**Universidade de São Paulo - USP**
**Instituto de Ciências Matemáticas e de Computação - ICMC**

# Relatório
# Trabalho 3
# Algoritmos e Estrutura de Dados II

**Professora:**
Elaine Parros Machado de Sousa
**Estagiários PAE:**
Diego Silva
Ivone Penque Matsuno

**Grupo:**
Guilherme José (7150306)
Leonardo Guarnieri de Bastiani (8910434)
Luiza Vilas Boas de Oliveira (8503972)
Ricardo Chagas (8957242)

**São Carlos**
**2015**

- **Descrição do problema:**

O problema apresentado neste trabalho consiste na criação de um arquivo de dados que armazene registros com os campos:

1. ID do usuário
2. Nome completo do usuário
3. Tipo do usuário (TU)

Sendo que o ID do usuário é uma chave numérica, e o tipo do usuário é um número que representa a política do site para oferecer serviços e propagandas para os usuários (1: gratuito; 2: comum; 3: premium).

Os registros deveriam ser indexados por meio de uma árvore-B de ordem 6.

O programa deveria ter as funcionalidades:

1. Criação do arquivo de índice a partir do arquivo de dados, caso este seja preexistente.
2. Inserção de novos usuários (inserindo também na árvore-B).
3. Remoção de usuários (utilizando a árvore-B para encontrar o usuário a ser removido).
4. Pesquisa por ID (por meio da árvore-B).
5. Mostrar a árvore-B.

A interação com o usuário deveria ser feita por meio do console/terminal, e a saída por meio da criação de uma arquivo de log (histórico).

- **Arquivo de dados**

Quando em memória principal o registro é manipulado com uma struct com campos:

```
int id;
char nome[50];
int tu;
```

Já o arquivo de dados, de nome “registo.reg”, é composto por registros de tamanhos variáveis. Antes de cada registro há um byte que indica o tamanho total deste registro. A seguir vêm as informações dos campos ID, nome e TU, separados pelo caractere “|”.

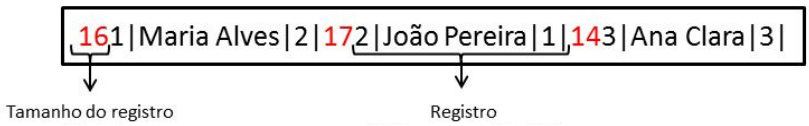


**Figura 1. Representação da organização dos registros no arquivo de dados.**

A inserção de novos registros é realizada sempre no final do arquivo de dados.

Para a remoção, primeiramente usa-se a estrutura da árvore-B para encontrar o offset no arquivo de dados do registro a ser removido. Em posse desse offset, encontra-se o registro no arquivo de dados e um marcador é inserido neste registro para indicar a remoção. Esse marcador consiste no caracter "*", que é adicionado logo após o campo ID, no lugar do primeiro caracter do nome.

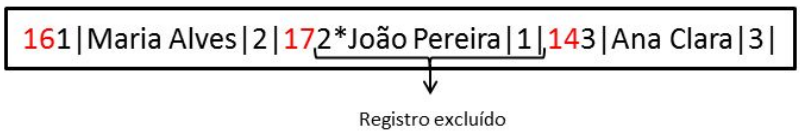


**Figura 2. Exemplo de arquivo removido.**

- ## Arquivo de índice (Árvore-B)

A indexação neste trabalho foi realizada com o uso de uma árvore-B de ordem 6 mantida em memória secundária (disco).

As páginas (nós) da árvore-B são estruturas de tamanho fixo com os seguinte campos:

❏ Número da página no disco (iniciando em 1).
❏ Quantidade de chaves que a página contem.
❏ Número booleano que indica se a página é folha ou não.
❏ Vetor de chaves (5 elementos).
❏ Vetor de filhos (6 elementos).

Cada uma das chaves é uma estrutura que contem o ID desta chave, e o offset que indica a posição do registro referente a este ID no arquivo de dados.

