МГТУ им. Баумана

Лабораторная работа №5

По курсу: "Анализ алгоритмов"

Конвейер

Работу выполнил: Подвашецкий Дмитрий, ИУ7-54Б

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

Москва, 2019

Оглавление

1 Аналитическая часть

1.1 Цели и задачи работы

Цель лабораторной работы: изучить алгоритмы поиска подстроки в строке.
В рамках выполнения работы необходимо решить следующие задачи:

- изучить стандартный алгоритм, алгоритмы Кнута-Морриса-Пратта и Бойера-Мура;
- реализовать данные алгоритмы;
- привести подробное описание работы каждого алгоритма.

1.2 Описание алгоритмов

Постановка задачи: пусть даны строки *source* и *pattern*, обозначим их s и p соответственно. Необходимо проверить входит ли строка p в s, если да, то найти индекс первого вхождения.

1.2.1 Стандартный алгоритм

Стандартный алгоритм основан на последовательном сравнении всех подстрок строки s с p, т.е. будет происходить сравнение всех подстрок размера $|p|$, начиная с индексов $i = 1, 2, ..., |s| - |p| + 1$.

Пусть $s =$ "$abcabccba$", $p =$ "cab". В таблце 1 показаны сравнения символов, выполняемые в ходе работы алгоритма.

Таблица 1.1: Таблица коэффициентов для класса данных №1

№	a	b	c	a	b	c	c	b	a
1	c	a	b						
2		c	a	b					
3			c	a	b				

Поэтапное описание алгоритма:

1. $i = j = 0$, $i \in [0, |s| - 1], j \in [0, |p| - 1]$.

2. Посимвольно сравниваем s с p, *s[i]* с *p[j]*.

3. Если найдено совпадение, то увеличиваем i и j, если $j = |p|$, то подстрока найдена, иначе прикладываем p к следующему символу s.

В листинге 1 представлена реализация стандартного алгоритма.

Листинг 1.1: Стандартный алгоритм

```
int standart(const std::string &str, const std::string &
   substr)
{
        const size_t str_len = str.length();
        const size_t sub_len = substr.length();

        for (size_t i = 0; i <= str_len - sub_len; i++)
        {
                size_t tmp = i;

                for (size_t j = 0; j < sub_len; j++)
                {
                        if (substr[j] != str[tmp])
                        {
                                break;
                        }

                        if (j == sub_len - 1)
                        {
                                return static_cast<int>(i)
                                   ;
                        }

                        tmp++;
                }
        }

        return -1;
}
```

1.2.2 Алгоритм Кнута-Морриса-Пратта

Алгоритм Кнута-Морриса-Пратта является оптимизацией стандартного алгоритма. Необходимо дать определения префикса, суффикса и префикс-функции.

Префикс pf строки s - последовательность символов строки такая, что $pf = s[0, ..., i), i \in [0, |s| - 1]$.

Суффикс sf строки s - последовательность символов строки такая, что $sf = s[i, ..., |s| - 1], i \in [1, |s| - 1]$.

Префикс-функция строки s на позиции i - функция, возвращающая длину k наибольшего собственного префикса строки s, совпадающего с суффиксом этой строки.

Пусть строка $s =$ "*cbcbc*", рассмотрим все ее суффиксы, префиксы и значения префикс-функции, представленные в таблице 2.

Таблица 1.2: Пример таблицы префиксов и суффиксов для строки s

i	Prefix(s,i)	префиксы	суффиксы
0	0	c	
1	0	c	b
2	1	c, cb	c, bc
3	3	c, cb, cbc	c, bc, cbc
4	3	c, cb, cbc, cbcb	c, bc, cbc, bcbc

Данный алгоритм использует автомат, прикладывание искомой подстроки к строке происходит при помощи вспомогательного массива сдвигов.

Алгоритм заполнения массива сдвигов:

1. Создать массив $shift, |shift| = |p|$;
2. $shift|0| = 0$;
3. $i = 1, j = 0$ – индексы для строки и массива сдвигов соответственно;
4. Сравниваем $p[i]$ и $p[j]$, при совпадении $shift[i] = j+1$, сдвигаемся на символ вперед; иначе если $j = 0$, длина префикса нулевая, то $shift[i] = 0$.
5. Если $j \neq 0$, то $j = shift[j-1]$.
6. Если просмотрена не вся строка, то возвращаемся на 4ый шаг.

Шаблонная строка устанавливается в начало исходной, затем аналогично стандартному алгоритму проводтся посимвольное сравнение. При нахождении несоответствия выполняется сдвиг p с помощью массива сдвигов. Таким образом удается снизить число сравнений.

Рассмотрим работу алгоритма на примере $s =$ "*abacababda*, $p =$ "*abab*", см. рис. 1.

