Алгоритмы поиска подстроки в строке

Определения

- **Алфавит** – конечное множество символов.
- **Строка (слово)** – последовательность символов из некоторого алфавита. Например **X=x[1]x[2]...x[n]** – строка длиной **n**, где **x[i]** – i-ый символ строки **X**, принадлежащий алфавиту.
- **Пустая строка** – строка, не содержащая ни одного символа.
- **Длина строки** – количество символов в строке.
- **Подстрока** – 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.
- **Префикс** – 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.
- **Суффикс** – 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.

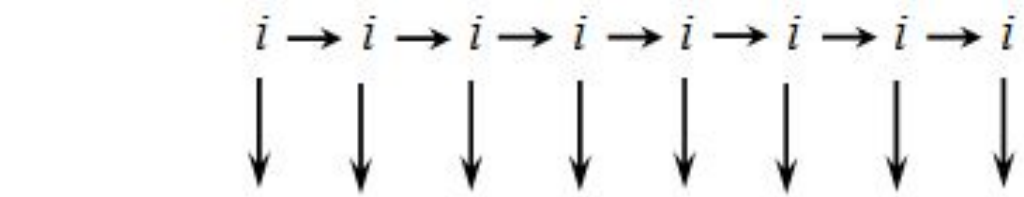
Задача поиска подстроки в строке

- Пусть задана строка, состоящая из некоторого количества символов.
- Проверим, входит ли заданная подстрока в данную строку.
- Если входит, то найдем номер символа строки (первое вхождение заданной подстроки в исходной строке или все вхождения подстроки в строку).

Прямой поиск

- Основная идея алгоритма прямым поиском **заключается в посимвольном сравнении строки с подстрокой.**
- В начальный момент происходит сравнение первого символа строки с первым символом подстроки, 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.
-

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.



Строка	A	B	C	A	B	C	A	A	B	C	A	B	D
Подстрока	**A**	**B**	**C**	**A**	**B**	**D**							
		A	B	C	A	B	D						
			A	B	C	A	B	D					
				A	**B**	**C**	**A**	**B**	D				
					A	B	C	A	B	D			
						A	B	C	A	B	D		
							A	**B**	C	A	B	D	
								A	**B**	**C**	**A**	**B**	**D**

```
Procedure Solve;
{P, T – глобальные величины типа String}
  Var i,j:Integer;
  Begin
    For i:=1 To n-m+1 Do Begin
      j:=1;
      While (j<=m) And (P[j]=T[i+j-1]) Do j:=j+1;
      If j=m+1 Then WriteLn('найдено вхождение P в T,
                             начиная с позиции ', i);
    End;
  End;
```

Оценим время работы этого алгоритма в количестве операций сравнения. Оно очевидно и имеет значение $O(n \cdot m)$. Такая оценка достигается при $n - m + 1$ совпадениях, т. е. когда и P, и T состоят из одного символа. При значениях $m > 10^3$, $n > 10^9$ время работы такого алгориитма становится неприемлемым для многих приложений.

- Алгоритм был открыт Д. Кнутом и В. Праттом и, независимо от них, Д. Моррисом. Результаты своей работы они совместно опубликовали в 1977 году. Алгоритм Кнута, Морриса и Пратта (КМП -алгоритм) является алгоритмом, который фактически требует только **O(n)** сравнений даже в самом худшем случае.
- Рассматриваемый алгоритм основывается на том, что после частичного совпадения начальной части подстроки с соответствующими символами строки фактически известна пройденная часть строки и можно, вычислить некоторые сведения, с помощью которых затем быстро продвинуться по строке.
- Основным отличием алгоритма Кнута, Морриса и Пратта от алгоритма прямого поиска заключается в том, что **сдвиг подстроки выполняется не на один символ на каждом шаге алгоритма, а на некоторое переменное количество символов.**
- Следовательно, 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. Если для произвольной подстроки определить все ее начала, одновременно являющиеся ее концами, и выбрать из них самую длинную (не считая, конечно, саму строку), то такую процедуру принято называть префикс -функцией.
- В реализации алгоритма Кнута, Морриса и Пратта используется предобработка искомой подстроки, которая заключается в создании префикс -функции на ее основе.

Определение

Гранью (*border*, *verge*, *brink*) *br* строки S называется любой собственный префикс этой строки, равный суффиксу S.

Строка S = `abaababaabaab` имеет две грани (не пустые) — `ab` и `abaab`. Строка S = `abaabaab` также имеет две грани — `ab` и `abaab`, но вторая грань — перекрывающаяся. Строка длины n из повторяющегося символа, например `aaaaaaaa` (или a^8), имеет $n - 1$ грань. Для $S = a^8$ это грани: `a`, `aa`, `aaa`, `aaaa`, `aaaaa`, `aaaaaa` и `aaaaaaa`.

