Классические алгоритмы решения задачи точного совпадения

Примитивный алгоритм имеет временную сложность $O(n \cdot m)$.

Суть всех последних достижений (конец 20 века и по настоящее время) – разработка алгоритмов с временной сложностью $O(n+m)$.

Основа, ключевой момент, предварительный анализ исходных данных (препроцессинг) с целью выявления неких закономерностей в них, позволяющих решать основную задачу за указанное время. Естественно, что алгоритм препроцессинга должен иметь линейную временную оценку.

Грани строки

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* (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» – наибольший (по количеству символов) собственный префикс строки, равный её суффиксу.

Простым алгоритмом вычисления наибольшей грани строки S является последовательная проверка совпадения префиксов S[1], S[1..2], S[1..3], …, S[1..n-1] с соответствующими суффиксами S[n], S[n-1..n], S[n-2..n], …, S[2..n].

```
функция maxBorder(S)
 n←длина(S)
 br←0
 для i от 1 до n-1: {Цикл по длине грани.}
  j←n-i+1
  пока (j≤n) и (S[i+j-n]=S[j]):j←j+1
  если j=n+1:br←i
 вернуть br
```

Временная сложность алгоритма $O(n^2)$.

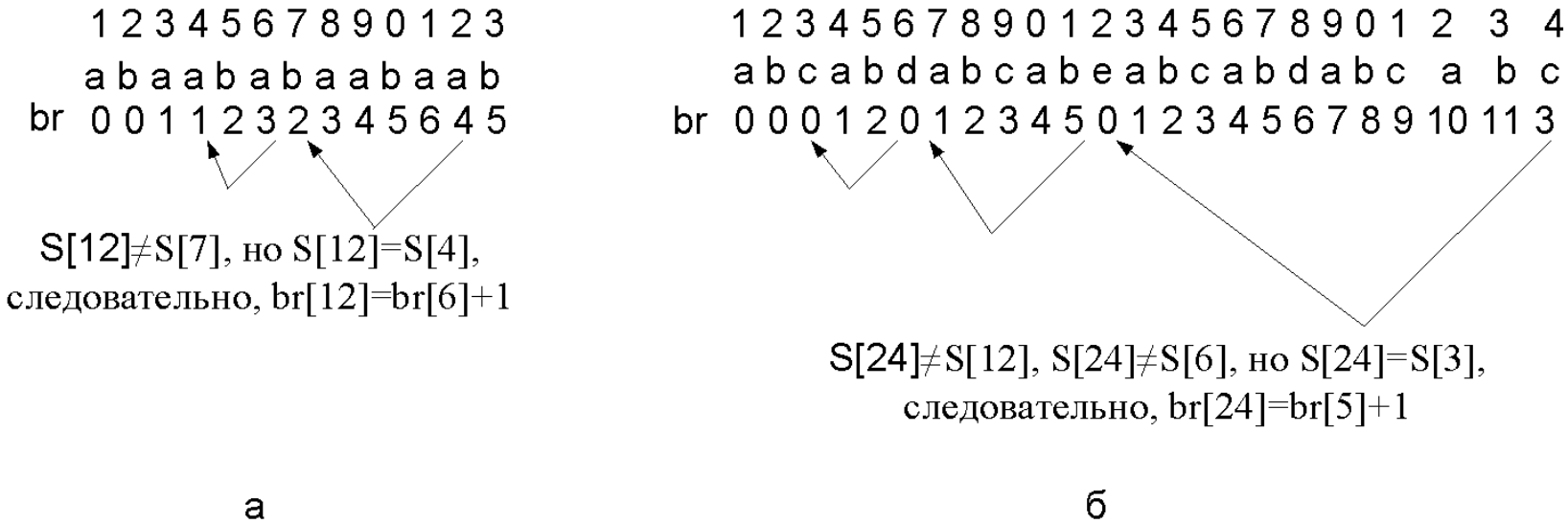
Вычисление значений наибольших граней для всех подстрок S[1..i] (i=1..n) – в массив br[1..n]. Значение br[1] равно 0, ибо подстрока S[1] не имеет собственных подстрок. Последовательное применение предыдущей логики к каждой подстроке приводит к алгоритму с временной сложностью $O(n^3)$.

```
процедура maxBorderArray(S)
 n←длина(S)
 br[1]←0
 для i от 2 до n:
  br[i]←maxBorder(подстрока S(1,i))
```

Примеры br для различных строк S.

