Relatório - REO 5

Arthur Henrique Sousa Cruz

Universidade Federal de Lavras

1 Execução

Um arquivo de entrada contendo os valores a serem inseridos devem ser passados por linha de comando. Os resultados serão exibidos na tela do terminal caso seja passado o parâmetro ou não. Para compilar e executar o código utilize os seguintes comandos:

gcc arvore_b.h -o arvore_b.o
gcc arvore_b.c -o arvore_b
./arvore_b [arquivo de entrada]

Para a execução de experimentos com diferentes ordens, é necessário alterar o valor de T no arquivo **arvore_b.h**. T é o número de filhos de uma página após uma divisão ocorrer (número máximo de filhos dividido por 2).

2 Organização de arquivos

Os arquivos deste projeto foram organizados da seguinte forma:

- **gerador_de_testes**: Arquivo que implementa um gerador de números aleatórios;
- **arvore_b.h**: *Header* do arquivo principal
- **arvore_b.c**: Arquivo principal com a implementação da Árvore B.
- **teste_tamanho_1000.in**: Arquivo de teste utilizado para este relatório;
- **bd_file**: Arquivo binário de saída do programa.

3 Implementação

Os algoritmos e estruturas foram implementados em *C*. A implementação é uma modificação de https://gist.github.com/lylemalik/4283086. Na implementação a raiz da árvore permanece em memória mas também foi utilizado o cabeçalho sugerido na atividade. Além disso, os acessos em memória secundária para a leitura da raiz não ocorrem, porém eles foram considerados nos cálculos de complexidade.

4 Análise do algoritmo

A instância gerada continha 1000 valores aleatórios para inserção. O primeiro dos testes foi realizado com uma árvore de ordem 100, e foram necessários 1985 acessos à memória para realizar as buscas de todos os elementos inseridos. Vale lembrar que, desses 1985 acessos, 1000 foram para buscar a raiz. Isso significa que em teoria seriam necessários 1985 acessos, mas como a raiz é mantida em memória, foram realizados apenas 985 acessos.

O segundo teste, com uma página de tamanho 50, necessitou de 1972 acessos, sendo 1000 para a raiz. O fato de uma ordem menor ter menor quantidade de acessos mostra que a medida que se aumenta excessivamente o número de chaves na página, a árvore começa a se aproximar de uma busca sequencial. É essencial lembrar que há um *trade-off* entre o as buscas verticais e horizontais na Árvore B.

O resultado obtido também comprova a complexidade de tempo da Árvore B: desconsiderando a ordem da árvore, $\Theta(\log n)$ (pois a altura será sempre $\log n$). Considerando a ordem, tem-se o pior caso de $O(T * \log n)$, pois seria necessário buscar e verificar T elementos a cada nível da árvore.