Algoritmos em sequências

Yan Soares Couto **Orientadora: Cristina Gomes Fernandes**
Departamento de Ciência da Computação, Instituto de Matemática e Estatística, Universidade de São Paulo

Introdução

O crescimento expressivo na capacidade de armazenamento de dados em meios digitais criou a demanda por algoritmos cada vez mais eficientes para realizar buscas por padrões em textos. Dependendo da situação tais buscas podem ser por ocorrências exatas ou por ocorrências aproximadas do padrão.

Algoritmos de busca de padrão têm aplicação não só em atividades do nosso dia a dia, quando frequentemente usamos, em um editor ou em um browser, comandos de busca por um padrão, como também por exemplo em problemas de biologia computacional, quando buscamos padrões em sequências de DNA ou de proteína.

Algoritmos clássicos como o KMP e o Boyer-Moore são abordados em disciplinas de um curso de graduação em Ciência da Computação. Nesse trabalho, abordamos algoritmos e estruturas de dados mais sofisticados e poderosos, que resolvem não apenas um problema específico de busca, mas uma classe de problemas, de maneira eficiente.

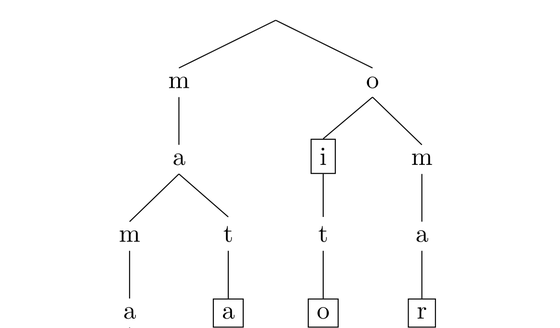
Objetivos

Estudo e implementação de algoritmos sofisticados e eficientes para problemas de busca de padrão e variantes.

Apresentação tão completa e didática quanto possível dos algoritmos e estruturas de dados estudados. Disponibilização das implementações desenvolvidas no github.

Dado um alfabeto Σ, uma string T é uma sequência finita de símbolos de Σ. Defina $T[i]$ como o i-ésimo simbolo dessa sequência.

Dizemos que $T[i\,..\,j] = T[i]T[i+1]\dots T[j]$ é uma **substring** de T.



Um prefixo de T é uma substring que começa na posição 1, e um sufixo de T é uma substring que termina na posição $|T|$. Dizemos que T_1 ocorre em T_2 se T_1 é uma substring de T_2.

Tries

Uma trie é uma estrutura de dados que armazena um conjunto de strings $\mathcal{S}$.

Operações

Contains(P): Determina se $P \in \mathcal{S}$. Tempo $\mathcal{O}(|P|)$.

LCP(P): Retorna o maior prefixo comum de P com **alguma** string de $\mathcal{S}$. Tempo $\mathcal{O}(|P|)$.

Figura: Uma trie exemplo T para $\mathcal{S}$ ={mamata, mata, oi, oito, omar}.

A imagem ilustra uma trie, os vértices marcados indicam os fins das strings no conjunto. É possível construir uma trie de $\mathcal{S} = \{S_1, \dots, S_k\}$ em tempo $\mathcal{O}(\sum_{i=1}^{k} |S_i| \cdot |\Sigma|)$.

Informações e contato

Para mais informações, acesse a página do trabalho:
http://www.linux.ime.usp.br/~yancouto/mac0499

Endereço para contato: yancouto@ime.usp.br

Estruturas em tries

É possível guardar algumas informações adicionais na trie, ou usar uma forma diferente de armazená-la para torná-la ainda mais poderosa. Estas versões alternativas são úteis para busca de ocorrências exatas de conjuntos.

Aho-Corasick

Para um conjunto $\mathcal{S} = \{S_1, \dots, S_k\}$ de strings, após preprocessamento na trie de $\mathcal{S}$, é possível realizar:

Operações

Any(P): Determina se alguma string de $\mathcal{S}$ ocorre em P. Tempo $\mathcal{O}(|P|)$.

