TP0: Pesquisa em Texto Utilizando Árvore Trie

Rúbia Reis Guerra
rubia-rg@ufmg.br

2016/02

1 Introdução

Tries, ou árvores digitais, são árvores que representam coleções de strings em que todos os descendentes de um nó possuem prefixo comum.

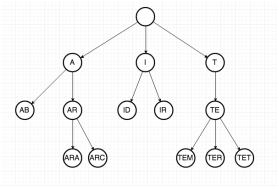

Figura 1: Exemplo de trie.

No presente trabalho, propôs-se o estudo da aplicação dessa estrutura de dados na solução de problemas envolvendo buscas textuais. Dado um conjunto de strings que formam um texto T, deseja-se, para cada entrada de um dicionário D previamente fornecido, identificar o número de ocorrências em T das palavras pertencentes à T. Como exemplo:

$$D = \{anexo, polo, forma, trigo, renda, anexo\}$$

$$T = \{monologo, renda, forma, acougueiro, anexar, forma, choque\}$$

$$Ocorr. = \{0, 0, 2, 0, 1, 0\}$$

A solução consiste na implementação de uma trie que guarda as palavras do dicionário e, internamente, os resultados da contagem de ocorrências das entradas de D em relação à T. A estrutura é formada por nós contendo arrays de c posições - sendo c o tamanho do alfabeto - e dois sinalizadores para indicar o final de uma palavra e sua respectiva contagem de ocorrências.

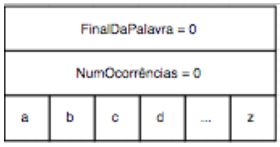

Figura 2: Exemplo da estrutura de um nó.

2 Descrição da solução

2.1 Estrutura de dados

A estrutura de dados utilizada no projeto da solução foi a árvore digital (trie). Sua implementação utilizou-se de um nó contendo um array *branches* de c posições, em que cada posição marca um possível caminho para percorrer a árvore; um sinalizador indicando se o final da palavra foi atingido e uma variável inteira que, no momento da pesquisa, guarda o número de ocorrências caso a palavra pesquisada esteja presente no dicionário.

2.2 Inserção

Na inserção, cada caractere é inserido como um nó individual. Caracteres são transformados em chaves do array *branches*, em que cada posição funciona como um ponteiro para um nível mais interno (ou nó-folha) da árvore. A figura abaixo exemplifica uma trie com as strings "aba"e "z":

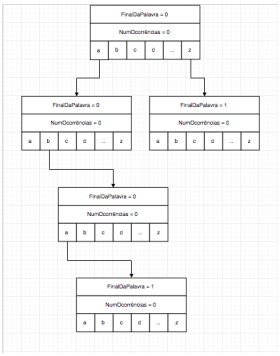

Figura 3: Exemplo de trie após a inserção de "aba"e "z".

A extensão de um prefixo existente ou a inserção de nova palavra consiste em identificar a posição em *branches* que o ponteiro deverá seguir e criar novos nós, posicionados mais longe da raiz conforme se avança na leitura da palavra. Ao inserir o último caracter de uma nova palavra, um nó-folha é criado na posição correspondente ao índice e FinalDaPalavra é marcado como verdadeiro. Caso a palavra inserida seja prefixo de outra já presente no dicionário, o nó-folha correspondente ao último caracter do prefixo é marcado como FinalDaPalavra.

A complexidade de tempo, no pior caso em que nenhum dos caracteres da string está presente na árvore, corresponde a:

$$O(L * C),$$

sendo L o tamanho do da cadeia de caracteres a ser inserida e C, o tamanho do alfabeto.

A complexidade de espaço para a inserção é:

$$O(L * C * N)$$

sendo L o tamanho da palavra a ser inserida, C, o tamanho do alfabeto e N, o número de palavras na árvore.

2.3 Pesquisa

A pesquisa na árvore ocorre de maneira similar à inserção, porém, ao deparar com um caminho inexistente referente ao índice do caracter por que se procura, a busca é terminada sem retorno. Caso encontre toda a string presente no dicionário, verifica-se se o nó final do caminho é um nó-folha. Caso seja, a variável que guarda o número de ocorrências (incialmente nula) é incrementada.

A complexidade de tempo, no pior caso em que todos os caracteres da string estão presentes na árvore, porém o nó final não é um nó-folha, é:

$$O(L),$$

sendo L o tamanho do da cadeia de caracteres a ser pesquisada.

A complexidade de espaço para a pesquisa é:

$$O(L * C * N)$$

sendo L o tamanho da palavra a ser inserida, C, o tamanho do alfabeto e N, o número de palavras na árvore.

2.4 Contagem de ocorrências

A contagem de ocorrências ocorre de maneira similar à pesquisa, exceto que ao final, caso a string esteja presente na trie e o último nó encontrado seja um nó-folha, a quantidade de ocorrências da chave em questão é retornada.

A complexidade de tempo, no pior caso em que todos os caracteres da string estão presentes na árvore, porém o nó final não é um nó-folha, é:

$$O(L),$$

sendo L o tamanho do da cadeia de caracteres a ser pesquisada.

