SCC 0503 – Algoritmos e Estruturas de Dados II (Sistemas de Informação)
SCC 0603 – Algoritmos e Estruturas de Dados II (Engenharia da Computação)

**Professora:** Elaine Parros Machado de Sousa – parros@icmc.usp.br
**Estagiários PAE:** Diego Silva – diegofsilva@icmc.usp.br
Ivone Penque Matsuno – ivone.matsuno@usp.br

# Trabalho 3 – Árvore-B

**1. Objetivo do Trabalho:**

Aplicar os conceitos estudados na disciplina de Algoritmos e Estrutura de Dados II para implementação de soluções para problemas práticos.
Neste trabalho, o aluno deverá desenvolver um programa em linguagem C ou C++ para resolução do problema especificado na **seção 4**.
O programa deve ser compilado no **gcc/g++ 4.2.7.** ou no **codeblocks 13.11**.

**2. Critérios de Avaliação**

O programa deve ser desenvolvido em grupo de no máximo 3 alunos e deve seguir rigorosamente os formatos de entrada e saída definidos. Quaisquer programas similares (parcialmente ou totalmente) terão **nota zero**, independente de qual for o original e qual for a cópia. Programas desenvolvidos em outras linguagens (diferentes de C/C++) não serão aceitos.

- A nota atribuída ao trabalho será de zero a dez inclusive.
- A nota máxima é atribuída se o trabalho avaliado atender a todos os requisitos e o trabalho estiver bem organizado e apresentado.
- Para cada requisito não atendido no trabalho, será descontada uma pontuação da nota até seu limite mínimo.
- Será atribuída nota ZERO quando:
  - o trabalho não for submetido até a data máxima possível;
  - ou/e o trabalho apresentar muitos erros;
  - ou/e não compilar;
  - ou/e for detectado plágio do trabalho (parcial ou total);
  - ou/e, não for entregue relatório conforme especificação;
  - ou/e a árvore-B não for implementada em disco;
  - ou/e as operações de inserção, busca e remoção forem realizadas sem o uso da árvore-B.

Os itens de avaliação dos programas são:

1. (50%) O programa funciona corretamente nas tarefas descritas para todos os casos de teste e processamento correto das entradas e saídas;
2. (20%) Qualidade da solução desenvolvida, modularização do código, bom uso das técnicas de programação e eficiência da solução em termos de espaço e tempo;
3. (10%) Boa identação e uso de comentários no código, além de boa estruturação e modularização;
4. (20%) Documentação externa: **relatório** curto (em pdf) explicando cada decisão tomada para a implementação da solução, sendo necessária a inclusão de detalhes sobre (pelo menos) a estrutura do arquivo de índice, estruturas de dados utilizadas, estratégias de implementação das operações de criação da árvore-B, inserção, remoção, e apresentação da árvore-B e estimativas de complexidade de tempo e espaço. Inclua figuras para ilustrar a organização do arquivo e as estruturas de apoio (nos moldes das figuras apresentadas em aula).

**3. Data e Forma de Entrega**

**Data de entrega:** 03/07/2015 (6a-feira) - até 23:55h.

A data de entrega será considerada a da última submissão do arquivo no Tidia-ae.

A cada dia de atraso será descontado 1 ponto da nota final do trabalho. Serão considerados os dias consecutivos independentemente se é dia útil ou não.

Após encerrado o prazo máximo de 10 dias de atraso, não serão aceitos mais trabalhos.

**Forma de entrega:**
A entrega será realizada no ambiente Tidia-ae na Atividade **Trabalho 3**.

**O que entregar?**
Você deve entregar os arquivos contendo **apenas o seu programa fonte e o relatório**. A pasta completa do projeto deve ser compactada em um único arquivo (com extensão “zip”).

O nome do arquivo compactado deve ser formado pela sigla “T3-” concatenada à 1ª letra do nome concatenada ao último sobrenome do integrante do grupo que fez a submissão do trabalho 3 no TIDIA.
Exemplo: T3-imatsuno, T3-dsilva, T3-esousa, etc.

**4. Descrição do Problema**

O projeto consiste em implementar um TAD para Árvore-B de ordem 6, com a função de indexação de dados, e as operações de pesquisa, inserção, e remoção. A árvore-B deve ser mantida em memória secundária (**em disco**).

Considere que os registros de dados (arquivo de dados) são formados pelos seguintes campos:

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

O campo tipo de usuário (TU) representa uma política do site para oferecer serviços e propaganda para os usuários. Os possíveis TU são: gratuito, comum e premium.

