Strings

Strings e *Hashes*

Prof. Edson Alves - UnB/FGA

2019

Sumário

Strings e *Hashes*

Comparação de strings

- Duas strings S e T são iguais se $S[i] = T[i]$, para $i \in [1, n]$, com $n = |S| = |T|$
- A comparação entre os caracteres de posições correspondentes faz com que esta verificação tem complexidade $O(n)$
- Uma maneira de realizar esta comparação de forma mais eficiente é utilizar uma função de *hash* h, que transforma uma string S em um inteiro $h(S)$, e comparar $h(S)$ com $h(T)$
- Como a comparação de inteiros, em geral, é feita em $O(1)$, a complexidade da comparação dependerá apenas do custo de se computar $h(S)$

Hash

- Seja $\mathcal{S}$ o conjunto de todas as strings possíveis e m um número natural
- Denominamos
$$h : \mathcal{S} \to [0, m]$$
uma função de *hash* em $\mathcal{S}$
- Observe que, como h é função, se $S = T$ então $h(S) = h(T)$
- A recíproca não é necessariamente verdadeira: pode acontecer $h(S) = h(T)$ com $S \neq T$
- Isto ocorre porque o número de strings possíveis é, em geral, muito maior do que o intervalo $[0, m]$, de modo que h não é injetiva
- Esta situação é denominada colisão
- O desafio é definir h de modo a minimizar o número de colisões

Polynomial Rolling Hash

Definição

Polynomial Rolling Hash

Seja S uma string de tamanho N, cujos elementos são indexados de 0 a $N-1$. A função

$$h(S) = \left(\sum_{i=0}^{N-1} S_i p^i\right) \bmod m$$
$$= \left(S_0 + S_1 p + S_2 p^2 + \ldots + S_{N-1} p^{N-1}\right) \bmod m,$$

onde p e m são dois inteiros positivos, é denominada *polynomial rolling hash*.

Escolha dos parâmetros

- Em geral, p é um número primo aproximadamente igual ao tamanho do alfabeto
- Para um alfabeto de 26 letras, uma escolha razoável seria $p = 31$
- Para maiúsculas e minúsculas pode-se adotar $p = 53$
- O valor de m deve ser grande, pois a chance de colisão entre duas strings sorteadas aleatoriamente é de $1/m$
- Usar um número primo para m também é uma boa escolha, no sentido de evitar colisões
- O valor $m = 10^9 + 7$ tem a vantagem de ser fácil de lembrar e digitar, e também de permitir a multiplicação sem *overflow* usando variáveis do tipo `long long`

Mapeamento de caracteres

- Na definição da função h o valor s_i corresponde ao mapeamento do caractere $S[i]$ da string para um inteiro
- Em termos formais, dado um alfabeto $\mathcal{A}$ e uma função

$$f : \mathcal{A} \to \mathbb{N},$$

então $s_i = f(S[i])$, onde $S[i] \in \mathcal{A}$ para todo $i = 0, 1, 2, \ldots, N - 1$
- Um mapeamento possível seria $f(\texttt{a}) = 1, f(\texttt{b}) = 2, \ldots, f(\texttt{z}) = 25$
- Veja que o caractere 'a' não é mapeado para zero, e sim para um, para evitar que todas as strings compostas por repetições deste caractere tenham o mesmo *hash* h

Implementação do *rolling hash* em Haskell

```haskell
import Data.Char

f :: Char -> Int
f c = (ord c) - (ord 'a') + 1

h :: String -> Int
h s = sum (zipWith (*) fs ps) `mod` m where
    p = 31
    m = 10^9 + 7
    fs = map f s
    ps = map (\x -> p ^ x) $ take (length s) [0..]
```

Implementação do *rolling hash* em C++

```cpp
int f(char c)
{
    return c - 'a' + 1;
}

int h(const string& s)
{
    long long ans = 0, p = 31, m = 1000000007;

    for (auto it = s.rbegin(); it != s.rend(); ++it)
    {
        ans = (ans * p) % m;
        ans = (ans + f(*it)) % m;
    }

    return ans;
}
```

Calculo do *hash* das substrings de S

- Dada uma string S, a definição de h permite computar o valor de $h(S[i..j])$, para qualquer par $i \leq j$ de índices válidos, em $O(1)$, se conhecidos os valores de h para todos os prefixos $S[0..i]$ de S
- A função h é definida por

$$h(S) = \left(\sum_{i=0}^{N-1} S_i p^i\right) \bmod m$$

- Deste modo,

$$\begin{aligned} h(S[i..j])p^i &= \left(\sum_{k=i}^{j} S_i p^i\right) \bmod m \\ &= (h(S[0..j]) - h(S[0..(i-1)])) \bmod m \end{aligned}$$

Calculo do *hash* das substrings de S

- Para obter o valor de $S[i..j]$, é necessário multiplicar a expressão acima pelo inverso multiplicativo $(p^i)^{-1}$ de p^i módulo m
- Este pode ser obtido pelo Pequeno Teorema de Fermat: se p é primo e $(a, p) = 1$, então
$$a^{p-1} \equiv 1 \pmod{p}$$
- Assim, como $p \geq 2$,
$$a \cdot a^{p-2} \equiv 1 \pmod{p},$$
de modo que
$$a^{-1} \equiv a^{p-2} \pmod{p}$$
- Se os inversos de p^i também forem precomputados, juntamente com os *hashes* dos prefixos $S[0..i]$, os valores $h(S[i..j])$ podem ser calculados em $O(1)$

Referências

1. CP-Algorithms. String Hashing, acesso em 06/08/2019.
2. **CROCHEMORE**, Maxime; **RYTTER**, Wojciech. *Jewels of Stringology: Text Algorithms*, WSPC, 2002.
3. **HALIM**, Steve; **HALIM**, Felix. *Competitive Programming 3*, Lulu, 2013.