Государственное образовательное учреждение высшего какого-то образования
“Московский государственный технический университет имени Н.Э.Баумана”

Дисциплина: Анализ алгоритмов

Лабораторная работа №7

ТЕМА

Алгоритмы поиска подстроки в строке

Студент группы ИУ7-54
Лозовский Алексей

2019 г.

Содержание

1 Аналитическая часть

1.1 Цель и задачи

Цель работы: Изучить работу алгоритмов поиска подстроки в строке. Для достижения посталенной цели выделим следующие задачи:

- разобрать стандартный алгоритм, алгоритмы Кнута-Морриса-Пратта и Бойера-Мура,
- реализовать разобранные алгоритмы,
- подробно описать работу каждого, с примерами.

1.2 Описание алгоритмов

В данном разделе будут рассмотрены три алгоритма поиска подстроки в строке: стандартный, Кнутта-Морриса-Пратта, а также Бойера-Мура.

Постановка задачи:
Пусть даны две строки:

- исходная строка – $Source$, в дальнейшнем будем обозначать ее S,
- строка-шаблон – $Pattern$, обозначим ее P.

Необходимо проверить содержится ли строка P в строке S. Если $P \in S$ вывести индекс первого вхождения, в противном случае -1.

1.2.1 Стандартный алгоритм поиска подстроки в строке

Идея стандартного алгоритма заключается в последовательном сравнении всех подстрок строки S с шаблонной. Пусть s, p – длины строк S, P соответственно. Тогда сравнение всех подстрок размера p будет происходить, начиная с $i = 1, \ldots, s - p + 1$.

Рассмотрим пример:

- строка $\boldsymbol{S}$ = "$cacabbbcabdb$";
- подстрока $\boldsymbol{P}$ = "$cabb$";

Ниже приведен пример пошаговой обработки строк S и P стандартным алгоритмом. Зеленым отмечены символы, которые совпали, красным, те, что нет.

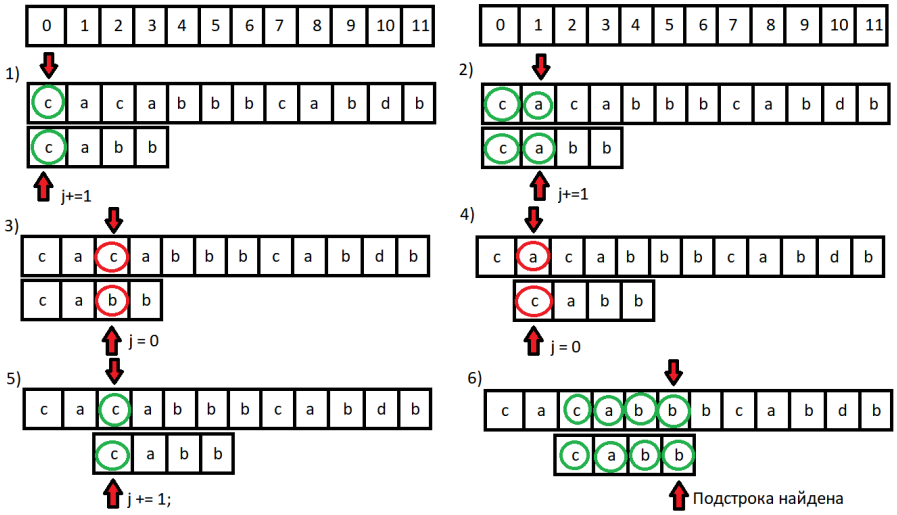

Рис. 1. Пример работы стандартного алгоритма

Опишем алгоритм пошагово:

1. $i = 0$, $j = 0$ индексы в строках S, P, $i \in [0, s)$ и $j \in [0, p)$,
2. Прикладываем подстроку к строке и сравниваем посимвольно $S[i], и P[j]$,
3. В случае совпадения увеличим индексы обеих строк и проверим следующие символы, иначе прикладываем строку к следующему символу строки,
4. Вернуться на шаг 2.

