Trabalho Prático 1
Biblioteca do Filipe

Gilson Urbano Ferreira Dias - 2015430959
guf@ufmg.br

1. Introdução

O problema de organização é motivo de estudo em várias áreas do conhecimento, na computação são propostas técnicas a fim de solucionar o problema utilizando o mínimo de recursos computacionais possível, o caso especial de organização que iremos propor uma solução, é sobre a organização dos livros de uma biblioteca e a recuperação dos detalhes dessa organização, para que possamos melhorar um sistema que recebe pedidos de empréstimos dos livros, em termos computacionais o problema principal será manipular o acervo, sendo que ele não pode ser transferido por completo para a memória principal, pois o número de livros no acervo é muito grande, teremos que ordenar os livros do acervo, separá-los em estantes, simulando a biblioteca física, por fim receberemos entradas com os nomes dos livros que os alunos desejam solicitar o empréstimo, deveremos retornar detalhes sobre a localização do livro se está disponível ou retornar uma mensagem em caso de indisponibilidade.

2. Solução do Problema

A implementação foi criada a partir da definição do Tipo Abstrato de Dados (TAD), com duas estruturas, uma de nome Livro, irá armazenar informações sobre os livros do acervo a ser trabalhado, nessa estrutura temos uma variável título que será preenchido com o título de cada livro, outra variável do tipo inteiro que irá armazena a informação sobre a disponibilidade de cada livro, a outra estrutura foi definida com intuito de melhorar o controle e encapsulamento do programa, ela tem o nome Variáveis, como o nome indica ela irá armazenar as variáveis que controlam a execução do programa e são definidas de acordo os requisitos da biblioteca, e do computador utilizado para executar o programa, ela tem cinco variáveis do tipo inteiro N, M, E, L, K, e elas indicam: número de livros da biblioteca, número de livros que o computador suporta em memória primaria, número de estantes da biblioteca, número de livros suportados por cada estante, número de consultas solicitadas pelos alunos, respectivamente.

2.1 Funções e Procedimentos

As funções foram declaradas da seguinte forma no Tipo Abstrato de Dados:

int comparador(const void *a, const void *b); : Essa função tem como objetivo auxiliar a função qsort(), que é uma função padrão de ordenação em memória primária da linguagem C, a função comparador() irá realizar as comparações das strings que contém os títulos dos livros a serem ordenadas, para que a ordenação ocorra de maneira de correta.

void salvaArquivo(char *endArquivo, Livro *livros, int tamanho); : Função simples, apenas salva os livros e seus detalhes em um arquivo binário, salvaArquivo(), será utilizada na função ordena().

int ordena(char *endArquivo, Variaveis controle); : Irá ordenar em memória primária a quantidade de livros que o computador suportar (M), para realizar essa ordenação

utilizamos a função qsort(), que é padrão da linguagem C, depois de ordenados salvamos os livros em arquivos temporários, que serão trabalhados pela função abaixo.

void merge(char *endArquivo, char *entrada1, char* entrada2); : Nessa função é feita uma etapa de intercalação (merge) dos arquivos temporários criados pela função ordena(), é feita a intercalação de dois arquivos de cada vez nessa função, gerando um terceiro arquivo ordenado com as chaves dos dois arquivos iniciais, essa é a última grande função que será utilizada para fazer a ordenação externa.

void ordenacaoExterna(char *endArquivo, Variaveis controle); : Função que faz as chamadas das outras funções declaradas acima com intuito de completar a ordenação externa do acervo de livros da biblioteca, primeiramente é chamada a função ordena(); criando vários arquivos menores ordenados, em um loop esses arquivos são intercalados utilizando a função merge(), ao fim desse loop teremos um único arquivo ordenado com todos os livros do acervo, esse arquivo será binário.

void livros_ordenados(char *endArquivo); : Nessa função começamos a tornar mais visual o problema de organização da biblioteca, pegamos o arquivo binário que foi ordenado pela função ordenacaoExterna(), e passamos para o formato de texto, que pode ser facilmente visualizado por um editor de texto comum, é feita a leitura do arquivo binário, então os livros são lidos e salvos em outro arquivo em formato de texto.

