**Лабораторная работа** **1. Поиск подстроки.**

**Цель работы: Закрепление теоретических знаний и получение практических навыков применения алгоритмов поиска подстрок.**

**Краткие теоретические сведения**

1. **Основные термины и определения.**

**Алфавит**– конечное множество символов.

**Строка** (слово) – это последовательность символов из некоторого алфавита.

***Длина строки*** – количество символов в строке.

Строку будем обозначать *символами алфавита*, например X=x[1]x[2]...x[n] – строка длиной n, где x[i] – i -ый символ строки Х, принадлежащий алфавиту. Строка, не содержащая ни одного символа, называется *пустой*.

Строка X называется **подстрокой**строки Y, если найдутся такие строки Z1 и Z2, что Y=Z1XZ2. При этом Z1 называют *левым*, а Z2 – *правым крылом подстроки*. Подстрокой может быть и сама строка. Иногда при этом строку X называют вхождением в строку Y. Например, строки hrf и fhr является подстроками строки abhrfhr.

*Подстрока* X называется **префиксом** строки Y, если есть такая *подстрока* Z, что Y=XZ. Причем сама строка является префиксом для себя самой (так как найдется нулевая строка L, что X=XL ). Например, *подстрока* ab является префиксом строки abcfa.

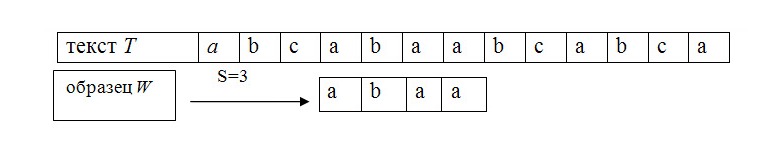
*Подстрока* X называется **суффиксом** строки Y, если есть такая *подстрока* Z, что Y=ZX. Аналогично, строка является *суффиксом* себя самой. Например, *подстрока* bfg является *суффиксом* строки vsenfbfg.

**Задача поиска подстроки в строке.**

Пусть задана строка, состоящая из некоторого количества символов. Необходимо определить, входит ли заданная *подстрока* в данную строку. Если входит, то требуется определить номер символа строки, начиная с которого подстрока входит в строку.

### По традиции шаблон поиска или образец принято обозначать как needle (англ. «иголка»), а строку, в которой ведётся поиск — как haystack (англ. «стог сена»).

#### Прямой поиск

Пусть есть некоторый текст Т и слово (или образ) W. Необходимо найти первое вхождение этого слова в указанном тексте. Это действие типично для любых систем обработки текстов. (Элементы массивов Т и W – символы некоторого конечного алфавита – например, {0, 1}, или {a, …, z}, или {а, …, я}.)  
Поиск строки формально определяется следующим образом. Пусть задан массив Т из N элементов и массив W из M элементов, причем 0<M≤N. Поиск строки обнаруживает первое вхождение W в Т, результатом будем считать индекс i, указывающий на первое с начала строки (с начала массива Т) совпадение с образом (словом).  
Пример. Требуется найти все вхождения образца W = abaa в текст T=abcabaabcabca.  
  
Образец входит в текст только один раз, со сдвигом S=3 .  
  
Основная идея алгоритма прямого поиска заключается в посимвольном сравнении текста со словом (строки с подстрокой).

В начальный момент производится сравнение первого символа строки с первым символом подстроки, при их совпадении производится сравнение второго символа строки со вторым символом подстроки и т. д. Если произошло совпадение всех символов, то фиксируется факт нахождения подстроки. В противном случае производится сдвиг подстроки на одну позицию вправо и повторяется посимвольное сравнение, то есть сравнивается второй символ строки с первым символом подстроки, третий символ строки со вторым символом подстроки и т. д. ( рис. 1). Символы, которые сравниваются, на рисунке выделены жирным. Рассматриваемые сдвиги подстроки повторяются до тех пор, пока конец подстроки не достиг конца строки или не произошло полное совпадение символов подстроки со строкой, то есть найдется *подстрока*.

