Trabalho Prático 1: Biblioteca do Filipe

Algoritmos e Estruturas de Dados III – 2016/2

Bruno Varella Peixoto

1 Introdução

Com o objetivo de praticar os novos conhecimentos sobre hierarquia de memória, este trabalho prático consiste basicamente em uma ordenação em memória externa e busca binária. A proposta é simular um sistema de busca de livros em uma biblioteca.
A entrada passa inteiros N, M, E, L, K, que correspondem respectivamente ao número de livros da biblioteca, o limite de livros que podem estar simultaneamente em memória, o número de estantes da biblioteca, o limite de livros por estante e o número de consultas. Em seguida são passados os N títulos dos livros que a biblioteca possui com uma flag indicando se estão disponíveis ou emprestados, e os K títulos que devem ser pesquisados.
O programa deve gerar um arquivo chamado *livros_ordenados* com todos os livros da biblioteca ordenados em ordem alfabética, E arquivos que representam as estantes, também ordenados e um índice que contém E linhas, cada uma com o título do primeiro e do último livro de cada estante.
A saída deve ser da forma:

- *disponivel na posicao P na estante X* (caso a biblioteca tenha o livro e ele não esteja emprestado. P é a posição do livro (a partir de 0) na estante X).
- *emprestado* (caso o livro exista na biblioteca mas esteja emprestado).
- *livro nao encontrado* (caso a biblioteca não tenha o livro em seu acervo).

Portanto, o desafio está em ordenar os livros, respeitando o limite M que pode estar em memória primária.

2 Solução do Problema

Inicialmente, os parâmetros da entrada são lidos e em seguida os livros são armazenados em um arquivo *all_files*. Este arquivo é temporário e serve como entrada para a função *void sort_all_books()*.
O próximo passo é justamente chamar essa função que irá gerar o arquivo *livros_ordenados*. O método de ordenação escolhido foi o merge sort externo por ser um método muito conhecido e utilizado em sistemas reais que precisam tratar entradas grandes. Além disso, é um algoritmo de fácil entendimento e implementação muito mais simples se comparado ao quicksort externo.
Basicamente foi feita a divisão da entrada em partes que cabem na memória, respeitando o limite M dado. Em seguida, cada uma dessas partes é ordenada em memória principal, usando a função qsort da biblioteca padrão. A medida que um pedaço é ordenado, ele é salvo em um arquivo temporário. Este processo é feito pela função *void create_sorted_temp_files()*. Estes arquivos são então passados para a função *void merge_sorted_files()* que faz o merge desses arquivos e cria efetivamente o *livros_ordenados*. O merge é feito de forma simples, todos os arquivos temporários são percorridos e o menor livro é extraído, isso se repete até todos os livros estarem ordenados no arquivo final. É importante notar que apenas o primeiro livro de cada arquivo temporário é comparado, pois estes já estão ordenados.
O próximo passo é criar as estantes e o índice. As estantes são criadas pela função *int create_shelves()* que recebe o arquivo *livros_ordenados* e grava de L em L livros em cada estante, até todos os livros terem sido lidos. O índice é criado pela função *void create_index()* que apenas lê o primeiro e último livro de cada estante e grava os devidos títulos no índice.

Finalmente, resta imprimir a saída. Essa parte do problema é tratada pela função *void answer_requests()*. Para cada requisição dos alunos, a função *int search_index()* é chamada e retorna a possível estante que o livro se encontra. O que foi feito é uma busca linear no índice, verificando em qual intervalo o título procurado se encaixa. Em seguida, com a possível estante encontrada, a função *int binary_search_shelf()* se encarrega de fazer uma busca binária para encontrar o livro.