void separa_livros(char *endArquivo, Variaveis controle); : Essa função também trabalha com o arquivo binário que foi gerado pela função ordenacaoExterna(), ela divide todos os livros ordenados do acervo em estantes numeradas, o formato do novo arquivo continua binário, para facilitar uma futura busca e a recuperação de informações, nessa mesma função também criamos um índice que contém o primeiro e último título dos livros de cada estante criada, o formato do arquivo índice é texto, as estantes que não foram preenchidas são representadas no índice pelo caractere '#'.

int buscaIndice(Livro *chave); : Função auxiliar da buscaBinaria(), ela pecorre o arquivo índice criado pela função separa_livros, e retorna o número da estante em que se encontra o pesquisado, caso o livro não esteja disponível a função retorna um valor negativo.

int buscaBinaria(Livro *chave); : A partir da informação do número da estante em que o livro desejado se encontra retornada pela função buscaIndice(), abrimos apenas o arquivo da estante especifica que o livro desejado se encontra , é efetuamos uma busca binário na estante, a fim de localizar o número que indica sua posição específica e retorná-lo para o usuário.

2.2 Programa Principal

No programa principal são declaradas algumas variáveis importantes, uma delas, que será utilizada ao longo de toda a execução, é a controle do tipo Variaveis. É feita a leitura da entrada padrão, pegamos os detalhes da biblioteca e dos recursos computacionais que poderemos utilizar sobre a memória primária e passamos para a variável controle.

Continuando a leitura da entrada padrão, recebemos todos os livros e seus detalhes e colocamos em arquivo binário, após isso é feita a chamada de três funções: ordenacaoExterna(), livros_ordenados(), separa_livros(), respectivamente. Nesse momento estamos aptos a recuperar as informações da biblioteca de maneira eficiente para os alunos, então é feita a leitura dos nomes dos livros que os alunos desejam buscar, então retornamos os detalhes de disponibilidade para empréstimo ou não, se o livro estiver disponível informamos sua localização exata na biblioteca, essas

informações sobre a busca dos livros são retornadas na saída padrão, por fim removemos o arquivo que auxiliou a execução do programa.

2.3 Organização do Código, Decisões de Implementação e Detalhes Técnicos

O código está dividido em três arquivos principais: TP1.c e TP1.h implementam o Tipo Abstrato de Dados enquanto o arquivo main.c implementa o programa principal.

O compilador utilizado foi o GCC 5.4.0 no Sistema operacional UBUNTU 16.04.

3. Análise de Complexidade

int comparador(const void *a, const void *b); : Na função comparador(), temos uma complexidade de tempo O(1), pois realizamos apenas comparações, essa mesma complexidade se repete para o tempo, declaramos apenas uma variável do tipo inteiro e não alocamos nenhum outro dado, com isso a complexidade de tempo é O(1).

void salvaArquivo(char *endArquivo, Livro *livros, int tamanho); : Aqui temos de salvar as estruturas que definem os livros da biblioteca em um arquivo binário, a complexidade de tempo é O((N/M)*M), sendo que N é o número de livros da biblioteca e M o número de livros que o computador suporta em memória primária, com isso temos que a complexidade de tempo é O(N), pois todos os livros da biblioteca devem ser salvos, a complexidade de espaço é O(1), pois o arquivo é salvo em memória secundaria não alocando memória primaria além da utilizada pelas variáveis locais.

int ordena(char *endArquivo, Variaveis controle); : Essa função faz a leitura dos livros do acervo e ordena M livros em memória primária então a complexidade de espaço é O(M), começamos efetuando a leitura do arquivo a ser ordenado a um custo O(N), para fazer a ordenação é utilizada a função qsort(), com complexidade de tempo O(log(M)), e depois salvamos o arquivo utilizando a função salvaArquivo() que é O(N), somando as complexidades temos: O(N) + O(log(N)) + O(N), pegando o limite superior temos que essa função é O(N).

void merge(char *endArquivo, char *entrada1, char* entrada2); : Nessa etapa da ordenação externa, armazenamos apenas duas varáveis do tipo Livro, então a complexidade de espaço é O(1), na questão do tempo é feita a leitura de dois arquivos é intercalamos eles em um outro arquivo de saída, isso dá um custo de tempo O(N), sendo que N é o quantidade de Livros nos arquivos que foram lidos e salvos.

void ordenacaoExterna(char *endArquivo, Variaveis controle); : Função principal de ordenação externa, ela chama as outras funções necessárias e completa o trabalho de ordenação, como chamamos as funções merge() e ordena(), podemos afirmar que a complexidade dessa função é: O(N) para o tempo e O(M) para espaço.

