Documentação Trabalho Prático 0 de AED's 3

Aluno: Marcus Vinícius de Oliveira Matricula: 2014144804

1. Introdução

Uma trie é uma estrutura de dados do tipo árvore de prefixos. Seu nome é derivado da palavra inglesa “retrivial”, que significa recuperação. Este nome lhe foi dado porque essa estrutura é basicamente usada na recuperação de informações.

Amplamente utilizada na implementação de dicionários, ela fornece toda a funcionalidade para inserir, procurar e excluir uma string. Embora exista outras soluções mais triviais para implementação de dicionários, a complexidade O(n) para as operações de inserção e exclusão de uma string na trie a torna um referencial. Ela também é utilizada na implementação de motores de busca na web, preenchimento automático e corretor ortográfico.

Dada sua importância, nosso trabalho é a contagem da frequência que palavras de um dado dicionário aparecem em um texto. Esta contagem deve ser feita e armazenada em uma implementação de uma árvore trie.

1.1. Modelagem do Problema

A modelagem do problema é implicitamente trivial. É necessário apenas a adaptação de um TAD trie para que este possa armazenar o número de vezes que uma chave é vista em um texto.

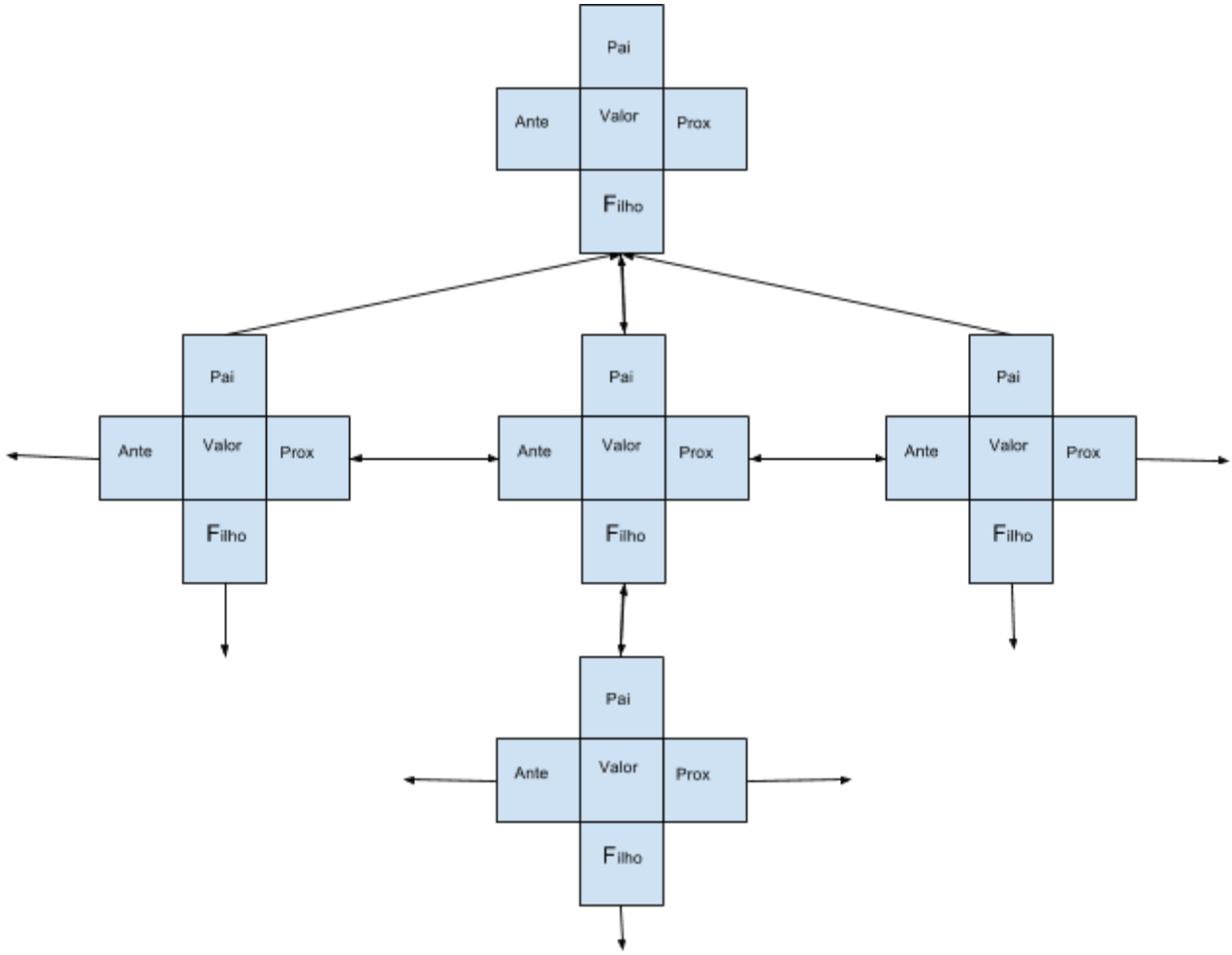
Embora sua modelagem seja trivial, sua implementação é complexa e exige o máximo de cuidados.

