МГТУ им. Н.Э.Баумана

Лабораторная работа №7

По курсу: "Анализ алгоритмов"

Поиск подстроки в строке

Работу выполнил: Луговой Дмитрий, ИУ7-51Б

Преподаватель: Волкова Л.Л.

Москва, 2019

Оглавление

Введение

Поиск подстроки в строке— одна из простейших задач поиска информации. Применяется в виде встроенной функции в текстовых редакторах, СУБД, поисковых машинах, языках программирования и т. п.

Цель работы: изучение алгоритмов поиска подстроки в строке.

Задачи работы

Задачами данной лабораторной являются:

1. Изучить алгоритмы основные алгоритмы поиска подстроки в строке;
2. Реализовать эти алгоритмы;
3. Провести сравнительный анализ алгоритмов.

1 | Аналитическая часть

В данном разделе содержится описание задачи поиска подстроки в строке и алгоритмы для её решения.

1.1 Задача поиска подстроки

Поиск подстроки в строке— класс алгоритмов над строками, которые позволяют найти паттерн в тексте. Пусть есть некоторый текст T и слово W. Необходимо найти первое вхождение этого слова в указанном тексте. Это действие типично для любых систем обработки текстов. (Элементы массивов T и W – символы некоторого конечного алфавита – например, $0, 1$, или $a, \ldots, z$, или а, ..., я.)

Наиболее типичным приложением такой задачи является документальный поиск: задан фонд документов, состоящих из последовательности библиографических ссылок, каждая ссылка сопровождается «дескриптором», указывающим тему соответствующей ссылки. Надо найти некоторые ключевые слова, встречающиеся среди дескрипторов.

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 рассмотрены основные алгоритмы сравнения с образцом:

- Стандартный алгоритм;
- Алгоритм Кнута-Морриса-Пратта;
- Алгоритм Бойера-Мура.

1.1.1 Стандартный алгоритм

Стандартный алгоритм начинается 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 встретятся несовпадающие символы. В первом случае задача решена, во втором указатель текущего положения сдвигается в тексте на один символ и заново начинается сравнение с подстрокой.

1.1.2 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 означает переход в следующий узел автомата, а в несовпадения - попадание в предыдущий узел, отвечающий образцу. В программной реализации этого алгоритма применяется массив сдвигов, создающийся для каждой подстроки, которая ищется в тексте. Для каждого символа из подстроки рассчитывается значение, равное максимальной длине совпадающего префикса и суффикса отеосительно конкретного элемента подстроки. Создание этого массива позволяет при несовпадении строки сдвигать ее на расстояние, большее, чем 1 (в отличие от стандартного алгоритма).

1.1.3 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 символа в подстроке. Для каждого встречающегося 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.

Вывод

В данном разделе была рассмотрена задача поиска в подстроке, а также выделены основные алгоритмы ее решения - стандартный алгоритм, алгоритм Кнута-Морриса-Пратта и алгоритм Бойера-Мура.

2 | Конструкторская часть

В данном разделе рассмотрены пошаговые примеры выполнения алгоритмов поиска подстроки в строке.

2.1 Пример работы алгоритмов

Пусть

- string = "ababacabaa исходная строка;
- substring = "abaa искомая подстрока.

2.1.1 Стандартный

Стандартный алгоритм сравнивает подстроку и строку посимвольно, и начинает все сравнение заново, если хоть 1 символ не совпал. Результат его работы продемонстрирован в таблице 2.1.

Таблица 2.1: Пошаговая работа стандартного алгоритма

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		
					a	b	a	a	
						a	b	a	a

2.1.2 Алгоритм Кнута-Морриса-Пратта

Для алгоритма Кнута-Морриса-Пратта вычисленный массив префиксов для заданой подстроки substring имеет вид [0, 0, 1, 1].

Таблица 2.2 отображает пошаговую работу алгоритма Кнута-Морриса-Пратта при данном массиве префиксов.

Таблица 2.2: 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

2.1.3 Алгоритм Бойера-Мура

Для алгоритма Бойера-Мура вычисленный массив суффиксов для заданой подстроки subsrting имеет значение: [2, 5, 5, 6].

Переходы алфавита для подстроки sub: ['a' = 0, 'b' = 2]. Если буквы нет в подстроке, то считается, что переход равен длине подстроки.