void livros_ordenados(char *endArquivo); : Como os arquivos ordenados foram salvos em arquivo binário é necessário fazer com que ele agora seja um arquivo de texto para poder visualizá-lo em editor de texto comun, com isso e feita a leitura do arquivo binário e logo após salvamos os livros em um arquivo de texto, isso dá um custo de tempo O(N), e O(1) para espaço, pois usamos apenas memória secundaria e variáveis para auxiliar a execução.

void separa_livros(char *endArquivo, Variaveis controle); : Para separar os livros continuamos com uma complexidade de tempo O(N), pois temos que ler o binário ordenado e separá-los em estantes, criamos também um arquivo índice nessa função, mas isso não altera a complexidade de tempo, como não alocamos memória primaria além das variáveis temos um custo O(1) para a complexidade de espaço.

int buscaIndice(Livro *chave); : Buscar o índice é feita de forma linear, pois percorremos o arquivo índice criado pela separa_livros() até encontrar a estante em que o livro buscado se encontra, com isso temos uma complexidade de tempo O(E), onde E é o número de estantes da biblioteca, temos uma complexidade O(1) para espaço.

int buscaBinaria(Livro *chave); : Iniciamos a busca binária chamando a função buscaIndice(), a um custo O(E), ela irá nos retornar a estante em que se devemos realizar a busca binária, no restante da busca temos um custo de tempo O(log(L)), pois podemos ter L livros em cada estante, somado as complexidade temos: O(E) + O(log(L), essa é a complexidade de tempo dessa função, a complexidade de espaço é O(1), pois apenas declaramos algumas variáveis locais.

4. Avaliação Experimental

4.1 Testes

Vários testes foram realizados com o programa de forma a verificar o seu perfeito funcionamento. Os testes foram realizados em Intel Core i5, 6Gb de memória com um Intel Core i5-4210U rodando a 1.70GHz, e sistema operacional Ubuntu 16.04 LTS.

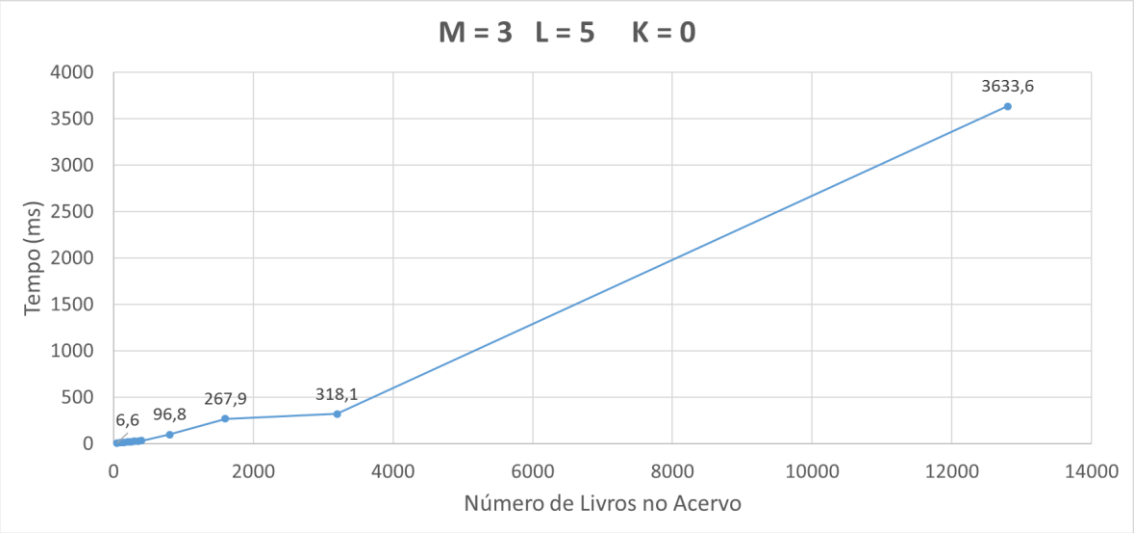

Figura 1 - Teste 1

Na figura 1 temos o resultado do primeiro teste, o número de livros suportados em memória primária foi fixando em três (M = 3), para simular um computador com poucos recursos computacionais, a quantidade de livros suportados por cada estante também é baixa (L = 5) e nenhuma pesquisa será realizada (K = 0), apenas ordenaremos os livros do acervo e dividiremos os livros em suas respectivas estantes.