Понятие «собственный префикс» исключает грань, совпадающую с самой строкой.

Длина грани — это количество символов в ней.

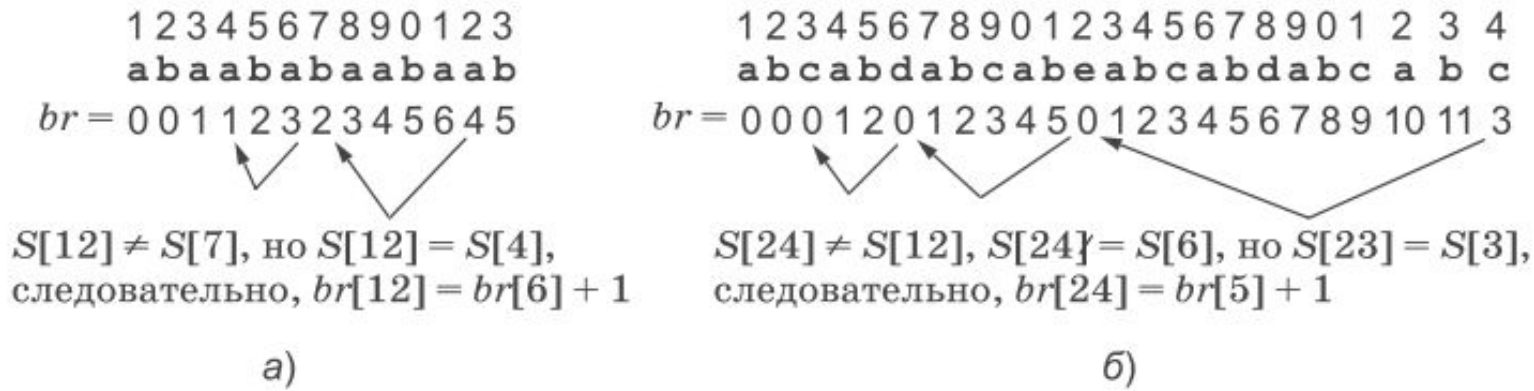
Естественным обобщением понятия «грань» является понятие «*наибольшей грани*» — это наибольший (по количеству символов) собственный префикс строки, равный ее суффиксу.

Примеры

В табл. 1.1 приведены примеры вычисления массива граней *br* для различных строк *S*.

Таблица 1.1

№	*S*	*Br*
1	aaaaaa	0, 1, 2, 3, 4, 5
2	abcdef	0, 0, 0, 0, 0, 0
3	abaababaabaab	0, 0, 1, 1, 2, 3, 2, 3, 4, 5, 6, 4, 5
4	abcabcabcabc=(abc)^4	0, 0, 0, 1, 2, 3, 4, 5, 6, 7, 8, 9
5	abcabdabcabeabcabd abcabc	0, 0, 0, 1, 2, 0, 1, 2, 3, 4, 5, 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 3



а) *б*)

Рис. 1.2. Примеры вычислений значений *br* при $S[i+1] \neq S[br[i]+1]$

```
Procedure MaxBorderArray(S:String);
{Массив br - глобальный}
  Var i,n,t:Word;
  Begin
    n:=Length(S);
    br[1]:=0;
    For i:=1 To n-1 Do Begin
      t:=br[i];
      While (t>0) And (S[i+1]<>S[t+1]) Do t:=br[t];
      If S[i+1]=S[t+1] Then br[i+1]:=t+1
        Else br[i+1]:=0;
    End;
  End;
```

Как массив br помогает сдвинуть подстроку более, чем на 1 символ?

Таблица 2.1

i	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
P	a	b	c	a	e	a	b	c	a	b	c	a					
br	**0**	**0**	**0**	**1**	**0**	**1**	**2**	**3**	**4**	**2**	**3**	**4**					
						a	b	c	a	e	a	b	c	a	b	c	a

Предположим, что произошло несовпадение символов при $i = 9$ и при некоторой позиции k текста T. Значение $br[8]$ говорит о наличии суффикса длиной 3, совпадающего с префиксом P. Значит, чтобы этот префикс совпал с суффиксом, следует сдвинуть P на $(8 - 3) = 5$ позиций. При этом гарантируется совпадение $br[8]$ (т. е. трех) символов P с соответствующими символами T, так что следующее сравнение следует выполнять между символами $T[k]$ и $P[br[8] + 1]$.

Пример

Пусть требуется в T=ababcxabdabcxabcxabcde найти вхождения P=abcxabcde. Массив граней br=(0, 0, 0, 0, 1, 2, 3, 0, 0). Процесс поиска представлен в табл. 2.2.

