Trabalho Prático 1: Biblioteca do Filipe

Algoritmos e Estruturas de Dados III - Lucas Fonseca Mundim - 2015042134

1 Introdução

O objetivo do TP1 é aprender a trabalhar com quantidades limitadas de memória principal, fazendo uso da memória secundária do computador para tarefas que seria mais simples se não houvesse essa limitação. Especificamente nesse trabalho, a tarefa a ser realizada é a ordenação de grandes quantidades de arquivos e, portanto, deve-se adaptar um método de ordenação interna para agir de forma externa.

2 Solução do Problema

Para solução do problema, inicialmente foram recebidos os dados quantitativos com os quais deve-se trabalhar. Depois disso, criou-se um arquivo temporário para preencher com os nomes e a disponibilidade de cada livro. Para isso foi criada uma função *fillTempFile*.

Feito isso, inicia-se o processo de ordenação externa. Para esse programa foi utilizado um método análogo ao *Merge Sort*. Nele, inicialmente particionou-se o arquivo original em vários arquivos contendo o número máximo possível de livros simultâneos na memória. Depois ordenou-se esses arquivos de forma interna utilizando a função *qsort* da biblioteca padrão. Após a ordenação interna, inicia-se o processo de intercalação, onde o pro-

grama compara arquivos dois a dois, copiando o menor *string* comparado para um arquivo de saída até acabarem os livros nos arquivos comparados. Isso é feito sucessivamente até o fim dos arquivos, onde o último arquivo, então ordenado, é renomeado para caracterizá-lo como tal e todos os arquivos anteriores são excluídos por uma chamada do sistema.

Após obter o arquivo final ordenado, o processo de separação em estantes é iniciado. Cria-se n (onde n é o número de estantes passado inicialmente ao programa) estantes e elas são preenchidas em sua totalidade com os livros presentes no arquivo (enquanto houver capacidade). Se a estante não for inteiramente preenchida, o resto de suas posições permanece vazio e, se a estante não possuir nem ao menos um livro, seu arquivo ainda é criado e mantido vazio. Enquanto as estantes são criadas, é criado simultâneamente um índice contendo, a cada linha, o primeiro e o último livros de cada estante e, caso a estante esteja vazia, imprime-se um único #.

Por último o programa procura os livros requisitados pelos alunos. Para isso, para cada livro requisitado é chamada uma função que procura em qual estante o livro estaria, comparando o nome dele com os livros presentes no índice daquela estante, uma a uma (busca sequencial). Caso o nome do livro venha antes ("seja menor") que o primeiro livro de uma estante, aquele livro não está presente na biblioteca e a busca é encerrada. Caso ele venha depois do último livro da estante, o programa segue procurando nas estantes seguintes. Caso o livro esteja entre ambos, o programa procura então na estante definida pelo nome do livro (novamente de maneira sequencial) e, caso encontre, retorna a posição (caso o livro esteja disponível) ou que ele está emprestado. Em último caso, se o programa procurar um livro em uma

estante vazia (marcada com um #), a busca é encerrada e o livro é dado como inexistente na biblioteca.

3 Análise de Complexidade

3.1 Complexidade de Tempo

3.1.1 Ordenação

Considerando que n seja o número de livros e m seja o tamanho da memória, temos as seguintes complexidades:

- Particionamento: $O(n*m)$, dada a distribuição de n livros em arquivos de tamanho m
- Ordenação interna: $O(n*log(n))$, dada a complexidade de caso médio do *quicksort*
- Intercalação: $O(n/m)$, dado o número de vezes que os arquivos devem ser intercalados dois a dois;

3.1.2 Estantes

Considerando que n seja o número de livros, que e seja o número de estantes e s seja o número de livros que cabem em cada estante, temos as seguintes complexidades:

- Criação das estantes: $O(e*s)$, dado que são feitas e estantes e preenchidas com s livros

- Criação do índice: $O(e)$, dado que para a criação do índice apenas são necessárias as estantes e ele é feito junto da criação delas.

3.1.3 Busca

Seja r o número de requisições, e o número de estantes e s o número de livros que cabem nas estantes:

- Busca de livros: $O(r * e * s)$, dado que a busca pelo livro é feita de forma sequencial, primeiramente no índice e depois nas estantes.

3.1.4 Complexidade Total

Dadas as ordens de complexidade de tempo acima, a complexidade total é a maior dentre elas, portanto, $O(max((n * m), (n * logn), (n/m), (r * e * s), (e * s)))$.