O resultado é que para um número baixo de livros o programa roda rápido, mas que para um número alto de livros no acervo o tempo se torna demasiadamente alto, isso ocorre por dois motivos a quantidade baixa de livros em memória primária requer mais leituras e criações de arquivos em memória secundária o que tem um custo computacional alto, o segundo motivo é quantidade baixa de livros suportada por cada estante, requer a criação de um alto número de arquivos em memória secundaria.

Um segundo teste foi feito a fim de verificar o quanto o número de livros em memória primária afeta desempenho do programa, o número de livros suportados em memória primária agora será a metade dos livros do acervo (M = N/2), o número de livros suportados por cada estante foi mantido em cinco, e não serão realizadas buscas nesse teste, podemos observar o resultado do teste na figura 2.

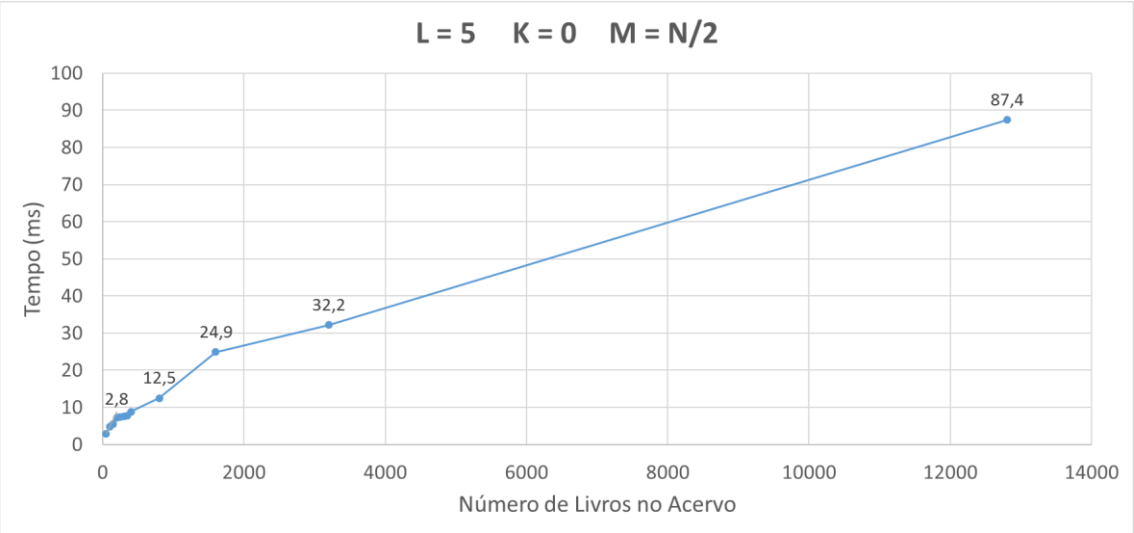

Figura 2 - Teste 2

Comparado ao teste 1, o tempo de execução caiu para menos da metade para entradas pequenas, e para a maior entrada testada de 12800 livros no acervo a quantidade de recursos computacionais utilizados foram reduzidos em incríveis 99,25%, com isso podemos afirmar que uma baixa quantidade de memória primária pode tornar o programa ineficiente, pois uma alta quantidade de leituras e escritas em memória secundaria será necessária e isso tem um custo computacional muito alto em relação a memória primária.

Continuando os testes tivemos a ideia de além do número de livros em memória primaria ser a metade do número de livros do acervo fazer também o número de livros suportados em cada estante ser a metade do número de livros do acervo, implicando o uso de duas estantes independentemente do número de livros no acervo, esse teste tenta reduzir o número de arquivos criados em memória secundária, e verificar o quanto isso afeta o tempo de execução do programa, nesse teste ainda não são realizadas as buscas dos livros, na figura abaixo temos os resultados dessas alterações.