Ниже приведена возможная реализация алгоритма.

Листинг 1. Реализация стандартного алгоритма

```
int standart(std::string str, std::string substr){

    auto str_len = str.length();        //Находим длину строк
    auto sub_len = substr.length();

    if (str_len < sub_len)
        return -1;

    //Проходим по всем символам строки
    for (size_t i=0; i<=str_len-sub_len; ++i){
        auto tmp_i = i;
        for (size_t j=0; j<sub_len; ++j){  //Сравниваем все символы подстроки

            if (substr[j] != str[tmp_i])   //Если несовпадение
                break;                     //Двигаем подстроку вправо на символ

            if (j == sub_len-1)            //Если дошли до конца подстроки
                return i;                  //Найдено вхождение, вернуть индекс

            ++tmp_i;
        }
    }
    return -1;
}
```

1.2.2 Алгоритм Кнута-Морриса-Пратта

Одной из оптимизаций стандартного алгоритма является алгоритм Кнута-Морриса-Пратта. Введем понятия **префикса**, **суффикса** и **префикс-функции**.

Префиксом строки называется последовательность символов рассматриваемой строки, такая, что $pr = P[0, \ldots, i], i \in [0, \ldots, p-1)$, где pr - префикс строки P.

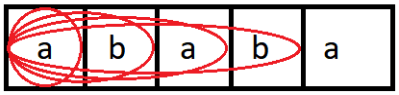

Рис. 2. Пример префиксов в строке.

Суффиксом строки называется последовательность символов рассматриваемой строки, такая что $sf = P[s-1, \ldots, i], i \in [1, \ldots, p-1]$, где sf - суффикс строки P.

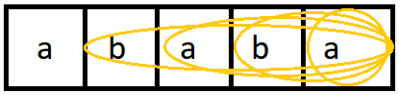

Рис. 3. Пример суффиксов в строке.

Префикс-функцией строки P на i позиции называется функция, возвращаюшая длину k наибольшего собственного префикса строки P, совпавшего с суффиксом этой строки.

Пусть дана строка $P = ababa$, рассмотрим все ее суффиксы, префиксы и вычислим префикс функции.

На рис. 4 изображена пошаговая работа префикс-функции для строки P. Зеленым обозначены наибольшие совпавшние с суффиксом sf префиксы pf. Желтым отмечены суффиксы, красным префиксы, зеленым - результат функции на текущей итерации.

