Trabalho Prático 0 de AEDS3

Frequência de Palavras

Douglas Rodrigues de Almeida
douglasralmeida@live.com

1. Introdução

O objetivo deste trabalho é implementar um contador de palavras específicas presentes em um determinado texto. A implementação usa um tipo abstrato de dados baseado em árvore que torna as pesquisas de cadeias de caracteres bastante rápidas pois, independentemente da quantidade de palavras a serem pesquisadas no texto, o algoritmo de pesquisa percorre todo o texto uma única vez.

2. Implementação

Estrutura de Dados

Para gravar a frequência de palavras no texto, foi implementada uma estrutura de dados conhecida como trie, ou árvore de prefixos, aqui chamada de Árvore Digital. A trie guarda palavras tal qual um dicionário ordenado.

A árvore digital é uma árvore M-ária cujos nós são vetores de m componentes, com campos correspondentes aos caracteres que formam uma palavra. No TAD implementado, cada nó possui um vetor de 26 itens, cada item representando uma letra do alfabeto. Para encontrar uma palavra, o algoritmo de pesquisa caminha pela árvore de acordo com os seus caracteres. A palavra é representada pelo caminho na árvore da raiz até um nó determinado.

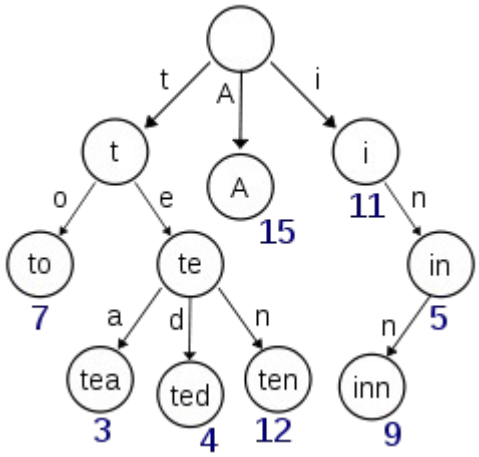

Figura 1: Exemplo de trie para as palavras "A", "to", "tea", "ted", "ten", "i", "in" e "inn".

Eventualmente, podemos ter prefixos armazenados na árvore que também são palavras. Para diferenciá-los, cada nó da árvore possui um marcador que informa se o caminho percorrido até ali indica uma palavra ou apenas um prefixo.

A estrutura de uma árvore digital não depende da ordem de inserção nem de remoção das palavras.

