Trabalho Prático 1 de AEDS3

Biblioteca do Filipe

Douglas Rodrigues de Almeida
douglasralmeida@live.com

1. Introdução

A necessidade de grande capacidade de memória aliada ao seu alto custo faz com que os computadores apresentem vários níveis de memória. A memória secundária, mais barata, possui grande capacidade de armazenamento, entretanto o custo para acessá-la é algumas ordens de grandeza menor que o custo de processamento da memória principal. Essa restrição leva a necessidade de algoritmos que façam ordenação e pesquisa de grandes quantidades de registros, maior que a capacidade da memória principal do computador, com a menor quantidade possível de acessos a memória secundária.

O objetivo deste trabalho é implementar um simulador de gerenciamento de biblioteca usando um computador com memória principal limitada. Para superar essa limitação, o aplicativo utiliza técnicas de ordenação e pesquisa que minimizam a quantidade de vezes que a memória secundária é acessada.

2. Implementação

Quicksort Externo

O quicksort é um algoritmo eficiente para ordenação em memória principal. Seu tempo de execução médio é O(n log n). A ideia básica do quicksort é dividir o problema de ordenar um conjunto de itens em problemas menores. Cada problema menor é ordenado de forma independente e os resultados combinados para produzir a solução do problema maior.

A ideia básica do quicksort foi adaptada para ordenação em memória secundária. Chamado de quicksort externo, o algoritmo utiliza um buffer na memória com capacidade de, pelo menos, 3 registros.

Seja um arquivo $A = \{R_1, \ldots, R_n\}$ de n registros.

Seja R_i, $1 < i < n$, o registro que se encontra na i-ésima posição de A.

1) Particionar A na seguinte forma:

$\{R_1, \ldots, R_i\} \leq R_{i+1} \leq R_{i+2} \leq \ldots \leq R_{j-2} \leq R_{j-1} \leq \{R_j, \ldots, R_n\}$

2) Chamar o algoritmo recursivamente para cada um dos subarquivos

$A_1 = \{R_1, \ldots, R_i\}$ e $A_2 = \{R_j, \ldots, R_n\}$

Código 1: Algoritmo do quicksort externo.

Inicialmente, são lidos para uma memória com capacidade de m registros, os primeiros e últimos m/2 registros da memória secundária onde são ordenados com algum algoritmo de ordenação interna. Os registros ordenados são armazenados de volta na memória secundária já na sua posição final, denominado pivô. O pivô possui o tamanho da memória principal. Todos os registros maiores que o maior registro do pivô são gravados à esquerda e todos os registros menores que o menor registro do pivô são gravados à direita. Logo depois, os lados esquerdos e direitos do pivô são

ordenados de forma recursiva. Essa operação se repete até que alcançar subconjuntos de um ou nenhum registro.