Таблица 2.3 показывает пошаговую работу алгоритма Бойера-Мура для приведенных массивов суффиксов и стоп-символов.

Таблица 2.3: 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 примеры работы стандартного алгоритма, алгоритма Кнута-Морриса-Пратта и алгоритма Бойера-Мура на конкретной строке и подстроке.

3 | Технологическая часть

В данном разделе приведены требования к программному обеспечению, средства реализации и листинги кода

3.1 Требования к ПО

Программа на вход получает строку и искомую подстроку. Выход программы: индекс первого вхождения подстроки либо -1 в случае ее невхождения в строку.

3.2 Средства реализации

Для реализации представленных алгоритмов был выбран язык C++. Время работы алгоритмов было замерено с помощью функции high_resolution_clock() из библиотеки chrono. Для тестирования использовался компьютер на базе процессора Intel Core i5 (4 физических ядра, 8 логических).

3.3 Листинги кода

В листинге 3.1 показана реализация стандартного алгоритма.

Листинг 3.1: Функция стандартного алгоритма

```
1  int standartSearch(const std::string &text, const std::string &substring)
2  {
3      for (int i = 0; i <= text.length() - substring.length(); i++)
4      {
5          bool found = true;
6          for (int j = 0; j < substring.length() && found; j++)
7          {
8              if (text[i + j] != substring[j])
9              {
10                 found = false;
11             }
12         }
13         if (found)
14         {
15             return i;
16         }
```

```
17        }
18        return -1;
19    }
```

В листингах 3.2 представлена реализация алгоритма Кнута-Морриса-Пратта.

Листинг 3.2: Главная функция алгоритма Кнута-Морриса-Пратта

```
1   int searchKMP(const std::string &text, const std::string &substring)
2   {
3       std::vector<int> prefix = getPrefix(substring);
4       int last = 0;
5       for (int i = 0; i < text.length(); i++)
6       {
7           while (last > 0 && substring[last] != text[i])
8               last = prefix[last - 1];
9
10          if (substring[last] == text[i])
11              last++;
12
13          if (last == substring.length())
14          {
15              return i + 1 - substring.length();
16          }
17      }
18      return -1;
19  }
```

Основная функция вызывает функцию поиска массива префиксов, представленную в листинге 3.3.

Листинг 3.3: Функция вычисления массива префиксов

```
1   std::vector<int> getPrefix(const std::string &substring)
2   {
3       std::vector<int> prefix(substring.length());
4       prefix[0] = 0;
5       for (int i = 1; i < substring.length(); i++)
6       {
7           int last = prefix[i-1];
8           while (last > 0 && substring[last] != substring[i])
9               last = prefix[last - 1];
10
11          if (substring[last] == substring[i])
12              last++;
13
14          prefix[i] = last;
15      }
16      return prefix;
17  }
```

В листинге 3.4 представлена реализация алгоритма Бойера-Мура.

Листинг 3.4: Главная функция алгоритма Бойера-Мура

```
int searchBM(const std::string &text, const std::string &substring)
{
    std::unordered_map<char, int> stopTable;
    int m = substring.length();
    int n = text.length();
    std::vector<int> suffix = getSuffix(substring);
    for (int i = 0; i < m; ++i)
    {
        stopTable[substring[i]] = m - 1 - i;
    }

    for (int i = m - 1; i < n;)
    {
        int j = m - 1;

        while (substring[j] == text[i])
        {
            if (j == 0) return i;
            i--;
            j--;
        }
        auto stop_symbol = stopTable.find(text[i]);
        int stopAdd = stop_symbol != stopTable.end() ? stop_symbol->second :
            m;
        i += std::max(suffix[m - j - 1], stopAdd);
    }

    return -1;
}
```

Основная функция вызывает функцию поиска массива суффиксов, представленную в листинге 3.5.

