Z-функция строки

Mike Mirzayanov

УНИВЕРСИТЕТ ИТМО

CODEFORCES

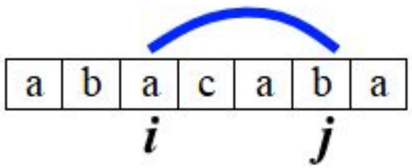
Сделано для раздела https://codeforces.com/edu/courses

Определения: строка, подстрока

Строка – это конечная (возможно, пустая) последовательность символов алфавита. Длина строки s обозначается как $|s|$. Символы будем считать пронумерованными от 0 до $|s|$-1.

Примеры строк: s_1=«`010101`», s_2=«`abacaba`», s_3=[31,34,41] (символы – целые числа), s_4=«».

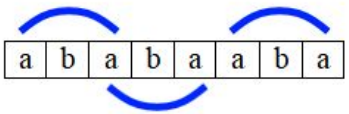
Подстрока – последовательность подряд идущих символов строки.
Например, все подстроки строки «`apple`»: «`apple`», «`appl`», «`pple`», «`app`», «`ppl`», «`ple`», «`ap`», «`pp`», «`pl`», «`le`», «`a`», «`p`», «`l`», «`e`», «».



Обозначим $s[i \ldots j]$ подстроку, которая начинается в i и заканчивается в j. Её длина равна j-i+1.

Определения: вхождение, префикс, суффикс

Вхождением строки p в строку t будем называть такую пару индексов (i, j), что $p = t[i \ldots j]$. Например, в строке t=«ababaaba» всего 3 вхождения строки p=«aba».



Префиксом строки s называется подстрока вида $s[0 \ldots i]$. Всего у строки $|s|+1$ префиксов (пустой и полный тоже учитываются). Например, если s=«abc», то префиксы это – «», «a», «ab», «abc».

Суффиксом строки s называется подстрока вида $s[j \ldots |s|-1]$. Всего у строки $|s|+1$ суффиксов (пустой и полный тоже учитываются). Например, если s= «abc», то суффиксы это – «», «c», «bc», «abc».

Поиск подстроки в строке (задача точного поиска)

Задана строка t (текст) и строка p (образец). Найдите все вхождения строки p в строку t.

Пример: t=«abacababa», p=«aba». Ответ содержит 3 вхождения:

- **aba**cababa: (0, 2)
- abac**aba**ba: (4, 6)
- abacab**aba**: (6, 8)

Если $n=|t|$ и $m=|p|$, то наивный алгоритм работает за время $O(nm)$.

Поиск подстроки в строке: наивный алгоритм

```
for i := 0 .. |t| - |p|:
    mismatch = false
    for k := 0 .. |p| - 1:
        if p[k] != t[i + k]:
            mismatch = true
            break
    if not mismatch:
        i is an occurrence
```

Поиск подстроки в строке (задача точного поиска)

Есть два основных подхода:

- препроцессинг образца (z-, префикс-функции)
- препроцессинг текста (суффиксные дерево, массив, автомат)

Z-функция: определение

Для заданной строки строки s=$s_0s_1\ldots s_{n\text{-}1}$ её **z-функцией** является массив z длины n, проиндексированный от 0 до n-1, что $z[i]$ – это длина наидлиннейшего общего префикса всей строки s и её суффикса $s[i \ldots n$-1].

Для i=0 обычно $z[0] = 0$ (иногда удобно считать, что $z[0] = n$).

Пример. Пусть s=«abacaba»

0	1	2	3	4	5	6
a	b	a	c	a	b	a

- $z[0] = 0$ – по определению
- $z[1] = 0$ – ищем длину наидл. общего префикса у abacaba и bacaba
- $z[2] = 1$ – ищем ДНОП у abacaba и acaba
- $z[3] = 0$ – ищем ДНОП у abacaba и caba
- $z[4] = 3$ – ищем ДНОП у abacaba и aba
- $z[5] = 0$ – ищем ДНОП у abacaba и ba
- $z[6] = 1$ – ищем ДНОП у abacaba и a

Z-функция: примеры

Для заданной строки строки $s=s_0s_1\ldots s_{n-1}$ её z**-функцией** является массив z длины n, проиндексированный от 0 до n-1, что $z[i]$ – это длина наидлиннейшего общего префикса всей строки s и её суффикса $s[i \ldots n-1]$. Для i=0 обычно $z[0] = 0$ (иногда удобно считать, что $z[0] = n$).