A complexidade de espaço é:

$$O(L * C * N)$$

sendo L o tamanho da palavra a ser inserida, C, o tamanho do alfabeto e N, o número de palavras na árvore.

2.5 Desalocando memória

O procedimento que libera a memória alocada para a trie consiste em uma recursão que recebe um ponteiro para o nó-raiz, caminha até o nó mais interno de cada posição em *branches* e desaloca, recursivamente, todos os nós até a raiz.

3 Testes de desempenho

Para verificar a validade das análises de complexidade, foram realizadas três baterias de testes alternando o tamanho da palavra (L), a quantidade de chaves do dicionário (D) e a quantidade de palavras do texto (T).

Observou-se, a partir dos tempos de execução, o comportamento linear com o tamanho da chave L, esperado a partir da análise do algoritmo. Ainda, nota-se o aumento do tempo de execução proporcional ao aumento da quantidade de palavras do texto e do dicionário, relacionado à alocação e liberação de memória para armazenamento da trie.

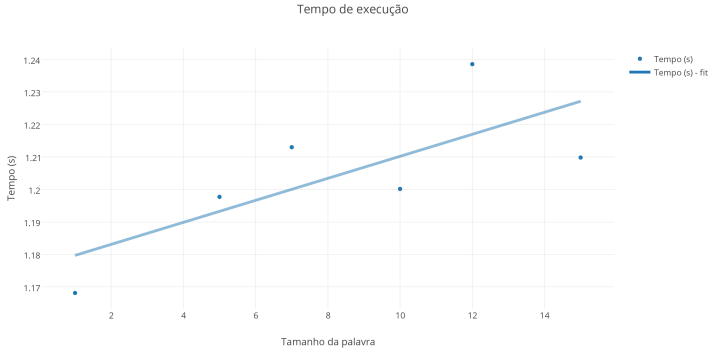

Figura 4: Tempo de execução para D = 100 e T = 1000.

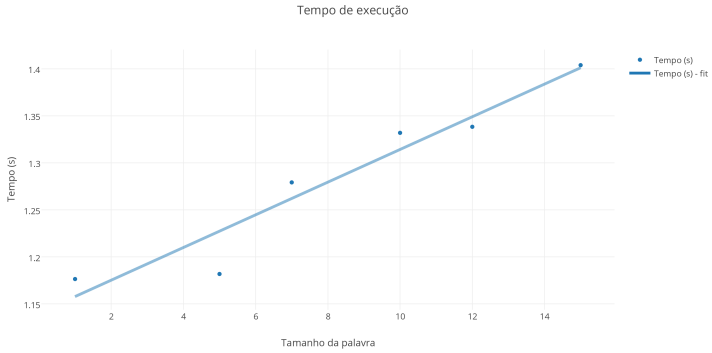

Figura 5: Tempo de execução para D = 10000 e T = 100000.

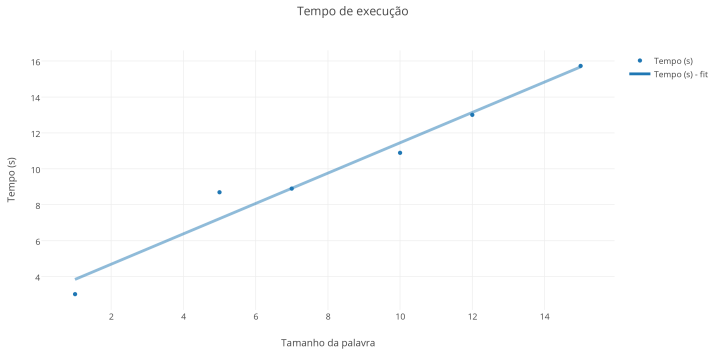

Figura 6: Tempo de execução para D = 100000 e T = 10000000.

4 Conclusão

Neste trabalho, foi proposto o estudo da implementação da estrutura de dados *trie* na solução de problemas de busca textual. O problema foi resolvido

adotando-se um modelo de árvore cujos nós formam arrays de apontadores para índices correspondetes aos caracteres que formam chaves de inserção e busca. Ao final, a análise das complexidades teóricas dos procedimentos do projeto foram confirmadas por experimentos utilizando parâmetros adequadamente dimensionados para permitir a observação do comportamento assintótico.

Referências

[1] R. Sedgewick and K. Wayne. *Algorithms*. 4th Edition, Addison-Wesley, 2011.

[2] U. Mamber. *Introduction to Algorithms*. 1st Edition, Addison-Wesley, 1989.

[3] V. Sangam. Trie Tree Implementation.
`http://theoryofprogramming.com/2015/01/16/trie-tree-implementation/`

[4] V. Sanka. Trie.
`http://www.geeksforgeeks.org/trie-insert-and-search/`

[5] E. Demaine. 6.851: Advanced Data Structures (Spring'12), Lecture 16.
`http://courses.csail.mit.edu/6.851/spring12/lectures/L16.pdf`