

Strings

Suffix Array – Aplicações

Prof. Edson Alves - UnB/FGA

2019

1. Busca em *array* de sufixos
2. Comparação de substrings de mesmo tamanho

Busca em *array* de sufixos

Busca em *array* de sufixos

- O problema de se determinar se a string P , de tamanho M , é ou não uma substring de S , de tamanho N , pode ser resolvido por meio de um *array* de sufixos
- Isto porque, se P é uma substring de S , ela será substring de algum dos prefixos $S[i..N]$ de S
- Assim, para localizar P em S basta fazer uma busca binária em $s_A(S)$
- Em cada etapa da busca binária, a comparação de T com o prefixo em questão tem complexidade $O(M)$
- Assim o algoritmo tem complexidade $O(M \log N)$, se $s_A(S)$ já estiver construído
- O número de ocorrências de P pode ser determinado por meio de uma segunda busca binária, pois todas elas estarão adjacentes no *array* de sufixos

Implementação da busca em *array* de sufixos em C++

```
84 int occurrences(const string& P, const string& S)
85 {
86     auto sa = suffix_array(S);
87
88     auto it = lower_bound(sa.begin(), sa.end(), P,
89         // retorna true S[sa[i]..N] de anteceder P na ordenação
90         [&](int i, const string& P) {
91             return S.compare(i, P.size(), P) < 0;
92         });
93
94     auto jt = upper_bound(sa.begin(), sa.end(), P,
95         // retorna true se P deve anteceder S[sa[i]..N] na ordenação
96         [&](const string& P, int i) {
97             return S.compare(i, P.size(), P) > 0;
98         });
99
100     return jt - it;
101 }
```

Comparação de substrings de mesmo tamanho

Comparação de strings de mesmo tamanho

- Seja S uma string de tamanho N e considere duas substrings de S :
 $a = S[i..(i + M - 1)]$ e $b = S[j..(j + M - 1)]$, ambas de tamanho M
- Uma função $f(a, b)$ é uma função de comparação de strings se
 $f(a, b) < 0$ se $a < b$, $f(a, b) = 0$ se $a = b$ e $f(a, b) > 0$ se $a > b$
- Usando a comparação caractere a caractere, uma função de comparação pode ser implementada com complexidade $O(M)$
- Contudo, uma vez construído o vetor de sufixos $s_A(S)$, esta função pode ser implementada em $O(1)$
- Para tal, é preciso armazenar os valores das classes de equivalência cs obtidos em todas as iterações da construção de $s_A(S)$
- Seja $cs[k][i]$ a classe de equivalência da substring cíclica de S , de tamanho 2^k , com início na posição i

Comparação de strings de mesmo tamanho



- Se $M = 2^k$, então a função $f(a, b)$ corresponde à comparação direta entre $cs[k][i]$ e $cs[k][j]$
- Caso contrário, M pode ser decomposto em dois blocos de tamanho 2^t , onde $2^t \leq M$ e $2^{t+1} > M$
- O primeiro bloco começa na posição inicial da substring (i , no caso da substring a)
- O segundo bloco começa 2^t posições antes da última posição (no caso da substring a , na posição $i + M - 2^t$)
- Se as classes de ambas strings em relação ao primeiro bloco são distintas, a comparação entre eles é suficiente
- Caso contrário, basta finalizar a comparação utilizando as classes de equivalência dos respectivos segundos blocos
- Assim, o algoritmo tem complexidade $O(N \log N)$, por conta da construção de $s_A(S)$, e complexidade de memória $O(N \log N)$, por conta da tabela de classes de equivalência cs

Visualização da comparação entre duas substrings de mesmo tamanho

a = programação

b = programados

Visualização da comparação entre duas substrings de mesmo tamanho

a =  $\text{programa}\text{ção}$
 b =  $\text{programa}\text{dos}$

The diagram illustrates the comparison of two substrings of equal length. String a is "programação" and string b is "programados". Both strings have their first 8 characters, "programa", highlighted in green. A bracket above the green part of a and a bracket below the green part of b are both labeled "1º bloco", indicating the first block of comparison.

Visualização da comparação entre duas substrings de mesmo tamanho

a = programação

b = programados

Implementação da comparação de substrings de mesmo tamanho em $O(1)$

```
88 int compare(int i, int j, int M, const vector<vector<int>>& cs)
89 {
90     int k = 0;
91
92     while ((1 << (k + 1)) <= M)
93         ++k;
94
95     auto a = ii(cs[k][i], cs[k][i + M - (1 << k)]);
96     auto b = ii(cs[k][j], cs[k][j + M - (1 << k)]);
97
98     return a == b ? 0 : (a < b ? -1 : 1);
99 }
```

1. CP Algorithms. [Suffix Array](#), acesso em 06/09/2019.
2. **CROCHEMORE**, Maxime; **RYTTER**, Wojciech. *Jewels of Stringology: Text Algorithms*, WSPC, 2002.
3. **HALIM**, Steve; **HALIM**, Felix. *Competitive Programming 3*, Lulu, 2013.