Count(P): Para cada string $S_i \in \mathcal{S}$, determina o número de ocorrências de S_i em P. Tempo $\mathcal{O}(|P| + \sum_{i=1}^{k} |S_i|)$.

Para cada vértice u, o seu **link de falha** é o nó mais profundo cuja string é um sufixo da string associada a u. Note que essa ideia é uma generalização do link de falha calculado no algoritmo KMP.

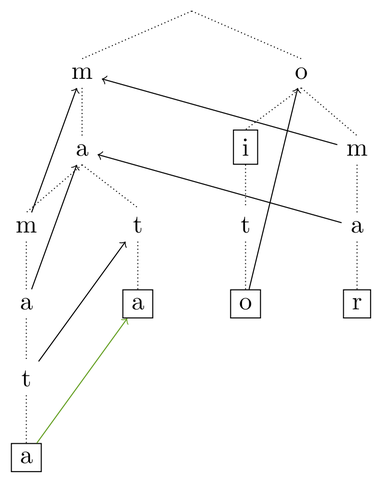


Figura: Links de falha para a trie exemplo. Os links para a raiz foram omitidos.

O **link de ocorrência** de u é o próximo vértice que pode ser visitado usando links de falha, a partir de u. O link destacado na figura é o único link de ocorrência.

A estrutura e algoritmo apresentado foi criado por Alfred V. Aho and Margaret J. Corasick [1].

Árvore de Sufixos

Uma árvore de sufixos é uma trie de todos os sufixos de S.

Operações

Count(P): Conta quantas ocorrências de P existem em S. Tempo $\mathcal{O}(|P|)$.

List(P): Retorna uma lista com as posições de todas as ocorrências de P em S. Tempo $\mathcal{O}(|P| + x)$, onde x é o número de tais ocorrências.

Comprimindo vértices que tem apenas um filho, o número de vértices da trie fica no máximo $2|S|$.

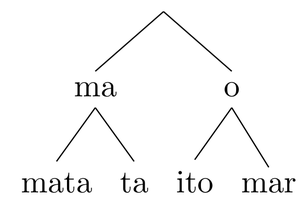


Figura: Trie comprimida para T

É possível construir esta árvore de sufixos com consumo de tempo e espaço $\mathcal{O}(|S||\Sigma|)$.

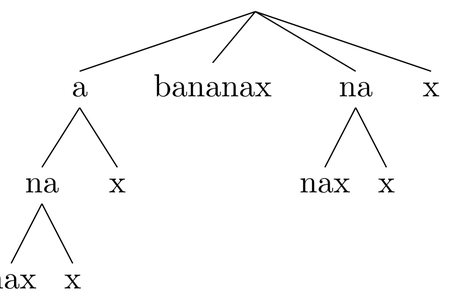


Figura: Árvore de sufixos de bananax.

O primeiro algoritmo para criação de árvores de sufixos em tempo linear foi proposto por Weiner [2], mas o algoritmo apresentado no texto é de Ukkonen [3].

Autômato de Sufixos

Autômato de sufixos é um autômato que reconhece todos os sufixos de uma string S.

Operações

Count(P): Conta quantas ocorrências de P existem em S. Tempo $\mathcal{O}(|P|)$.

Cada nó em um autômato de sufixos identifica um conjunto de ocorrências à direita.

$D(P) = \{i : P \text{ é sufixo de } S[1\,..\,i]\}$ é o conjunto de ocorrências à direita de P.

Figura: Construção do autômato de sufixos de banana, falhas desenhadas pontilhadas.

O número de vértices do autômato é no máximo $2|S|$. A estrutura tem usos bem similares aos da árvore de sufixos, mas a implementação é mais simples.

O **link de falha** de um nó aponta para o nó cujo conjunto de ocorrências à direita o contém propriamente.

Referências

Aho, Alfred V. and Corasick, Margaret J., "**Efficient string matching: an aid to bibliographic search**," in *Commun. ACM*, 1975, 18 (6), pp. 333–340.

Peter Weiner, "**Linear pattern matching algorithms**," in *14th Symposium on Switching and Automata Theory*, 1973.

Ukkonen, Esko, "**On-line construction of suffix trees**", in *Algorithmica*, 1995, 14 (3), pp. 249–260.