№	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	abcabdabcabeabcabdabcabc	0, 0, 0, 1, 2, 0, 1, 2, 3, 4, 5, 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 3

Какие наблюдения можно сделать на основании этих данных? Во-первых, $br[i+1] \le br[i]+1$ для любых i от 1 до n-1 и если значения элементов br возрастают, то они возрастают с шагом 1. Когда $br[i+1] = br[i]+1$? Ответ однозначен – при $S[i+1] = S[br[i]+1]$. Если символ S в позиции i+1 совпадает с символом, следующим за максимальным префиксом S[1..i] совпадающим с суффиксом S[1..i], то наибольшая грань br[i+1] для S[1..i+1] просто увеличивается на 1. А если $S[i+1] \neq S[br[i]+1]$? Ситуация для строки S[1..12] третьего примера поясняется на рис. 1а, а для строки S[1..24] пятого примера из табл. 1.1 – на рис. 1б.



а б

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

Временные параметры алгоритма. Цикл For выполняется n-1 раз. Количество шагов вложенного цикла While различно. Время выполнения алгоритма пропорционально общему количеству присваиваний значений переменной t. Оно равно n-1 (в цикле For), плюс число этих операций внутри цикла While. В цикле While происходит уменьшение значения переменной t. На каждой же итерации For значение t, а оно всегда неотрицательное, или остается равным нулю, или увеличивается на единицу. Таким образом, количество увеличений пропорционально (n-1), но общее количество уменьшений в цикле While, а оно всегда осуществляется не менее, чем на 1, не может превосходить количества увеличений. Следовательно, t изменяется внутри цикла While не более (n-1) раза и полное число присваиваний t ограничено сверху величиной 2(n-1)=O(n). Итак, массив br для S формируется не за время $O(n^3)$, как было рассмотрено ранее, а за время O(n).

Таким образом, в массиве br фиксируются грани всех подстрок S[1..i], i=1…n. Другими словами, вычисляются грани всех префиксов строки S. Аналогичную задачу можно решить и для суффиксов строки S, например, в массиве bw сформировать грани S[i..n], i=1..n. На рис. 2 показано, в чем заключается отличие понятий.

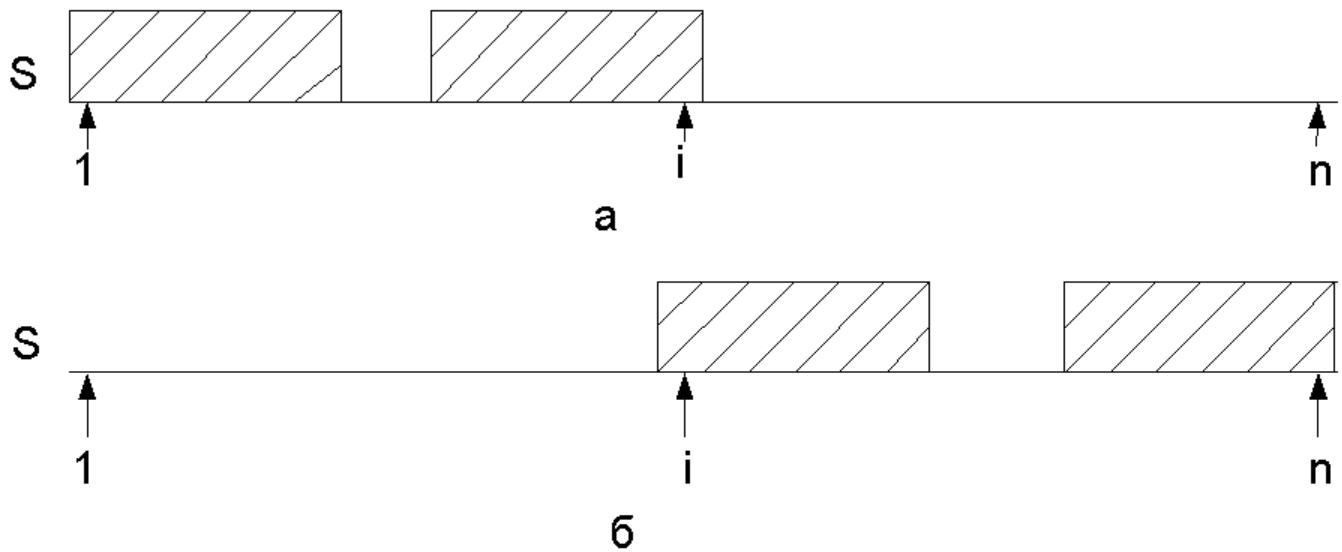


Рис. 2. Грани: а) грань префикса S[1..i] – br[i]; б) грань суффикса S[i..n] – bw[i]

Пример.