Примеры

- s=«abacaba», z=[0, 0, 1, 0, 3, 0, 1]
- s=«aaaaaaaa», z=[0, 7, 6, 5, 4, 3, 2, 1]
- s=«abababab», z=[0, 0, 6, 0, 4, 0, 2, 0]

Z-функция: упражнения

Для заданной строки строки $s=s_0s_1\ldots s_{n\text{-}1}$ её z**-функцией** является массив z длины n, проиндексированный от 0 до n-1, что $z[i]$ – это длина наидлиннейшего общего префикса всей строки s и её суффикса $s[i \ldots n\text{-}1]$. Для i=0 обычно $z[0] = 0$ (иногда удобно считать, что $z[0] = n$).

Упражнения

- найдите z-функцию строки s=«abaababa»;
- найдите z-функцию строки s=«baababaab».

Z-функция: упражнения (ответы)

Для заданной строки строки $s=s_0s_1 \ldots s_{n-1}$ её z**-функцией** является массив z длины n, проиндексированный от 0 до n-1, что $z[i]$ – это длина наидлиннейшего общего префикса всей строки s и её суффикса $s[i \ldots n-1]$. Для i=0 принято, что $z[0]$ = 0 (иногда удобно считать, что $z[0]$ = n).

Упражнения (ответы)

- s=«abaababa», z=[0, 0, 1, 3, 0, 3, 0, 1]
- s=«baababaab», z=[0, 0, 0, 2, 0, 4, 0, 0, 1]

Z-функция: наивный алгоритм

Разработайте и реализуйте простой наивный (квадратичный) алгоритм для нахождения z-функции строки.

Z-функция: наивный алгоритм

Разработайте и реализуйте простой наивный (квадратичный) алгоритм для нахождения z-функции строки.

Простая реализация:

```
for i = 1..n-1:
    while z[i] + i < n && s[z[i] + i] == s[z[i]]:
        z[i]++
```

Z-функция: использования для поиска вхождений

Предположим, что умеем находить z-функцию эффективно. Тогда и задачу о нахождении подстроки в строке можно решить эффективно.

Пусть дан текст t и образец p. Требуется найти все вхождения образца p в текст t.

Построим новую строку s следующим образом: $s := p + \text{‘\$’} + t$, где ‘\$’ – это такой символ, которого нет ни в t ни в p.

$t =$	a	a	b	a	a	b	a	b	a	a

$p =$	a	b	a	a

$s =$	a	b	a	a	\$	a	a	b	a	a	b	a	b	a	a

Z-функция: использования для поиска вхождений

Пусть дан текст t и образец p. Требуется найти все вхождения образца p в текст t.

$t =$

a	a	b	a	a	b	a	b	a	a

$p =$

a	b	a	a

$s =$

a	b	a	a	$	a	a	b	a	a	b	a	b	a	a

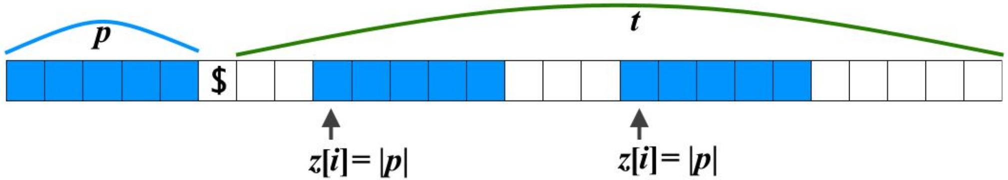
Если $z[i]=|p|$, то подстрока длины $|p|$, которая начинается в i совпадает с первыми $|p|$ символами s, то есть она равна p. Следовательно, есть вхождение p в t с позиции i-$|p|$-1. Обратное тоже верно.

