МГТУ им. Баумана

Лабораторная работа №7

По курсу: "Анализ алгоритмов"

Поиск подстроки в строке

Работу выполнила: Оберган Татьяна, ИУ7-55Б

Преподаватели: Волкова Л.Л., Строганов Ю.В.

Москва, 2019

Оглавление

Введение

Цель работы: изучение алгоритмов поиска подстроки в строке.
Задачи данной лабораторной работы:

1. изучить алгоритмы Бойера-Мура и Кнута-Морриса-Пратта;
2. реализовать эти алгоритмы;
3. провести тестирование ПО.

1 Аналитическая часть

В данной части будут рассмотрены алгоритмы поиска подстроки в строке.

1.1 Общие сведения об алгоритмах поиска подстроки

Поиск подстроки в строке — одна из простейших задач поиска информации. Применяется в виде встроенной функции в текстовых редакторах, СУБД, поисковых машинах, языках программирования, программы определения плагиата осуществляют онлайн-проверку, используя алгоритмы поиска подстроки среди большого количества документов, хранящихся в собственной базе[1]. На сегодняшний день существует огромное разнообразие алгоритмов поиска подстроки. Программисту приходится выбирать подходящий в зависимости от таких факторов: длина строки, в которой происходит поиск, необходимость оптимизации, размер алфавита, возможность проиндексировать текст, требуется ли одновременный поиск нескольких строк. В данной лабораторной работе будут рассмотремы два алгоритма сравнения с образцом, алгоритм Кнута-Морриса-Пратта и алгоритм Бойера-Мура.

1.1.1 Стандартный алгоритм

Стандартный алгоритм начинает со сравнения первого символа текста с первым символом подстроки. Если они совпадают, то происходит переход ко второму символу текста и подстроки. При совпадении сравниваются следующие символы. Так продолжается до тех пор, пока не окажется, что подстрока целиком совпала с отрезком текста, или пока не встретят-

ся несовпадающие символы. В первом случае задача решена, во втором мы сдвигаем указатель текущего положения в тексте на один символ и заново начинаем сравнение с подстрокой[2].

1.1.2 Алгоритм Бойера-Мура

Алгоритм Бойера-Мура осуществляет сравнение с образцом справа налево, а не слева направо. Исследуя искомый образец, можно осуществлять более эффективные прыжки в тексте при обнаружении несовпадения. В этом алгоритме кроме таблицы суффиксов применяется таблица стоп-символов. Она заполняется для каждого сивола в алфавите. Для каждогостречающегося в подстроке символа таблица заполняется по принципу максимальной позиции символа в строке, за исключением последнего символа. При определении сдвига при очередном несовпадении строк, выбирается максимальное значение из таблицы суффиксов и стоп-символов[2].

1.1.3 Алгоритм Кнута-Морриса-Пратта

Алгоритм Кнута-Морриса-Пратта основан на принципе конечного автомата, однако он использует более простой метод обработки неподходящих символов. В этом алгоритме состояния помечаются символами, совпадение с которыми должно в данный момент произойти. Из каждого состояния имеется два перехода: один соответствует успешному сравнению, другой - несовпадению. Успешное сравнение переводит нас в следующий узел автомата, а в случае несовпадения мы попадаем в предыдущий узел, отвечающий образцу. В программной реализации этого алгоритма применяется массив сдвигов, который создается для каждой подстроки, которая ищется в тексте. Для каждого символа из подстроки рассчитывается значение, равное максимальной длине совпадающего префикса и суффикса отсительно конкретного элемента подстроки. Создание этого массива позволяет при несовпадении строки сдвигать ее на расстояние, большее, чем 1 (в отличие от стандартного алгоритма).

Вывод

В данном разделе были рассмотрены основные алгоритмы поиска подстроки в строке.

2 | Конструкторская часть

В данном разделе будут рассмотрены основные требования к программе и пошлаговая работа алгоритмов.

