result.Add(-1);

return result.ToArray();

}

/// <summary>

/// Класс, содержащий морфемы для стемминга

/// </summary>

private static class StammingConstrains

{

/// <summary>

/// Массив приставок

/// </summary>

public static string[] Pristavki =

{

"пере", "подо", "о", "в", "с", "у",

"а","на", "за",

"о", "по" , "со" , "до" , "во" , "от", "об",

"вы" , "вз", "из" , "ис" ,

"о", "обо", "под", "воз",

"а", "над", "раз","рас",

"при", "пре", "изо", "пере", "про"

};

/// <summary>

/// Массив окончаний

/// </summary>

public static string[] Okonchaniya =

{

"ер", "ик", "ок", "ук", "ак", "ов",

"ев", "ий", "их", "ый", "ой", "ая",

"яя", "ея", "ия", "ым", "им", "ем",

"ам", "ям", "ею", "ию", "ою", "аю",

"яю", "ей", "ий", "ой", "ый", "ая",

"яя", "ее", "ие", "ое", "ые", "ть",

"тся", "му", "ми", "мя"

};

/// <summary>

/// Массив суффиксов

/// </summary>

public static string[] Suffiksy =

{

"один", "два", "три", "четыре", "пять",

"десять", "сто", "тысяча", "миллион",

"миллиард", "ы", "и", "ах", "ях", "а",

"я", "у", "ю", "у", "ю", "а", "я", "ом",

"ой", "е", "и", "ый", "ой", "ий", "ая",

"ое", "ие", "ейший", "ейшая", "ейшее",

"ее", "ей", "ого", "его", "их", "ых",

"ой", "его", "их", "ем", "ете", "ут",

"ю", "ешь", "ет", "ем", "ете", "ут", "л",

"ла", "ло", "ли", "у", "ешь", "ет", "и",

"ите", "не", "ни", "я", "вши", "ши", "ший",

"вш", "ющ", "о", "е", "ей", "ее", "ым",

"им", "й", "го", "му", "м", "х", "я", "е",

"и", "его", "их" , "то" , "нибудь",

"ни", "некто", "ничто", "ой", "его", "их",

"ся","сь", "ин", "ов", "ев", "енн"

};

}

/// <summary>

/// Осуществляет примитивный стэмминг данного слова

/// </summary>

/// <param name="word">Слово</param>

/// <returns>Основа слова</returns>

public static string Stamm(string word)

{

Array.Sort(StammingConstrains.Pristavki, (x, y) => -x.Length.CompareTo(y.Length));

Array.Sort(StammingConstrains.Suffiksy, (x, y) => -x.Length.CompareTo(y.Length));

Array.Sort(StammingConstrains.Okonchaniya, (x, y) => -x.Length.CompareTo(y.Length));

string stammedWord = word;

foreach (string rule in StammingConstrains.Okonchaniya)

{

if (stammedWord.EndsWith(rule))

{

stammedWord = stammedWord.Remove(stammedWord.Length - rule.Length - 1);

break;

}

}

foreach (string rule in StammingConstrains.Suffiksy)

{

if (stammedWord.EndsWith(rule))

{

stammedWord = stammedWord.Remove(stammedWord.Length - rule.Length);

break;

}

}

foreach (string rule in StammingConstrains.Suffiksy)

{

if (stammedWord.EndsWith(rule))

{

stammedWord = stammedWord.Remove(stammedWord.Length - rule.Length);

break;

}

}

foreach (string rule in StammingConstrains.Suffiksy)

{

if (stammedWord.EndsWith(rule))

{

stammedWord = stammedWord.Remove(stammedWord.Length - rule.Length);

break;

}

}

foreach (string rule in StammingConstrains.Pristavki)

{

if (stammedWord.StartsWith(rule))

{

stammedWord = stammedWord.Substring(rule.Length);

break;