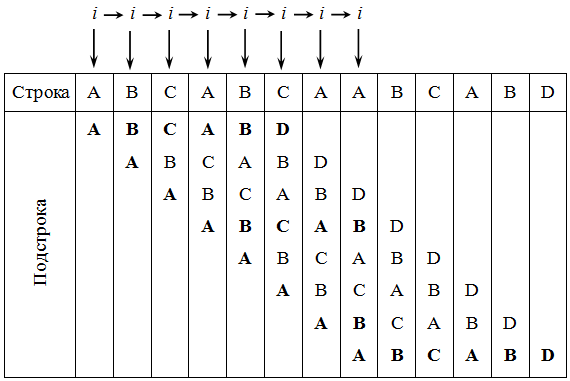


Рис. 1. Демонстрация алгоритма прямого поиска

#### Алгоритм прямого поиска

1. I=1,
2. J=1
3. Если Т[I]=W[J], то I=I+1, J=J+1 переход на п.5 иначе J=J+1 и переход на п.4
4. Если J+M>N, то конец «Текст не найден» иначе переход на п.3
5. Если I<M, то переход на п.3 иначе вывод J-M и конец (Текст найден)

Недостатки алгоритма:  
1. высокая сложность — O(N\*M), в худшем случае – Θ((N-M+1)\*M);  
2. после несовпадения просмотр всегда начинается с первого символа образца и поэтому может включать символы T, которые ранее уже просматривались (если строка читается из вторичной памяти, то такие возвраты занимают много времени);  
3. информация о тексте T, получаемая при проверке данного сдвига S, никак не используется при проверке последующих сдвигов.

#### Алгоритм Боуера – Мура – Хорспула (БМХ-поиск)

Процедура алгоритма очень простая. Сначала строится таблица смещений для каждого символа шаблона. Затем исходная строка и шаблон совмещаются по началу, сравнение ведется справа налево, начиная с последнего символа шаблона. Если последние символы совпадают, то сравнение идет по предпоследнему символу и так далее. Если же символы не совпали, то шаблон смещается вправо, на число позиций, взятое из таблицы смещений для символа из исходной строки, и снова сравниваются последний символ шаблона и соответствующий символ строки. И так далее, пока шаблон полностью не совпадет с подстрокой исходной строки, или не будет достигнут конец строки.

**Описание алгоритма**

1. Сдвиг шаблона производится слева направо, сравнение символов строки и шаблона производится справа налево. Совмещается начало текста (строки) и шаблона, проверка начинается с последнего символа шаблона. Если символы совпадают, производится сравнение предпоследнего символа шаблона и т. д. Если все символы шаблона совпали с наложенными символами строки, значит, подстрока найдена, и поиск окончен.

Если же какой-то символ шаблона не совпадает с соответствующим символом строки, шаблон сдвигается на несколько символов вправо, и проверка снова начинается с последнего символа.

Количество символов, на которые сдвигается шаблон вычисляются заранее и хранятся в таблице смещений.

Таблица смещений строится по принципу – пропускать столько символов, сколько возможно, но не более. В общем случае в таблице смещений каждому символу шаблона, кроме последнего, ставится в соответствие величина смещения, равная разности длины шаблона и порядкового номера символа, то есть равная номеру символа от конца строки, считая, что первым является предпоследний символ. Для последнего символа и всех символов, отсутствующих в шаблоне, величина смещения устанавливается равной длине шаблона. Например для шаблона string таблица смещений будет иметь вид:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Символ | s | t | r | i | n | g |
| Смещение | 5 | 4 | 3 | 2 | 1 | 6 |

Суть алгоритма заключается в следующем.

1. Искомый текст (шаблон) совмещается со строкой, в которой производится поиск так, чтобы первые символы строки и шаблона совпадали.
2. Сравнение символов строки и шаблона производится слева направо, начиная с последнего символа шаблона.
3. Если происходит несовпадение символов строки и шаблона, то запоминается символ СТРОКИ, который не совпал с символом шаблона.
4. Используя этот символ по таблице смещений определяется на сколько символов можно сместить шаблон для дальнейшего поиска.
5. Шаблон смещается и переход к пункту 2.
6. Если все символы шаблона совпали с соответствующими символами строки, значит искомый текст (шаблон) содержится в строке.