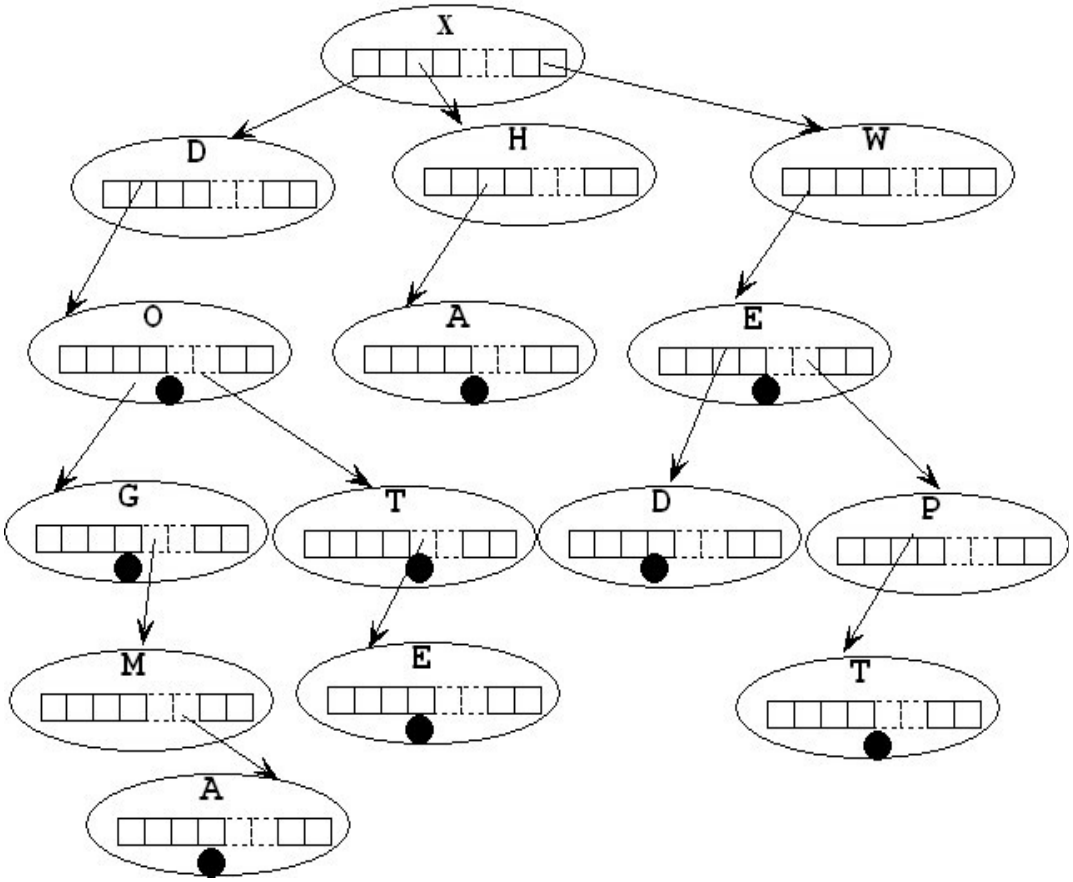

Figura 2:Um marcador indica que os prefixos “XDO”, “XDOG”, “XDOT”, por exemplo, são, também, palavras armazenadas na árvore. O prefixo “XDOGM” não é uma palavra armazenada na árvore.

Campos

O TAD Árvore Digital é composto apenas pela pela raiz da árvore. Cada nó da árvore possui os seguintes campos:

`Alfabeto: Vetor que representa os 26 caracteres do alfabeto.`

`Contador: Guarda a frequência da palavra em um determinado texto representada pelo caminho até o nó.`

`Prefixo: Diferencia prefixos de palavras armazenadas na árvore.`

Funções e Procedimentos

O TAD Árvore Digital possui as seguintes funções:

`TArvoreDigitalNo_Criar`
Aloca o nó da árvore na memória.

```
void TArvoreDigitalNo_Destruir(TArvoreDigitalNo* PArvoreDigitalNo);
```
Desaloca um no da árvore da memória e limpa a variável.

`TArvoreDigital_Criar`
Aloca uma árvore digital vazia na memória.

`TArvoreDigital_Destruir`
Desaloca a árvore digital da memória e limpa a variável.

`TArvoreDigital_Adicionar`
Adiciona uma palavra na árvore.

`TArvoreDigital_CarregarArquivo`
Carrega um dicionário de palavras na árvore armazenadas em um arquivo.

`TArvoreDigital_CarregarString`
Carrega um dicionário de palavras a partir uma cadeia de caracteres.

`TArvoreDigital_CarregarTela`
Carrega um dicionário de palavras digitado no prompt de comando.

`TArvoreDigital_ContarPalavrasArquivo`
Carrega um texto para contagem de palavras de um arquivo.

`TArvoreDigital_ContarPalavrasString`
Carrega um texto para contagem de palavras de uma cadeia de caracteres.

`TArvoreDigital_ContarPalavrasTela`
Carrega um texto para contagem de palavras do prompt de comando.

`TArvoreDigital_ExibirContador`
Exibe o contador de uma determinada palavra armazenada na árvore.

Programa Principal

O programa principal solicita ao usuário a quantidade de palavras que serão gravadas na árvore. Em seguida, deverão ser informadas as palavras que serão adicionadas à árvore. Como pré-requisito do trabalho, os contadores deverão ser exibidos na tela conforme a ordem de entrada de cada palavra no árvore. Assim, elas são registradas em um vetor a parte para buscar a ordem de entrada no final do programa.