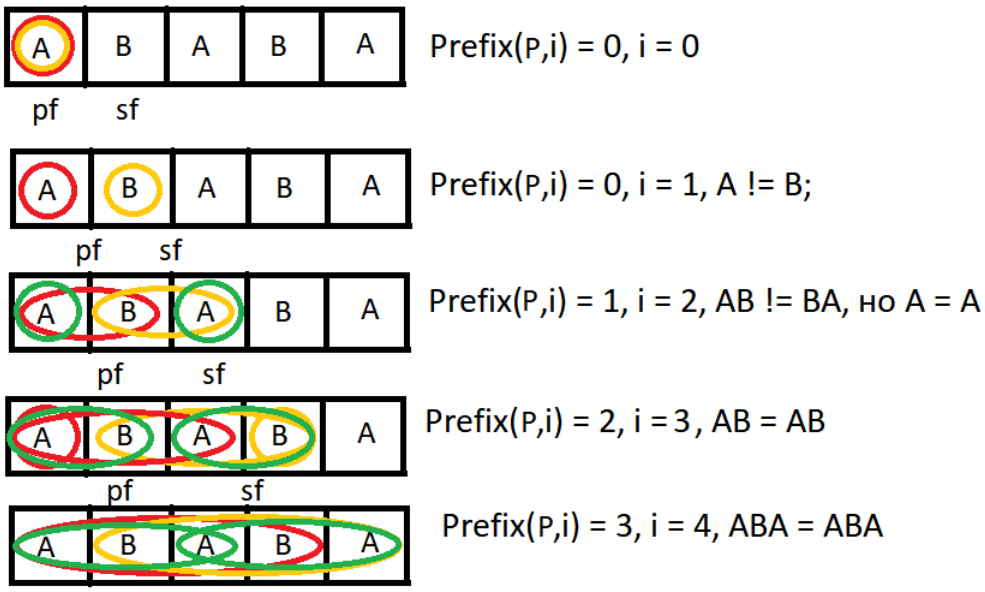

Рис. 4. Схема работы префикс-функции для строки P.

Пошаговый алгоритм заполнения массива сдвигов:

1. создать массив *shift*, $|shift| = |P|$,
2. shift[0] = 0,
3. i=1, j=0 индексы по строке и массиву сдвигов соответственно,
4. сравниваем P[i] и P[j],
5. если совпали, shift[i] = j+1; сдвигаемся на символ вперед,
6. если не совпали и j == 0, длина префикса нулевая, shift[i] = 0,
7. если j != 0 обновляем j = shift[j-1],
8. возвращаться на 4 шаг пока не будет рассмотрена вся строка.

Ниже приведена одна из возможных реализаций данного алгоритма, индекс i - отвечает за текущий рассматриваемый элемент, j за размер префикса:

Листинг 2. Алгоритм генерации массива сдвигов

```
using shift_t = std::vector<size_t>;

shift_t get_shift(const std::string &substr, const size_t &size){

    shift_t shift(size);  //Создаем массив размера size = substr.length();
    shift[0] = 0;         //Инициализируем 1 элемент нулем;

    for (size_t i=1; i<size; ++i){     //Цикл по всей строке, со ого2 символа
        size_t j = shift[i-1];    //Инициализируем j размером префика на j-1

        while (j > 0 && substr[i] != substr[j]) //Ищем совпадение
            j = shift[j-1];

        if (substr[i] == substr[j]) j++; //Если строки совпали увеличиваем
   счетчик

        shift[i] = j; //Записываем значение в массив
    }
    return shift;
}
```

Рассмотрим алгоритм КМП с использованием массива сдвигов. Данный алгоритм использует автомат, где перемещение строки-шаблона по исходной строке происходит при помощи вспомогательного массива сдвигов. Шаблонная строка устанавливается в начало сходной, после чего, как и в стандартном алгоритме производится посимвольное сравнение. В случае несоответствия символов выполняется сдвиг подстроки при помощи массива shift, определяющего какой символ сравнивать следующим. Таким образом удается снизить число сравнений.

Ниже на рисунках (5) и (6) приведены примеры работы алгоритма на примере строк $S =$ "$hello$" $P =$ "el" и $S =$ "$abacababda$ $P =$ "$abab$" соответственно.

