AED II - Trabalho II

Bruno Tomé - 0011254[1], **Cláudio Menezes - 0011255**[1]

[1]Instituto Federal de Minas Gerais (IFMG)
São Luiz Gonzaga, s/nº - Formiga / MG - Brasil

ibrunotome@gmail.com, claudiomenezio@gmail.com

Abstract. *Report on the Second Working AED II , Making hum empirical Study of A tall Behavior binary search tree on Wed Are Made inserts and -form random removals and implementing ONE application que Misspellings identify a text file.*

Resumo. *Relatório sobre o segundo trabalho de AED II, fazendo um estudo empírico sobre comportamento da altura de uma árvore binária de busca na qual são feitas inserções e remoções de forma aleatória e implementando uma aplicação que identifique erros ortográficos num arquivo texto.*

1. Introdução

O primeiro objetivo deste trabalho é realizar um estudo empírico sobre comportamento da altura de uma árvore binária de busca na qual são feitas inserções e remoções de forma aleatória. O segundo objetivo desse trabalho é a implementação de uma aplicação que identifique erros ortográficos num arquivo texto.

2. Implementação

Utilizamos tanto para a parte 1, quanto para a parte 2 do trabalho, o TAD básico de árvore binária de busca disponibilizado pelo www.GeeksBR.com, utilizamos suas funções de inserção, verificação se a árvore está vazia ou não, verificação se o elemento pertence à árvore e a função que libera a árvore.

Implementamos as funções: altura, remover, inserir_palavra, quantidadeNos e busca baseados nessa agenda programada em C que encontramos: http://www.vivaolinux.com.br/script/Agenda-feita-em-C-usando-arvore-binaria.

Esse algoritmo acima deu a complementação da ideia do TAD árvore para trabalhar com string.

2.1. Funcionamento

Há um menu onde escolhe-se a execução da parte 1 ou parte 2 do trabalho:

Parte 1: Outro menu aparece, selecionamos a quantidade de nós na qual os testes serão feitos, após a seleção a função parte1 é chamada, os cálculos são feitos e impressos na tela.

Parte 2: O arquivo é lido via um diretório fixo (pasta na qual se encontra o código fonte), suas palavras são lidas por linha (há uma palavra por linha no dicionario.txt) e são inseridas na árvore. Logo após o arquivo com o texto é lido por

linha, quebrado usando a função strtok e cada palavra obtida de tiver 4 ou mais caracteres será comparado com as palavras que foram inseridas na árvore anteriormente.

2.2. Como executar o programa

Abra o Terminal e digite:

cd <DIRETÓRIO>

gcc main.c arvore.c -omain.bin -Wall -pedantic -ansi

./main.bin

No windows, você pode rodar via netbeans ou se tiver um terminal batch basta seguir os comandos acima.

3. Descrição dos testes realizados

3.1. Parte 1

	1000	5000	10000	16000	32000	50000
Altura inicial	22	24	31	29	35	37
Altura média após operações rodarem 100 vezes	22	28	31	33	36	38
Quantidade de nós média após operações rodarem 100 vezes	1499	7501	15002	24000	48000	74996