Graficamente, temos:

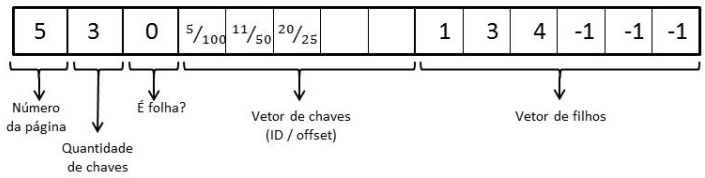


**Figura 3. Representação física de uma página da árvore-B.**

Lembrando que na árvore-B as chaves de uma página são ordenadas (no caso pelo ID), e que o filho da esquerda de uma chave leva à páginas que possuem chaves com IDs menores que o dela, e o filho da direita leva à páginas que possuem chaves com IDs maiores que o dela. Vale ressaltar que os elementos do vetor filhos não são ponteiros propriamente ditos, mas possuem o RRN da página que eles “apontam”. Como as páginas têm tamanho fixo, basta o RRN para encontrar uma página no arquivo de índice.

A representação lógica de uma página é a seguinte (para facilitar as representação, as chaves são representadas apenas pelos seus IDs, não aparecendo nas figuras os offsets, nem o cabeçalho da página):

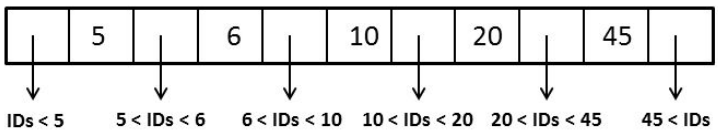


**Figura 4. Representação lógica de uma página da árvore-B.**

No arquivo da árvore, conforme ela vai crescendo, as páginas são posicionadas na sequência em que são criadas. Caso alguma página seja excluída, o espaço anteriormente ocupado por ela no arquivo é reaproveitado na criação de uma nova página. Para tanto há uma lista de espaços vazios. O RRN do primeiro espaço vazio é indicado no início do arquivo, em um cabeçalho. O RRN do próximo espaço vazio da fila está na primeira posição do vetor de filhos de cada página excluída.

Além da informação do primeiro espaço vazio, o cabeçalho no início do arquivo contem ainda o RRN da página raiz e o número de páginas da árvore. Por exemplo:

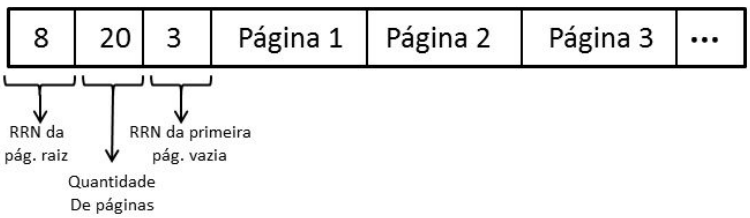


**Figura 5. Representação gráfica do arquivo de índice.**

- Inicialização:

Primeiramente, verifica-se se o arquivo de índice já existe. Em caso afirmativo, este arquivo é simplesmente aberto pelo programa.

Caso contrário o arquivo de índice é criado e a árvore começa a ser “montada”. A informação da raiz e do primeiro espaço vazio no início do arquivo recebem -1, e o número de páginas recebe o valor zero.

- Busca:

Na operação de busca, primeiramente a página do nó raiz é trazida para a memória. Lembrando que isso é possível pois no cabeçalho do arquivo está armazenado o RRN desta página no arquivo de índice. É realizada uma busca binária no vetor de chaves desta página, a fim de encontrar o ID pesquisado. Se o ID for coincidente ao de alguma chave, é retornado o offset desta chave. Caso contrário, a função analisa entre quais duas chaves do vetor deveria estar a chave que possuiria o ID pesquisado. É então carregada para a memória a página que é “apontada” pelo RRN do filho que está entre estas duas chaves (conforme a representação lógica da figura 4). O processo então continua recursivamente. A página que foi carregada, têm seu vetor de chaves buscados pelo ID, se houver coincidência, retorna o offset, se não carrega a próxima página, e assim sucessivamente. A operação de busca só termina quando a chave for encontrada ou caso chegar nos nós folhas (não há mais filhos, e todo as as posições do vetor filhos são -1). Se ocorrer o segundo caso, o ID pesquisado não pertence a nenhum registro no índice.

- Inserção:

Primeiramente é necessário saber se uma chave com o ID que se deseja inserir já está presente, pois ,por ser uma chave primária, não pode haver dois IDs iguais.