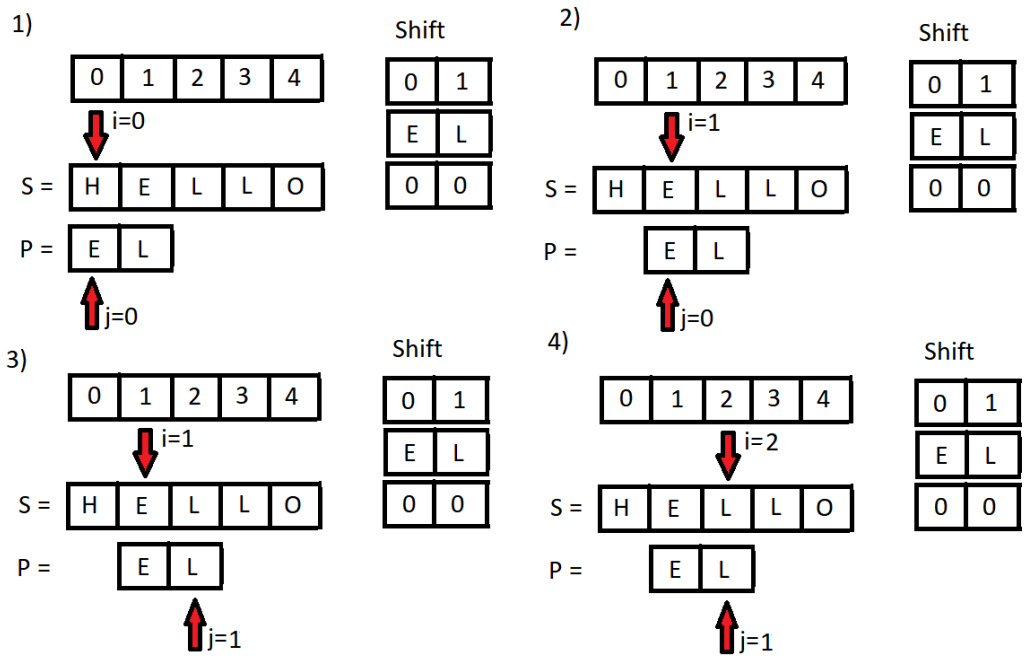

Рис. 5. Пример работы алгоритма КМП на строках $S =$ "$hello''$, $P =$ "el".