}

}

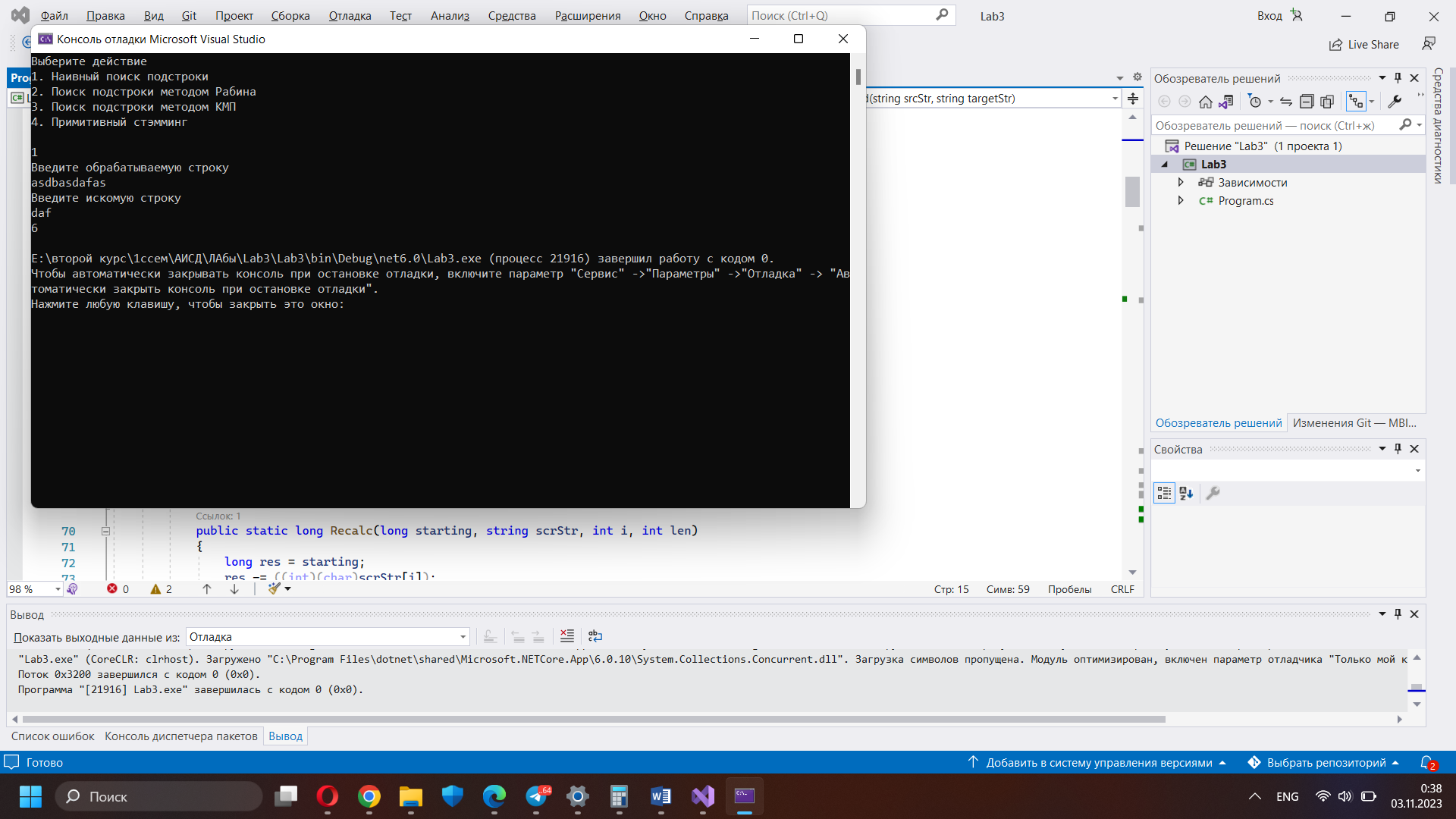
return stammedWord;