2.1 Требования к программе

Требования к вводу: Длина подстроки должна быть больше, чем длина строки.
Требования к программе:

- каждая из функций должна выдавать первый индекс вхождения подстроки в строку;
- если строка не содержит подстроку, то функция выдает -1.

2.2 Пример работы алгоритмов

В таблице 1 и таблице 2 буде рассмотрена пошаговая работа алгоритмов Кнута-Морриса-Пратта и Бойера-Мура на значениях строки s и подстроки sub.

string s = "ababacabaa"; string sub = "abaa";

2.2.1 Алгоритм Кнута-Морриса-Пратта

Для алгоритма Кнута-Морриса-Пратта вычисленный массив префиксов для заданой подстроки sub имеет значение: prefix = [0, 0, 1, 1]

Таблица 1 отображает пошаговую работу алгоритма Кнута-Морриса-Пратта при данном массиве префиксов.

Таблица 1. Пошаговая работа алгоритма Кнута-Морриса-Пратта.

a	b	a	b	a	c	a	b	a	a
a	b	a	a						
		a	b	a	a				
				a	b	a	a		
					a	b	a	a	
						a	b	a	a

2.2.2 Алгоритм Бойера-Мура

Для алгоритма Бойера-Мура вычисленный массив суффиксов для заданной подстроки sub имеет значение: suffix = [2, 5, 5, 6]. Переходы алфавита для подстроки sub: letters = ['a' = 0, 'b' = 2] Если буквы нет в letters, будет считаться, что переход равен длиние sub.

Таблица 2. Пошаговая работа алгоритма Бойера-Мура.

a	b	a	b	a	c	a	b	a	a
a	b	a	a						
		a	b	a	a				
						a	b	a	a

Вывод

В данном разделе были рассмотрены основные требования к программе, разобрана работа алгоритмов на конкретной строке и подстроке.

3 | Технологическая часть

Замеры времени были произведены на: Intel(R) Core(TM) i5-8300H, 4 ядра, 8 логических процессоров.

3.1 Выбор ЯП

В качестве языка программирования был выбран C# [3]. Средой разработки Visual Studio. Время работы алгоритмов было замерено с помощью класса Stopwatch. Тестирование было реализовано с помощью стандартного шаблона модульных тестов[4].

3.2 Сведения о модулях программы

Программа состоит из:

- Program.cs - главный файл программы, в котором располагается точка входа в программу
- StrMatching.cs - функции поиска подстроки

3.3 Листинг кода алгоритмов

В этой части будут рассмотрены листинги кода (листинг 3.1 - 3.5) реализованых алгоритмов.

Листинг 3.1: Стандартная функция

```
1 public static int Standard(string str, string substr)
2         {
```

```
3               for (int i = 0; i <= str.Length - substr.Length
                    ; i++)
4               {
5                   bool correct = true;
6                   for (int j = 0; j < substr.Length &&
                        correct; j++)
7                   {
8                       if (str[i + j] != substr[j])
9                           correct = false;
10                  }
11                  if (correct)
12                      return i;
13              }
14              return -1;
15          }
```

Листинг 3.2: Алгоритм KMP

```
1  public static int KMP(string str, string substr)
2          {
3              int[] prefix = PrefixFunction(substr);
4              int last_prefix = 0;
5              for (int i = 0; i < str.Length; i++)
6              {
7                  while (last_prefix > 0 && substr[
                       last_prefix] != str[i])
8                      last_prefix = prefix[last_prefix - 1];
9
10                 if (substr[last_prefix] == str[i])
11                     last_prefix++;
12
13                 if (last_prefix == substr.Length)
14                 {
15                     return i + 1 - substr.Length;
16                 }
17             }
18             return -1;
19         }
```