Caso a árvore esteja vazia (RRN da raiz -1), a página do nó raiz é criada em memória principal. Esta página recebe o número 1, por ser a primeira da árvore, e o booleano recebe 1 também, pois a primeira chave é tanto raiz como folha. O número de chaves recebe 1, e a chave a ser inserida (ID e offset) são colocados na primeira posição do vetor chaves. Após os dados da página estarem todos corretos, a página é passada para o arquivo de índice, após as informações do cabeçalho. Por fim, o RRN da raiz no cabeçalho do arquivo recebe 1, bem como o número de páginas.

Caso a árvore não esteja vazia, analisamos em qual posição a nova chave deve estar inserida. Isso é feito descendo pelos nós e analisando qual a posição que a chave deveria ocupar, em um processo igual à busca. Se a raiz do ramo da

árvore em que a nova chave deveria estar está totalmente ocupada, ela sobre uma divisão (split). Caso o split tenha ocorrido, é analisado em qual das páginas resultantes da divisão a nova chave deveria estar e o processo é repetido recursivamente até chegar em um nó folha. Quando o fundo da árvore foi atingido, como os possíveis splits foram realizados no caminho, basta adicionar a chave na posição correta em que deve ocupar no vetor de chaves.

Vale lembrar que no processo de split são criadas novas páginas, e isso deve ser realizado levando em conta os espaços vazios deixados pelas aglutinações (merge) das exclusões. Isso é realizado pela lista de espaços vazios, como mencionado anteriormente.

- Remoção:

A remoção na árvore-B deve ser feita de tal maneira que respeite a ocupação mínima de cerca de 50% em cada página, com exceção do nó raiz. Sendo assim, novamente se incia carregando a página do nó raiz para a memória principal. Em um processo semelhante a busca, procuramos a chave que possui o ID que se deseja remover. Caso se chegue em uma página de nó folha, e não se encontre a chave que possui o ID, não é possível remover a chave, pois ela não existe no índice.

Caso seja encontrada a chave que possui o ID de interesse, deve-se analisar se a página na qual essa chave se encontra é de nó folha ou de nó não-folha.

Caso não seja de nó folha, a posição da chave que se deseja excluir recebe a chave com maior ID da subárvore a esquerda ou a chave de menor ID da subárvore a direita. Após isso, é feita a remoção da chave original que foi copiada.

Caso seja nó folha, há duas situações. Se a retirada da chave não faz com que a ocupação dessa página seja menor que a mínima, deve-se apenas excluir está chave (reordenando as outras). Se a ocupação ficar menor que a mínima, deve-se primeiramente redistribuir as chaves das páginas irmãs. Caso isso não seja possível (pois as irmãs ficariam com menos que a ocupação mínima), deve-se aglutinar a página com uma das irmãs. Isso, é recursivo, subindo a árvore. Como esse processo faz com que uma das páginas seja apagada, é necessário inserí-la na lista de espaços vazios, para que este seja reaproveitado em uma possível futura inserção.

- Impressão:

Primeiramente são impressas as informações do cabeçalho do arquivo de dados, ou seja, o RRN da raiz, q quantidade de páginas na árvore e o RRN do primeiro espaço vazio. Em seguida são impressos os RRNs de todos os espaços vazios. Isso é feito simplesmente seguindo a fila de espaços vazios já mencionada anteriormente.

Em seguida são impressas as páginas que não estão vazias, ou seja, os nós da árvore. Iniciando na raiz, todas as informações do nó são impressas: número da página, quantidade de chaves, se é nó folha, bem como as chaves (ID e offset) e os filhos). Após a impressão das informações de uma página, são impressas as páginas filhas, uma a uma, em um processo recursivo. Dessa maneira, toda a árvore é percorrida.

Por possuir uma ocupação mínima de chaves de cerca de 50%, número máximo de níveis (d) que a árvore-B pode possuir é de:

$$d = 1 + \log_{\lceil m/2 \rceil}\left(\frac{(N+1)}{2}\right)$$

Em que, m é a ordem da árvore e N é o número de chaves. Sendo assim, a busca, que no pior caso tem que percorrer toda a altura da árvore para a encontrar uma chave, tem complexidade de tempo de:

$$O\left(\log_{\lceil m/2 \rceil}(N)\right)$$

Lembrando que as operações que são realizadas dentro de uma página, são irrelevantes para a complexidade de tempo, pois são feitas em memória principal. Apenas é relevante o número de acessos a disco.