Nossa Estrutura de Trie foi implementada com os seguintes campos:

Chave: Parte da string a ser inserida na Trie (A letra ‘a’ de aeds por exemplo)
Valor: Número de vezes em que a palavra do dicionario é vista no texto.
Prox: Contém endereço do próximo nó (node) no mesmo nível.
Ante: Contém o endereço do nó anterior no mesmo nível.
Filho: Contém o endereço do nó filho.
Pai: Contém o endereço do nó pai.

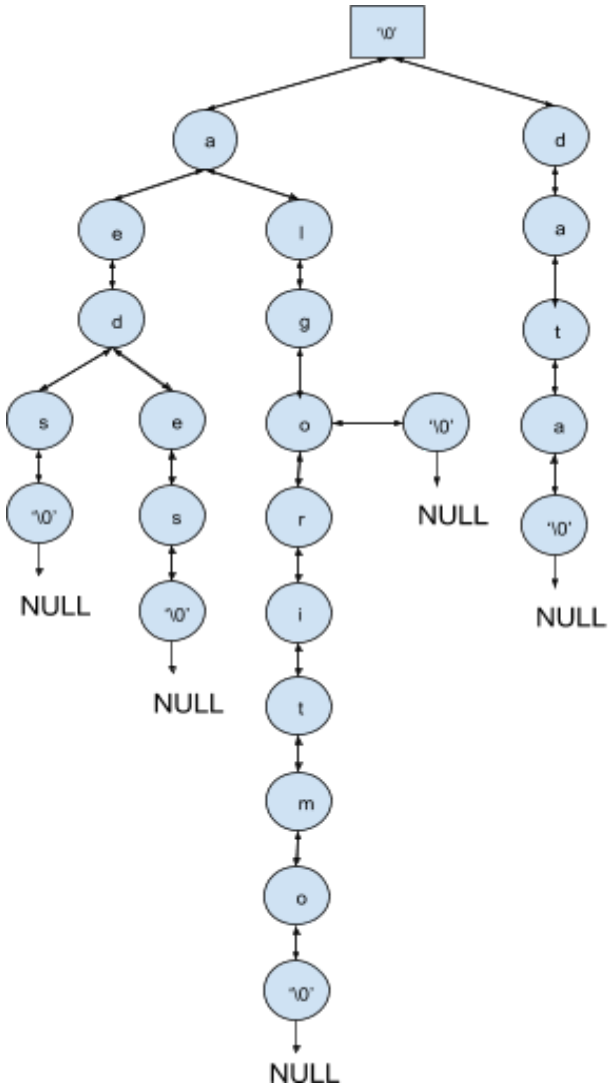
Nós adicionais como “ante” e “pai” foram adicionados para facilitar a pesquisa e o desalocamento da Trie. Estes não estão presentes em uma implementação clássica.

A seguir um desenho esquemático de um nó de nossa implementação:



1.2. Funcionamento da Trie

As strings contidas no dicionário não são armazenadas explicitamente. Elas ficam codificadas nos caminhos que começam na raiz e finalizam em uma folha, aqui representado pelo nó que contem a chave ‘/0’. Neste mesmo nó, está contido o inteiro “valor”, que armazena o número de vezes que a string foi encontrada no texto. Conforme observado nas figuras a seguir:



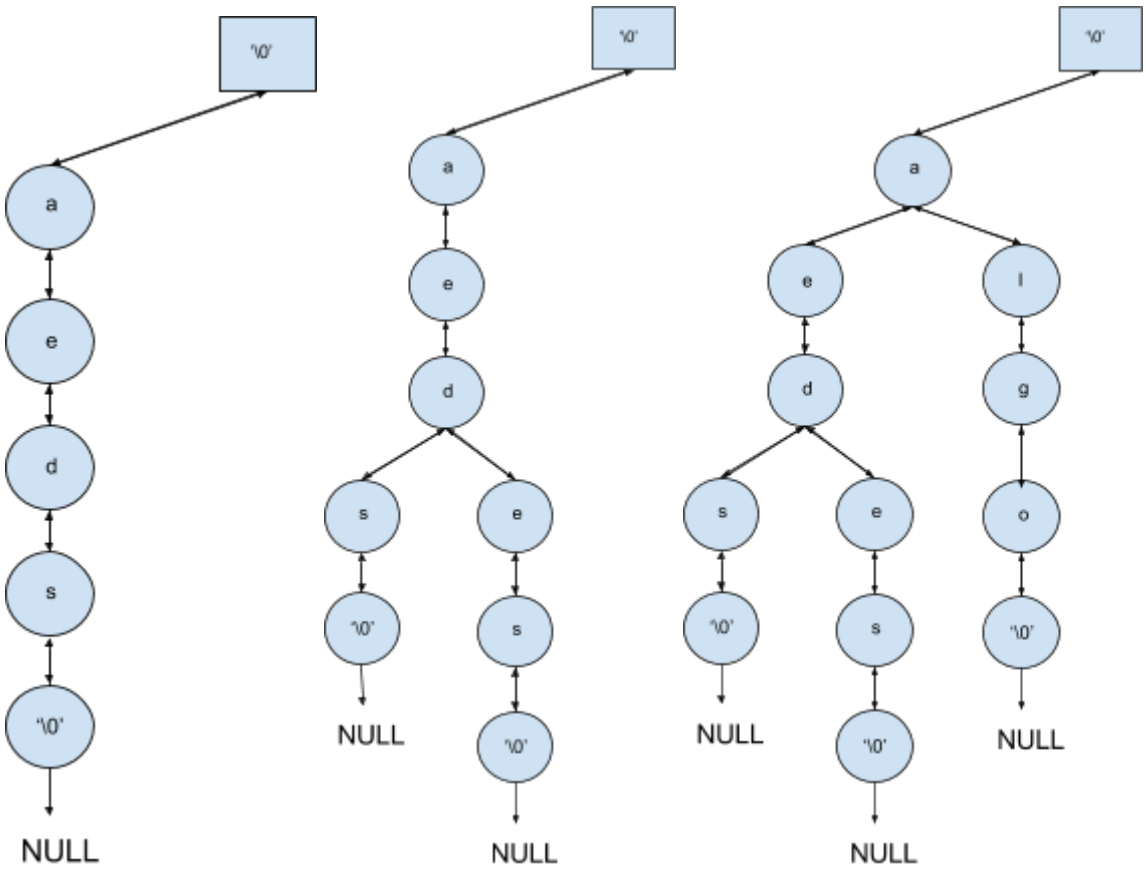
1.2.1. Inserindo String na Trie

A inserção acontece da seguinte maneira:

- Caso 1 - o nó/raiz não possui filhos: é criado um novo nó em que é inserido o primeiro caractere da string. Deste novo nó, é criado um novo nó filho em que é inserido o segundo caractere da string. O processo se repere até que toda a string seja inserida.

- Caso 2 - o nó/raiz possui filhos: é checado se existe um filho com o mesmo caractere que a string. Caso exista, o algoritmo desce de nível na Trie e é repetido o Caso2. Caso não exista, um novo nó é criado no mesmo nível e então é executado o Caso1.

As ilustrações abaixo representam muito bem a inserção:



1.2.2. Pesquisando String na Trie

A pesquisa ocorre da seguinte forma:

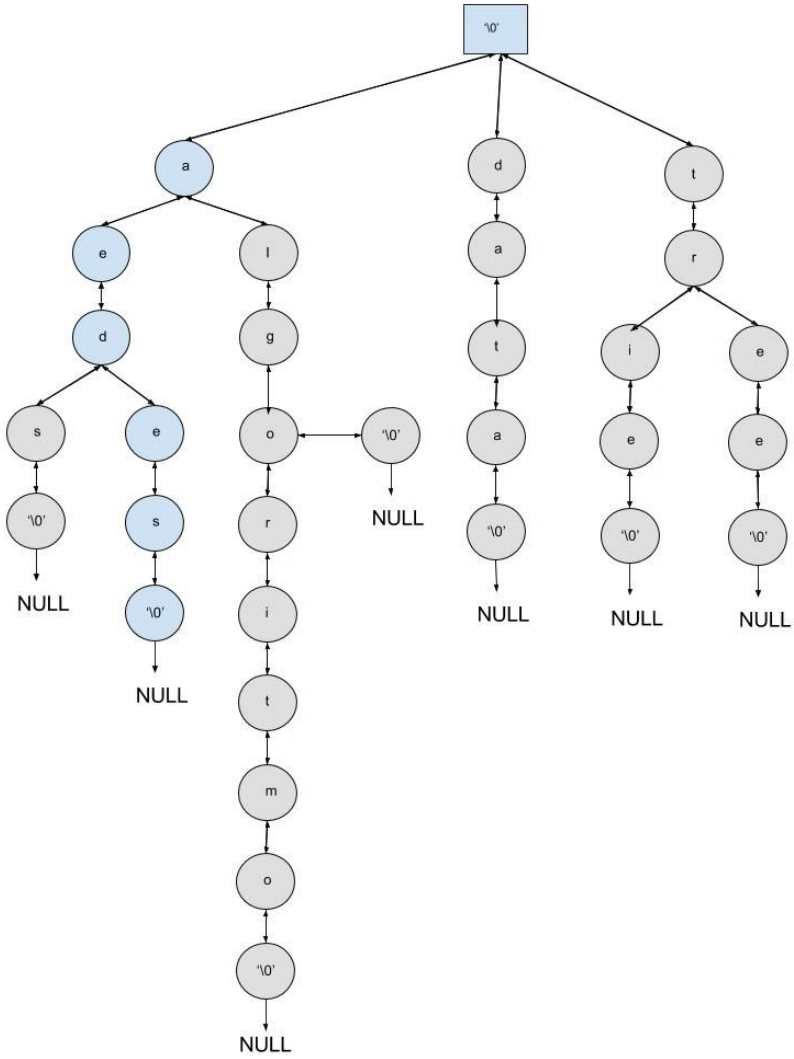
Dada a raiz e uma chave, é checado se a raiz possui um filho com o mesmo caractere que a string. Caso possua, o algoritmo desce de nível na Trie e o processo é repetido até que se encontre a folha da árvore (‘/0’). Caso não possua, não existe tal string na Trie e a função é finalizada.

Uma representação da busca em uma árvore Trie.

Caso 1 - o nó/raiz não possui filhos: é criado um novo nó em que é inserido o primeiro caractere da string. Deste novo nó, é criado um novo nó filho em que é inserido o segundo caractere da string. O processo se repere até que toda a string seja inserida.

Caso 2 - o nó/raiz possui filhos: é checado se existe um filho com o mesmo caractere que a

string. Caso exista, o algoritmo desce de nível na Trie e é repetido o Caso2. Caso não exista, um novo nó é criado no mesmo nível e então é executado o Caso1.



2 Análise de Complexidade

Nesta seção será apresentada a análise do custo teórico de tempo e de espaço

2.1 Tempo

Como notado nos esquemas anteriores, a altura da árvore é igual ao comprimento da string mais longa mais 1 (referente ao elemento ‘/0’). Notamos também que o número de nós filhos está relacionado ao tamanho máximo do alfabeto.

Dado o funcionamento supracitado da trie, temos que o tempo de execução das operações não depende do número de elementos da árvore e sim do comprimento das strings e do alfabeto. Sua complexidade é O(AS), onde A representa o tamanho do alfabeto e C o tamanho da maior string.

2.2 Espaço

Em arvores Trie com implementações do tipo R-Way cada nó aloca espaço para todo o alfabeto. Isso gera um grande desperdício de espaço, uma vez que nem sempre é usado todo o alfabeto em um nível. Sua Complexidade é O(AN), onde A corresponde ao tamanho do alfabeto e N ao número de string.

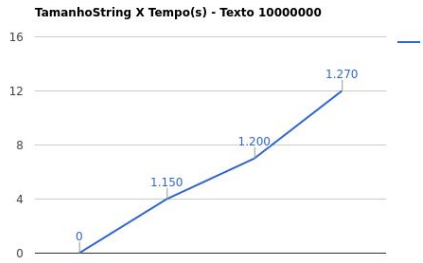
Nossa implementação utiliza uma pequena otimização. Graças a estrutura utilizada não é necessário alocar espaço para todo o alfabeto, somente se necessário. Logo, em cado médio nossa implementação utiliza menos espaço. Entretanto no pior caso (aquele em que cada nó terá que alocar todo o alfabeto), nossa complexidade iguala ao do tipo R-Way, sendo: O(AN), onde A corresponde ao tamanho do alfabeto e N ao número de string.