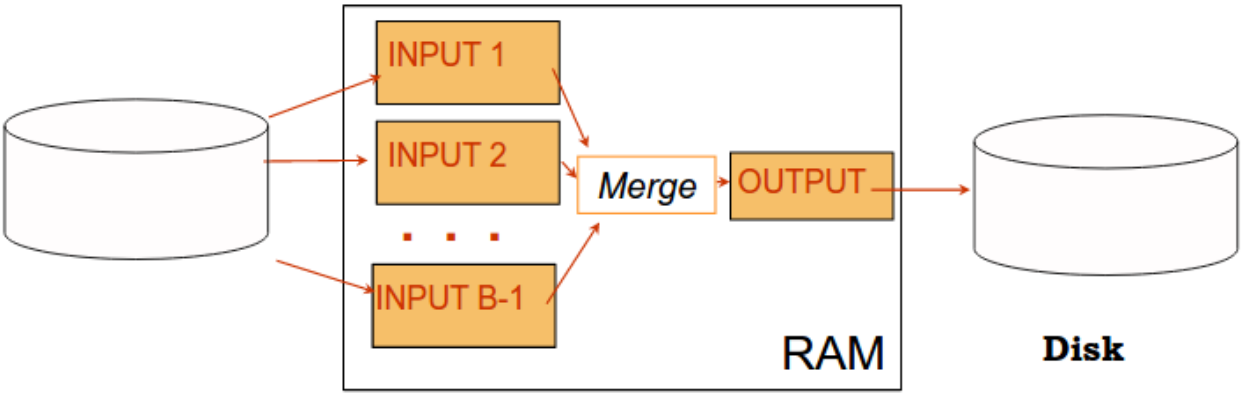


Figura 1: Exemplo de Merge Sort Externo

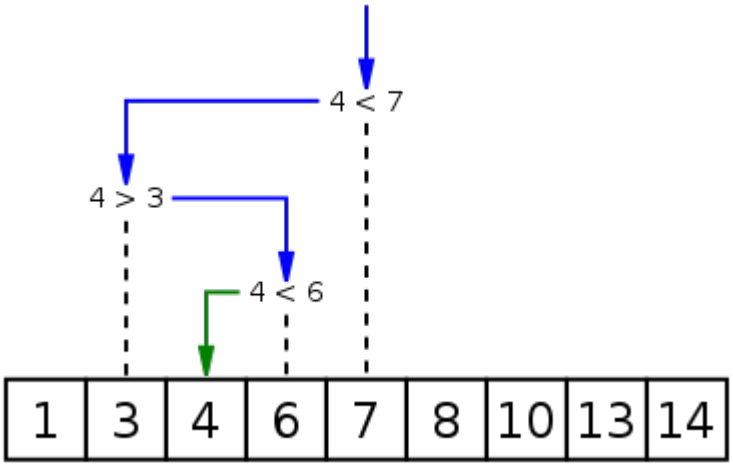


Figura 2: Exemplo genérico de Busca Binária

3 Análise de Complexidade de Tempo e Espaço

Para ser melhor analisado, este trabalho deve ser dividido em duas partes: ordenação e pesquisa.
A ordenação é feita pelas funções:

- *(1) void create_sorted_temp_files()*
- *(2) void merge_sorted_files()*

A função (1) chama a função *qsort* N/M vezes. Esta função não tem complexidade definida, mas acredita-se que seja uma implementação do quicksort. Desta forma, a complexidade de tempo da função (1) é O(N/M * (M*log(M))).
A função (2) lê 1 livro de cada arquivo temporário criado em (1) para selecionar o menor e repete esse processo até todos os livros terem sidos copiados para o arquivo final. A complexidade de tempo é então O(N * N/M).
Como a N/M pode ser N no pior caso, A ordenação é de ordem quadrática $O(N^2)$.