Найдем z-функцию строки s.

$s =$	a	b	a	a	$	a	a	b	a	a	b	a	b	a	a
$z =$	0	0	1	1	0	1	4	0	1	3	0	4	0	1	1

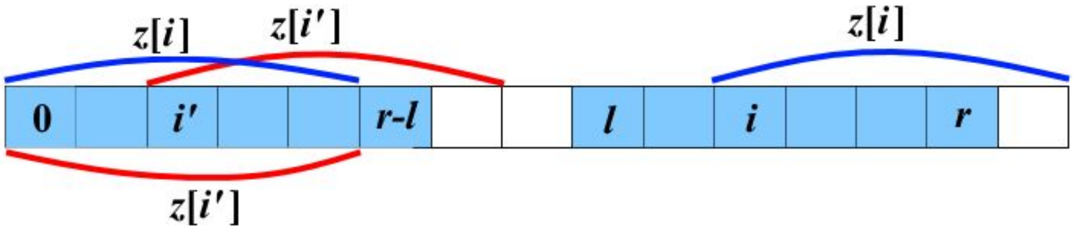
Z-функция: использования для поиска вхождений

Пусть дан текст t и образец p. Требуется найти все вхождения образца p в текст t.



1. Построим $s := p$ + ‘$’ + t
2. Найдем z-функцию строки s
3. $z[i] = |p|$ тогда и только тогда, когда в t есть вхождение p с позиции i-$|p|$-1.

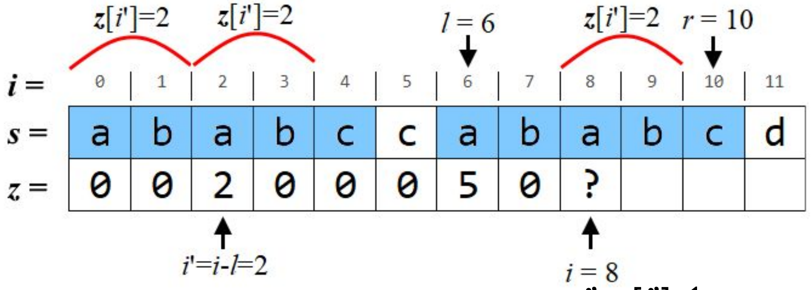
Z-алгоритм



- Будем вычислять значение $z[i]$ слева направо, начиная с i=1.
- Будем поддерживать такие два индекса l и r, что $s[l\,..\,r]$ – это префикс строки s, а значение r – максимально.
- В частности, это означает, что $s[l]=s[0]$, $s[l+1]=s[1]$, …, $s[i]=s[i-l]$, …, $s[r]=s[r-l]$.
- Если $i \leqslant r$, то пусть $i'=i-l$, посмотрим на $z[i']$ – это значение поможет найти $z[i]$.
- По определению $z[i']$ верно: $s[0]=s[i']$, $s[1]=s[i'+1]$, …, $s[z[i']-1]=s[i'+z[i']-1]$, но помним, что $s[i']=s[i]$, $s[i'+1]=s[i+1]$, …, $s[r-l]=s[r]$. Следовательно, $s[0]=s[i]$, $s[1]=s[i+1]$, … и так далее всего $\mathbf{min}(z[i'], r-i+1)$ раз. То есть $z[i]$ частично посчитана ($z[i] \geqslant \mathbf{min}(z[i'], r-i+1)$).

Z-алгоритм: случай 1.1

Предположим $i \leqslant r$, то есть i принадлежит $[l, r]$. Тогда $s[i]=s[i-l]$, $s[i+1]=s[i-l+1]$, ..., $s[r]=s[r-l]$.
Пусть $i'=i-l$ — в некотором смысле отражение символа i в обработанной ранее части строки.



