Documentação - Trabalho Prático 1

Esthefanie Pessoa Lanza - esthefanielanza@gmail.com

Outubro, 2016

1 Introdução

O objetivo principal deste trabalho é simular o funcionamento de um sistema onde os recursos computacionais são limitados, de modo que não é possível armazenar mais que determinado número de dados na memória primária. Usando os conceitos vistos na disciplina de AEDS III foram implementados algoritmos de ordenação e pesquisa externos, para simular o funcionamento de um sistema de consulta de uma biblioteca.

Entretanto, apesar dos algoritmos externos conseguirem armazenar mais registros do que a memória interna pode suportar, suas operações são mais custosas de modo que devemos minimizar o número de acessos para não prejudicar a eficiência e velocidade do programa. Dessa forma, foi escolhido utilizar o quicksort externo, que faz uso do paradigma da divisão e conquista, para minimizar o número de operações realizadas. Além disso, o algoritmo não utiliza nenhuma memória externa adicional.

O sistema que será descrito neste documento, terá como função ordenar um grupo de livros em ordem lexicográfica, considerando a limitação de memória dos computadores da biblioteca, separá-los em arquivos binários correspondentes as estantes, que vão de 0 a E, e, por fim, realizar uma pesquisa binária para determinar se o livro se encontra disponível, emprestado ou se não existe um exemplar nesta biblioteca.

2 Solução do Problema

2.1 Modelagem

Para a implementação do sistema foi utilizado um struct do tipo TLivro, que por possuir tamanho constante, permitiu que fossem usadas as funções do C fseek, fread, fwrite e ftell para caminhar no arquivo binário. Um livro ocupa em memória um espaço equivalente a 54 bytes preenchidos da seguinte maneira:

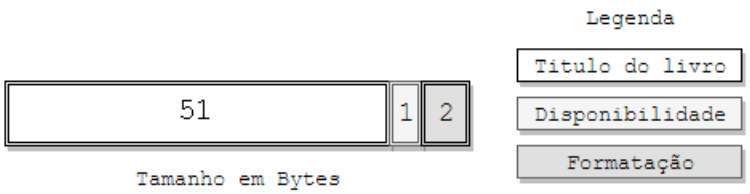


Figure 1: Representação do struct TLivro

O titulo do livro pode ocupar até 50 caracteres e foi adicionado mais um para representar o final da string, o vetor é inicializado com '\0'. Quanto à disponibilidade, esta pode ser 1, caso o livro esteja na biblioteca, e 0, caso este esteja emprestado. Além disso, por questões de estética, foram adicionados mais dois bytes para a formatação sendo o primeiro deles para representar um espaço entre o livro e sua disponibilidade e o segundo uma quebra de linha entre os diferentes exemplares.

Em relação aos arquivos, foram usados arquivos binários para as estantes e para a ordenação dos livros em um arquivo temporário. Dessa forma, seria possível gravar objetos inteiros, os livros, e não apenas informações textuais. O uso de arquivos binários também é mais rápido para gravações.

O algoritmo de ordenação escolhido para esta operação foi o quicksort externo pois não é necessário a implementação de várias fitas e o método se mostra bastante eficiente para ordenar arquivos de dados.

2.2 Implementação

O fluxo do programa é iniciado com a leitura dos livros pela entrada padrão que são escritos um a um em um arquivo binário temporário. Após a leitura de todos os n livros é realizada a ordenação usando o quicksort externo cujo o pseudocódigo de suas partições está descrito abaixo no Algorithm 1. Esse código é chamado recursivamente a partir dos apontadores i e j que delimita as novas partições criadas até que todos os elementos tenham sido ordenados. Para a execução do algoritmo são inicializados quatro apontadores: escrita superior e inferior, leitura superior e inferior. Inicialmente enchemos a memória com o número máximo de livros permitidos enquanto ainda houverem livros disponíveis, alternando entre os apontadores de leitura inferior e superior. Com a memória preenchida, esses livros são ordenados por meio de um quicksort interno.

Após isso cada livro é lido individualmente e comparado com o maior e menor livro, de acordo com a ordem lexicográfica. Se o livro vem antes, escrevemos seus dados na posição do apontador de escrita inferior. Por outro lado, se este vem depois, escrevemos na posição do apontador de escrita superior. Caso o livro se encontre no meio do mínimo e do máximo, devemos substituí-lo com um dos elementos da memória usando como critério de desempate qual apontador

andou menos até o momento. Por fim, os dados da memória são escritos no arquivo de saída.