Para o arquivo de dados, implemente:

- O arquivo de dados é composto de registros de tamanho variável. No início de cada registro tem um byte que indica o tamanho total do registro. Na sequência são as informações referentes aos campos ID, NOME e TU que são separados pelo caractere "|".

O registro é definido da seguinte forma:

```
typedef struct{
      int id;
      char nome[50];
      int tu;
}tRegistro;
```

- **inserção**: somente no final do arquivo (para simplificar);
- **remoção**: somente com "marcador" de registro removido (para simplificar).

**OBS:** a busca para remoção no arquivo de dados deve ser feita a partir da estrutura de indexação.

O sistema deve armazenar seus dados e a estrutura de indexação (árvore-B) de modo a realizar consultas **eficientes**. As **funcionalidades** requeridas são:
Funcionalidade 1: Criação do (arquivo de) índice a partir do arquivo de dados (considerando a ordem dos registros no arquivo de dados)
Funcionalidade 2: Inserção de novos usuários
Funcionalidade 3: Remoção de usuários
Funcionalidade 4: Pesquisa por ID
Funcionalidade 5: Mostrar Árvore-B (funcionalidade bônus - valendo 2 pontos extras)

**IMPORTANTE:** apesar de estar especificada ordem 6 para criar a árvore-B, sua solução deve ser escalável. Em outras palavras, em uma situação real a árvore poderia ter uma ordem MUITO maior (e portanto a busca por chaves na página deve ser eficiente). Além disso, considere que o índice e o arquivo de dados não "cabem" em memória principal.

O programa deve permitir a interação pelo console/terminal (modo texto). Os requisitos funcionais do programa, necessários para realização das demais operações, são:

- Verificar se existem arquivos de índice (arvore.idx) e de dados (dados.dad). Se não existirem índices, mas existirem os arquivos de dados, então criar o índice.
- Verificar se o índice está atualizado. Caso não esteja, refazer o índice corretamente.
- Assuma que os arquivos de dados e índice estão no diretório raiz do programa.
- O programa deve fazer tratamento/validação de dados informados, como ID e tipo de usuário.
- A entrada de dados será ASCII e não serão considerados caracteres especiais (~, ç, ^, `, ’, etc.).

**5. Entrada**

A interação com o usuário por meio do console/terminal, deve seguir um formato padrão de entrada. Essas informações serão fornecidas no formato descrito a seguir.

O programa deve iniciar com a leitura de um código numérico inteiro **C**, indicando a operação a ser realizada (listada num menu). Considere os seguintes códigos de operação:

1. Inserir usuário (funcionalidade 2)
2. Remover usuário (funcionalidade 3)
3. Pesquisar por ID (funcionalidade 4)
4. Mostrar Árvore-B (funcionalidade 5)
5. Fechar o programa

Caso a operação escolhida seja a de código **1**, seu programa deve fazer a leitura de dados nessa ordem (e separados por \n):

- Número inteiro com ID do usuário
- Nome completo do usuário
- Número inteiro indicando o tipo de usuário (1 para gratuito, 2 para comum e 3 para premium)

Caso a operação escolhida seja a de código **2**, seu programa deve ler apenas um número inteiro discriminando o ID do usuário a ser removido. O mesmo vale para a operação de código **3,** em que o ID lido será buscado no arquivo.

Caso a operação escolhida seja a de código **4**, seu programa deve apresentar a árvore-B. Veja detalhes na seção **6.2**.

Após cada operação (exceto a de código 5), o programa deve retornar à leitura do código de operação **C**.

## 6. Saída

A saída deste trabalho deverá ser em um arquivo texto que contenha as mensagens para cada operação realizada.

### 6.1 Arquivo de *Log* (Histórico)

O programa deve criar um arquivo texto com o nome de "**log_X.txt**", em que X é 1ª letra do nome concatenada ao último sobrenome do integrante do grupo que fez a submissão do trabalho 3 no TIDIA.

Este arquivo deve conter os registros das operações executadas pelo programa e os principais passos executados pelo programa para a manipulação da Árvore-B.

### 6.2 Mensagens de Saída a Serem Gravadas no Arquivo de *Log*

Caso execução de criação de índice, gravar no arquivo de *log*:
"Execucao da criacao do arquivo de indice <NOME1> com base no arquivo de dados <NOME2>." <NOME1> e <NOME2> indicam, respectivamente, o nome dos arquivos de índice e de dados.

Caso operação **1**, gravar no arquivo de *log*:
"Execucao de operacao de INSERCAO de <ID>, <NomeCompleto>, <tipo>." <ID>, <NomeCompleto> e <tipo> indicam os dados do usuário a ser inserido.