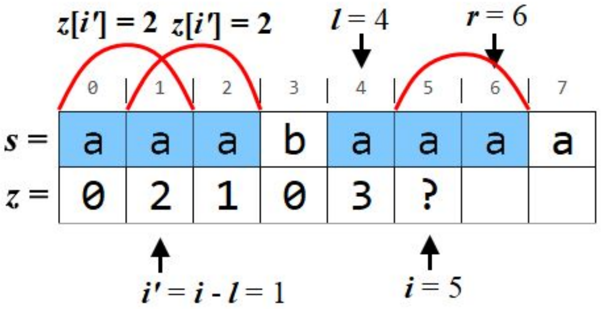
Тогда если значение $z[i']$ такое, что правая граница совпадения $i'+z[i']-1$ строго меньше $r-l$, то и начиная с символа i есть точно такое же совпадение, но нет совпадения длиннее. По определению $z[i']$ символы $s[z[i']]$ и $s[i' + z[i']]$ различаются (иначе $z[i']$ можно увеличить), но так как $s[i' + z[i']] = s[i + z[i']]$, то значит и различаются $s[z[i']]$ и $s[i + z[i']]$. Следовательно, значение $z[i]$ в точности равно $z[i']$.

Например, на рисунке $z[i']=2$. В самом деле, все все красные дуги обозначают одинаковые подстроки, $s[2] \neq s[4]$ (иначе $z[i']$ было бы 3). Однако так как $s[4]=s[10]$, то и $s[2]\neq s[10]$. Следовательно, $z[8]=z[2]=2$.

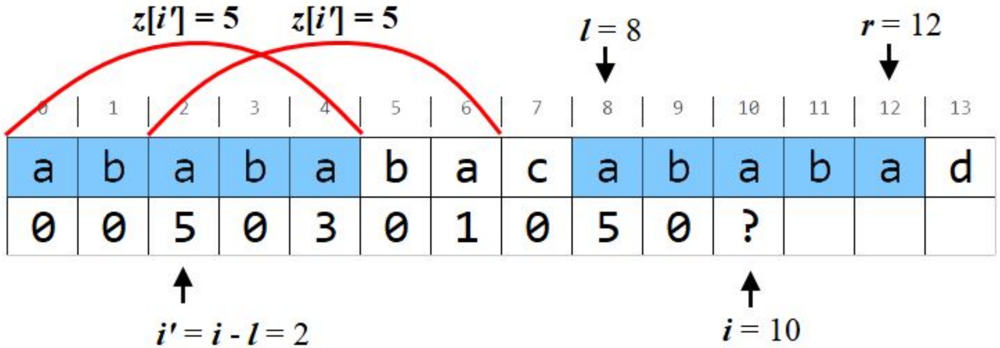
Z-алгоритм: случай 1.2

Предположим $i \leqslant r$, то есть i принадлежит $[l, r]$. Тогда $s[i]=s[i\text{-}l]$, $s[i+1]=s[i\text{-}l+1]$, …, $s[r]=s[r\text{-}l]$.

Пусть $i'=i\text{-}l$ — в некотором смысле отражение символа i в обработанной ранее части строки.



На этом рисунке показано, что после $z[i]:=r\text{-}i+1$ значение $z[i]$ надо увеличить, чтобы оно стало корректным.

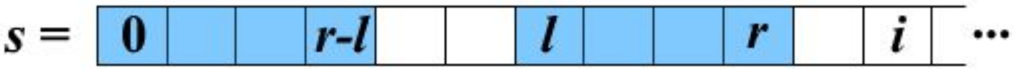


На этом рисунке показано, что иногда для $z[i]$ нельзя использовать $z[i']$, здесь $z[i']=5$, но $z[i]=3$.

Если значение $z[i']$ такое, что правая граница совпадения $i'+z[i']\text{-}1 \geqslant r\text{-}l$, то всё совпадение от $z[i']$ использовать нельзя, а только ту часть, что попала в префикс длины $r\text{-}l+1$ (голубую часть). Иными словами мы можем быть только уверенными, что $z[i] \geqslant r\text{-}l\text{-}i'+1= r\text{-}l\text{-}(i\text{-}l)+1 = r\text{-}i+1$. Поэтому присвоим $z[i]:=r\text{-}l+1$ и потом попробуем увеличить (до корректного значения) наивным образом.