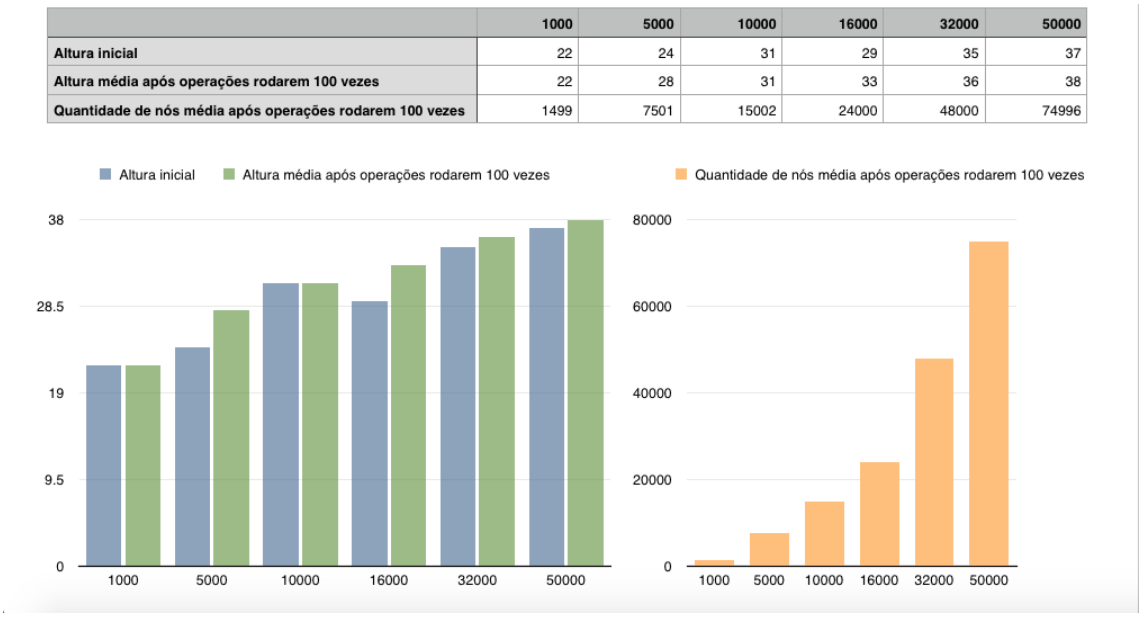

Figura 1. Resultado dos testes realizados, analisando altura média e quantidade média de nós após 100 vezes

Os testes comprovam que a teoria está correta, a quantidade de nós da árvore após o número inicial de nós, alternando inserções e remoções, aumenta na faixa de 50%. Em nossos testes, esse valor alterou apenas numa margem de erro de 4 nós para baixo ou para cima.

3.2. Parte 2

A segunda parte do trabalho consistia em criar uma aplicação que identifique erros em um texto, com base em um arquivo texto com palavras corretas. A princípio pensamos que seriam necessárias centenas de comparações, mas a sacada é que, com as palavras devidamente inseridas na árvore, basta ler o

texto e ir comparando as palavras com 4 caracteres ou mais com as palavras já inseridas na árvore, se ele não encontrá-la, assinalamos como um erro. Usamos a função strncmp, esse n a mais é o tamanho da string. Apenas com a strcmp não estava funcionando.

Feita a comparação, foi preciso criar um contador para a linha e um para a coluna de onde está a palavra não encontrada. A cada impressão de palavra com erro, uma variável é incrementada, a partir dela sabemos quantas palavras estão erradas após o término do texto.

4. Conclusão

Dificuldades encontradas neste trabalho: Demoramos bastante a encontrar o erro da função busca na parte 2 do trabalho, foi preciso utilizar a strncmp para corrigir o problema. Na parte 1, percebemos que utilizando o rand() % 65535, e rodando a árvore com 50000 nós, o número final de nós diminuía em vez de aumentar. Isso acontecia devido a seguinte situação: A maioria dos números de 0 a 65535 já havia sido inserida na árvore, então ele encontrava mais nós para remover do que para inserir. Resolvemos isso deixando apenas o rand().

A partir dos testes realizados na parte 1 do trabalho, também pudemos afirmar que a teoria está correta. A quantidade de nós final da árvore é em média 50% maior do que a original, antes das inserções e remoções aleatórias.

5. Bibliografia

TAD arvore_binaria = http://www.geeksbr.com/2012/01/programacao-em-c-arvore-binaria.html

Agenda telefônica que utiliza árvore binária de busca, código em C = http://www.vivaolinux.com.br/script/Agenda-feita-em-C-usando-arvore-binaria