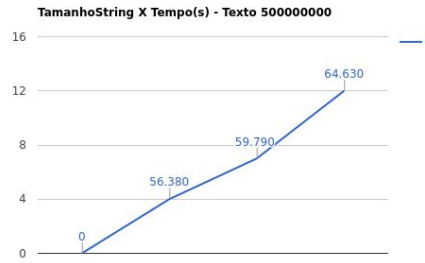
3 Análise Experimental

Para esta seção, foram feitas uma série de testes, ora fixando o tamanho do texto e variando o tamanho das strings (word size), tamanho das stringse variando o tamanho do texto. O tamanho do dicionario foi mantido em 100 sempre. As tabelas de dados se encontram em anexo (no final desse arquivo).

3.1 Tempo x Tamanho da String

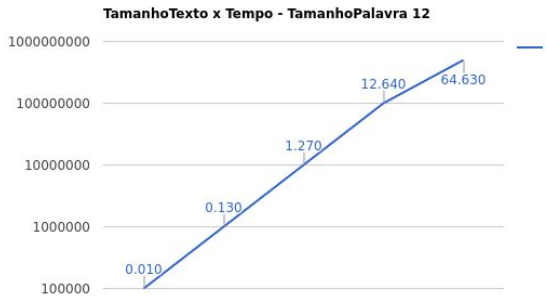
Observe o gráficos gerados a partir dos dados colhidos.





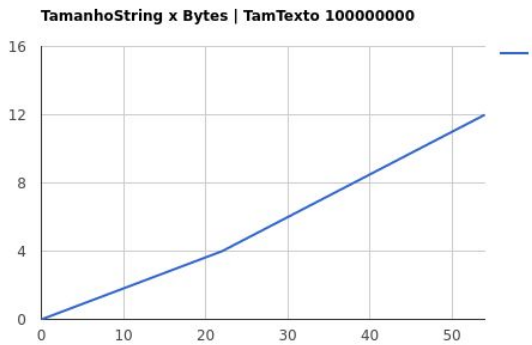
Como podemos notar, a função de tempo com relação ao tamanho da string se comporta de forma linear O(n), comprovando nossa analise no item anterior. As pequenas variações são frutos da quantidade amostral.

Observando o gráfico a seguir, temos que o tempo gasto com relação ao tamanho do texto também é linear .



3.1 Espaço x Tamanho da String

Conforme esperado, o tamanho de bytes alocados em memoria, cresce linearmente conforme cresce o tamanho da string. Isso pode ser visto no gráfico a seguir:



4 Conclusão

Nesse trabalho foi resolvido o problema de contagem de palavras de um dado dicionario em um texto através da implementação de uma árvore Trie. Embora a estrutura adotada seja diferente do convencional, a implementação foi bem sucedida e sua eficiência provada na analise experimental. A análise de complexidade teórica de tempo foi comprovada através de experimentos com entradas grandes o suficiente para analisar o comportamento assintótico.

5 Anexo

Word size 4			
Tamanho do texto	**Tempo**	**Bytes Allocated**	**Allocs**
100000	0.010	21,92	422
1000000	0.110	21,96	423
10000000	1.150	21,84	420
100000000	11.530	21,84	420
500000000	56.380	27,72	417

Word size 7			
Tamanho do texto	**Tempo**	**Bytes Allocated**	**Allocs**
100000	0.010	33,92	722
1000000	0.120	33,6	714
10000000	1.200	33,88	721
100000000	11.750	33,84	720
500000000	59.790	34,04	725

Word size 12			
Tamanho do texto	**Tempo**	**Bytes Allocated**	**Allocs**
100000	0.010	54	1,224
1000000	0.130	53,76	1,218
10000000	1.270	53,8	1,219
100000000	12.640	54	1,224
500000000	64.630	53,92	1,222

6 Referências

http://simplestcodings.blogspot.com.br/2012/11/trie-implementation-in-c.html

https://pt.wikipedia.org/wiki/Trie

https://www.cs.bu.edu/teaching/c/tree/trie/

http://www.ime.usp.br/~pf/estruturas-de-dados/aulas/tries.html

http://algs4.cs.princeton.edu/lectures/52Tries.pdf