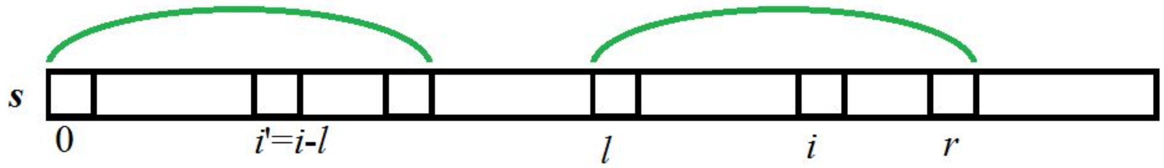
Z-алгоритм: случай 2

Предположим $i > r$, тогда предыдущие вычисленные значение z использовать нельзя.



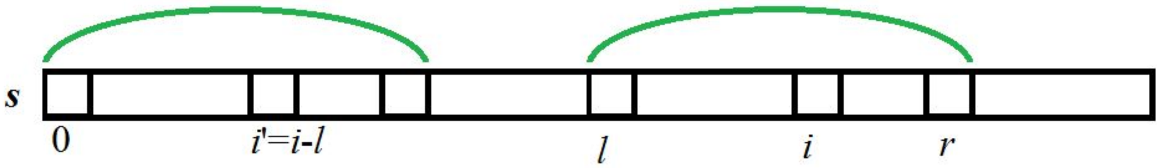
В этом случае будем увеличивать $z[i]$ начиная со значения 0 наивным образом (пока можно).

Z-алгоритм



- Будем вычислять значение $z[i]$ слева направо, начиная с i=1.
- Будем поддерживать такие два индекса l и r, что $s[l \,..\, r]$ — это префикс строки s, а значение r — максимально.
- В частности, это означает, что $s[l]=s[0]$, $s[l+1]=s[1]$, …, $s[i]=s[i-l]$, …, $s[r]=s[r-l]$.
- Если $i \leqslant r$, то посмотрим на $z[i-l]$ — это значение поможет найти $z[i]$.
  - В самом деле подстроки длины $r-i+1$, которые начинаются в i и $i-l$ равны.
  - Если $z[i-l]<r-i+1$, то тогда и $z[i]=z[i-l]$. В этом случае значение $z[i]$ подсчитано.
  - Если $z[i-l]\geqslant r-i+1$, то $z[i]$ точно больше или равно $r-i+1$, то точное значение неизвестно. Наивным способом будем увеличивать $z[i]$ пока можно, получим точное значение.
- Если $i>r$, то доп. информации для $z[i]$ нет и будем вычислять $z[i]$ наивно.
- Заметим, что подстрока длины $s[i \ldots i + z[i] - 1]$ совпадает с префиксом, пересчитаем l и r, если нужно (то есть, если $r < i + z[i] - 1$).

Z-алгоритм




```
n = |s|, l = 0, r = 0
for i := 1 .. n - 1:
    if r >= i: // можем ли воспользоваться [l,r]-блоком?
        if z[i - l] < r - i + 1: // значение z[i - l] короче правого конца?
            z[i] = z[i - l]
        else:
            z[i] = r - i + 1 // совпадение до правого конца или дальше
            while z[i] + i < n && s[z[i]] == s[z[i] + i]:
                z[i]++
    else:
        while z[i] + i < n && s[z[i]] == s[z[i] + i]:
            z[i]++
    if r < i + z[i] - 1: // пересчитаем [l,r]? поддерживаем r->max
        l = i
        r = i + z[i] - 1
```