}

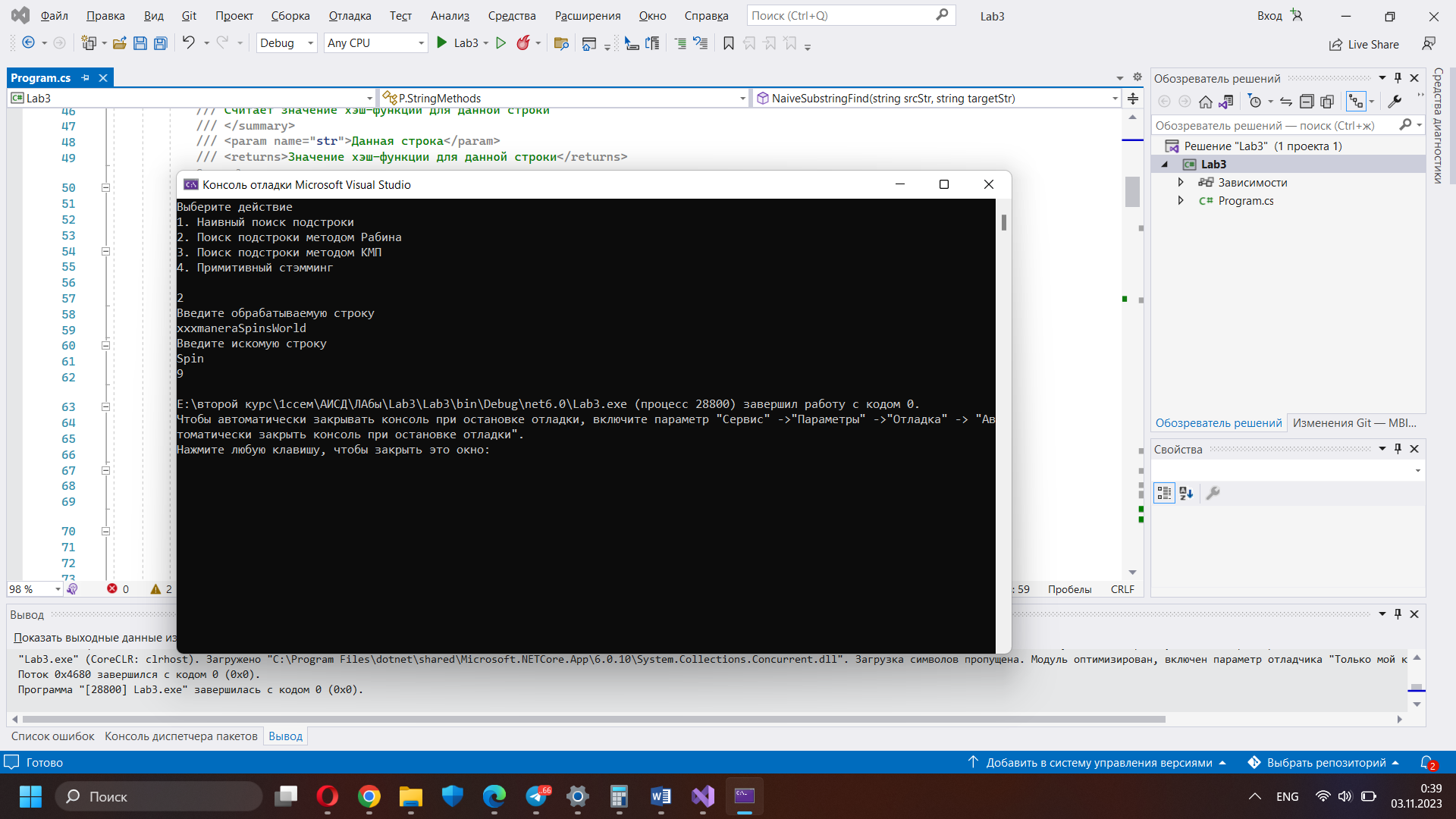
}

**Часть 5. Тестирование и результаты работы.**

Наивный поиск



Алгоритм Рабина



КМП