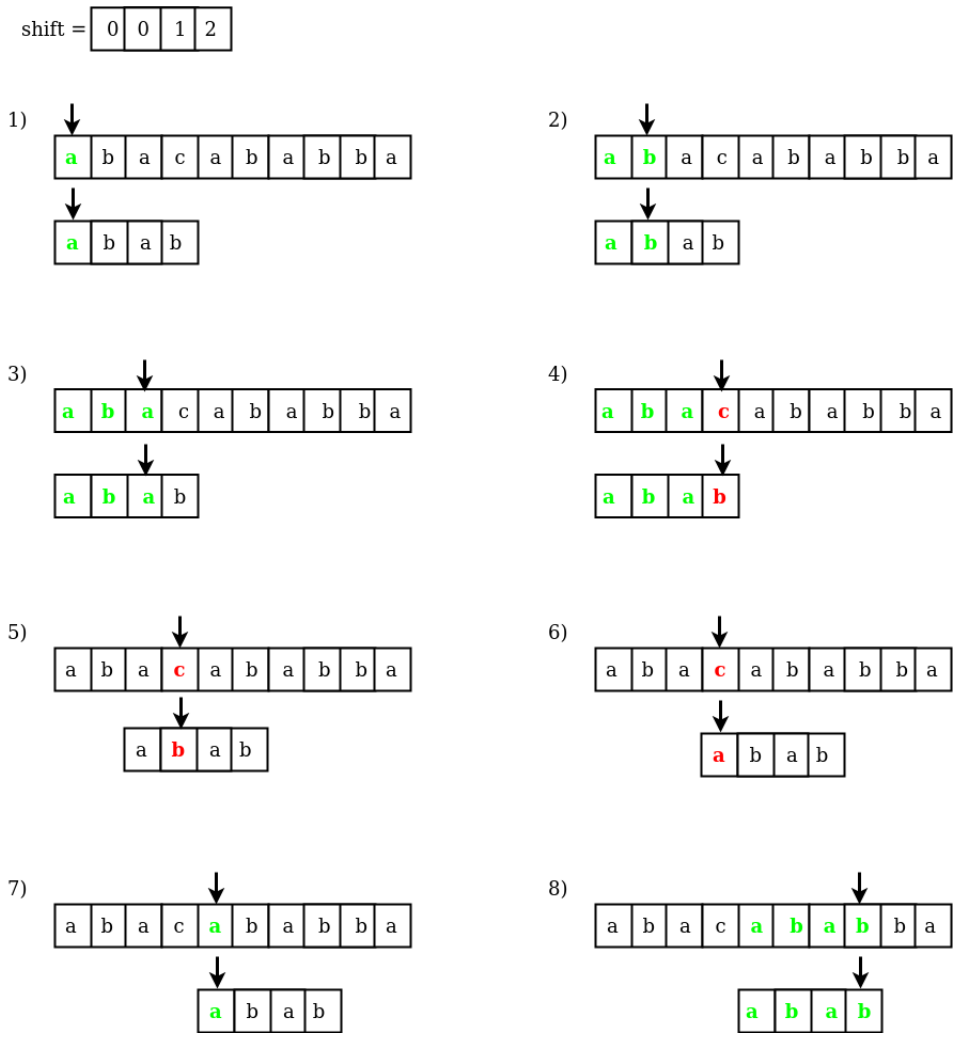

Рис. 1.1: Пример работы алгоритма Кнута-Морриса-Пратта.

В листинге 2 представлена реализации алгоритма Кнута-Морриса-Пратта.

Листинг 1.2: Алгоритм Кнута-Морриса-Пратта

```
1  int kmp(std::string &str, std::string &substr)
2  {
3          auto str_len = str.length();
4          auto sub_len = substr.length();
5
6          if (str_len < sub_len) {
7                  return -1;
8          }
9
10         std::vector<size_t> shift(sub_len);
11         shift[0] = 0;
12
13         for (size_t i = 1; i < sub_len; i++) {
14                 size_t j = shift[i - 1];
15                 while (j > 0 && substr[i] != substr[j]) {
```

```
16                          j = shift[j - 1];
17                  }
18                  if (substr[i] == substr[j]) {
19                          j++;
20                  }
21                  shift[i] = j;
22          }
23
24          for (size_t j = 0, i = 0; i < str_len; i++) {
25                  while (j > 0 && str[i] != substr[j]) {
26                          j = shift[j - 1];
27                  }
28                  if (str[i] == substr[j]) {
29                          j++;
30                  }
31                  if (j == sub_len) {
32                          return static_cast<int>(i - j + 1)
                               ;
33                  }
34          }
35
36          return -1;
37  }
```

1.2.3 Алгоритм Бойера-Мура

Рассмотрим пошагово алгоритм Бойера-Мура:

1. Построить таблицу смещений для каждого символа.
2. Совмещение s и p по началу.
3. Сравнение символов справа налево. Если найдено несовпадение, то p смещается вправо на число символов, взятое из таблицы смещений по символу исходной строки, иначе переходим к следующему символу.
4. Если просмотрена не вся исходная строка, то переход к пункту 3.