Листинг 3.3: Функция нахождения массива сдвигов

```
static int[] PrefixFunction(string substr)
        {
            int[] prefix = new int[substr.Length];

            int lastPrefix = prefix[0] = 0;
            for (int i = 1; i < substr.Length; i++)
            {
                while (lastPrefix > 0 && substr[lastPrefix]
                    != substr[i])
                    lastPrefix = prefix[lastPrefix - 1];

                if (substr[lastPrefix] == substr[i])
                    lastPrefix++;

                prefix[i] = lastPrefix;
            }
            return prefix;
        }
```

Листинг 3.4: Алгоритм Бойера-Мура

```
public static int BM(string str, string substr)
        {
            if (substr.Length == 0)
                return -1;

            Dictionary<char, int> letters = new Dictionary<
               char, int>();
            for (int i = 0; i < substr.Length; i++)
                if (letters.ContainsKey(substr[i]))
                    letters[substr[i]] = substr.Length - 1
                       - i;
                else
                    letters.Add(substr[i], substr.Length -
                       1 - i);

            int[] suffix = GetSuffix(substr);

            for (int i = substr.Length - 1; i < str.Length
               ;)
            {
```

```
17                int j = substr.Length - 1;
18                while (substr[j] == str[i])
19                {
20                    if (j == 0)
21                        return i;
22                    i--;
23                    j--;
24                }
25                var a = letters.ContainsKey(str[i]) ?
                      letters[str[i]] : substr.Length;
26                var b = suffix[substr.Length - 1 - j];
27                i += Math.Max(a, b);
28            }
29            return -1;
30        }
```

Листинг 3.5: Функция вычисления сдвигов суффиксов

```
1         static int[] GetSuffix(string substr)
2         {
3             int[] table = new int[substr.Length];
4             int lastPrefixPosition = substr.Length;
5
6             for (int i = substr.Length - 1; i >= 0; i--)
7             {
8                 if (IsPrefix(substr, i + 1))
9                     lastPrefixPosition = i + 1;
10                table[substr.Length - 1 - i] =
                      lastPrefixPosition - i + substr.Length -
                      1;
11            }
12
13            for (int i = 0; i < substr.Length - 1; i++)
14            {
15                int slen = SuffixLength(substr, i);
16                table[slen] = substr.Length - 1 - i + slen;
17            }
18
19            return table;
20        }
```

3.4 Тестирование программы

В этой части будет рассмотрен листинг функций тестирования алгоритмов (листинг 3.6 -3.7).

Листинг 3.6: Тестирование функции случайными значениями

```
[TestMethod]
        public void TestStandardRandom()
        {
            for (int i = 0; i < N; i++)
            {
                string s = rand.Next(1000, 9999999).
                   ToString();
                string sub = rand.Next(100, 999).ToString()
                   ;

                int iCorrect = s.IndexOf(sub);
                int res = lab_7_Substr.StrMatching.Standard
                   (s, sub);
                Assert.AreEqual(iCorrect, res, "str: " + s
                   + " sub: " + sub);
            }
        }
```

Листинг 3.7: Тестирование функции случайными значениями одной длины

```
[TestMethod]
        public void TestStandardSameLength()
        {
            for (int i = 0; i < N; i++)
            {
                string s = rand.Next(100, 999).ToString();
                string sub = rand.Next(100, 999).ToString()
                   ;

                int iCorrect = s.IndexOf(sub);
                int res = lab_7_Substr.StrMatching.Standard
                   (s, sub);
                Assert.AreEqual(iCorrect, res, "str: " + s
                   + " sub: " + sub);
```

```
12                  }
13              }
```

На рис. 3.1 предоставлен скриншот результатов работы тестов, на котором видно, что алгоритмы работают правильно.

