Strings

z-Function

Prof. Edson Alves - UnB/FGA

2019

Sumário

z-Function

Definição

- Seja S uma string de tamanho n
- A função z (z-*function*) é definida por

$$z_S : \mathbb{N} \to \mathbb{N} \cup \{0\}$$
$$i \to z_S(i) = \max\{k \mid S[1..k] = S[i..n]\}$$

- O caso especial $z_S(1)$ depende se o conjunto usado na definição acima inclui apenas prefixos próprios ou não
- Em geral, considera-se apenas sufixos próprios, de modo que $z_S(1) = 0$
- A tabela abaixo ilustra a função z para a string $S =$ "abaaba":

i	1	2	3	4	5	6
S	a	b	a	a	b	a
$z_S(i)$	0	0	1	3	0	1

Pseudocódigo da função z – Naive

Algoritmo 1 Função z

Input: Uma string S

Output: Um vetor zs tal que $zs[i] = z_S(i)$

```
1:  function z(S)
2:      n ← |S|
3:      zs[1] ← 0
4:
5:      for i ← 2 to n do
6:          j ← 0
7:          while i + j ≤ n and S[1 + j] = S[i + j] do
8:              j ← j + 1
9:          zs[i] ← j
10:
11:     return zs
```

Cálculo da função z em $O(n)$

- O pseudocódigo para a função z apresentado anteriormente tem complexidade $O(n^2)$
- É possível modificar este algoritmo de modo que seja possível computar todos os valores $z_S(i)$, $i = 1, 2, \ldots, n$ em $O(n)$
- A ideia central é utilizar os valores já computados da função z evitar comparações já feitas
- De fato, a implementação difere da versão *naive* em apenas dois pontos, referentes a duas condicionais
- A seguir será apresentada a implementação em C++ desta versão modificada, e as mudanças promovidas serão explicadas adiante

Implementação da função z em C++

```cpp
vector<int> z(const string &s)
{
    int n = s.size(), L = 0, R = 0;
    vector<int> zs(n, 0);

    for (int i = 1; i < n; i++)
    {
        if (i <= R)
            zs[i] = min(zs[i - L], R - i + 1);

        while (zs[i] + i < n && s[zs[i]] == s[i + zs[i]])
            zs[i]++;

        if (R < i + zs[i] - 1)
            L = i, R = i + zs[i] - 1;
    }

    return zs;
}
```

Detalhamento da implementação da função z

- A ideia principal da implementação é o uso dos dois ponteiros L e R
- Para qualquer posição i (na implementação a string é indexada de 0 a $n-1$), L e R representam o início e o fim de prefixo comum entre S e algum sufixo $S[k..(n-1)]$, para $k < i$
- Este prefixo deve ser não nulo, e caso exista mais de um prefixo comum já identificado, deve ser escolhido aquele termina mais à direita possível
- Por exemplo, $S =$ "abacababac", a função z assumiria os seguintes valores:

i	0	1	2	3	4	5	6	7	8	9
S	a	b	a	c	a	b	a	b	a	c
$z_S(i)$	0	0	1	0	3	0	4	0	1	0

Detalhamento da implementação da função z

- Suponha que o laço **for** está na nona iteração (isto é, $i = 8$)
- Neste ponto, os prefixos comuns não nulos já encontrados são:

Prefixo	Posição	Tamanho
"a"	$S[2..2]$	1
"aba"	$S[4..6]$	3
"abac"	$S[6..9]$	8

- As posições destas substrings são os candidatos a valores de L e R
- Como a substring que termina mais à direita é $S[6..9]$, nesta iteração vale que $L = 6$ e $R = 9$

Detalhamento da implementação da função z

- Nesta iteração o primeiro **if** tem sua condição verdadeira:

```
8         if (i <= R)
9             zs[i] = min(zs[i - L], R - i + 1);
```

- O fato de que i pertence ao intervalo $[L, R]$ significa que a substring $S[i..R] = S[(i - L)..(R - L)]$, pois $S[0..(R - L)] = S[L..R]$
- O índice $i - L$ corresponde à posição do caractere de $S[0..(R - L)]$ equivalente i em $S[L..R]$
- Como $zs[i - L]$ já foi computado, é conhecido o tamanho do maior prefixo comum entre S e $S[i..R]$
- Assim, $zs[i]$ será mínimo entre $zs[i - L]$ e $|S[i..R] = R - i + 1$, pois caso $zs[i - L]$ seja maior que o tamanho de $S[i..R]$, não há garantias de que os caracteres que sucedem $S[R]$ coincidam com os caracteres que sucedem $S[R - L]$

Detalhamento da implementação da função z

- O segundo **if** atualiza os ponteiros L e R caso o prefixo comum da substring $S[i..(i + z[i] - 1)]$ termine mais à direita do que o prefixo comum armazenado em L e R
- Assim, se $i + z[i] - 1 > R$, então o L e R passam a apontar para a substring $S[i..i + z[i] - 1]$

```
14          if (R < i + zs[i] - 1)
15              L = i, R = i + zs[i] - 1;
```

- Observe que, caso $z[i] = 0$, $S[i..i - 1]$ é uma string vazia, e os valores de L e R correspondem a um intervalo degenerado
- Pode ser mostrado que a condição do laço **while** pode ser verdadeira, no máximo, $n - 1$ vezes, de modo que o algoritmo tem complexidade $O(n)$

Referências

1. **CHARRAS**, Christian; **LECROQ**, Thierry. *Handbook of Exact String-Matching Algorithms*[1]
2. CP Algorithms. *z-Function*, acesso em 16/08/2019.
3. **CROCHEMORE**, Maxime; **RYTTER**, Wojciech. *Jewels of Stringology: Text Algorithms*, WSPC, 2002.
4. **HALIM**, Steve; **HALIM**, Felix. *Competitive Programming 3*, Lulu, 2013.

[1] Morris-Pratt Algorithm