Busca de padrões em texto

Marco Alves de Alcantara
Supervisora: Cristina Gomes Fernandes

Introdução

Nesse trabalho iremos estudar algoritmos e estruturas de dados para buscas de padrões em textos. Estaremos pensando em aplicações onde desejamos buscar várias palavras em um texto que seja fixo e bastante longo. Um exemplo de aplicação com essas condições poderia ser buscar certas sequências de DNA no genoma humano.

Em vez de pré-processar cada palavra P a ser buscada, como seria feito em outros algoritmos como Boyer-Moore ou o KMP, iremos pré-processar o texto T uma única vez e realizar cada busca em tempo $\mathcal{O}(|P|)$ em vez de $\mathcal{O}(|T|)$.

Preliminares

Antes de começar o pré-processamento, é necessário acrescentar um caractere especial ao final do texto. Esse caractere tem a propriedade de que ele é lexicograficamente menor que qualquer outro caractere do alfabeto do texto e das palavras que serão buscadas nele. Iremos representá-lo com \$.

Para uma palavra P (que, em particular pode ser o texto T), denotaremos por $P[i]$ o caractere na i-ésima posição em P (indexada a partir de 0). Também denotaremos por $P[i..j]$ a palavra contida entre o i-ésimo e o j-ésimo caractere de P. Além disso, $P[i..]$ será o sufixo de P a partir do i-ésimo caractere, e $P[..j]$ será o prefixo de P até o j-ésimo caractere.

Vetor de Sufixos

Definimos o vetor de sufixos de T, denotado por VS(T), ou simplesmente VS, como sendo o vetor dos índices dos sufixos de T ordenados lexicograficamente. Ou seja, para todo par de índices i e j de VS, se $i < j$, então T[VS[i]..] é lexicograficamente menor que T[VS[j]..].

Por exemplo, para T = "abracadabra\$", temos o seguinte vetor de sufixos:

$T[i..]$	i	VS[i]	T[VS[i]..]
abracadabra\$	0	11	\$
bracadabra\$	1	10	a\$
racadabra\$	2	7	abra\$
acadabra\$	3	0	abracadabra\$
cadabra\$	4	3	acadabra\$
adabra\$	5	5	adabra\$
dabra\$	6	8	bra\$
abra\$	7	1	bracadabra\$
bra\$	8	4	cadabra\$
ra\$	9	6	dabra\$
a\$	10	9	ra\$
\$	11	2	racadabra\$

O Vetor de sufixos pode ser construído em tempo linear em $|T|$ por meio do DC3 Skew Algorithm[KS06], um algoritmo de divisão e conquista no qual o texto é dividido em três partes, sendo criado um alfabeto novo onde cada símbolo é uma tripla de caracteres. O VS é então calculado recursivamente para duas dessas três partes do texto, e o algoritmo realiza algumas comparações dos caracteres para construí-lo para a terceira parte. Esse algoritmo será explicado com mais detalhes na monografia pois ele foge ao escopo desse pôster.

Vetor LCP

Para realizar a busca eficientemente, é necessário também um outro vetor, denominado LCP (*longest common prefix*). Trata-se de um vetor de comprimento $|T| - 1$, indexado a partir de 1, tal que LCP[i] é o comprimento do prefixo comum mais longo entre os sufixos de índices VS[$i-1$] e VS[i].

Ainda para o texto T = "abracadabra\$", tem-se o seguinte vetor LCP:

i	VS[i]	LCP[i]	T[VS[i]..]
0	11		\$
1	10	0	a\$
2	7	1	**a**bra\$
3	0	4	**abra**cadabra\$
4	3	1	**a**cadabra\$
5	5	1	**a**dabra\$
6	8	0	bra\$
7	1	3	**bra**cadabra\$
8	4	0	cadabra\$
9	6	0	dabra\$
10	9	0	ra\$
11	2	2	**ra**cadabra\$

O vetor LCP pode ser construído em tempo linear em $|T|$ a partir do VS por meio do Algoritmo de Kasai[Kas+01], descrito abaixo.

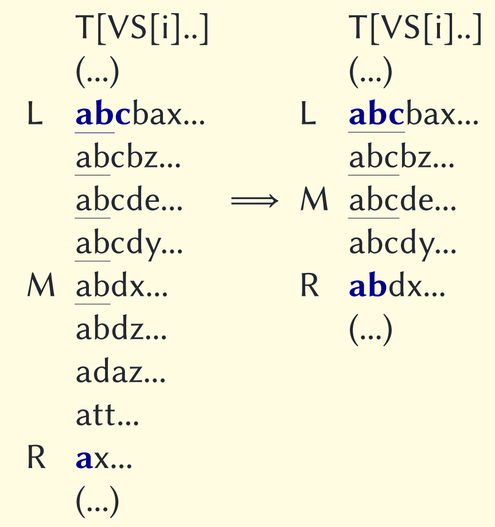
Algoritmo: Construção linear do LCP
Dados: Texto T, vetor de sufixos VS
Resultado: Vetor LCP de T

```
i ← 0
enquanto i < |T| faça
    rank[VS[i]] ← i
    i ← i + 1
h, i ← 0
enquanto i < |T| faça
    se rank[i] > 0 então
        k ← VS[rank[i] − 1]
        enquanto T[i + h] = T[k + h] faça
            h ← h + 1
        LCP[rank[i]] ← h
        se h > 0 então
            h ← h − 1
    i ← i + 1
retorna LCP
```

Sobre a Busca

Tendo os vetores de sufixos e o LCP, podemos encontrar as ocorrências de P no texto. São feitas duas buscas binárias em VS, uma pelo predecessor de P, que é o maior sufixo de T que é lexicograficamente menor que P, e na segunda busca, incrementamos o último caractere de P e procuramos o predecessor dessa nova palavra. Esses predecessores nos fornecem o intervalo com os índices de todas as ocorrência de P em VS.

