Prefix-функция, Z-функция и алгоритм Кнута-Морриса-Пратта

Волков Сергей, Б06-806

19 декабря 2019 г.

Содержание

1 Prefix-функция

1.1 Описание

Prefix-функция – функция двух аргументов – строки (s) и номера (i) – выдающая на выходе число – длину максимально больших совпадающих собственных (то есть их длина меньше длины самой строки) префикса и суффикса строки $s[: i + 1]$.

Математически это можно выразить так:

$$\pi[i] = \max_{k=0...i-1} \{k : s[0 : k + 1] = s[i - k + 1 : i + 1]\} \tag{1}$$

1.2 Пример расчета prefix-функции для строки

Строка – «abacaba».

Элементы строки	a	b	a	c	a	b	a
Значения prefix-функции	0	0	1	0	1	2	3
Комментарии	Нет совпадений	Нет совпадений	a=a	Нет совпадений	a=a	ab=ab	aba=aba

1.3 Тривиальный алгоритм

Тривиальный алгоритм – расчет значения prefix-функции для каждого элемента строки в отдельности.

Асимптотика алгоритма – $O(n^3)$, т.к. всего $O(n^2)$ итераций цикла, на каждой из который происходит сравнение строк за $O(n)$, что дает в итоге $O(n^3)$.

Реализация на Python 3.7:

```
def prefix_trivial(string):

        prefix_values = [0] * len(string) ## list of values
        for i in range(len(string):
                for k in range(i):
                        if string[:k] == s[i-k, i]:
                                prefix_values[i] = k
        return prefix_values
```

1.4 Оптимальный алгоритм

В основе оптимального алгоритма расчета prefix-функции лежит такая идея:

Если в данный момент времени рассчитано значение $\pi[i]$ и требуется рассчитать $\pi[i+1]$, то можно использовать значение на i-ом элементе:

Посмотрим на элемент $s[\pi[i]]$, и в том случае, когда он равен $s[i+1]$, значение $\pi[i+1] = \pi[i] + 1$.

Если эти элементы не равны, то следует посмотреть на префикс максимальной длины на $s[:\pi[i]]$ и сравнить следующий элемент после этого префикса с $s[i+1]$, если они окажутся равны, то значение $\pi[i+1] = length + 1$, где $length$ – длина максимально большого префикса.

Если эти элементы опять различны, то повторяем алгоритм, пока не найдем равные элементы или не дойдем до $s[0]$.

Асимптотика алгоритма – $O(n)$. Это связано с тем, что при вычислении $\pi[i+1]$ мы делаем линейное количество сравнений [не более, чем $n-1$ в худшем случае (n – длина строки)], например, в случае «aaaab» для символа b.

Реализация на Python 3.7:

```
def prefix(string):

    prefix_values = [0] * len(string) ## list of values
    k = 0
    for i in range( 1, len(string) ):
        while k > 0 and string[k] != string[i]:
            k = prefix_values[k - 1]
        if string[k] == string[i]:
            k += 1
        prefix_values[i] = k
    return prefix_values
```

2 Z-функция

2.1 Описание

Z-функция – функция двух аргументов – строки (s) и номера (i) – выдающая на выходе число – длину максимально больших совпадающих собственных (то есть их длина меньше длины самой строки) префиксов строк s и $s[i:]$.

Математическое описание:

$$z[i] = \max_{k=0...len(s)-i-1} \{k : s[0 : k+1] = s[i : i+k+1]\} \tag{2}$$

2.2 Пример расчета Z-функции для строки

Строка – «abacaba». $z[0] = 0$ по договоренности (доопределение).

Элементы строки	a	b	a	c	a	b	a
Значения Z-функции	0	0	1	0	3	0	1
Комментарии	Нет совпадений	Нет совпадений	a=a	Нет совпадений	aba=aba	Нет совпадений	a=a

2.3 Тривиальный алгоритм

Тривиальный алгоритм – расчет значения Z-функции для каждого элемента строки в отдельности.

