ПОИСК ПОДСТРОК

Поиск точно заданной подстроки в строке

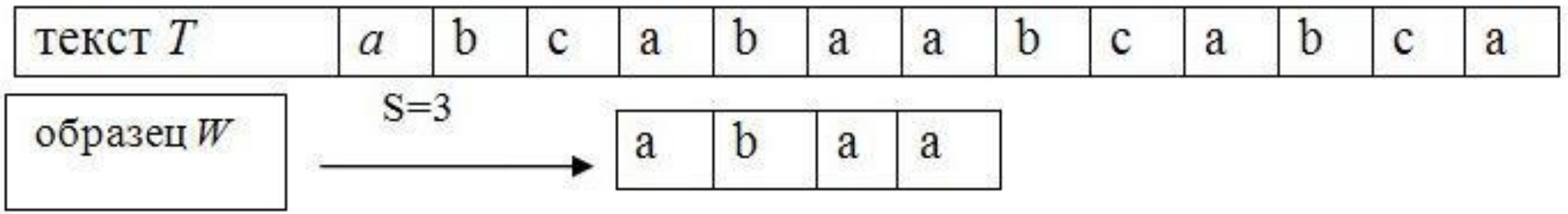
В задачах поиска традиционно принято обозначать шаблон поиска как needle (англ.В задачах поиска традиционно принято обозначать шаблон поиска
как needle (англ. «иголка»), а строку,

Поиск строки формально определяется следующим образом. Пусть задан массив T из N элементов и массив W из M элементов, причем $0<M\leq N$.

Поиск строки обнаруживает первое вхождение W в T, результатом будем считать индекс i, указывающий на первое с начала строки совпадение с шаблоном.

Пример:

Требуется найти все вхождения образца W = abaa в текст T=abcabaabcabca



Образец входит в текст только один раз, со сдвигом S=3, индекс i=4.

Алгоритм прямого поиска

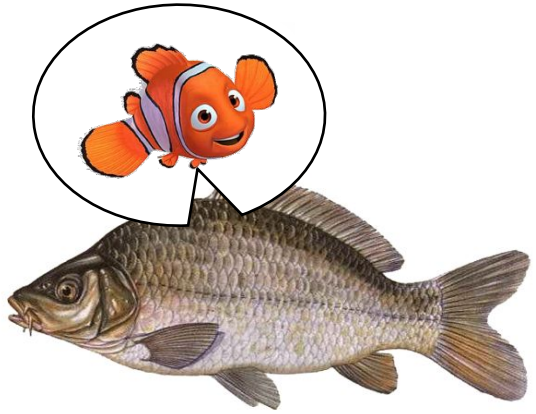
- Идея алгоритма:
  1. I=1,
  2. сравнить I-й символ массива T с первым символом массива W,
  3. совпадение → сравнить вторые символы и так далее,
  4. несовпадение → I:=I+1 и переход на пункт 2,

- Условие окончания алгоритма:
  1. подряд M сравнений удачны,
  2. I+M>N, то есть слово не найдено.

Недостатки алгоритма:

1. Высокая сложность — O(N*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.

```
i = –1; //n – длина строки m-подстроки
do  {
i++;
j = 0;
while((j < m) && (haystack[i + j] == needle[j]))
j++;
}
while ((j != m) && (i < n – m));
```



Алгоритм Рабина-Карпа (РК-поиск)

ИДЕЯ 

Пусть алфавит $D=\{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$, то есть каждый символ в алфавите есть d–ичная цифра, где $d=|D|$.

На примере русского алфавита:

А	Б	В	Г	Д	Е	Ж	З	И	Й	К	Л	М	Н	О	П	Р
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
С	Т	У	Ф	Х	Ц	Ч	Ш	Щ	Ъ	Ы	Ь	Э	Ю	Я	_	.
17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33

ЕГОР = 5 3 14 16 = $5*34^3+ 3*34^2 +14*34+ 16 =$ 200480

Пусть алфавит D={0, 1, 2, 3, 4, 5, 6, 7, 8, 9}

Пример. Пусть шаблон имеет вид W = 3 1 4 1 5
Вычисляем значения чисел из окна длины |W|=5 по mod q, q — простое число.

T	=	2	3	5	9	0	2	3	1	4	1	5	2	6	7	3	9	9	2	1
Значения по mod 13 =						8	9	3	11	0	1	7	8	4	5	10	11	7	9	11

23590(mod 13)=8, 35902(mod 13)=9, 59023(mod 13)=3 …
31415(mod 13)=7

T	=	2	3	5	9	0	2	3	1	4	1	5	2	6	7	3	9	9	2	1
Значения по mod 13 =						8	9	3	11	0	1	7	8	4	5	10	11	7	9	11

k1=31415(mod 13)=7 – вхождение подстроки,

k2=67399(mod 13)=7 – холостое срабатывание.

Из равенства ki= kj (mod q) не следует, что ki= kj (например, 31415=67399(mod 13), но это не значит, что 31415=67399). Если ki= kj (mod q), то ещё надо проверить, совпадают ли строки W[1…m] и T[s+1…s+m] на самом деле.

Трудоемкость

Если простое число q достаточно велико, то дополнительные затраты на анализ холостых срабатываний будут невелики.
В худшем случае время работы алгоритма РК — Θ((N-M+1)*M), в среднем же он работает достаточно быстро – за время O (N+M).

Очевидно, что количество холостых срабатываний k является функцией от величины простого числа q (если функция обработки образца mod q) и, в общем случае, от вида функции для обработки образца W и текста T.

Пример:

Сколько холостых срабатываний k сделает алгоритм РК, если q= 11, 13, 17. Пусть W={2 6}

T=	3	1	4	1	5	9	**2**	**6**	5	3	5	8	9	7	9	3
mod 11		9	3	8	***4***	***4***	***4***	***4***	10	9	2	3	1	9	2	5
mod 13		5	1	2	2	7	1	***0***	***0***	1	9	6	1	6	1	2
mod 17		14	14	7	15	8	7	***9***	*14*	2	1	7	4	12	11	8

26 mod 11=4 → k =3 холостых срабатывания,
26 mod 13=0 → k =1 холостое срабатывание,
26 mod 17=9 → k =0 холостых срабатываний.

```
int RabinKarp(char* haystack, char* needle)
    hash_needle = hash(needle[1..m])
    hash_haystack = hash(haystack [1..m])


        for i=1 to (n-m+1) {
      if hash_haystack = hash_needle
        if haystack[i..i+m-1] = needle
        return i
      hash_haystack = hash(haystack[i+1..i+m])   }
 return -1 // не найдено 🙁
```