Universidade Federal de Ouro Preto – UFOP
Instituto de Ciências Exatas e Biológicas – ICEB
Departamento de Computação – DECOM
Disciplina: BCC203 – Estrutura de Dados II
Professor: Guilherme Tavares de Assis
Aluno: Bernardo Cavanellas Biondini - 20.1.4112

Resumo - Pesquisa Externa

Em programação, precisamos lidar com dados que estão localizados em memória principal e em memória secundária. Dessa forma, contamos com estrutura de dados específicas capazes de manipulá-los em ambas formas de armazenamento de dados.

Pesquisa externa, trata sobre métodos de pesquisa em memória secundária, quando os dados não cabem todos em memória principal. Assim, é preciso levar em consideração, em relação ao custo do algoritmo, não só o número de comparações, mas também o número de transferências de dados da memória secundária para a memória principal, uma vez que esses dados são acessados de forma sequencial, desse modo, existem estratégias que tentam melhorar ou diminuir esse acesso.

1) Sistema de Paginação

Uma estratégia eficiente que promove a implementação de métodos de pesquisa externa que consiste em dividir os dados igualmente, em páginas na memória principal, podendo realizar transferências entre as memórias e mapear os endereços. Tal método é chamado de sistema de paginação. Para endereçar um item de uma página, uma parte dos bits é utilizada para representar o número da página e a outra parte, o número do byte do item na página.

- Caso o programa precise de uma página que não se encontra na memória principal, ela precisa ser trazida da memória secundária
- Se não houver uma moldura vazia, alguma página tem que ser removida
- A remoção pode ser feita seguindo alguns critérios:
    - LRU: remove a página menos recentemente utilizada.
    - LFU: remove a página menos frequentemente utilizada
    - FIFO: remove a página que está há mais tempo em memória principal

2) Acesso Sequencial Indexado

É um método que utiliza da pesquisa sequencial o qual cada item é lido até encontrar uma chave maior ou igual a chave desejada. Esse método utiliza de um índice para que o acesso seja realizado, facilitando-o. Dessa forma, é preciso criar uma estrutura que contenha o índice da página e sua primeira chave, a menor delas, uma vez que o arquivo deve estar ordenado. Assim, para realizar a pesquisa, é preciso comparar apenas as chaves da estrutura, deixando explícito em que página o item se encontra.

3) Árvores de Pesquisa

Árvores binárias são estruturas eficientes quando é possível armazenar os dados em memória principal. Em consideração a memória secundária, a árvore binária pode ser utilizada com algumas modificações, os nodos são armazenados em disco e seus apontadores à esquerda e à direita armazenam endereços de disco. Para uma eficiência maior, os nodos podem ser agrupados em páginas, porém, é difícil organizar essa estrutura de forma ótima.

O algoritmo de alocação sequencial é capaz de simular uma árvore binária sem considerar seu formato físico, porém os nodos ficam ordenados pela entrada de itens e não pela localidade na árvore.

3.1 Árvore B: solução mais eficiente em relação aos problemas das estruturas antes mencionadas. A árvore B é uma árvore n-ária, ou seja, possui mais de dois descendentes por nodos.

Em uma árvore de ordem m, a página raiz contém entre 1 e 2m itens e as demais páginas no mínimo m itens e m+1 descendentes e, no máximo, 2m itens e 2m + 1 descendentes. Uma característica dessa estrutura é seu balanceamento, todas as folhas aparecem no mesmo nível, e seus itens aparecem em ordem crescente.

A pesquisa em uma árvore B é semelhante à pesquisa em uma árvore binária. Há uma comparação do item x com as chaves da página raiz, até encontrar a chave desejada, caso ela não seja encontrada, o programa segue para o apontador da subárvore.

A inserção de itens nessa estrutura ocorre da seguinte maneira:

- Localiza-se na árvore o local apropriado para ser inserido o item
- Caso a página encontrada tenha menos de 2m itens, a inserção acontece nela.
- Se a página estiver cheia, é criada uma nova página, dividindo-se os itens entre elas e o item do meio sobe para a página pai. *Caso a página pai estiver cheia, o processo continua. *Caso esse processo ocorra até a raiz, a árvore aumenta de tamanho.

Já a remoção:

- Caso o item esteja em uma página folha, a remoção é direta
- Caso contrário, deve-se substituir o item por outro, no caso, pode ser o item mais à direita da subárvore à esquerda ou o item mais à esquerda da subárvore à direita
- Após a remoção é necessário ver se as propriedades da árvore B foram mantidas
- Caso não tenham sido mantidas, a página precisa pegar um item emprestado. Nesse caso, se a página vizinha possui m itens, as páginas devem ser fundidas, tomando emprestado da página do pai o item do meio, permitindo a liberação da página. Caso a página vizinha tenha um número de intens maior que m, pega-se emprestado um item da página vizinha a partir da página pai

3.1.2 Árvore B*: uma alternativa para implementação da árvore B. Nela, todos os itens se encontram no último nível, enquanto que os níveis acima só são índices. Assim, é preciso categorizar a página da árvore como sendo internas ou externas.

A pesquisa nessa estrutura ocorre de forma semelhante a de uma árvore B, ela sempre leva a uma página folha e os valores encontrados pelo caminho não são relevantes.

A inserção na Árvore B* é essencialmente igual a inserção da árvore B, exceto que, quando a folha é dividida em duas, o algoritmo faz uma cópia da chave do item do meio para a página de nível anterior, retendo o próprio item do meio na página folha da direita.

A remoção é um pouco mais simples, caso a folha fique com pelo menos m itens, as páginas do índice não precisam ser modificadas e as páginas do índice

precisarão ser modificadas apenas se a folha ficar com uma quantidade de itens menor que m