	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9	0	1
S	*a*	*b*	*a*	*a*	*b*	*a*	*b*	*a*	*a*	*b*	*a*	*a*	*b*	*a*	*b*	*a*	*a*	*b*	*a*	*b*	*a*
br	0	0	1	1	2	3	2	3	4	5	6	4	5	6	7	8	9	10	11	7	8
bw	8	7	6	5	4	3	2	1	8	7	6	5	4	3	2	1	3	2	1	0	0

Логика формирования значений элементов массива bw аналогична той, что рассмотрена в процедуре MaxBorderArray.

```
процедура borderRigth(S)
 n←длина(S)
 bw[n]←0
 для i от n до 2:
  t←bw[i]
  пока (t>0) и (S[i-1]≠S[n-t]:t←bw[n-t]
  если S[i-1]=S[n-t]:bw[i-1]←t+1
   иначе: bw[i-1]←0
```

Алгоритм Д. Кнута – Дж. Морриса – В. Пратта

Дан образец (образцы). На обработку поступает текст. Требуется выявлять вхождение образцов в текст.

Пусть требуется в *T=ababcxabdabcxabcxabcde* найти вхождения *P=abcxabcde*. Какую информацию о P необходимо иметь, чтобы выполнять «сдвиги» так, как это показано в таблице?

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*

Массив граней br=(0, 0, 0, 0, 1, 2, 3, 0, 0).

Пусть вычислен массив граней P. Другими словами для каждой позиции i в P определена br[i] – длина наибольшего собственного суффикса P[1..i], совпадающего с префиксом P.

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 несравнение символов при i=9 и некоторой позиции k текста T. Значение br[8] говорит о наличии суффикса, длиной 3, совпадающего с префиксом P. И для того, чтобы этот префикс совпал с суффиксом, следует сдвинуть P на 8–3=5 позиций. Гарантируется совпадение br[8] (трех) символов P с соответствующими символами T и следующее сравнение следует выполнять между символами T[k] и P[br[8]+1]. В этом вся суть алгоритма – за счет знания структуры образца P, анализ которой выполняется за линейное время, сдвигать при поиске вхождения P в T более чем на одну позицию. При этом символ T[k] участвовал в сравнении как минимум два раза.

Алгоритм Кнута – Морриса – Пратта

```
процедура solve(T,P)
 n←длина(T)
 m←длина(P)
 maxBorderArray(P); {Вычисляем
значения элементов массива граней
br.}
 q←0
 для i от 1 до n:
  пока (q>0) и (P[q+1]<>T[i]):q←br[q]
  если P[q+1]=T[i]:q←q+1
  если q=m:
    вывод(вхождение P в T с позиции
i-m+1)
    q←br[m]
```

В первом примере «жирным» шрифтом выделен случай, дающий основания для дальнейшего усовершенствования алгоритма. На этот момент времени сравнивается символ T[9] с P[7] (i=9, q=6). Работает цикл While (формализованная запись) и значение q становится равным 2 (br[6]=2). Происходит сравнение T[9] с P[3]. Результат заведомо известен, ибо P[3]=P[7], и уже было несовпадение символа P[7] с T[9]. Затем q присваивается значение 0 (br[2]=0) и осуществляется сравнение T[9] с P[1], но это уже следующая строка таблицы. Как исключить эти лишние сдвиги?

Уточним понятие грани. Для каждой позиции i строки S определим brs[i] как длину наибольшего собственного суффикса S[1..i], совпадающего с префиксом S, и такого, что S[i+1]≠S[brs[i]+1]. Другими словами, следующий символ за префиксом, равным суффиксу, не должен совпадать с символом S[i+1].

Три примера вычисления значений уточненных граней. Естественно, что значения элементов массива brs, получены на основе вычисленного ранее массива br.

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21
S	a	b	c	x	a	b	c	d	e												
br	0	0	0	0	1	2	3	0	0												
brs	0	0	0	0	0	0	3	0	0												
S	a	b	a	a	b	a	b	a	a	b	a	a	b								
br	0	0	1	1	2	3	2	3	4	5	6	4	5								
brs	0	0	1	0	0	3	0	1	0	0	6	0	5								
S	a	b	a	a	b	a	b	a	a	b	a	a	b	a	b	a	a	b	a	b	a
br	0	0	1	1	2	3	2	3	4	5	6	4	5	6	7	8	9	10	11	7	8
brs	0	0	1	0	0	3	0	1	0	0	6	0	0	3	0	1	0	0	11	0	8

Примечание. Значение brs[n] получено при условии, что к S приписывается символ, которого нет в алфавите.