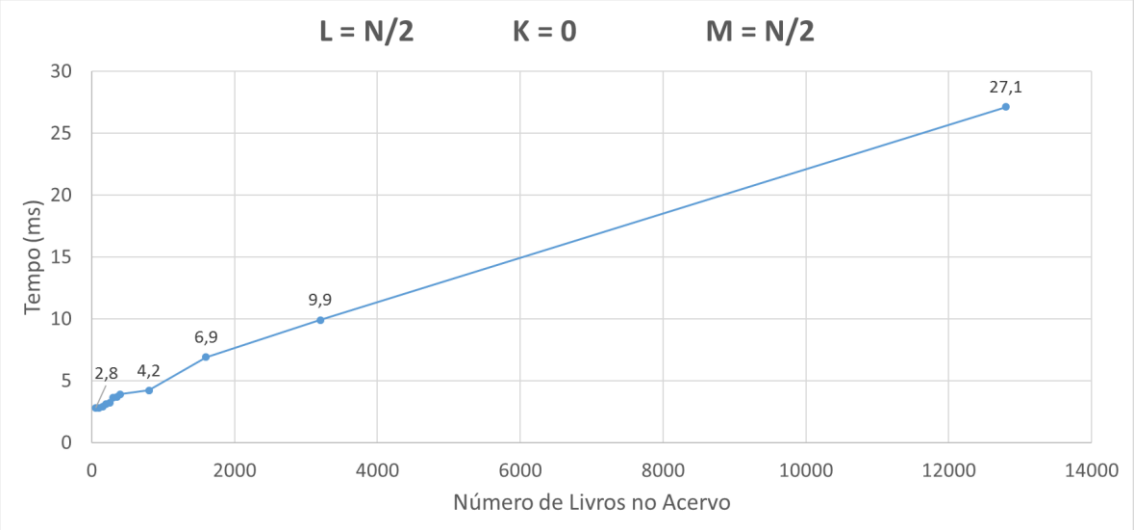

Figura 3 - Teste 3

A forma do gráfico se manteve bem parecida com o teste 2, mas a quantidade de tempo gasta diminuiu diminuiu, a redução da quantidade tempo para a maior entrada foi de 68,99%, mostrando que o custo de escrever os arquivos em memória secundaria que simulam as estantes da biblioteca é alto.

No próximo teste começamos a realizar as buscas solicitadas pelos alunos que desejam solicitar empréstimo dos livros, no primeiro teste onde serão solicitadas as buscas o número de livros suportados em memória primaria e o número de livros em cada estante será a metade do acervo (M = N/2) e (L = N/2), fixamos a quantidade de buscas em oitocentos (K = 800), o resultado pode ser observado na figura 4.

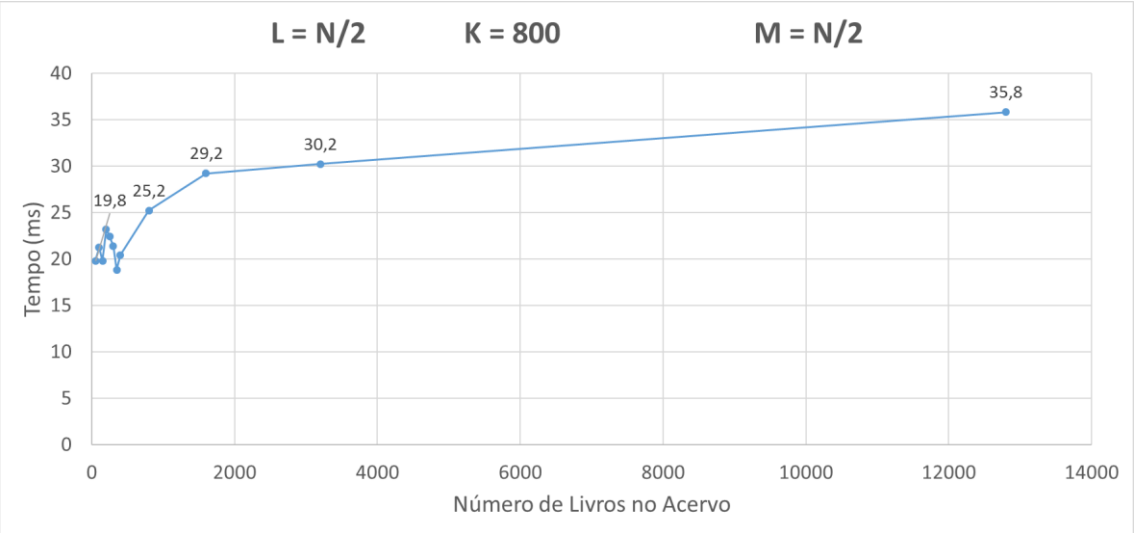

Figura 4 - Teste 4

O gráfico tem um início bastante indefinido para entradas pequenas depois estabiliza e segue um padrão logarítmico, o que pode ter acontecido pelo fato de que a busca binária tem uma complexidade logarítmica, comparado com o teste 3 onde temos os mesmos parâmetros a menos do termo de busca, o tempo nesse teste aumentou muito para pequenas entradas de livros no acervo, mas que para uma grande quantidade de livros no acervo a diferença de tempo foi pequena, na maior entrada o aumento de tempo foi de apenas 24,30% ou 8,7 ms.