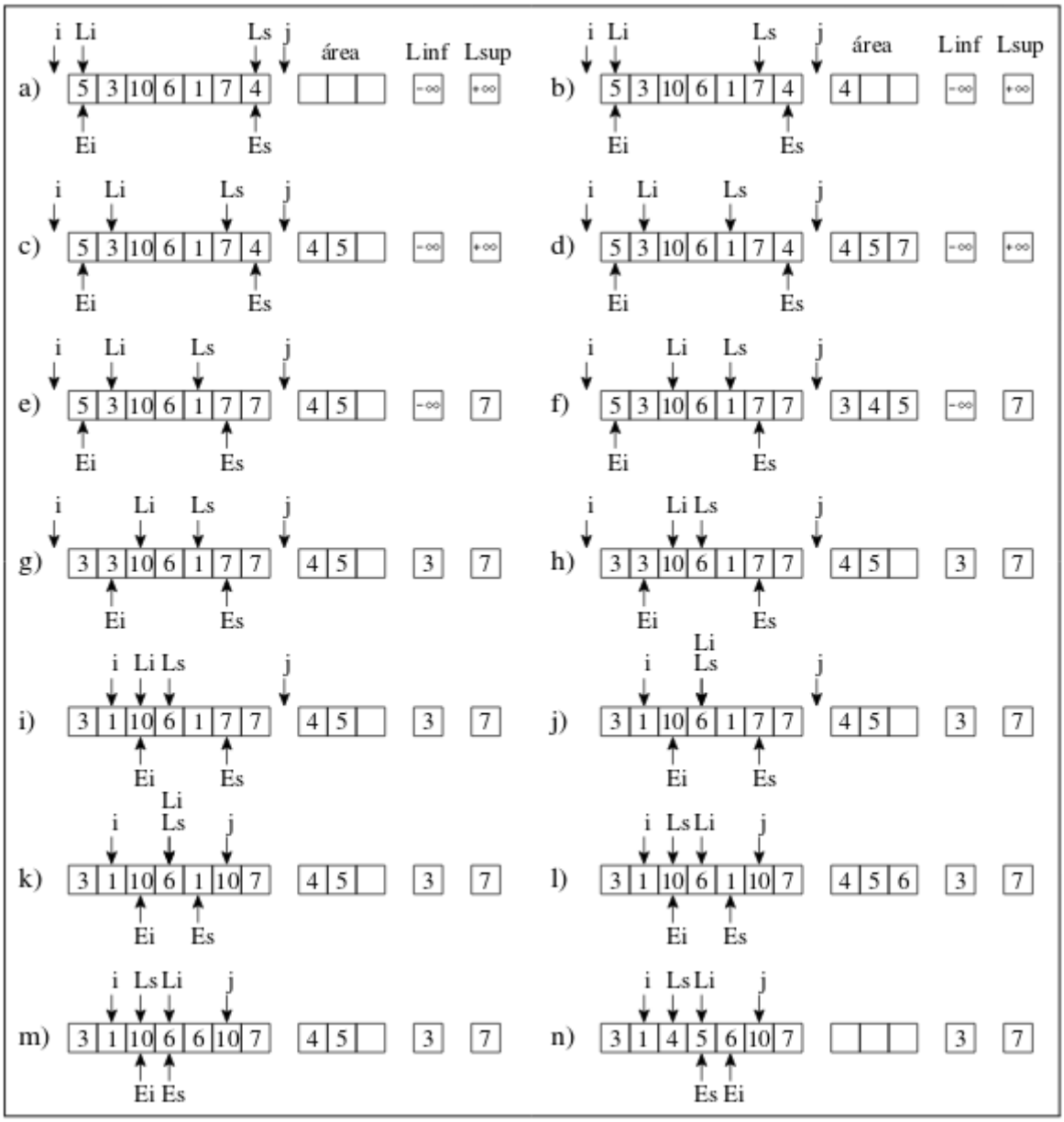

Figura 1: Processo de geração do pivô ordenado no Quicksort Externo

Árvore B

Árvores B de ordem m são árvores **m-árias** e balanceadas que são usadas para organizar grandes quantidades de registros armazenados em memória secundária. A árvore B permite acesso sequencial eficiente e, o custo para recuperar, inserir e remover registros é logarítmico. O espaço utilizado pelos dados é, pelo menos, 50% do espaço reservado.

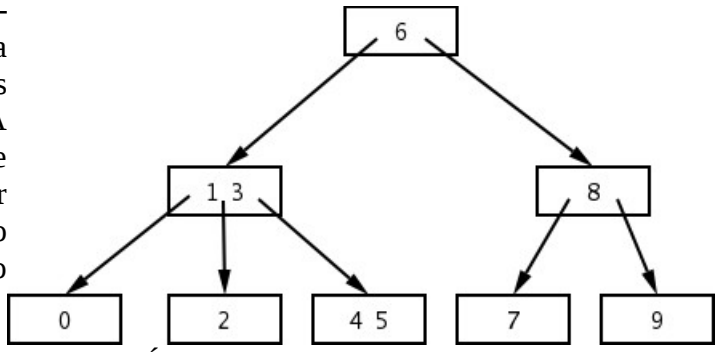

Figura 2: Árvore B de ordem 1.

Uma árvore B de ordem m possui as seguintes características:

- Cada caminho da raiz até a folha da árvore possui o mesmo comprimento.
- Todos os nós da árvore, com exceção da raiz, possuem entre m e 2m registros. A raiz possui entre 1 e 2m registros.
- Cada nó tem entre m+1 e 2m+1 filhos.
- Os registros armazenados no mesmo nó estão em ordem crescente.

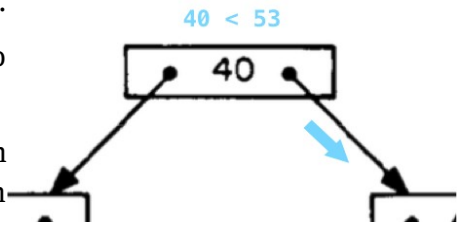

Figura 3: Pesquisa a chave 53. Por ser maior que 40, a pesquisa caminha para a subárvore da direita.