Таблица 2.2

i	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**	**16**	**17**	**18**	**19**	**20**	**21**	**22**
T	a	b	a	b	c	x	a	b	d	a	b	c	x	a	b	c	x	a	b	c	d	e
P	a	b	c	x	a	b	c	d	e													
			a	b	c	x	a	b	c	d	e											
							a	**b**	**c**	**x**	**a**	**b**	**c**	**d**	**e**							
									a	b	c	x	a	b	c	d	e					
										a	b	c	x	a	b	c	d	e				
														a	b	c	x	a	b	c	d	e

Формализованная запись алгоритма Д. Кнута – Дж. Морриса – В. Пратта поиска P в T имеет вид:

```
Procedure KMP(T,P:String);
  Var n, m, i, q:Word;
  Begin
    n:=Length(T);
    m:=Length(P);
    MaxBorderArray(P);
    {Вычисляем значения элементов массива граней br
     (п. 1.2)}
    q:=0;
    {Индекс сравниваемого символа образца P}
    For i:=1 To n Do Begin
    {Индекс символа текста T}
      While (q>0) And (P[q+1]<>T[i]) Do q:=br[q];
      {Имитация сдвига}
      If P[q+1]=T[i] Then q:=q+1;
      If q=m Then Begin
        WriteLn('Найдено вхождение P в T с позиции ',
                i-m+1);
        q:=br[m];
      End;
    End;
  End;
```

Алгоритм Д. Кнута – Дж. Морриса – В. Пратта выполняется за время $O(n)$; при этом время, необходимое для предварительной обработки образца P (для формирования массива граней), равно $O(m)$. Тогда общее время — $O(n+m)$.

- Он был разработан Р.Бойером и Д. Муром в 1977 году. Преимущество этого алгоритма в том, сравнение подстроки с исходной строкой осуществляется не во всех позициях – часть проверок пропускаются как заведомо не дающие результата.

Существует множество вариаций алгоритма Бойера и Мура, рассмотрим одну из простых:

- Первоначально строится таблица смещений для искомой подстроки.
- Далее идет совмещение начала строки и подстроки и начинается проверка с последнего символа подстроки.
- Если последний символ подстроки и соответствующий ему при наложении символ строки не совпадают, подстрока сдвигается относительно строки на величину, полученную из таблицы смещений, и снова проводится сравнение, начиная с последнего символа подстроки.
- Если же символы совпадают, производится сравнение предпоследнего символа подстроки и т.д. Если все символы подстроки совпали с наложенными символами строки, значит, найдена подстрока и поиск окончен.
- Если же какой-то (не последний) символ подстроки не совпадает с соответствующим символом строки, далее производим сдвиг подстроки вправо (? На сколько символов?) и снова начинаем проверку с последнего символа.

		1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21
	T	a	x	b	c	f	a	b	a	x	b	a	d	a	b	a	x	a	x	a	a	d
1	***P***	a	b	a	x	a																
2							a	b	a	x	a											
3										a	b	a	x	a								
4														a	b	a	x	a				
5															a	b	a	x	a			
6																a	b	a	x	a		
7																	a	b	a	x	a	
8																		a	b	a	x	a

		1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23
	T	a	b	a	b	c	x	c	d	e	d	e	a	x	a	a	b	c	x	a	b	c	d	e
1	***P***	a	b	c	x	a	b	c	d	e														
2				a	b	c	x	a	b	c	d	e												
3					a	b	c	x	a	b	c	d	e											
4									a	b	c	x	a	b	c	d	e							
5												a	b	c	x	a	b	c	d	e				
6																a	b	c	x	a	b	c	d	e

Перейдем теперь к формальной реализации описанной эвристики. Для фиксации самого правого вхождения символов алфавита (предположим, что это буквы латинского алфавита от a до z) в *P* нам необходим массив — обозначим его как *bs*. Формирование его значений может осуществляться следующим образом.

```
Procedure ShiftBadSymbol;
{Массив bs (bs:Array['a'..'z'] Of Integer),
так же как образец P и количество символов m в P, -
глобальные переменные}
  Var i:Integer;
      q:Char;
  Begin
    For q:='a' To 'z' Do bs[q]:=0;
     {Начальная инициализация}
    For i:=1 To m Do bs[P[i]]:=i;
  End;
```

```
Procedure BadSymbol;
  Var i,j:Word;
  Begin
    i:=1;
    While i<=(n-m+1) Do Begin
      j:=m;
      While (j>=1) And (P[j]=T[i+j-1]) Do j:=j-1;
        If j=0 Then Begin
          WriteLn('образец ',P,' входит в ',T,
                  ' с позиции ',i);
          i:=i+1;
        End
        Else i:=i+Max(1,j-bs[t[i+j-1]]);
        {Функция Max - нахождение максимального
         из двух целых чисел}
    End;
  End;
```