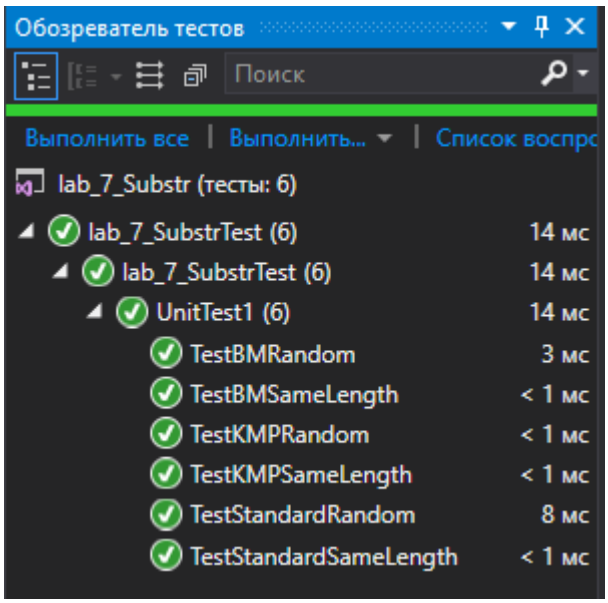

Рис. 3.1: Результат работы тестов

Вывод

В данном разделе были рассмотрены основные сведения о модулях программы, листинг кода алгоритмов и тестов, результаты тестирования, которое показало, что алгоритмы реализованы корректно.

4 | Исследовательская часть

В данном разделе будет проведен временной анализ работы алгоритмов.

4.1 Сравнительный анализ на основе замеров времени

Был проведен замер времени работы алгоритмов при разных размерах строки и фиксированом размере подстроки.

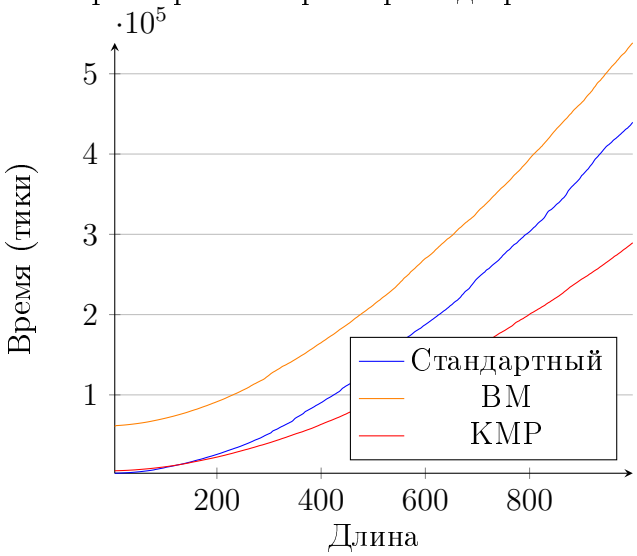

Рис. 4.1: Сравнение времени работы алгоритмов при увеличении длины строки. Длина подстроки 3

На рисунке 4.1 видно, что моя реализация алгоритма Бойера-Мура проигрывает стандартному и Кнута-Морриса-Пратта. Это происходит

потому что в моей реализации Бойера-Мура используется словарь и в каждом цикле происходит проверка наличия ключа в словаре, из-за чего и замедляется работа.

Вывод

Сравнительный анализ по времени показал, что внедрение словаря сильно замедляет алгоритм Бойера-Мура.

Заключение

В ходе лабораторной работы я изучила возможности применения и реализовала алгоритмы поиска подстроки в строке.

Было проведено тестирование, показавшее, что алгоритмы реализованы правильно.

Временной анализ показал, что неэффективно использовать структуру словаря для реализации алгоритма Бойера-Мура.

Литература

[1] Окулов С. М. Алгоритмы обработки строк. — М.: Бином, 2013. — 255 с.

[2] Дж. Макконнелл. Анализ лгоритмов. Активный обучающий подход

[3] Руководство по языку C#[Электронный ресурс], - режим доступа: https://docs.microsoft.com/ru-ru/dotnet/csharp/

[4] Основные сведения о модульных тестах [Электронный ресурс], - режим доступа: https://docs.microsoft.com/ru-ru/visualstudio/test/unit-test-basics?view=vs-2019