Algorithm 1: Partição(int esq, int dir, FILE lInf, FILE lSup, FILE eInf, FILE eSup, TLivro **memoria, int *i, int *j)

```
// Ajustamos os ponteiros para partições
ajustaPonteirosIniciais(lInf, eInf)
ajustaPonteirosParticoes(i, j)
// Preenchemos a memória pela primeira vez
preencheMemoria(Arquivos, memoria)
quicksort(memoria)
while Enquanto existirem livros a serem lidos do
    // Continuamos lendo os próximos livros, tomando cuidado
       pra não sobreescrer nenhum deles
    leProximoLivro(livroAtual)
    // Se vem antes do min, escrevemos embaixo
    if livroAtual < minMemoria then
        *i = ei
        escreveMin(eInf, livroAtual, ei)
    end
    // Se vem depois do max, escrevemos em cima
    else if livroAtual > maxMemoria then
        *j = es
        escreveMax(eSup, livroAtual, es)
    end
    // Se está entre os dois devemos substituir um da memória
    else
        // Se o apontador de escrita sup andou mais que o inf
        if ei − esq < dir − es then
            trocaLivroMemoria(memoria[0], livroAtual)
        end
        else
            trocaMemoria(memoria[ultimo], livroAtual)
        end
        insertionSort(memoria)
    end
end
// Escrevemos os dados da memória no arquivo de saída
descarregaMemoria(eSup, memoria, es)
```

Para a ordenação interna foram utilizados dois algoritmos, quicksort e insertion sort. O quicksort é realizado quando enchemos a memória pela primeira vez e o vetor está completamente desordenado, para alterações menores foi utilizado o insertion sort pois sua complexidade é $O(N)$ para vetores semi-ordenados.

Depois de ordenados, os livros são separados em grupos de tamanho L, sendo L o número de livros por estante, e são colocados em arquivos binários nomeados de estante$E$, onde E é um número que varia de 0 ao número total de

estantes presentes na biblioteca. Durante a separação dos livros, escrevemos em um arquivo txt, nomeado indice, o título do menor e maior livro presente na estante. Caso a estante esteja vazia, escrevemos o símbolo #.

Por fim, com o sistema da biblioteca criado já é possível realizar consultas dos alunos. Primeiramente, é lido o livro que está sendo buscado e, a partir do arquivo de índices, identificamos a qual estante ele pertence ou poderia pertencer a partir de uma pesquisa sequencial. Se passarmos por todas as estantes e a possível posição do livro não for encontrada, quer dizer que este não faz parte da coleção da biblioteca. Por outro lado, se encontramos uma estante válida é realizada uma pesquisa binária entre seus elementos para verificar a situação do livro.

Algorithm 2: char PesquisaBinária(char *livroBuscado, int esq, int dir, FILE *estante, int *posição

```
// Enquanto o ponteiro da esquerda não ultrapassar o da
   direita
while dir ≥ esq do
    pivo = (dir + esq)/2
    // Encontramos o livro correspondente à posição do pivô
    ajustaPosicao(estante, pivo)
    leLivro(estante, livroAtual)
    // Caso encontremos o livro
    if livroBuscado == livroAtual then
        *posicao = pivo
        retorna a disponibilidade do livro
    end
    // Se o livro buscado é menor que o livro atual
    if livroBuscado < livroAtual then
        dir = pivo - 1
    end
    else
        esq = pivo + 1
    end
end
retorna que o livro não existe na biblioteca
```

A partir do resultado retornado pela pesquisa binária o algoritmo imprimirá se o livro está disponível e em qual posição, se está emprestado ou se ele faz parte da coleção no stdout.

3 Análise de complexidade

Nesta seção, será apresentada a análise do custo teórico de tempo e de espaço.

3.1 Análise Teórica do Custo Assintótico de Tempo

Para a análise assintótica foram observadas as funções presentes no código principal que definem o fluxo do programa.

- A função incializaLivros tem como objetivo ler os livros presentes na biblioteca diretamente do stdin e escrevê-los em um arquivo temporário. Sua complexidade é referente ao número de livros inseridos, ou seja, $O(N)$.

- A função ordena utiliza o quicksort externo para ordenar o arquivo temporário criado pela função de inicializar os livros. Por fazer o uso do paradigma de divisão e conquista as partições criadas são cada vez menores. A cada nova chamada recursiva os elementos que ainda devem ser ordenados são diminuídos em M, sendo M o tamanho da memória disponível. Serão realizadas $\log \frac{N}{M}$ chamadas recursivas, assim no caso médio, a complexidade será equivalente a $O(N \times \log \frac{N}{M})$.