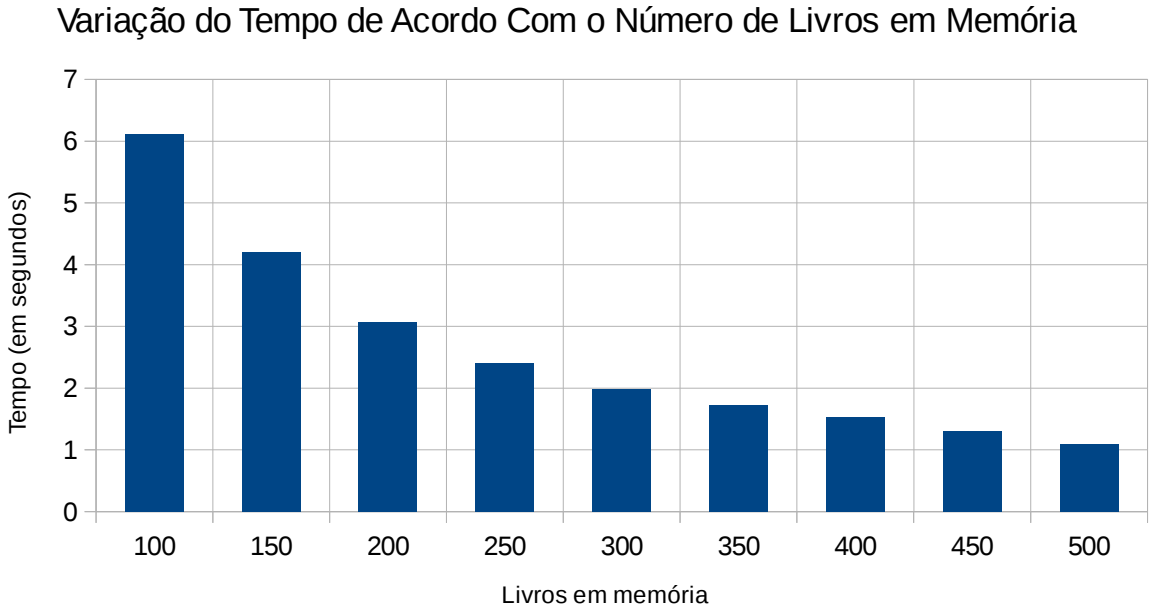
A pesquisa é divida em pesquisar qual estante o livro possivelmente está e depois, procurar o livro dentro dela.
Para encontrar a possível estante, é feita uma busca linear no índice, o que custa O(E).
A busca dentro da estante é binária e tem custo O(log(L)).

Assim, a complexidade de tempo total do programa fica por conta da ordenação $O(N^2)$ no pior caso. Entretanto, na prática, o pior caso nunca irá ocorrer pois sempre irá caber mais de 1 livro em memória (O(N * N/M)).

A complexidade de espaço desse programa é linear de acordo que a quantidade total de livros da biblioteca, pois todos os arquivos criados dependem diretamente de N. Portanto, a complexidade total de espaço é O(N).

4 Análise Experimental

Para fazer a análise experimental deste trabalho, foram criadas entradas do tipo:
100000 M 1000 100 500
Ou seja, todos os parâmetros foram fixados, exceto o número de livros em memória que foi variado entre 100 a 500, em intervalos de tamanho 50. Podemos ver o resultado no gráfico abaixo:



O teste correu dentro do esperado, uma vez que já sabíamos que quanto maior a quantidade de livros em memória primária, menor seria o tempo gasto na execução do programa. O sistema operacional utilizado foi o Fedora 24, com compilador gcc. O computador é um Core i7 4770 4,0Ghz com 16Gb de memória RAM.

5 Conclusão

Neste trabalho foi criado um sistema de busca de livros em uma biblioteca. A maior dificuldade foi em implementar o algoritmo de ordenação externa de forma eficiente, pela dificuldade de lidar com vários arquivos temporários no merge sort. Porém, tudo ocorreu dentro do esperado e a análise experimental confirmou a análise teórica. É importante perceber que este programa irá rodar muito bem em qualquer computador com uma quantidade razoável de memória, pois como foi testado, em uma biblioteca com 100000 livros, para um M > 500 o programa executa em menos de 1 segundo. Outra observação é que caso isso fosse um sistema real, o tempo de pesquisa importaria muito mais que o tempo de ordenação, pois seria necessário ordenar apenas 1 vez os livros e todas as operações seguintes seriam de busca.