Листинг 3.5: Функция вычисления массива суффиксов

```
std::vector<int> getSuffix(const std::string &substring)
{
    int n = substring.length();
    std::vector<int> table(n);
    int lastPrefixPosition = n;

    for (int i = n - 1; i >= 0; i--)
    {
        if (isPrefix(substring, i + 1))
            lastPrefixPosition = i + 1;
        table[n - 1 - i] = lastPrefixPosition - i + n - 1;
    }

    for (int i = 0; i < n - 1; i++)
    {
```

```
16              int slen = suffixLength(substring, i);
17              table[slen] = n - 1 - i + slen;
18          }
19
20          return table;
21      }
```

3.4 Вывод

Были реализованы стандартный алгоритм, алгоритм Бойера-Мура и алгоритм Кнута-Морриса-Пратта, решающие задачу поиска подстроки в строке.

4 | Экспериментальная часть

В данном разделе сравнительный анализ алгоритмов поиска подстроки в строке.

4.1 Сравнение временных характристик

Для сравнения стандартного алгоритма, алгоритма Бойера-Мура и алгоритма Кнута-Морриса-Пратта строки и подстроки генерировались случайным образом. Длина подстроки была фиксирована и равнялась 3, а размер строки менялся от 1000 до 100000 с шагом в 1000 символов. На рисунке 4.1 представлен график зависимости времени работы каждого из реализованных алгоритмов от размера строки.

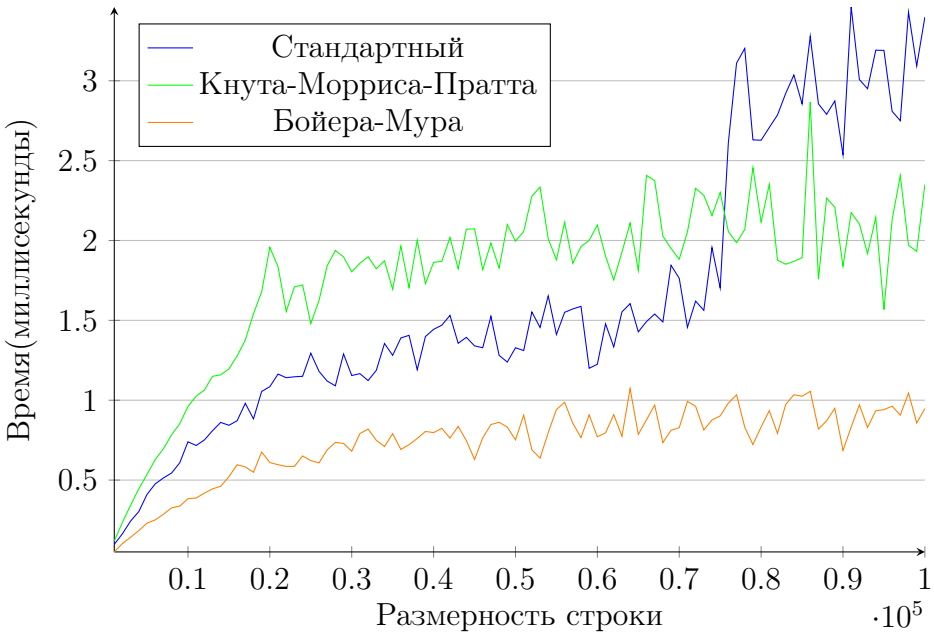

Рис. 4.1: Алгоритмы решения задачи поиска подстроки в строке

Как можно наблюдать на графике, алгоритм Бойера-Мура показывает лучшие результаты, время его работы практически перестает расти на размерах строк более 60000 символов. Алгоритм Кнута-Морриса-Пратта проигрывает стандартному во времени до строк размеров $\approx$ 75000 символов, после этого он становится эффективнее стандартного.

4.2 Вывод

В результате проведенных экспериментов был выявлен оптимальный для использования алгоритм, который показывает наименьшее время работы на любых размерах строк - алгоритм Бойера-Мура.

Заключение

В ходе работы были изучены и реализованы основные алгоритмы для решения задачи поиска подстроки в строке: стандартный алгоритм, алгоритм Бойера-Мура и алгоритм Кнута-Морриса-Пратта. Был проведен их сравнительный анализ, в ходе которого были получены зависимости времени выполнения алгоритмов от размеров входной строки. В результате экспериментов было получено, что алгоритм алгоритм Бойера-Мура работает одинаково быстро на всех размерах строк, время его работы практически перестает расти при размрах строк более 60000 символов, и он является оптимальным алгоритмом для решения поставленной задачи.