Os limites calculados por Comer para altura máxima e altura mínima de uma árvore B de ordem m contendo N registros são:

$$\log_{2m+1}(N+1) \leq altura \leq 1+\log_{m+1}\left(\frac{N+1}{2}\right)$$

O procedimento de pesquisa de um registro na árvore B é recursivo. Primeiro compara-se sequencialmente a chave de pesquisa com as chaves da raiz em busca do intervalo compatível com o registro. Caminhamos, então, à subárvore correspondente ao intervalo e repetimos o processo até encontrarmos o registro procurado ou chegarmos a uma folha, no caso do registro não estiver na árvore.

A inserção de um registro na árvore B de ordem m ocorre da seguinte maneira:

```
1) A partir da raiz, procura-se o nó apropriado para inserir o registro.

2) Se o nó onde o registro será inserido tiver menos de 2m nós, insere o
registro em ordem crescente neste nó.

3)Se o nó onde o registro será inserido tiver 2m nós, divide-se em dois,
redistribuindo igualmente os registros entre os nós e promovendo o registro
do meio para o nó pai.

4)Se o nó pai estiver cheio repete-se o procedimento do item anterior no nó
pai.
```

Código 2: Algoritmo de inserção na árvore B.

No pior caso, a divisão pode propagar-se até a raiz aumentando a altura da árvore. A árvore B somente aumenta sua altura com a divisão da raiz.

Existem variações da árvore B, como árvore B* que otimiza o uso de memória e árvore B+ que facilita o acesso concorrente aos registros.

Um sistema com memória virtual pode diminuir o custo de pesquisa em árvores B. Dependendo do esquema de reposição de páginas implementado, a raiz da árvore, por exemplo, sempre estará na memória principal.

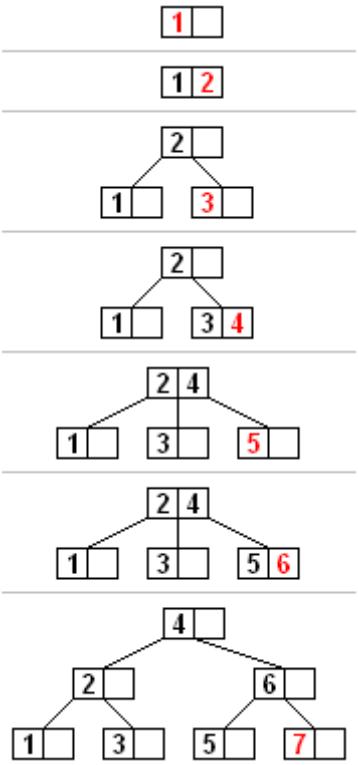

Figura 4: Inserção de itens em uma árvore de ordem 1.

Sistema

O sistema foi desenvolvido em módulos que facilitam sua manutenção.

O TAD TArvoreB implementa uma árvore B para pesquisas conforme descrito nesta documentação.

O TAD TFila implementa uma fila usando vetores. Esta fila é usada para armazenar as consultas de livros.

O TAD Livro é usado pelo sistema para trabalhar com os livros da biblioteca.

O TAD Memoria simula a memória principal do sistema. Cada registro é inserido na memória de forma ordenada. É possível remover o maior ou menor item da memória em O(1).

O TAD Ordenador implementa a ordenação externa usando a técnica de Quicksort Externo.

O TAD TSistemaConsulta armazenda uma dados para consulta de livros.

O TAD TSistemaMotorBusca é responsável pela pesquisa de livros. Ele usa uma árvore binária com um índice para verificar se um livro está na bilbioteca, está emprestado ou não está registrado.

O TAD TSistemaManipuladorES é responsável pela leitura e escrita de dados na memória secundária.

Por fim, o TAD Sistema é responsável pelo funcionamento efetivo do sistema, realizando a leitura dos dados, manipulando e escrevendo os resultados em arquivos e na tela.

Programa Principal

O sistema recebe do aviso de comando do usuário cinco variáveis, N, M, E, L e K para efetuar a a simulação de gestão de biblioteca. A variável N indica o número de livros da biblioteca, E indica o número de estantes e L indica o número de livros por estante. A variável M indica a capacidade da memória principal do computador e K indica o número de consultas a serem realizadas no sistema.

Em seguida, o aplicativo recebe a listagem nos N livros com a informação se estão emprestados. Cada livro deve ter no máximo 50 caracteres. Estes livros são ordenados e seus nomes salvos num arquivo de texto de nome *livros_ordenados*. Também são gerados arquivos binários representando as estantes da biblioteca. Estantes vazias não são geradas. Cada arquivo recebe o nome *estanteX*, onde x é o identificador da estante e possui os registros dos livros que lá estão armazenados.

A simulação também gera um arquivo-texto chamado *indice* com o registro do primeiro e último livro de cada estante. Estantes vazias são representadas pelo caractere #. O índice é usado para a realização das consultas que serão realizadas a seguir. Se o livro estiver registrado e armazenado em alguma estante, a consulta retorna o identificador da estante e a posição do livro na própria estante.