Отдельно рассмотрим построение таблицы смещений. Необходимо построить ее так, чтобы пропустить максимально возможное количество незначащих символов. Для этого каждому символу ставится в соответствие величина, равная разности длины шаблона и порядкового номера символа (если символ повторяется, то берется самое правое вхождение, при этом последний символ не учитывается), что равносильно порядковому номеру символа, если считать его с конца строки. Это дает возможность смещаться вправо на максимальное число позиций.

На рис. 2 приведен пример генерации таблицы смещений символов для $s =$ "*cbcbc*".

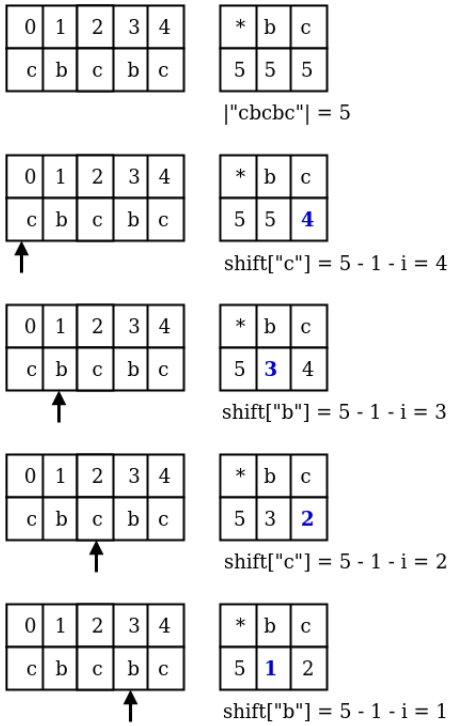

Рис. 1.2: Пример генерации таблицы сдвигов.

На рис.3 приведен пример работы алгоритма Бойера-Мура при $s = "abcdefgabc\, p = "efg"$.

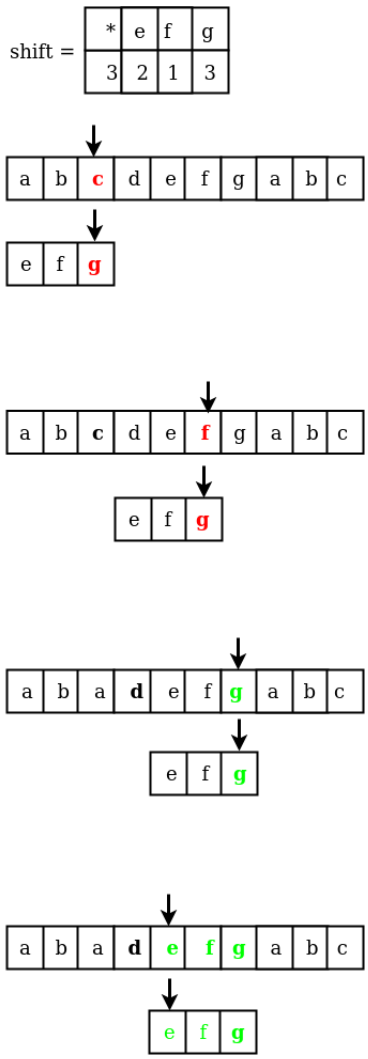

Рис. 1.3: Пример работы алгоритма Бойера-Мура.

На листинге 3 представлена реализация алгоритма Бойера-Мура.

Листинг 1.3: Стандартный алгоритм

```
1  std::map<char, size_t> get_shift(const std::string &substr
       ) {
2          size_t alphabet_size = 256;
3          auto sub_size = substr.length();
4          std::map<char, size_t> shift;
5
6          for (size_t symb = 0; symb < alphabet_size; ++symb
               ) {
7                  shift[static_cast<char>(symb)] = sub_size;
8          }
9
10         for (size_t symb = 0; symb < sub_size - 1; ++symb)
               {
11                 shift[static_cast<char>(substr[symb])] =
```

```
                    sub_size - symb - 1;
        }

        return shift;
}

int bm(std::string &str, std::string &substr) {
        auto str_len = str.length();
        auto sub_len = substr.length();
        if (str_len < sub_len) {
                return -1;
        }

        auto shift = get_shift(substr);
        auto start = sub_len - 1;
        auto i = start;
        auto j = start;
        auto k = start;

        while (j >= 0 && i < str_len) {
                j = start;
                k = i;
        while (j >= 0 && str[k] == substr[j]) {
                --k;
                --j;
        }

                i += shift[str[i]];
        }

        return static_cast<int>(k >= str_len - sub_len ?
           -1 : k + 1);
}
```

Вывод

По итогам аналитического раздела были описаны стандартный алгоритм, алгоритм Кнута-Морриса-Пратта и алгоритм Бойера-Мура для нахождения подстроки в строке.

Заключение

Таким образом, в ходе лабораторной работы были изучены, описаны и реализованы стандартный алгоритм, алгоритм Кнута-Морриса-Пратта и алгоритм Бойера-Мура для нахождения подстроки в строке.