4 Avaliação Experimental

4.1 Complexidade de Espaço

São criados n livros separados em e estantes, portanto temos as seguintes complexidades de espaço:

- Livros: $O(n)$
- Estantes: $O(e)$

4.2 Análise Experimental

Para uma análise experimental do programa, foram consideradas duas variáveis como importante na hora de avaliar o tempo: a quantidade de livros na biblioteca e a quantidade de requisições dos alunos.

4.2.1 Variação de Número de Livros

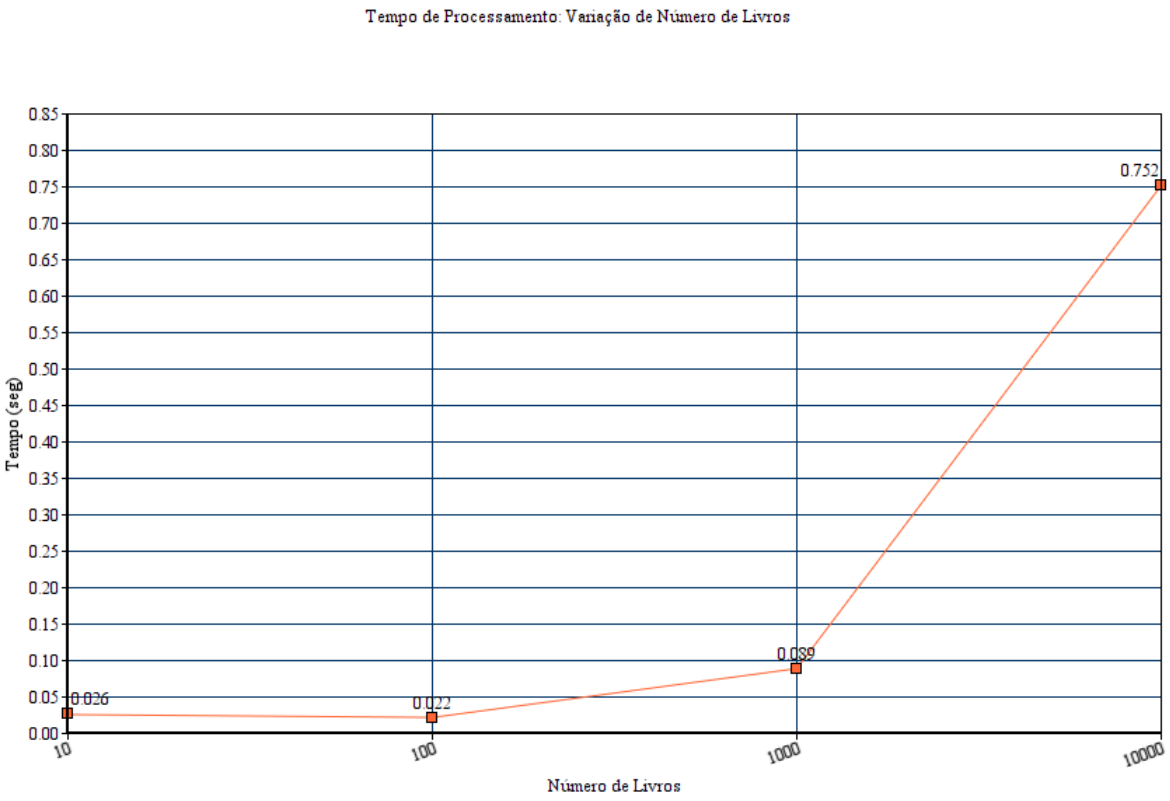
Ao testar a variação de número de livros, foram mantidos constantes os seguintes dados:

- Memória: 3 livros por vez
- Requisições: 150 requisições
- Tamanho das Estantes: 5 livros

Dado isso, foram alterados os seguintes dados:

- Número de Livros: $b = 10^n$, com b indicando total de livros e n iniciado em 1 e incrementado em 1 a cada teste.
- Número de Estantes: $b/5$, para que todos os livros coubessem nas estantes. Vale notar que, nesses testes, todas as estantes estão sempre cheias em sua totalidade.

Segue abaixo um gráfico com alguns os dados de alguns testes feitos.



Com o gráfico podemos ver a natureza exponencial do tempo de execução do programa relativo à quantidade de livros inseridos.