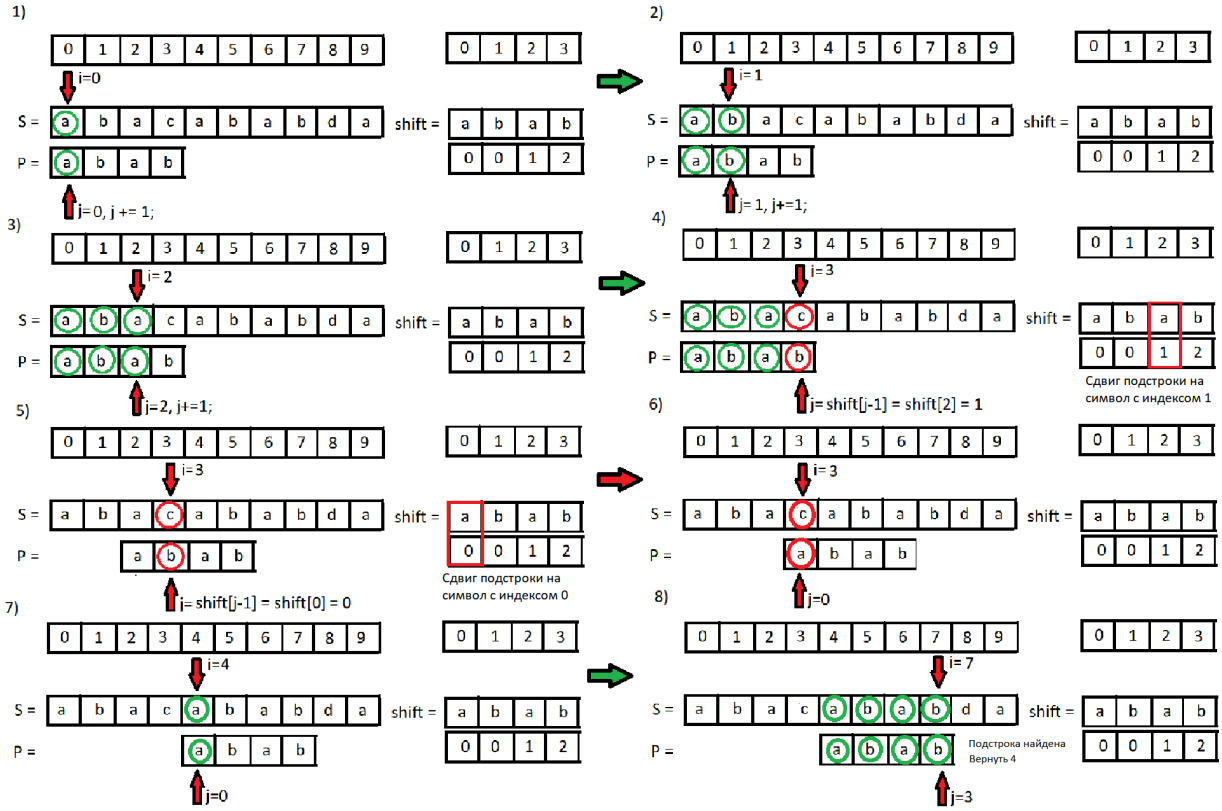

Рис. 6. Пример работы алгоритма КМП на строках $S =''$ $abacababda''$, $P =''$ $abab''$.

Ниже приведена возможная реализация алгоритма.

Листинг 3. Алгоритм КМП

```
using shift_t = std::vector<size_t>;

int kmp(std::string str, std::string substr){
    auto str_len = str.length();   // вычисляем длины строк
    auto sub_len = substr.length();

    if (str_len < sub_len) //Подстрока должна быть не больше строки
        return -1;

    shift_t shift = get_shift(substr, sub_len); //Вычисляем массив сдвигов


    for (size_t j=0, i=0; i<str_len; ++i){  //Идем по всей строке

        while (j > 0 && str[i] != substr[j])  //Делаем откат при необходимости
            j = shift[j-1];

            if (str[i] == substr[j]) j++; //При совпадении смещаем вправо

            if (j == sub_len)             // если весь шаблон был пройден
                return (i - j + 1);       // подстрока найдена, вернем
        }                                 // индекс первого вхождения
    return -1;
}
```

1.2.3 Алгоритм Бойера-Мура

Идея алгоритма.
Сначала строится таблица смещений для каждого символа. Затем исходная строка и шаблон совмещаются по началу, сравнение ведется по последнему символу справа налево. Если же символы не совпали, то шаблон смещается вправо, на число позиций, взятое из таблицы смещений по символу из исходной строки, и тогда снова сравниваются последние символы исходной строки и шаблона до тех пор, пока не будет найдено вхождение или не встретится конец исходной строки.

Построение таблицы смещений.
Таблица смещений строится так, чтобы пропускать максимальное число незначащих символов, но не большее, чем необходимо. Например, если на каком-то шаге алгоритма последние символы не совпали, и символ, находящийся в исходной строке не присутствует в шаблоне вообще, то можно сдвинуться вправо на полную длину шаблона. В общем случае, 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). Эта величина будет в точности равна порядковому номеру символа, если считать от конца строки, что и дает возможность смещаться вправо на максимально возможное число позиций.

Ниже приведен пример генераци массива сдвигов на строке.Звездочкой обозначены все остальные символы алфавита .