Пример. Требуется определить имеется ли слово «**данные»** в строке **«строка данных передана»** .

Таблица смещений для шаблона «данные» имеет вид

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Символ | д | а | н | ы | х |
| Смещение | 5 | 4 | 2 | 1 | 6 |

Длина слова «данные» составляет 6 символов, поэтому для всех символов, не входящих в шаблон и для последнего символа шаблона смещение равно 6. Символ «н» входит в шаблон дважды, поэтому для него выбирается минимальное смещение, равное 2.

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **с** | **т** | **р** | **о** | **к** | **а** |  | **д** | **а** | **н** | **н** | **ы** | **х** |  | **п** | **е** | **р** | **е** | **д** | **а** | **н** | **а** |
| д | а | н | н | ы | х |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  | д | а | н | н | ы | х |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  | д | а | н | н | ы | х |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  | д | а | н | н | ы | х |  |  |  |  |  |  |  |  |  |

Шаг 1. На первом шаге сравнения не совпали символы «х» в шаблоне и «а» в строке. Так как для символа «а» смещение равно 4, то на следующем шаге сдвигаем шаблон на 4 символа.

Щаг 2. На втором шаге не совпали символы «х» в шаблоне и «н» в строке. Так как для символа «н» смещение равно 2, то на следующем шаге сдвигаем шаблон на 2 символа.

Щаг 3. На третьем шаге не совпали символы «х» в шаблоне и «ы» в строке. Так как для символа «н» смещение равно 1, то на следующем шаге сдвигаем шаблон на 1 символа.

Шаг 4. Все символы шаблона совпали с соответствующими символами строки, значит шаблон найден в строке. Сумма сдвигов на всех шагах равна номеру символа в строке, начиная с которого шаблон входит в строку.

В наихудшем случае трудоемкость рассматриваемого алгоритма O(m+n).

#### Алгоритм Д. Кнута, Д. Мориса и В. Пратта (КМП-поиск)

Приблизительно в 1970 году Д.Кнут, Д.Морис и В.Пратт изобрели алгоритм (КМП-алгоритм), фактически требующий только O(N) сравнений даже в самом плохом случае. Новый алгоритм основывается на том соображении, что после частичного совпадения начальной части слова с соответствующими символами текста фактически известна пройденная часть текста и можно «вычислить» некоторые сведения (на основе самого слова), с помощью которых затем быстро продвинуться по тексту.

Основным отличием КМП-алгоритма от алгоритма прямого поиска является выполнение сдвига слова не на один символ на каждом шаге алгоритма, а на некоторое переменное количество символов. Таким образом, перед тем как осуществлять очередной сдвиг, необходимо определить величину сдвига. Для повышения эффективности алгоритма необходимо, чтобы сдвиг на каждом шаге был бы как можно большим.

Первое (и возможно самое сложно) что нужно сделать -- понять что такое префикс-функция.

Но сначала нужно разобрать что значит вообще префикс и суффикс?  
Если грубо префикс - это любая подстрока, которая начинается с первого символа. Для строки "**happy**", строки **ha**, **happ**, **h** будут являться префиксами.

Суффикс – это любая подстрока, заканчивающаяся последним символом. Например, для строки "**happy**", строки "**appy**", **“py”,** "**y**"  будут суффиксами.

Префикс и суффиксы, которые совпадают со всей строкой целиком нас не интересуют.

Теперь введем понятие **префикс-функции**.

**Префикс-функция** от строки — это зависимость длины наибольшего префикса, равного суффиксу от количества символов подстроки, взятой от начала строки. То есть значения префикс-функции строки - это длины самых больших одинаковых кусков спереди и сзади подстрок из N (N=1,2, …) символов

Например, длины префиксов, равных суффиксам для следующих строк:

**pi("abacaba") = 3**

**pi("**aba") = 1

pi("ab") = 0

Используется префикс-функция от двух аргументов - строки и числа. Строка - это исследуемая строка, а число - указывает длину подстроки, для которой нужно определить наибольшую длину префикса, равного суффиксу.