Logo após, é solicitado a quantidade de palavras do texto onde será realizado a pesquisa. Por fim, é solicitado o texto em si, onde o aplicativo efetuará a pesquisa.

Após a realização da pesquisa é exibido um vetor numérico com as frequências de cada palavra do dicionário conforme sua ordem de entrada na árvore.

Organização do código

O código está dividido em 4 arquivos principais. Boolutils.h implementa o tipo booleano para facilitar a programação e o arquivo principal.c implementa o programa principal. Por fim, a árvore digital é implementada nos arquivos trie.h e trie.c.

Para executar o programa, digite o seguinte comando:

./tp0

3. Análise de Complexidade

Em cada nó da árvore digital existem R ponteiros alocados, ainda que apontando para nulo, onde R é o tamanho do alfabeto. Contando com a raiz, temos R(+1)N ponteiros, o que implica muita memória alocada para árvores grandes. Por exemplo, se usarmos caracteres Unicode, podemos ter 65.536 alocações para cada nó da árvore. Se muitas palavras curtas compartilham prefixos em comum, a complexidade de espaço pode ser sublinear.

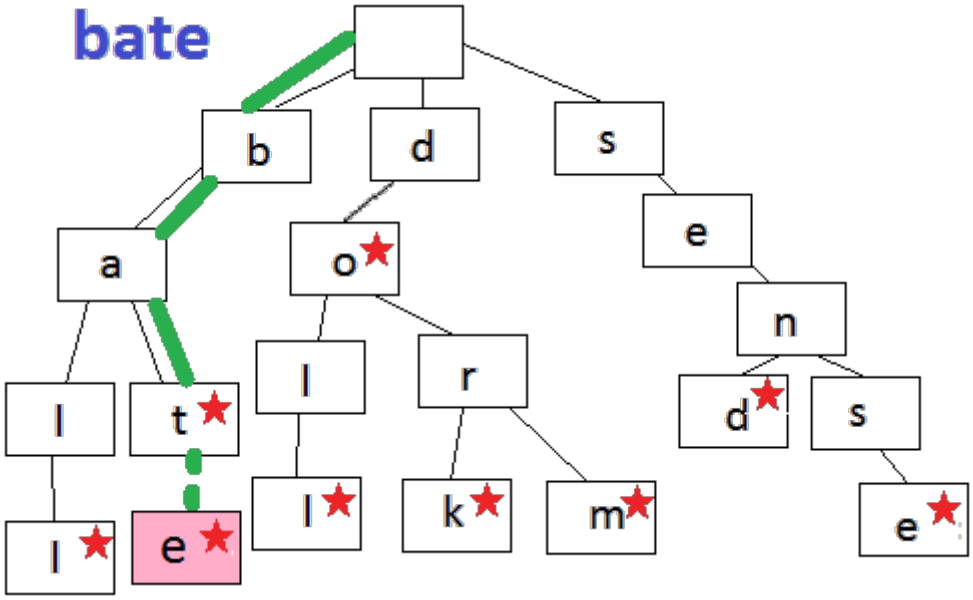

Figura 3: Pesquisa com sucesso da palavra 'bate' na árvore digital.

Para uma pesquisa na árvore com sucesso, caminha-se pela árvore a partir da raiz passando por cada nó que representa cada caractere da palavra. Assim, a complexidade de tempo é de O(m), onde m é o tamanho da palavra que está sendo pesquisada.

Para uma pesquisa com insucesso, o caminhamento é feito, no pior caso, da raiz até o nó mais externo da árvore. Neste caso, a complexidade de tempo da pesquisa é O(log n), onde n é número de nós da árvore.

O algoritmo de de inserção de palavras na árvore é semelhante ao algoritmo de pesquisa e, portanto, não depende do número de chaves mas, sim, do tamanho da palavra. Eles são O(m), onde m é o número de caracteres de cada palavra.

Pesquisa no texto

A pesquisa de palavras no texto possui complexidade linear. Independentemente do número de palavras a serem pesquisadas, o texto é percorrido uma única vez, portanto, o algoritmo de pesquisa é O(n), onde n é o número de caracteres do texto.