Z-алгоритм: компактная реализация

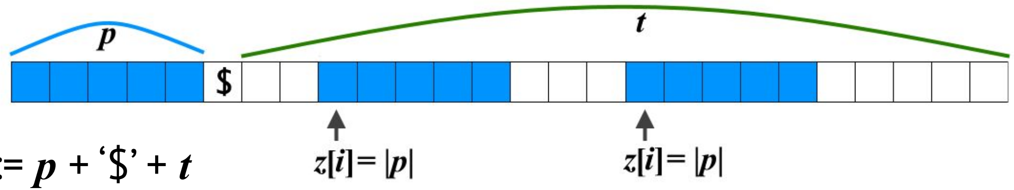
```
l = r = 0
for i = 1..n-1:
    if r >= i:
        z[i] = min(z[i - l], r - i + 1)
    while z[i] + i < n && s[z[i]] == s[z[i] + i]:
        z[i]++
    if i + z[i] - 1 > r:
        l = i, r = i + z[i] - 1
```

- Работает за $O(n)$ потому что при каждом увеличении $z[i]$ внутри while индекс r тоже сдвигается вправо. Сдвинутся более n раз он не может.
- Можно сократить реализацию, если правую границу делать “не включительно”, то есть использовать парадигму $[l, r)$.

Z-функция: использование для поиска вхождений

Поиск подстроки в строке

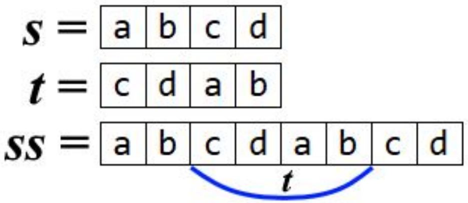
Пусть дан текст t и образец p. Требуется найти все вхождения образца p в текст t.



1. Построим $s := p$ + '\$' + t
2. Найдем z-функцию строки s
3. $z[i] = |p|$ тогда и только тогда, когда в t есть вхождение p с позиции i-$|p|$-1.

Задача. Заданы две строки s и t. Надо проверить является t циклическим сдвигом строки s. Например, для abcd её циклические сдвиги: abcd, bcda, cdab и dabc.

Решение. Запишем строку s подряд два раза. В получившейся строке найдём t как подстроку.



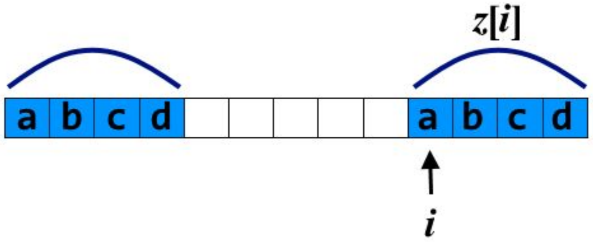
Z-функция: наибольшая грань строки

Гранью строки называется каждый её такой собственный префикс, который одновременно является и её суффиксом. Собственный префикс (суффикс) – это префикс (суффикс) отличный от самой строки.

Например, гранями строки s=«abacaba» являются три строки: «», «a» и «aba».

Задача. Задана строка s. Найдите её самую длинную грань. Например, для s= «abacaba» надо вернуть «aba».

Решение. Для нахождения наибольшей грани с помощью z-функции достаточно найти наименьшее i, что $i+z[i]=|s|$. Тогда соответствующее $z[i]$ – это наидлиннейшая грань строки s.



Z-функция: количество разл. подстрок за $O(n^2)$

Задана строка s, выведите количество различных непустых подстрок в ней.

Например, если s=abaaba, то все её различные подстроки — это `a, aa, aab, aaba, ab, aba, abaa, abaab, abaaba, b, ba, baa, baab, baaba`. Всего 14 штук.

Начнём с пустой строки t и будем "набирать" строку s справа налево (каждый раз дописывая к t слева очередной символ: $t = s[i] + t$), пока не дойдем то полной строки t=s.

В ходе этого будет поддерживать ответ — количество различных подстрок для текущей строки t.