Например для abacaba, Pi("abacaba",i)  
P("abacaba", 1) = 0 (“a”)  
P("abacaba", 2) = 0 ("ab“)  
P("abacaba", 3) = 1 ("aba“)  
P("abacaba", 4) = 0 ("abac“)  
P("abacaba", 5) = 1 ("abaca“)   
P("abacaba", 6) = 2 ("abacab“)  
P("abacaba", 7) = 3 ("abacaba“)

получаем P("abacaba") =0010123

Пусть требуется найти подстроку t в строке s. С помощью префикс-функции это делается тривиально: находится префикс-функцию от строки t+#+s (решётка обозначает символ, гарантированно не встречающийся ни в одной из строк). Если эта префикс-функция содержит значения равные длине t, значит t входит в s . А именно, пусть π[i]=|t||. Значит s[i−|t|−1] – последний символ вхождения t в s.

Пример - нахождение образца ааbаа в строке ааbааbааааbааbаааb.

1. Записываем образец, затем символ, гарантированно отсутствующий в образце и в строке.

Например символ @ или # или &. Справа от этого символа записываем строку, в которой производится поиск.

1. Вычисляем все префикс функции полученной строки и помещаем их в массив.

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| a | a | b | a | a | @ | a | a | b | a | **a** | b | a | **a** | a | a | b | a | **a** | b | а | **a** | a | b |
| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 | 24 |
| 0 | 1 | 0 | 1 | 2 | 0 | 1 | 2 | 3 | 4 | 5 | 3 | 4 | 5 | 2 | 2 | 3 | 4 | 5 | 3 | 4 | 5 | 2 | 3 |

Символ **'@**' играет роль разделителя, его заведомо нет ни в образце, ни в строке поиска. Посмотрим на позиции 11, 14, 19, 22 массива. Значения в массиве равны 5, это означает, что суффикс длиной 5 (фрагмент строки поиска) совпадает с 5 символами префикса. А 5 символов префикса — это и есть образец для поиска! Алгоритм поиска получатся такой — склеиваем с помощью разделителя образец и строку поиска, передаем «склейку» префиксной функции и потом ищем в массиве элементы, равные длине образца, *Можно заметить, что значений больше длины образца не будет из-за символа-разделителя, а значения, равные длине образца могут появиться только в позициях, соответствующих исходной строке поиска.* Склеенная строка имеет длину <длина образца>+<длина строки>, поэтому время расчета оценивается как О(<длина образца>+<длина строки>).

**Задания на лабораторную работу**.

1. Изобразить в виде таблиц строку и шаблон со всеми сдвигами при поиске для всех алгоритмов поиска подстроки.

2. Создать программу, выполняющую следующие действия.

2.1. Ввод строки, содержащей фамилию, имя и отчество студента.

2.2. Ввод шаблона для поиска в соответствии с вариантом.

2.3. Подсчет числа сравнений символов при прямом поиске, при БМХ, КМР поиске.

Варианты задания

|  |  |
| --- | --- |
| Вариант | Шаблон |
| 1 | Имя студента |
| 2 | Имя отца |
| 3 | Пять первых символов фамилии |
| 4 | Пять последних символов фамилии |
| 5 | Первые четыре символа имени |
| 6 | Первые пять символов отчества |
| 7 | Первые четыре символа отчества |
| 8 | Четыре последних символа отчества |
| 9 | Первые пять символов имени |
| 10 | Последние пять символов отчества |
| 11 | Последние пять символов имени |
| 12 | Последние четыре символа фамилии |
| 13 | Средние четыре символа имени |
| 14 | Средние пять символов отчества |
| 15 | Средние четыре символа отчества |

**Требования к отчету**.

Отчет по лабораторной работе должен соответствовать следующей структуре.

* Титульный лист.
* Словесная постановка задачи.
* Таблицы, содержащие строку и шаблон со всеми сдвигами при поиске для всех алгоритмов поиска подстроки.
* Листинг программы.
* Результат подсчета числа сравнений символов.
* Выводы по лабораторной работе.

**Контрольные вопросы**

1. В чем заключается сущность прямого поиска текста?
2. В чем заключается сущность БМХ- поиска текста?
3. Как строится таблица смещений для БМХ поиска?
4. В чем заключается сущность КМП- поиска текста?
5. Что такое префикс-функция?