Programa principal

A analise do complexidade do programa principal em função do tamanho do texto e da quantidade de palavras do texto é O(n + m), onde n é o número de palavras do dicionário e m é o número de palavras do texto.

Análise do tempo de execução

O compilador utilizado para testes foi o GCC 4.8 no sistema operacional Ubuntu 14.04 em uma máquina Intel Core 2 Duo 2.93 GHz com 4.0 GB de memória.

Os testes foi realizado variando a quantidade de palavras do dicionário e a quantidade de palavras no texto. Cada teste foi executado trinta vezes, para cada quantidade de palavras. O resultado final foi calculado baseado na média aritmética do tempo de execução de cada teste.

No primeiro teste, o tempo de execução da pesquisa foi medido em microssegundos, variando a quantidade de palavras do texto e mantendo a quantidade de palavras do dicionário fixa. Verifica-se que o tempo médio de execução varia proporcionalmente conforme o número de palavras, conforme listado na Tabela 1.

Nº de Palavras	10^1	10^2	10^3	10^4	10^5	10^6	10^7
Tempo médio (ms)	2.2	15.3	144.2	1039.2	11465.4	105300.9	1044413.0

Tabela 1: Tempo médio de execução do algoritmo de contagem de palavras em função do número de palavras do texto.

O gráfico que representa os tempos médios de execução se aproxima a uma reta. Os resultados estão dentro do esperado pela análise de complexidade onde é afirmado que o algoritmo de pesquisa possui complexidade linear.

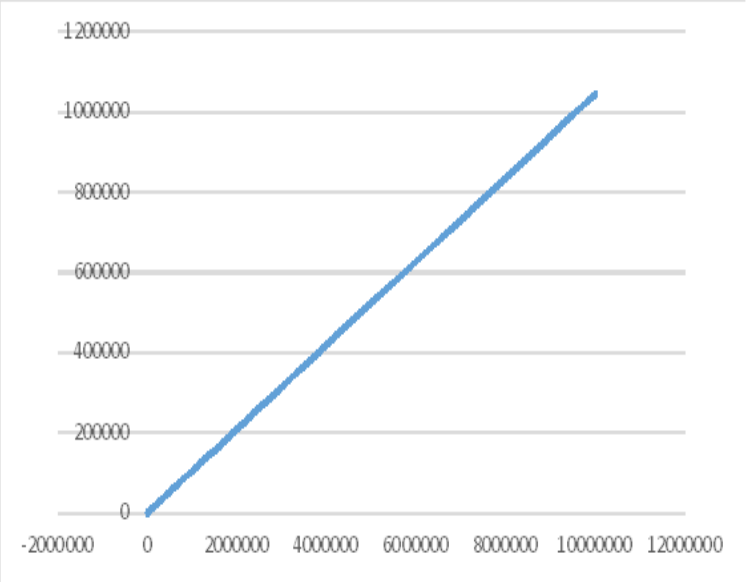

Gráfico 1: Função do tempo médio de execução se aproxima a uma função de primeiro grau.

O segundo teste foi realizado variando a quantidade de palavras do dicionário, mantendo-se fixo o número de palavras do texto.

Assim como no teste 1, verifica-se que o tempo médio de execução varia proporcionalmente conforme o número de palavras do dicionário, conforme listado na Tabela 2. Isso confirma o fato do algoritmo de inserção ser linear.

Nº de Palavras	10^2	10^3	10^4	10^5	10^6	10^7
Tempo médio (ms)	67.7	283.9	2163.2	16052.6	117216.0	1101531.8

Tabela 2: Tempo médio de execução do algoritmo de inserção na árvore função do número de palavras.