4.2.2 Variação de Número de Requisições

o testar a variação de número de requisições, foram mantidos constantes os seguintes dados:

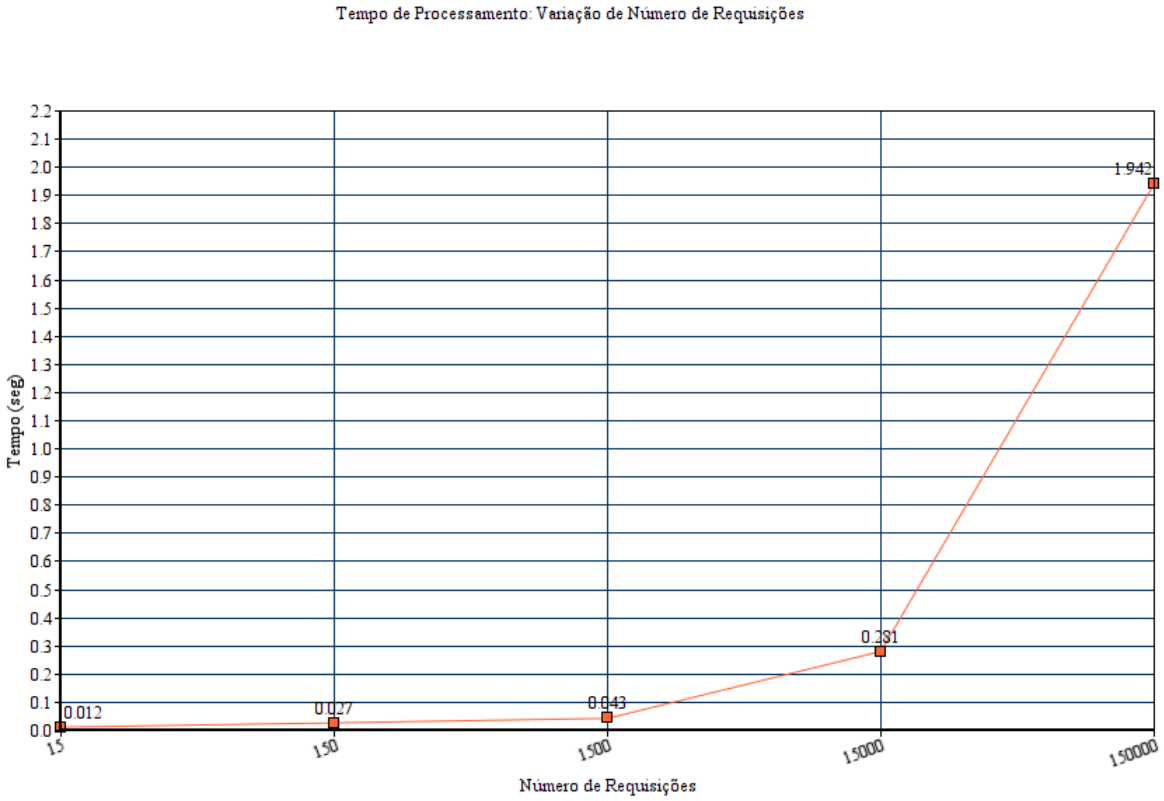
- Memória: 3 livros por vez;
- Número de Livros: 100 livros;

- Tamanho das Estantes: 5 livros;
- Número de Estantes: 30 estantes;

Dado isso, foram alterados os seguintes dados:

- Número de Requisições: $r = 15 * 10^n$, com r indicando total de requisições e n iniciado em 0 e incrementado em 1 a cada teste;

Segue abaixo um gráfico com alguns os dados de alguns testes feitos.



Com o gráfico podemos ver a natureza exponencial do tempo de execução do programa relativo à quantidade de requisições feitas pelo usuário.

5 Conclusão

O trabalho permitiu concluir que lidar com uma grande quantidade de dados pode se tornar algo demorado quando lidamos com memória limitada, tanto para ordenação quanto para a pesquisa em um arquivo já ordenado. Isso se deve à quantidade de acessos à memória secundária necessária, já que nem todo o arquivo cabe de uma vez na memória principal.

O método de ordenação selecionado utilizou os conhecimentos prévios obtidos em AEDSII junto com o algoritmo de intercalação balanceada visto em AEDSIII, gerando uma espécie de *Merge Sort Externo* para ordenar todos os dados recebidos. O programa criado viabiliza a ordenação de livros em estantes de forma alfabética, visto que retorna a posição do livro requisitado de forma imediata e, mesmo com números massivos, retorna o resultado em tempo hábil.