A consulta também informa se o livro está disponível para empréstimo, se está emprestado ou se não está registrado na biblioteca.

Para executar o programa, digite o seguinte comando:

./tp1

3. Análise de Complexidade

O custo de acessar um registro em memória secundária é algumas ordens de grandeza maior que o custo de processamento na memória principal. Logo, a medida de complexidade aqui está relacionada com o custo para transferir os dados da memória principal e a memória secundária. A ênfase aqui dever ser na minimização do número de vezes que cada registro é transferido entre a memória interna e a memória principal.

Quicksort Externo

A complexidade de melhor caso do quicksort externo é $O(\frac{n}{b})$, em que n é o número de registros a serem ordenados e b é a quantidade de registros lidos por vez na memória secundária. O melhor caso ocorre, por exemplo, em arquivos já ordenados.

O pior caso ocorre quando as chamadas recursivas são árvores degeneradas. Uma partição é vazia e a outra partição é a maior possível. Nesse caso, a complexidade é $O(\frac{n^2}{m})$, onde m é a quantidade de registros que pode ser gravados na memória principal ao mesmo tempo.

A complexidade no caso médio é $O(\frac{n}{b}\log(\frac{n}{m}))$.

Árvore B

Caminhar pela árvore da raiz até uma folha tem um custo igual à sua altura. Numa árvore m-aria cada nó possui, no mínimo, m+1 filhos. Assim, o número de nós aumentam exponencialmente em cada nível da árvore. Assim, o custo para pesquisar um registro cresce com o logaritmo base m da ordem da árvore. É possível pesquisar numa árvore de ordem 50 com um milhão de registros com apenas 4 acessos em disco no pior caso.

O custo de inserção de um registro em uma árvore B de ordem m com n registros é também proporcional a $\log_m n$ no pior caso.

Programa principal

As funções de ordenação externa e pesquisa de livros são executas sequencialmente, logo a medida de complexidade do aplicativo é $O(\frac{n}{b}\log(\frac{n}{m}))$ + $O(\log_m n)$. Logo, a complexidade final é $O(n\log n)$.

4. Análise de Execução

Foram realizados diversos testes para verificar o comportamento do programa conforme a variação da entrada.

No primeiro teste variamos a quantidade de livros, mantendo as demais variáveis constantes.

Nº livros	1000	2000	3000	4000	5000	6000
Tempo (s)	0.032	0.047	0.075	0.098	0.111	0.130

No segundo teste variamos o tamanho da memória principal, mantendo as demais variáveis contantes.

Tamanho	10	20	30	40	50	60
Tempo (s)	0.011	0.010	0.009	0.009	0.009	0.008

No terceiro teste variamos o número de estantes, mantendo as demais variáveis contantes.

Tamanho	100	200	300	400	500	600
Tempo (s)	0.013	0.013	0.012	0.013	0.030	0.029

No quarto teste, variamos o número de livros por estantes, mantendo as demais variáveis constantes.

Tamanho	100	200	300	400	500	600
Tempo (s)	0.012	0.012	0.011	0.011	0.011	0.012

No último teste, variamos o número de consultas a serem realizadas, mantendo as demais variáveis constantes.

Tamanho	100	200	300	400	500	600
Tempo (s)	0.052	0.053	0.069	0.072	0.073	0.075

5. Conclusão

A árvore B é simples, de fácil manutenção, eficiente e versátil. Por isso é amplamente utilizada organização de registros em memória secundária como bancos de dados e sistemas de arquivos. Existem, ainda, variações da árvore B, como B* e B+, que ampliam sua utilidade e adicionam novos recursos, como concorrência. Sistemas de paginação de memória virtual pode aumentar sua eficiência em pesquisas de registros.

O Quicksort Externo também é bastante eficiente para ordenação em memória secundária, desde que o pivô seja bem escolhido. Tal como a árvore B, a ordenação é ainda mais eficiente se utilizada com sistema de paginação de memória virtual.

Referências

BAYER, B. AND MC CREIGHT, E. M. 1972. Organization and maintenance of large ordered indices. Acta Inform. 1, 173–189.

COMER, D. 1979. The ubiquitous B-tree. ACM Comput. Surv. 11, 2, 121–137.

CUNTO, W. GONNET, G. H.. MONARD, M. C. External Quicksort. 1981.

ZIVIANI, N. 2009. Projeto de Algoritmos com Implementações em Pascal e C, 2ª Edição, Editora Cengage Learning.