As inserção e remoção necessitam de uma maneira ou de outra de realizar buscas (para encontrar o lugar em que a chave deve ser inserida, e para encontrar a chave a ser removida, respectivamente). Isso, somado ao fato de que a quantidade de operações adicionais realizadas por elas são independentes de N, leva que a complexidade delas são idênticas a da busca.

A impressão terá uma complexidade um pouco maior. Considerando que em cada página há no mínimo cerca de 50% de ocupação (ou [m/2] - 1), e considerando que a impressão deve passar por todas as páginas, a complexidade será:

$$O\left(\frac{N}{\lceil m/2 \rceil - 1}\right)$$

A complexidade de espaço é:

$$O(N)$$

Pois quanto mais chaves mais folhas são necessárias de maneira proporcional.

- **Programa principal (main):**

O programa principal inicia declarando uma estrutura *arvoreb_t*. Nessa estrutura está o ponteiro para o arquivo de índice, e variáveis que irão armazenar as informações do cabeçalho deste arquivo.

Assim, a árvore-B é inicializada. Lembrando que como mencionado anteriormente, isso pode significar criar uma árvore nova, ou abrir um arquivo de índice já preexistente.

Em seguida, verifica-se a hipótese de já haver uma arquivo de dados, mas de não haver o de índice. Se for o caso, uma função específica monta o arquivo de dados, lendo cada um dos registros e inserindo na árvore-B. Lembrando que a chave é formada pelo ID e pelo offset, então ambas as informações devem se extraídas do arquivo de dados.

Só a partir daí que se inicia a interação com o usuário. O programa oferece 5 opções ao usuário, que deve digitar sua escolha por meio do código:

1. Inserir usuario
2. Remover usuario
3. Pesquisar por ID
4. Mostrar Arvore-B
5. Fechar o programa

Vale ressaltar que neste programa as saídas são realizadas por meio de um arquivo de log (histórico). Se trata de um arquivo de texto (.txt), em que as mensagens de erro, de confirmação e os retornos de informações são impressos.

1. Inserir usuário

O programa solicita que o usuário digite o ID, o nome e o TU do usuário. É tratado o caso de tipos de usuários que não sejam os válidos. As informações são concatenadas em uma string, com campos separados por "|", que constitui o registro a ser adicionado ao arquivo de dados.

O arquivo de dados é aberto no modo append, pois a inserção é sempre no final do arquivo. O offset do fim do arquivo (que será o offset do registro) é obtido.

Antes de realmente adicionar o registro na arquivo de dados, o programa tenta adicionar sua chave (ID e offset) na árvore-B (arquivo de índice). Isso, pois caso o ID já exista nos dados, o retorno da função de inserir na árvore irá indicar isso.

Se a chave for inserida com sucesso na árvore, significa que o ID não é repetido, e pode ser inserido no arquivo de dados. Lembrando que primeiro coloca-se o tamanho do registro, e depois o registro em si no arquivo de dados.

2. Remover usuário

Primeiramente, pesquisa-se o ID do usuário a ser removido. Caso ele não existe, envia a mensagem de erro. Caso ele exista, o offset retornado pela função de busca já é usado para encontrar o registro no arquivo de dados e marcá-lo como removido, como mencionado anteriormente. Posteriormente, a chave que corresponde ao ID é removida na árvore-B (arquivo de índice).

3. Pesquisar por ID

A busca é feita por meio da árvore-B (arquivo de índice). A função busca, que toma por parâmetro o arquivo de índice e o ID pesquisado, retorna o offset do registro correspondente ao ID no arquivo de dados. Em posse do offset, basta recuperar o registro no arquivo de dados, separar os campos e imprimir as informações.

4. Mostrar Árvore-B

O programa faz uso da função de impressão da árvore-B já mencioando.

5. Fechar o programa

São fechados todos os arquivos abertos e liberada a memória de todas estruturas temporárias usadas.

O programa de encerra.