Na sequência gravar no arquivo de *log*:

- "Divisao de no - página X" deve ser impressa sempre que um nó for dividido - X é // chave numericada página dividida;
- "Chave <*ID*> promovida" deve ser impressa sempre que uma chave for promovida.
- "Chave <*ID*> inserida com sucesso" deve ser impressa ao final da inserção indicando sucesso da operação.
- "Chave <*ID*> duplicada" deve ser impressa ao final da inserção e indica que a operação de inserção não foi realizada.

Caso operação **2**, gravar no arquivo de *log* :
"Execucao de operacao de REMOCAO de <ID>." <ID> indica o número a ser removido.
Na sequência gravar as mensagens:

- "Redistribuicao de chaves - entre as páginas irmas X e Y" deve ser impressa sempre que as chaves de um nó forem redistribuídas;

- "Concatenacao- entre as páginas irmas X e Y" deve ser impressa sempre que nós forem concatenados;
- "Chave <*ID*> promovida" deve ser impressa sempre que uma chave for promovida. <*ID*> é o valor da chave promovida;
- "Chave <*ID*> rebaixada" deve ser impressa sempre que uma chave for rebaixada. <*ID*> é o valor da chave promovida;
- "Chave <*ID*> removida com sucesso" deve ser impressa ao final da remoção indicando sucesso da operação.
- "Chave <*ID*> não cadastrada" deve ser impressa ao final da remoção e indica que a operação de remoção não foi realizada.

Caso operação **3**, gravar no arquivo de *log*:
"Execucao de operacao de PESQUISA de <ID>". <ID> indica o número a ser pesquisado.
Na sequência gravar no arquivo de *log*:

- "Chave <*ID*> encontrada, offset <*Y*>
  Nome: <nome_usuario>, Tipo Usuário: <TU>" , indica que a chave <*ID*> foi encontrada e possui offset associado <*Y*> e os dados do registro são <nome_usuario> e <TU>.
- "Chave <*ID*> nao encontrada" indica que a chave <*ID*> não está presente na árvore-B.

Caso operação **4**, gravar no arquivo de *log*:
"Execucao de operacao para mostrar a arvore-B gerada:"

Na sequência gravar as informações da seguinte maneira: cada nó (página) da árvore deve ser impressa em uma linha, precedida pelo seu nível. A raiz é considerada nível 0, seus filhos como nível 1 e assim sucessivamente. A impressão deve ser por largura, ou seja, nó, e todas as respectivas chaves, de nível 0, seguidos por todos os nós, e respectivas chaves, de nível 1, etc. A cada página, deve-se imprimir um número inteiro descrevendo o nível da árvore, seguido de outro inteiro *n*, representando o número de chaves desse nó. Após isso, deve-se imprimir *n* elementos do tipo <chave/*offset*>, representando a chave e o *offset* onde ela está localizada no arquivo de dados.

Exemplo: Considere uma árvore de ordem 3 como a apresentado na Figura 1.

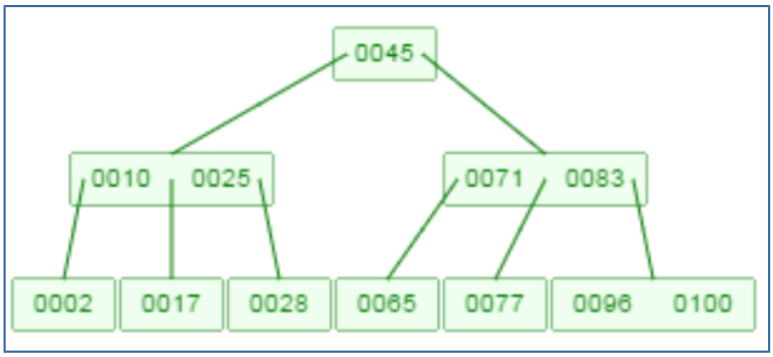


**Figura 1 - Exemplo de Árvore-B (ordem 3)**

Saída após execução de Mostrar a Árvore-B apresentada na Figura 1.

```
Execucao de operacao para mostrar a arvore-B gerada:
0  1 <45/205>
1  2 <10/149>  <25/100>
1  2 <71/210>  <83/160>
2  1 <2/130>
2  1 <17/500>
2  1 <28/230>
2  1 <65/450>
2  1 <77/300>
2  2 <96/400> <100/270>
```

**ATENÇÃO**

- Não deverá ser utilizada qualquer variável global.
- Não poderão ser utilizadas bibliotecas com funções prontas (a não ser aquelas para entrada, saída e alocação dinâmica de memória).
- Dúvidas conceituais deverão ser colocadas nos horários de atendimento.