Асимптотика алгоритма – $O(n^3)$, т.к. всего $O(n^2)$ итераций цикла, на каждой из который происходит сравнение строк за $O(n)$, что дает в итоге $O(n^3)$.

Реализация на Python 3.7:

```
def z_func_trivial(string):

        z_values = [0] * len(string)
        for i in range(1, len(string)):
                while (i+z_values[i] < len(string) and
                        string[z_values[i]] == string[i+z_values[i]]):
                         z_values[i] += 1
        return z_values
```

2.4 Оптимальный алгоритм

В основе оптимального алгоритма расчета Z-функции лежит такая идея:

Если мы для некоторого i нашли префикс длины такой, что для нескольких дальнейших итераций будет выполнено, что номер итерации меньше, чем номер правой границы получившегося префикса, то мы можем заполнить часть дальнейших значений Z-функции значениями из соответствующего префикса строки s.

Это можно сделать для всех $k = i - left : z[k] < right - i + 1$, где $left, right$ – левая и правая границы т.н. Z-блока для i-го элемента. В случае, если это условие не выполнено, префикс k-го элемента может выходить за границы Z-блока, и нужно провести сравнение следующих элементов.

Асимптотика алгоритма – $O(n)$, доказательство аналогично доказательству линейности времени работы prefix-функции.

Реализация на Python 3.7:

```
def z_func(string):

    z_values = [0] * len(string)
    left, right = 0, 0
    for i in range(1, len(string)):
        if i > right:
            left = right = i
            while (right < len(string) and
                    string[right] == string[right - left]):
                right += 1
            z_values[i] = right - left
            right -= 1
        else: ## checking values from [left, right] interval
            j = i - left
            if z_values[j] < right - i + 1: ## if it's not
                z_values[i] = z_values[j] ## last element
            else: ## if it's last element, compare next ones
                left = i
                while (right < len(string) and
                           string[right] == string[right - left]):
                    right += 1
                z_values[i] = right - left
                right -= 1
    return z_values
```

3 Алгоритм Кнута-Морриса-Пратта (КМП)

3.1 Описание алгоритма

Алгоритм КМП нужен для поиска подстроки в строке, с его помощью можно не только определить, присутствует ли искомая подстрока в заданной строке, но и также сказать, на какой позиции в строке она находится.

Реализация алгоритма КМП состоит в применении prefix- или Z-функции на строке вида $target + @ + string$, где $target$ – искомая подстрока, $string$ – заданная строка, а @ – некоторый символ, не встречающийся ни в $target$, ни в $string$. Этот символ нужен для того, чтобы значение prefix- или Z-функции преждевременно не достигло значения равного либо даже большего, чем $length(target)$.

Если на некотором элементе такой строки в области *string* значение prefix- или Z-функции окажется равным длине *target*, то тогда *target* входит в *string*. Для prefix-функции место, где $\pi[i] = length(target)$, определяет конец вхождения *target* в *string*, а для Z-функции – начало вхождения. Соответственно для prefix-функции элемент под номером $i - length(target)$ будет началом вхождения, а для Z-функции элемент под номером $i + length(target)$ будет концом вхождения. Чтобы найти место вхождения в исходную строку, нужно вычесть (прибавить) соответствующие значения (см. реализацию).

Реализация на Python 3.7:

```
def kmp_prefix(string, target):

        new_string = target + '@' + string
        enterings = []
        prefix_list = prefix(new_string)
        for i in range(len(prefix_list)):
                if prefix_list[i] == len(target):
                        ## appending entering of target
                        ## counting string from 0
                        enterings.append(
                        (i - 2*len(target), i - len(target) - 1) )
        return enterings

def kmp_z(string, target):

        new_string = target + '@' + string
        enterings = []
        z_list = z_func(new_string)
        for i in range(len(z_list)):
                if z_list[i] == len(target):
                        ## appending entering of target
                        ## counting string from 0
                        enterings.append(
                        (i - len(target) - 1, i - 2) )
        return enterings
```