МГТУ им. Баумана

Лабораторная работа №7

По курсу: "Анализ алгоритмов"

Поиск подстроки в строке

Работу выполнила: Серёгина Дарья, ИУ7-54Б

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

Москва, 2020

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 Python [?]. Средой разработки IDLE 3.8. Тестирование было реализовано с помощью стандартного шаблона модульных тестов[3].

3.2 Листинг кода алгоритмов

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

Листинг 3.1: Алгоритм КМР

```
1  def getFail(substring):
2  fail = [0]*len(substring)
3  for i in range(1,len(substring)):
4  k = fail[i-1]
5  while k > 0 and substring[k] != substring[i]:
6  k = fail[k-1]
7  if substring[k] == substring[i]:
8  k = k + 1
9  fail[i] = k
10 return fail
11
12 def kmp(substring, text):
13 index = -1
```

```
f = getFail(substring)
k = 0
for i in range(len(text)):
while k > 0 and substring[k] != text[i]:
k = f[k-1]
if substring[k] == text[i]:
k = k + 1
if k == len(substring):
index = i - len(substring) + 1
break

# print("string    = '" + text + "'")
# print("substring = '" + substring + "'")
return index
```

Листинг 3.2: Алгоритм Бойера-Мура

```
def getSlide(pattern, m):
slide = 256*[-1]
for i in range(m):
slide[ ord(pattern[i]) ] = i;
return slide

def bm(pattern, text):
patternLoc = len(pattern)
textLoc = len(text)

slide = getSlide(pattern, patternLoc)
length = 0

while(length <= (textLoc - patternLoc)):
tmp = patternLoc - 1
while (tmp >= 0 and pattern[tmp] == text[length + tmp]):
tmp -= 1
if (tmp < 0):
return length
else:
length += max(1, tmp-slide[ord(text[length + tmp])])
```

3.3 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.

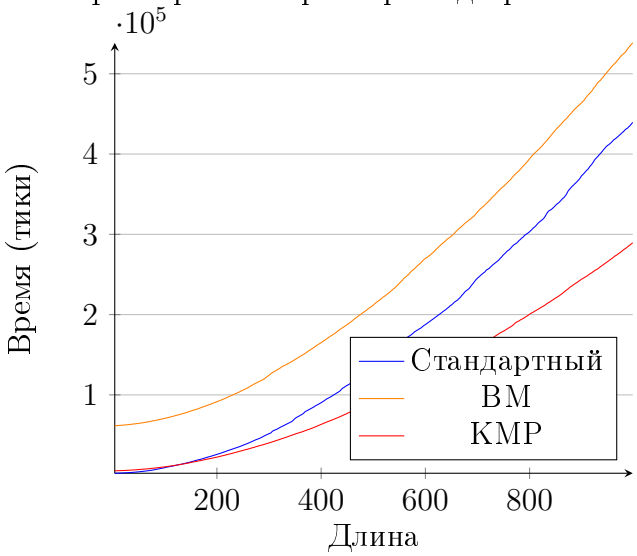

Рис. 4.1: Сравнение времени работы алгоритмов при увеличении длины строки. Длина подстроки 3

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