  No melhor caso, quando o vetor já está ordenado, não são criadas novas partições pois sempre estamos substituindo os elementos na memória primária. Assim, sua complexidade será equivalente ao número livros ordenados, ou seja, $O(N)$.

  Por outro lado, no pior caso teremos partições de tamanho incorreto, uma delas será vazia e a outra possuirá todos os elementos menos o tamanho da memória a cada nova partição. Desta forma, não tiraremos proveito do paradigma da divisão e conquista, realizando $\frac{N}{M}$ chamadas recursivas e elevando sua complexidade para $O(\frac{N^2}{M})$.

  Na teoria, o custo de ordenações internas torna-se irrelevante ao ser comparado com as operações de manipulação do arquivo, portanto sua complexidade não influencia na execução do algoritmo.

- A função criaLivrosOrdenados apenas transfere os dados do arquivo binário para um arquivo de texto. Sendo assim, sua complexidade é referente ao número de livros presentes na biblioteca $O(N)$.

- A função separaLivrosEstantes é responsável por abrir os arquivos referentes as estantes, organizar os N livros nas mesmas e escrever o arquivo de índices. Para realizar esses procedimentos é utilizado um único loop que vai de 0 a N, ou seja, a complexidade desta função será equivalente a $O(N)$.

- A função consultaLivros utiliza dois tipos de pesquisa. Em um primeiro momento, é realizada uma busca sequencial entre os índices para encontrar a estante que provavelmente possui o livro com custo $O(E)$, sendo E o número de estantes não vazias. Para encontrar o livro, é utilizada uma busca binária que possui complexidade $O(1)$ no melhor caso e $O(\log N)$ para o pior. Entretanto, existe um loop externo as duas pesquisas correspondente ao número de consultas K, assim a complexidade total da função é $O(K \times max(E, logN))$.

- A complexidade total do programa é a maior complexidade dentre as citadas acima.

3.2 Análise Teórica do Custo Assintótico de Espaço

Por se tratar de um algoritmo que trabalha com memória secundária, para a análise assintótica de espaço é necessário analisar somente as função de ordenação e a de separar os livros em estantes, pois são as únicas que utilizam alocação dinâmica.

- A função ordena possui um vetor de tamanho M para a ordenação interna das chaves. Dessa forma, a complexidade é $O(M)$.
- Para a criação das estantes, a função separaLivrosEstantes, aloca espaço para guardar os arquivos das E estantes. Assim, sua complexidade espacial é $O(E)$.

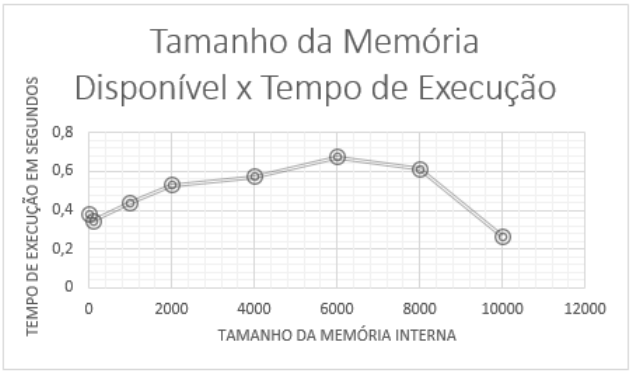
4 Avaliação Experimental

4.1 Metodologia

A implementação da solução proposta foi compilada utilizando gcc 4.8.4. Os experimentos foram executados em um sistema com 4 GB de memória e um processador Intel Core i3 2310K @ 2.1 GHz. Cada teste foi realizado em torno de 30 vezes e foi realizada uma média entre os valores obtidos.

4.2 Análise do tempo da ordenação inicial com relação à varição da memória disponível

Para a análise da variação do tempo em relação a memória, foi criado um teste com 10000 livros na biblioteca, distribuídos em 100 estantes de tamanho 100, e 10000 consultas. Estes valores permaneceram constantes durante a realização dos testes enquanto a memória variou de 10 a 10000.



Número de Palavras	Tempo de Execução
10	0,379
100	0,343
1000	0,440
2000	0,529
4000	0,572
6000	0,673
8000	0,611
10000	0,263

Figure 2: Análise da variação do tempo em relação ao tamanho da memória disponível

Diferente do que foi descrito na análise de complexidade, o tempo de execução não diminuiu ao aumentar o número de livros que cabem na memória. Uma hipótese levantada sobre a causa desse comportamento é a de que a ordenação interna, por ser executada um número grande de vezes, acaba afetando o comportamento do algoritmo. Dessa forma, temos que considerar a complexidade do quicksort intero $O(n \times \log n)$ para a primeira ordenação de cada partição e do insertion sort, considerando que o vetor está quase ordenado, $O(n)$ a cada alteração realizada na memória. Na figura 3, temos um gráfico exemplificando o número de elementos ordenados em memória interna para os mesmos testes realizados acima.

