

# Z функция

## 1. Какво е Z функция

Нека имаме низ  $S$  с дължина  $N$ . Z функция на този стринг наричаме масив с размер  $N$ , в който  $i$ -тия елемент показва максималната дължина на подниз на  $S$ , започващ в  $i$ -тата позиция, който е префикс на  $S$ .

С други думи, ако кръстим този масив  $Z$ , то за всяко  $i$  всички символи от  $S[i]$  до  $S[i + Z[i] - 1]$  съвпадат съответно със символите от  $S[0]$  до  $S[Z[i] - 1]$ . Да разгледаме един пример:

$S = \text{"banbban"}, N = 7$

$Z[0]$  – Този елемент от масива не се използва, защото на практика  $Z[0] = N$  (целият е низ е префикс сам на себе си).

$Z[1] = 0$ , защото  $S[1] \neq S[0]$ . (banbban)

$Z[2] = 0$ , защото  $S[2] \neq S[1]$ . (banbban)

$Z[3] = 1$ , защото  $S[3] = S[0]$ , но  $S[4] \neq S[1]$ . (banbban)

$Z[4] = 3$ , защото  $S[4] = S[0]$ ,  $S[5] = S[1]$  и  $S[6] = S[2]$ . (banbban)

$Z[5] = 0$ , защото  $S[5] \neq S[0]$ . (banbban)

$Z[6] = 0$ , защото  $S[6] \neq S[0]$ . (banbban)

(в червено са съвпадащите символи, а в синьо различните)

## 2. Алгоритъм

Лесно можем да се досетим за алгоритъм със сложност  $O(N^2)$ , който намира стойностите на масива  $Z$ : Обхождаме  $S$  и от всяка позиция (освен нулевата) с втори цикъл ходим напред и сравняваме стойности, докато не стигнем до различие в символите, при което прекъсваме този вътрешен цикъл и сме готови със  $Z[i]$ .

```

void z_function(string s)
{
    for(int i=1; i<s.size(); i++)
    {
        while(i+z[i]<s.size() && s[z[i]]==s[i+z[i]]) z[i]++;
    }
}

```

Проблемът с този алгоритъм е, че при правилно подбрани стрингове с големи размери е прекалено бавен. Нека разгледаме низа  $S = "aaaaaa"$ . Тогава за всяко  $i$  програмата ще обхожда низа докрай, което ще отнеме общо  $5+4+3+2+1=15$  операции, а низът е дълъг само 6 символа. Очевидно такъв низ с размер няколко стотин хиляди символа ще отнеме огромен брой операции, което е неефективно.

Ако се замислим, можем да забележим че всеки път когато изчислим някоя стойност  $Z[i]$ , не я използваме за нищо след това. Какво ако помислим за начин да намалим операциите, като използваме вече сметнатите стойности? Оказва се изключително лесно да подобрим тривиалното решение. Нека с  $l$  и  $r$  означим съответно индексите на левия и десния край на последния подниз-префикс, който сме изчислили (в началото  $l=r=0$ ). В процеса на работа  $r$  ще служи като граница, която ни показва индекса на първия неразгледан символ. Това означава, че интервалът е от вида  $[l, r)$ , тоест  $r$  е с 1 по-голямо от последния елемент на последния подниз. Нека в момента сме на позиция  $i$  от низа:

- Ако  $i < r$ , това означава че текущия символ е част от подниз, който съвпада с префикс. От дефиницията за  $Z$  функция следва, че този символ съвпада с някой от първите символи на стринга. Индексът на този символ всъщност е  $i - l$ :  $S[l] = S[0]$ ,  $S[l+1] = S[1]$  и т.н., тоест  $S[l+x] = S[x]$ . За да получим  $S[i]$  вдясно, заместваем  $x=i-l$  и получаваме  $S[i] = S[i-l]$ . Следователно като начална стойност за  $Z[i]$  можем да вземем  $Z[i-l]$  и да продължим изчисленията с тривиалния алгоритъм оттам, като по този начин изпускаме обхождането на вече изчислени символи. Единствената уловка е, че има шанс да вземем повече символи отколкото остават до границата  $r$ , а ние не знаем какви символи следват след нея (не можем да броим неразгледани символи

като прегледани). Затова залагаме като ограничение, че началната стойност на  $Z[i]$  не може да надхвърля броя символи, които се намират между  $i$  и  $r$ . С други думи, началната стойност на  $Z[i]$  не може да надхвърля  $r-i$ . Лесен пример за това е  $S = "aaabaa"$ . Когато сме на последния символ ( $i = 5$ ), знаем че  $l=4$  и  $r=6$ . Използвайки горните разсъждения,  $S[5] = S[1]$  и  $Z[5] = Z[1]$ . Но  $Z[1]=2$  и добавяйки 2 към индекс 5 получаваме индекс 7, който надхвърля  $r$  (и в конкретния пример надхвърля размера на самия стринг). Затова казваме, че  $Z[5] = \text{по-малкото между } Z[1]=2 \text{ и } r-i=6-5=1$ , следователно  $Z[5]=1$ .