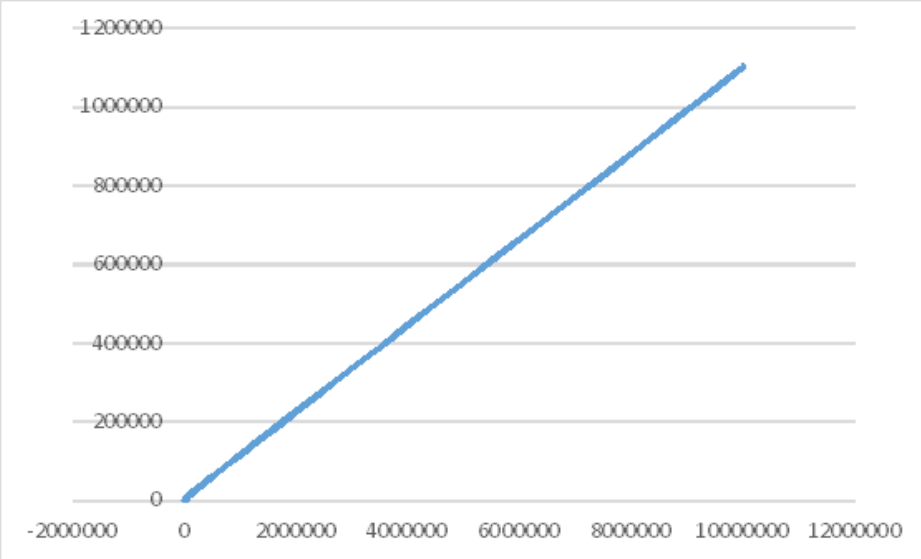

Figura 4: Gráfico da função de tempo de execução em função do tamanho do dicionário

Ainda foi realizado um teste de ocupação de memória. Comparado a um vetor de palavras de tamanho médio de 10 caracteres a árvore digital ocupa mais espaçõ, numa razão de 1 para 45, aproximadamente.

Nº de palavras	Árvore Digital	Vetor
10	25.6 KB	0.4 KB
10^2	193.8 KB	3.9 KB
10^3	2.0 MB	39.1 KB
10^4	17.3 MB	391 KB
10^5	172.7 MB	3.8 MB

Tabela 3: Comparação da memória alocada pela árvore digital e um vetor

4. Conclusão

Árvore digitais podem usadas para uma variedade de aplicações como corretores ortográficos, hifenização de textos, autocomplemento de palavras, compressão de dados, tabela de roteamento de endereços IP, manipulação de arquivos XML, índice de motor de busca, entre outros.

A grande vantagem da árvore digital é permitir a realização pesquisa de várias palavras em textos com apenas uma única caminhada. O caminhamento pela árvore é, no máximo, o tamanho da palavra a ser encontrada.

Inserir, pesquisar e remover palavras em árvores digitais são algoritmos bem rápidos que dependem somente do tamanho da palavra em questão.

Umas das desvantagens da árvore digital é seu alto consumo de memória, em grande parte desnecessário. Alfabetos grandes aumentam consideravelmente a necessidade de memória, pois cada nó da árvore aloca espaço para representar todos os caracteres deste alfabeto. Muitas palavras que compartilham prefixos podem melhorar o uso do espaço alocado da árvore.

Outro problema é a formação de caminhos de uma só direção para palavras com um grande número de caracteres em comum. Se duas palavras diferirem apenas no último caractere, elas formarão um caminho cujo o comprimento é igual ao tamanho delas, não importando quantas chaves existem na árvore.

Referências

M. Thenmozhi and H. Strimathi, n Analysis on the Performance of Tree and Trie based Dictionary Implementations with Different Data Usage Models, Indian Journal of Science and Technology, Vol 8(4) 364-375, Feb 2015.

R. Sedgewick and K. Wayne, Algorithms, 4th Edition, Addison-Wesley, 2011.

Ziviani, N., Projeto de Algoritmos com Implementações em Pascal e C, 2ª Edição, Editora Cengage Learning, 2009.

Trie. (2016, Set 05). In Wikipedia, The Free Encyclopedia. Acessado em 05/09/2016.

Anexos

Listagem dos arquivos:

```
makefile
principal.c
trie.c
trie.h
boolutils.h
```