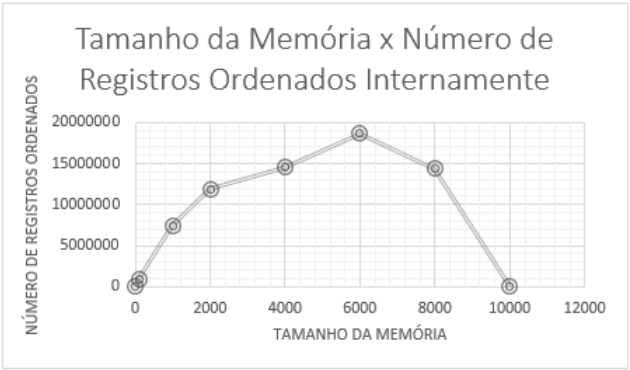


Figure 3: Análise da variação do número de registros ordenados internamente em relação ao tamanho da memória disponível

Esse gráfico foi gerado multiplicando o número de vezes que o código foi ordenado internamente vezes o tamanho da memória. Ao sobrepor os gráficos

podemos ver a influência que a ordenação interna faz sobre o tempo de execução. Entretanto, é importante ressaltar que o programa apresentará comportamentos diferentes para testes diferentes, já que o número de ordenações feitas internamente depende do número de livros, da sua ordem inicial, dentre outros fatores.

4.3 Análise do tempo com relação ao número de estantes e suas devidas capacidades

Para esta análise foi criado um teste com 10000 livros na biblioteca, distribuídos em estantes que variam de 10 a 1000 em conjunto com suas capacidades e 10000 consultas.

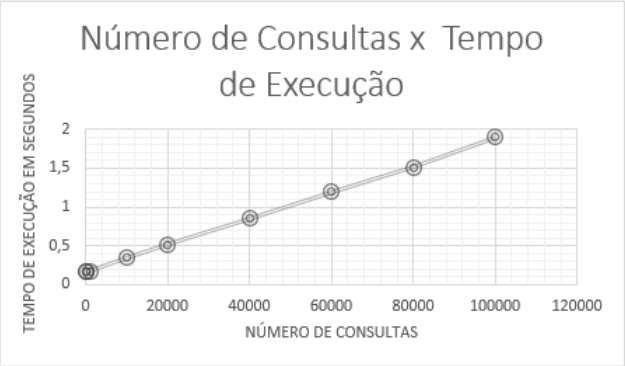
Como o número de estantes influencia apenas na criação de estantes e nas consultas realizadas pelos alunos, temos um gráfico linear. Isso ocorre pois a pesquisa sequencial, em relação a binária, tem um maior custo para ser realizada $O(E)$.

Número de Palavras	Tempo de Execução
10	0,252
100	0,339
200	0,4365
400	0,691
600	0,9285
800	1,1655
1000	1,45

Figure 4: Análise da variação do tempo em relação ao número de estantes da biblioteca

4.4 Análise do tempo com relação ao número de pesquisas a serem feitas.

Para esta análise foi criado um teste com 10000 livros na biblioteca, distribuídos em 100 estantes de 1000 livros e variamos o número de consultas de 10 a 100000.



Número de Palavras	Tempo de Execução
10	0,162
100	0,178
1000	0,171
10000	0,346
20000	0,517
40000	0,851
60000	1,193
80000	1,516
100000	1,938

Figure 5: Análise da variação do tempo em relação ao número de consultas realizadas

Como mostrado no gráfico acima o algoritmo reagiu como previsto. Ao aumentar o número de consultas realizadas o tempo aumenta de maneira linear.

5 Conclusão

A ordenação externa, apesar de ser útil ao lidar com sistemas de capacidade de armazenamento limitada, é difícil de ser implementada e possui um custo mais alto ao realizar a manipulação da memória secundária. Nesta implementação foi possível perceber que, quando aumentamos o tamanho da memória disponível, a ordenação interna também têm influência sobre os tempos do algoritmo, pois esta é realizada inúmeras vezes em vetor de tamanho razoável.

É importante ressaltar que o número de vezes que a memória interna é ordenada independe dos inteiros iniciais de forma que na analise experimental obtivemos tempos muito diferentes enquanto aumentávamos o tamanho da memória. Se outro teste fosse realizado poderíamos obter valores diferentes, pois o número de registros ordenados internamente depende do número de livros que foram substituídos, ao ler um novo livro, durante o processo do quicksort externo.