- Ако  $i \geq r$ , то вече сме в непозната територия и трябва да използваме тривиалния алгоритъм, за да разгледаме още символи.
- Ако в някой момент намерим нов подниз-префикс (с край по-голям от  $r$ ), обновяваме стойностите на  $l$  и  $r$  подobaващо.

Може да се докаже, че крайната стойност на този алгоритъм е  $O(N)$ , което е значително подобрене спрямо предната сложност.

```
void z_function(string s)
{
    int l=0, r=0;
    for(int i=1; i<s.size(); i++)
    {
        if(i<r) z[i]=min(r-i, z[i-l]);

        while(i+z[i]<s.size() && s[z[i]]==s[i+z[i]]) z[i]++;

        if(i+z[i]>r)
        {
            l=i;
            r=i+z[i];
        }
    }
}
```

### 3. Приложения

#### Намиране на всички срещания на стринг $P$ в текст $T$ :

Тъй като  $Z$  функцията работи с префикси, логично е да потърсим решение в тази посока. Можем да създадем един низ  $S = P + \text{'\#'} + T$ , където  $\#$  може да стои какъвто и да е символ, стига да сме сигурни че той не се среща нито в  $P$ , нито в  $T$  (той ще служи като разделител). По този начин, когато направим  $Z$  функция на  $S$ , ние на практика ще гледаме поднизове на  $T$ , който съвпадат частично или изцяло с  $P$ , а разделителят ни гарантира че не можем да смесим  $P$  с част от  $T$ . Накрая в получения масив търсим всички  $i$ -та, за които  $Z[i] = \text{размера на } P$ .

#### Брой различни последователни поднизове на стринг $S$ :

Ще изчислим бройката последователно – започваме с един символ и постепенно добавяме повече и повече символи от  $S$ , като на  $i$ -тата стъпка пресмятаме колко нови подниза са се появили след добавянето на  $i$ -тия символ.

Нека имаме низ, който съдържа първите  $i$  символа от  $S$ , и кръстим този низ  $P$ . На  $i$ -тата стъпка добавяме  $i$ -тия символ към  $P$ , като е ясно че новопоявилите се поднизове задължително завършват на този символ. За да използваме  $Z$  функция, обръщаме  $P$  наобратно и сега гледаме низовете които започват с новия символ вместо това. Построяваме масива  $Z$  за стринга  $P$ . Сега, за да разберем колко префикса вече съществуват, намираме  $Z[i]_{\max}$ . Тази стойност ни показва, че някъде във вътрешността на  $P$  съществува подниз-префикс с такава дължина, което значи че този подниз вече е броен на предна итерация (при предно  $i$ ). Освен това, всички поднизове-префикси с дължина по-малка от  $Z[i]_{\max}$  също вече са броени, защото те имат същото начало като най-дългия подниз-префикс.

Нека размера на  $P$  в стъпка  $i$  означим с  $M_i$ . Всички възможности за нови стрингове на дадена стъпка са  $M_i - 1$ . Както установихме, от тях  $Z[i]_{max}$  вече са броени. Следователно след всяка стъпка имаме общо  $M_i - Z[i]_{max}$  на брой нови поднизове.

Продължавайки по този начин (като не забравяме да занулим масива  $Z$  и да завъртим  $P$  обратно след всяка стъпка), на всяка стъпка правим  $M_i$  операции, затова сложността на програмата е  $M_0 + M_1 + M_2 + \dots + M_{N-1} + M_N = 0 + 1 + 2 + \dots + N-1 + N$  ( $N$  е размерът на  $S$ ). Следователно крайната сложност е приблизително  $O(N^2)$ .

#### 4. Задачи

<https://www.spoj.com/problems/DISUBSTR/>

<https://codeforces.com/problemset/problem/126/B>

<https://codeforces.com/problemset/problem/432/D>

<https://codeforces.com/problemset/problem/631/D>

<https://codeforces.com/problemset/problem/471/D>