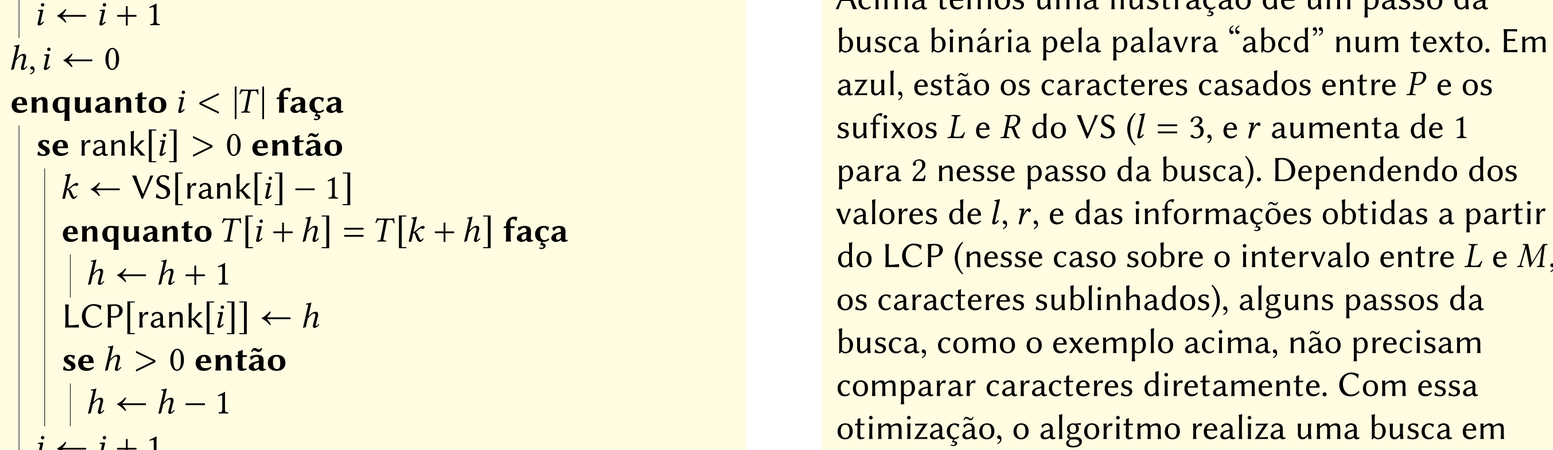
Em cada busca binária, além do intervalo $[L, R]$ de índices de VS onde a busca está sendo realizada, também são utilizadas variáveis auxiliares l e r com o comprimento do prefixo comum mais longo entre P e os sufixos iniciados em VS[L] e VS[R] respectivamente. Isso é, l e r servem para contar o número de caracteres casados entre P e as extremidades da busca binária.



Acima temos uma ilustração de um passo da busca binária pela palavra "abcd" num texto. Em azul, estão os caracteres casados entre P e os sufixos L e R do VS ($l = 3$, e r aumenta de 1 para 2 nesse passo da busca). Dependendo dos valores de l, r, e das informações obtidas a partir do LCP (nesse caso sobre o intervalo entre L e M, os caracteres sublinhados), alguns passos da busca, como o exemplo acima, não precisam comparar caracteres diretamente. Com essa otimização, o algoritmo realiza uma busca em tempo $\mathcal{O}(|P| + \log(|T|))$.

Árvores de Sufixos

Uma alternativa ao vetor de sufixos para realizar as buscas é construir a árvore de sufixos do texto T. Numa árvore de sufixos, os nós filhos de um determinado nó são indexados por caracteres do texto T. A partir da raiz, cada aresta da árvore contém a descrição de um intervalo que deve ser "soletrado" quando a busca percorrer essa aresta.



A busca consiste em encontrar o nó menos fundo n cujo caminho até a raiz contém P. Se n não existir, então P não ocorre em T, e, caso contrário, os índices das ocorrências de P serão os índices das folhas da árvore a partir de n. A árvore pode ser construída a partir do vetor de sufixos e LCP, ou diretamente com o Algoritmo de Ukkonen, ambas as possibilidades em tempo linear em $|T|$.

Referências

[Kas+01] Toru Kasai et al. "Linear-Time Longest-Common-Prefix Computation in Suffix Arrays and Its Applications". Em: (2001).
[KS06] Juha Kärkkäinen e Peter Sanders. "Simple Linear Work Suffix Array Construction". Em: (2006).

https://www.linux.ime.usp.br/~marcoalves/
marcoalves1504@usp.br
Instituto de Matemática e Estatística, Universidade de São Paulo (IME-USP)