Seguindo com os testes vamos fazer as buscas em cima dos mesmos parâmetros definidos para o teste 2, com o número de buscas igual a 800 (K = 800), na figura abaixo podemos obter o resultado.

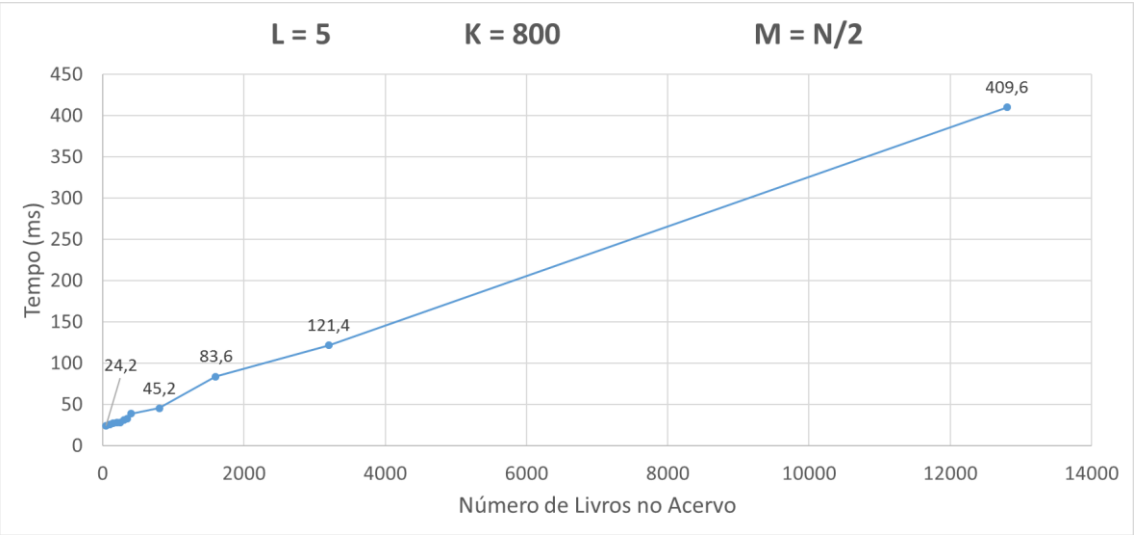

Figura 5 - Teste 5

No teste 5 foi fixado um número baixo de livros suportados por estantes (L = 5), com isso temos um aumento expressivo no tempo em relação ao teste 2, isso ocorre, pois, o número de arquivos criados para simular as estantes é muito alto, no maior caso

de teste o número de arquivos criados chega a 2560 arquivos, simulando 2560 estantes físicas, criar essa quantidade de arquivos em memória secundaria e preenche-los é muito caro, computacionalmente falando, por isso um aumento no tempo de execução.

No último teste, simulamos um computador bem limitado, como no teste 1, e iremos realizar a busca de oitocentos livros, na figura temos os resultados desse teste.