Текущая t	Количество различных подстрок	Как изменился ответ, новые подстроки
""	`0`	–
`a`	`1`	`+1: a`
`ba`	`3`	`+2: b, ba`
`aba`	`5`	`+2: ab, aba`
`aaba`	`8`	`+3: aa, aab, aaba`
`baaba`	`11`	`+3: baa, baab, baaba`
`abaaba`	`14`	`+3: abaa, abaab, abaaba`

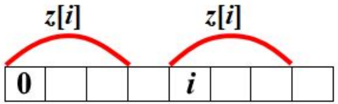
Z-функция: количество разл. подстрок за O(n^2)

Каждый раз к ответу добавляются “новые” префиксы строки t. То есть такие префиксы, которые не были добавлены ранее. То есть такие префиксы, которые не встречаются далее в t.

Добавляются такие префиксы t, которые длиннее самого длинного префикса, который встречается правее.

Подзадача. В строке t найти длину наидлин. префикса, который встречается еще хотя бы раз в t.

Решение. Найдем z-функцию строки t, тогда максимум из значений в ней – это искомая длина наиболее длинного префикса, который встречается в t как не-префикс. Тогда каждый раз к ответу прибавляется $|t|$-$max\{z(t)\}$.



Текущая t	Количество различных подстрок	Как изменился ответ, новые подстроки
“”	0	–
a	1	+1: a
ba	3	+2: b, ba
aba	5	+2: ab, aba
aaba	8	+3: aa, aab, aaba
baaba	11	+3: baa, baab, baaba
abaaba	14	+3: abaa, abaab, abaaba

Z-функция: количество разл. подстрок за O(n^2)

Задана строка s, выведите количество различных непустых подстрок в ней.

Алгоритм решения:

- t = “”
- для i от n-1 до 0
  - $t = s[i] + t$
  - увеличить ответ на i+1-max{$z(t)$}, так как $|t|=i+1$

Всего O(n) шагов, каждый шаг выполняется за линейное время. Итого, общее время работы алгоритма составляет O(n^2).

Текущая t	Количество различных подстрок	Как изменился ответ, новые подстроки
“”	0	–
a	1	+1: a
ba	3	+2: b, ba
aba	5	+2: ab, aba
aaba	8	+3: aa, aab, aaba
baaba	11	+3: baa, baab, baaba
abaaba	14	+3: abaa, abaab, abaaba

Z-функция: период строки

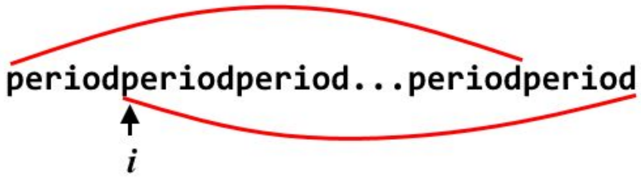
Длиной периода строки s называется такое минимальное число k, что строка s представима в виде конкатенации нескольких копий префикса длины k. Например, для строки s=abcabc длина периода равна 3, для строки s=aaaaa длина периода равна 1, а для строки s=abcab длина периода равна 5.

Очевидно, что k является делителем длины строки.

Найдите длину периода заданной строки за линейное время.

Z-функция: период строки

Алгоритм. Найдем z-функцию строки s. Тогда наименьшее положительное такое i, что $i+z[i]=|s|$ и одновременно $|s|$ делится на i – длина периода строки. Если таких i не существует, то длина периода равна $|s|$.



Достаточность. Если i такое, что $i+z[i]=|s|$ и одновременно $|s|$ делится на i, то покажем, что $s[j] = s[j + i]$ для любого $j < |s|-i$. В самом деле эти два символа равны как соответствующие в префиксе длины $z[i]$ и равной ему подстроке, которая начинается в i. Если для любого $j < |s|-i$ верно $s[j] = s[j + i]$ (и i делитель $|s|$), то, очевидно, что строка периодична с периодом i.

Необходимость. Пусть i – длина периода строки, тогда префикс длины $|s|-i$ равен подстроке такой
же длины, которая стартует из i. Следовательно, $z[i] = |s|-i$. Кроме того, очевидно, i является делителем $|s|$.

