Strings

z-Function

Prof. Edson Alves - UnB/FGA

Sumário

z-Function

Definição

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

$$\begin{aligned} z_S &: \mathbb{N} \to \mathbb{N} \cup \{0\} \\ i &\to z_S(i) = \max\{k \mid S[1..k] \text{ é prefixo de } S[i..n]\} \end{aligned}$$

- O caso especial $z_S(1)$ depende se o conjunto usado na definição acima inclui apenas sufixos 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 $zs[i] = z_S(i)$

function $\mathrm{Z}(S)$
    $n \leftarrow |S|$
    $zs[1] \leftarrow 0$

    for $i \leftarrow 2$ **to** n **do**
        $j \leftarrow 0$
        while $i + j \leq n$ **and** $S[1 + j] = S[i + j]$ **do**
            $j \leftarrow j + 1$
        $zs[i] \leftarrow j$

    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 para 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"	$S[6..9]$	4

- 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:

```
        if (i <= R)
            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 $S[i..i + z[i] - 1]$

```
        if (R < i + zs[i] - 1)
            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 $O(n)$

Aplicações da z-*Function*

Aplicação #1 – Número de ocorrências de P em S

- A função z também pode ser utilizada para determinar o número de ocorrências de uma string P, de tamanho m, em uma string S, de tamanho n
- Defina a string T como

$$T = P + \text{`\#'} + S$$

- Assim, T é a concatenação da string P, o caractere separador '#' e a string S
- O separador pode ser qualquer caractere que não apareça nem em S e nem em T
- A string P ocorre na posição i de S se, e somente se,

$$z_T(i + m + 1) = m$$

- Esta abordagem tem complexidade $O(n + m)$

Exemplo de uso da função z para contagem de ocorrências de P em S

i	1	2	3	4	5	6	7	8	9	10
T	a	n	a	#	b	a	n	a	n	a
$z_T(i)$	0	0	1	0	0	$\underline{3}$	0	$\underline{3}$	0	1

$$m = 3$$

$$occ = 2$$

$$pos = 2\ (6 - 3 - 1) \text{ e } 4\ (8 - 3 - 1)$$

Implementação do número de ocorrências de P em S 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;
}
```

Implementação do número de ocorrências de P em S em C++

```cpp
int search(const string& S, const string& P, char delim = '#')
{
    string T { P + delim + S };
    auto zs = z(T);
    int occ = 0, m = P.size();

    for (const auto x : zs)
        occ += (x == m ? 1 : 0);

    return occ;
}
```

Aplicação #2 – Busca inexata

- A função z também pode ser utilizada para localizar o número de ocorrências de uma substring P, de tamanho m, em S, de tamanho n, permitindo que P e a substring de S sejam distintas em, no máximo, um caractere
- Sejam A e B duas strings de mesmo tamanho n. A distância de Hamming $\mathrm{dist}(A, B)$ de A a B é dada por
$$\mathrm{dist}(A, B) = |\{i \mid i \in [1, n] \text{ e } A[i] \neq B[i]\}|$$
- Em termos mais precisos, é possível determinar o tamanho do conjunto
$$M = \{S[i..j] \mid j - i + 1 = m \text{ e } \mathrm{dist}(P, S[i..j]) \leq 1\}$$

Aplicação #2 – Busca inexata

- Para computar o tamanho de M é preciso montar duas strings
- A primeira delas é a string T, definida na primeira aplicação, onde

$$T = P + \text{`\#'} + S$$

- Deste modo, o caractere $S[i]$ corresponde ao caractere $T[i+m+1]$
- Seja S' a string reversa de S, isto é, $S'[i] = S[n-i+1]$, para todo $i \in [1, n]$
- A segunda string R é definida por

$$R = P' + \text{`\#'} + S'$$

- O caractere $S[i]$ corresponde ao caractere $S'[i']$ da string reversa, com $i' = n - i + 1$
- Como $j = i + m - 1$, segue que $j' = i' - m + 1$

Aplicação #2 – Busca inexata

- Assim, o último caractere da substring $S[i..j]$ corresponde ao caractere $R[k]$, onde

$$\begin{aligned} k &= j' + m + 1 \\ &= (i' - m + 1) + m + 1 \\ &= (n - i + 1) + 2 \\ &= n - i + 3 \end{aligned}$$

- Portanto, $S[i..j] \in M$ se, e somente se

$$z_T[i + m + 1] + z_R[n - i + 3] \geq m - 1$$

- Outras palavras, se o tamanhos do prefixo comum entre $S[i..j]$ e P, somado ao tamanho do sufixo comum entre $S[i..j]$ e P, resultar em $m - 1$, há apenas um caractere de diferença entre ambos
- Se a soma for maior ou igual a m (de fato, igual a $2m$), então $S[i..j] = P$

Exemplo de uso da função z para busca inexata

i	1	2	3	4	5	6	7	8	9	10	11	12
T	a	n	a	#	r	a	b	a	n	e	t	e
$z_T(i)$	0	0	1	0	0	1	0	2	0	0	0	0
R	a	n	a	#	e	t	e	n	a	b	a	r
$z_R(i)$	0	0	1	0	0	0	0	0	1	0	1	0

Exemplo de uso da função z para busca inexata

i	1	2	3	4	5	6	7	8	9	10	11	12
T	a	n	a	#	r	a	b	a	n	e	t	e
$z_T(i)$	0	0	1	0	0	$\underline{1}$	0	2	0	0	0	0
R	a	n	a	#	e	t	e	n	a	b	a	r
$z_R(i)$	0	0	1	0	0	0	0	0	$\underline{1}$	0	1	0

$$m = 3$$

$$n = 8$$

$$occ = 1$$

$$i = 2$$

$$i_T = i + m + 1 = 6$$

$$j_R = n - i + 3 = 9$$

$$z_T(6) + z_R(9) = 2$$

Exemplo de uso da função z para busca inexata

i	1	2	3	4	5	6	7	8	9	10	11	12
T	a	n	a	#	r	a	b	a	n	e	t	e
$z_T(i)$	0	0	1	0	0	1	0	$\underline{2}$	0	0	0	0
R	a	n	a	#	e	t	e	n	a	b	a	r
$z_R(i)$	0	0	1	0	0	0	$\underline{0}$	0	1	0	1	0

$$m = 3$$

$$n = 8$$

$$occ = 2$$

$$i = 4$$

$$i_T = i + m + 1 = 8$$

$$j_R = n - i + 3 = 7$$

$$z_T(8) + z_R(7) = 2$$

Implementação da busca inexata 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;
}
```

Implementação da busca inexata em C++

```
string rev(const string& S)
{
    auto P { S };
    reverse(P.begin(), P.end());

    return P;
}

int search(const string& S, const string& P, char delim = '#')
{
    string T { P + delim + S }, R { rev(P) + delim + rev(S) };
    auto zT = z(T), zR = z(R);
    int occ = 0, n = S.size(), m = P.size();

    // Como as string estão indexadas a partir de 0, o índice de k de j em R é igual a n - i + 1
    for (int i = 0; i < n; i++)
        occ += (zT[i + m + 1] + zR[n - i + 1] >= m - 1) ? 1 : 0;

    return occ;
}
```

Referências

1. CP Algorithms. *z-Function*, acesso em 16/08/2019.
2. **CROCHEMORE**, Maxime; **RYTTER**, Wojciech. *Jewels of Stringology: Text Algorithms*, WSPC, 2002.
3. **HALIM**, Steve; **HALIM**, Felix. *Competitive Programming 3*, Lulu, 2013.
4. Wikipédia. Hamming distance, acesso em 20/08/2019.