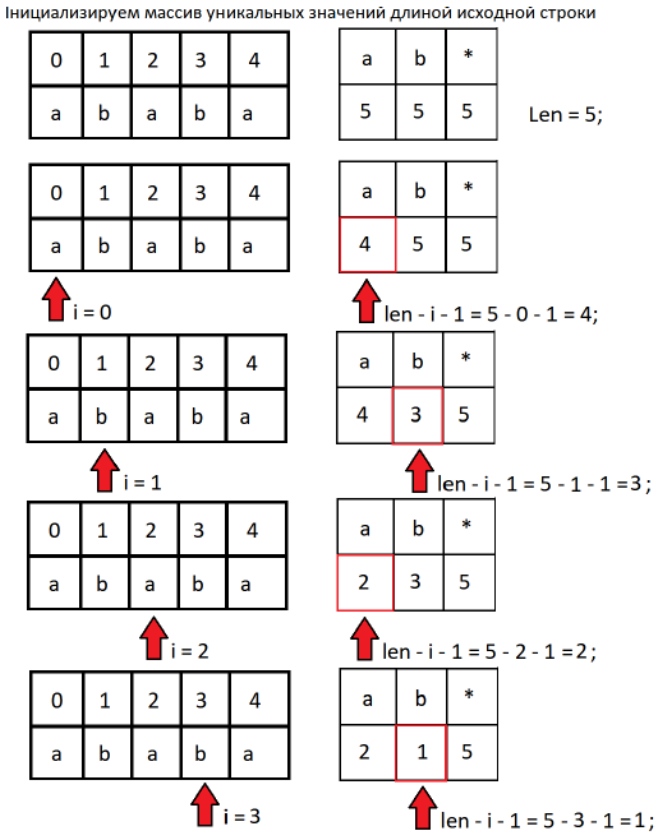

Рис. 7. Генерация массива сдвигов

Рассмотрим пошаговую работу алгоритма Бойера-Мура для строк $S =$ "$verysimple$" $P =$ "sim". Зеленым обозначены совпавшие символы, красным те, которые не совпали.

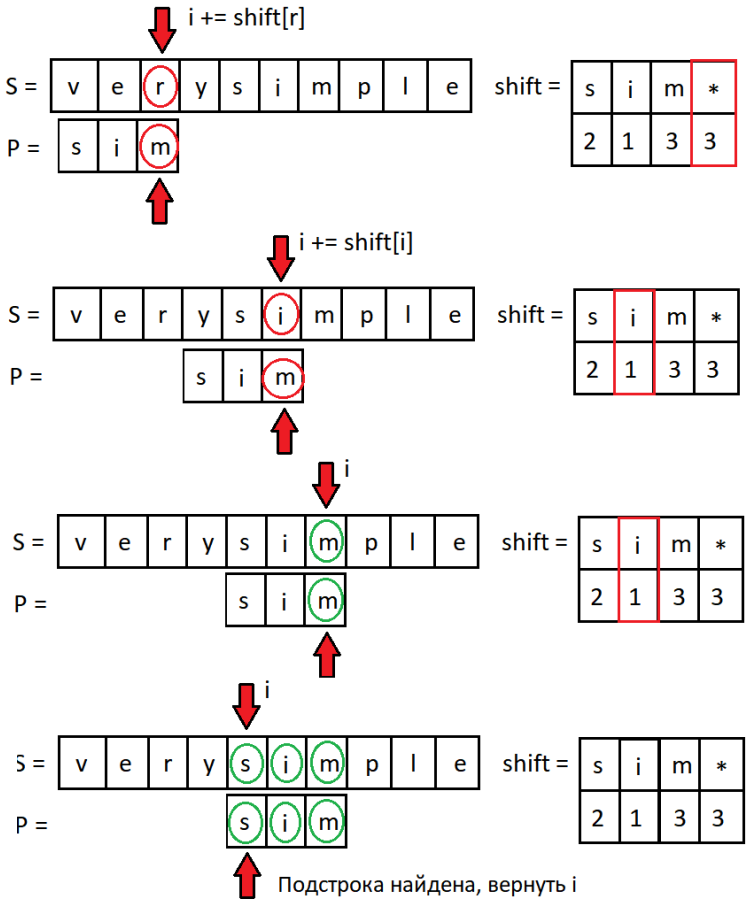

Рис. 8. Пример работы алгоритма Бойера-Мура

Приведем пример возможной реализации алгоритма.

Листинг 4. Алгоритм БМ

```
using shift_t = std::vector<size_t>;

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);

    int start = sub_len - 1; // последний символ искомой подстроки
    int i = start;
    int j = i;
    int k = i;

    while (j>=0 && i < str_len){
        j = start;
        k = i;

        // проход по совпавшим символам
        while (j>=0 && str[k] == substr[j]) { --k; --j;}

        i += shift[str[i]]; // сдвиг по главной строке
    }
    return k >= str_len-sub_len ? -1 : k+1;
}
```

2 Вывод

Таким образом, были изучены и описаны стандартный алгоритм, алгоритм Кнута-Морриса-Пратта и Бойера-Мура поиска подстроки в строке.