Z-функция: поиск с одной ошибкой

t	o	p	u	n	i	v	e	r	c	i	t	y	e	v	e	r
			u	n	i	v	e	r	s	i	t	y				

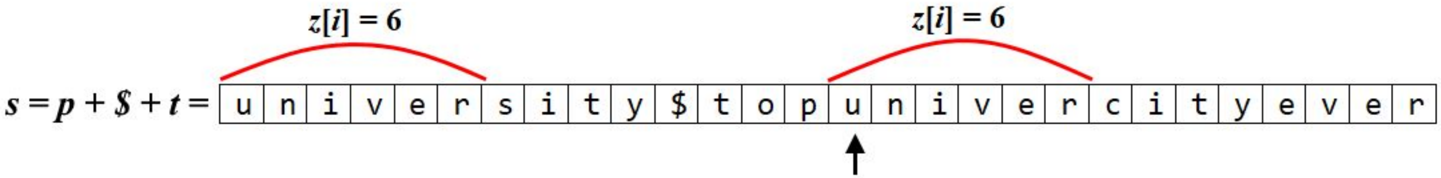
Задан текст t и образец p. Неточным вхождением с одной ошибкой будем называть такую пару индексов (l, r), что $p[i]=t[i+l]$ для всех i от 0 до $|p|$-1, кроме ровно одного случая неравенства $p[i]\neq t[i+l]$.

Пусть $n=|t|$, $m=|p|$. Фактически, для каждого l от 0 до n-m надо проверить является ли l стартом неточного вхождения. Для этого найдем две величины:

- целое число a – длину наибольшего совпадения слева направо (на рисунке выше a=6);
- целое число b – длину наибольшего совпадения справа налево (на рисунке выше b=3).

Заметим, что l является началом неточного вхождения с одной ошибкой, если и только если a+b=m-1.

Найти a проще. Построим новую строку s = p + ’$’ + t, посчитаем для неё z-функцию (как в задаче поиска точного совпадения). Тогда a=z[m+1+l].



Z-функция: поиск с одной ошибкой

t	o	p	u	n	i	v	e	r	c	i	t	y	e	v	e	r
			u	n	i	v	e	r	s	i	t	y				

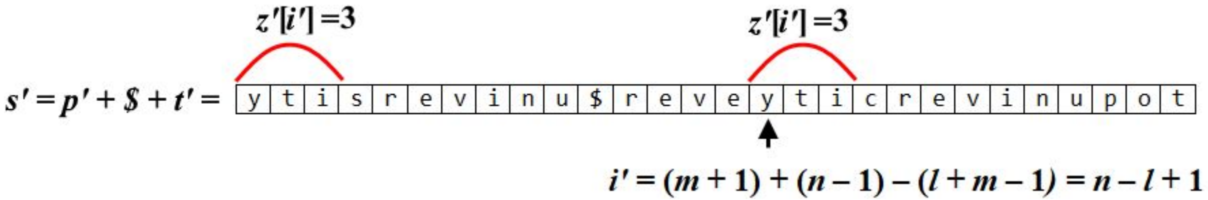
Пусть n=|t|, m=|p|. Фактически, для каждого l от 0 до n-m надо проверить является ли l стартом неточного вхождения. Для этого найдем две величины:

- целое число a – длину наибольшего совпадения слева направо (на рисунке выше a=6);
- целое число b – длину наибольшего совпадения справа налево (на рисунке выше b=3).

Заметим, что l является началом неточного вхождения с одной ошибкой, если и только если a+b=m-1.

Найти a проще. Построим новую строку s = p + '$' + t, посчитаем для неё z-функцию (как в задаче поиска точного совпадения). Тогда a=z[m+1+l].

Чтобы найти b, построим построим p'=rev(p), t'=rev(t) (где rev(x) – это переворот строки x) и s' = p' + $ + t'. Посчитаем для z-функцию z' для s'. Тогда искомое значение b равно просто z'[n-l+1], так как (m + 1) + (n - 1) - (l + m - 1) = n - l + 1.



Спасибо за внимание

И не забудьте решить серию учебных практических задач.