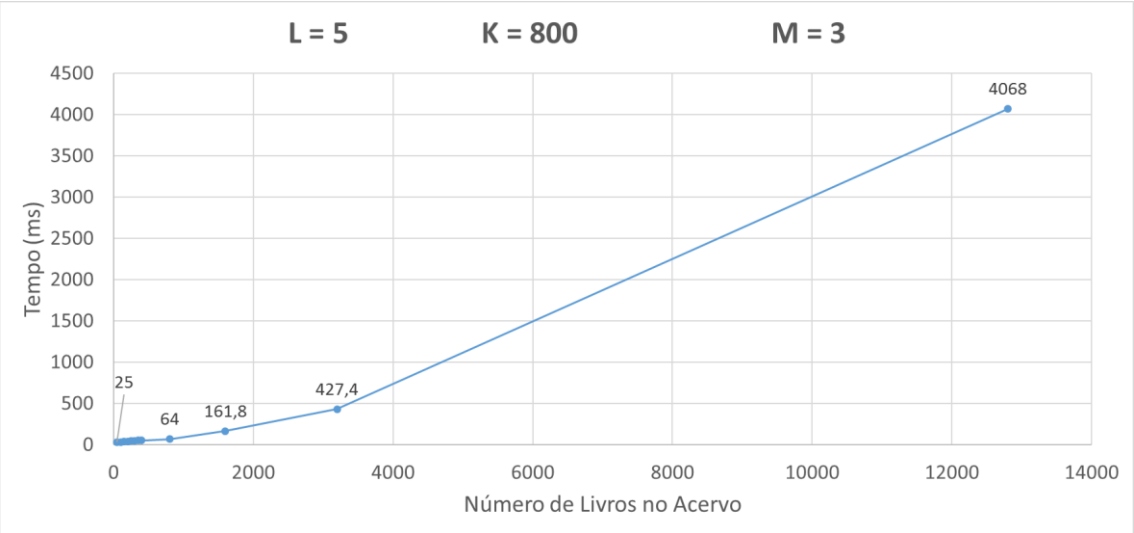

Figura 6 - Teste 6

Com um computador limitado a apenas três livros em memória primária e cinco livros em cada estante, o tempo de execução é muito alto, na maior entrada de teste o tempo de execução foi maior do que no teste 1 como esperado, pois aplicamos a busca, mas não foi muito grande isso se deve a complexidade da função de busca ser logarítmica.

5. Conclusão

A implementação ocorreu de forma tranquila, os algoritmos já haviam sidos estudados na disciplina de Algoritmos e Estruturas de Dados III, foi necessário algumas pesquisas e revisões sobre o assunto a fim de verificar alguns detalhes sobre as estruturas de dados a serem utilizadas e manipulação de utilizando as funções propostas, mas por fim pudemos concluir o trabalho, com o resultado ficando dentro do esperado.

Para obter os valores e gerar os gráficos dos testes, foram feitas dez medidas do tempo de execução para cada entrada, e então fizemos uma média simples desses tempos, os nomes dos livros foram gerados de forma aleatória tanto os inseridos no acervo quanto os livros que que desejamos realizar as buscas.

Por fim o trabalho pode ser solucionado, alguns detalhes devem ressaltados;
Como já havíamos estudado em sala de aula, operações em memória secundaria tem um custo muito alto se comparado ao custo computacional da mesma operação em memória principal, um exemplo disso e que utilizamos aqui, foi a ordenação, pudemos constatar essa diferença de custo no tempo de execução dos testes.

Algumas formas de melhorar o tempo de execução podem ser: Um computador com maior disponibilidade de memória primária, ou investir em algoritmos que sejam mais eficientes, reduzindo o número de operações em memória secundaria, com isso uma área de estudos emerge, pois vivemos em uma era que podemos obter facilmente uma grande quantidade de informações, que pode já ter ultrapassado 1 bilhão de gigabytes, sendo a maior dificuldade manipular essa informação, já que não podemos contar com a quantidade de 1 bilhão de gigabytes em memória principal para manipular esses dados.

6. Referências

[1] Ziviani, N., Projeto de Algoritmos com Implementações em Pascal e C, 4ª Edição, Editora Pionera, 1999
[2] André Backes, Linguagem C: Completa e Descomplicada, 1ª Edição, Elsevier, 2012
[3] Kyle Loudon, Mastering Algorithms with C, 1ª Edição, Editora O'Reilly
[4] Dennis M. Ritchi, C Programming Language, 2ª Edição
[5] Robert Sedgewick, Algorithms, Algorithms in C, 1990, ADDISON-WESLEY
[6] Thomas H. Cormen ..., Algoritmos - Teoria e Prática, 2ª Edição, Elsevier, 2002
[7] Jon Kleinberg, Eva Tardos, Algorithm design, 1ª Edição, Pearson Education, 2006
[8] Jayme Luiz Szwarcfiter e Lilian Markenzon, Estruturas de Dados e Seus Algoritmos, 1994, LTC
[9] http://www.tecmundo.com.br/curiosidade/8567-pesquisadores-estimam-a-quantidade-de-informacao-existente-no-mundo.htm