



Universidade de São Paulo

Instituto de Ciências Matemáticas e de Computação

Departamento de Ciências de Computação

SCC0215 - Organização de Arquivos – Projeto 2

## Projeto 2 : Implementação do Índice Árvore B\*.

GABRIEL PINTO DE CAMARGO N<sup>o</sup> USP: 9293456

GABRIEL SIMMEL NASCIMENTO N<sup>o</sup> USP: 9050232

MARCOS CESAR RIBEIRO DE CAMARGO N<sup>o</sup> USP: 9278045

VICTOR LUIZ ROQUETE FORBES N<sup>o</sup> USP: 9293394

Profa. Dra. Cristina D. A. Ciferri

São Carlos – SP

13 de Junho de 2016



# Sumário

<b>1</b>	<b>Introdução</b>	<b>3</b>
<b>2</b>	<b>Macros</b>	<b>3</b>
<b>3</b>	<b>Modificações em relação a primeira parte do projeto</b>	<b>4</b>
<b>4</b>	<b>Definição de Árvore B*</b>	<b>4</b>
<b>5</b>	<b>Estrutura de Dados</b>	<b>5</b>
<b>6</b>	<b>Ordem da Árvore B*</b>	<b>5</b>
6.1	Problema	6
<b>7</b>	<b>Implementação</b>	<b>7</b>
7.1	Inicialização	7
7.2	Impressão	7
7.3	Busca	8
7.4	Inserção	8
<b>8</b>	<b>Interface</b>	<b>9</b>
<b>9</b>	<b>Testes</b>	<b>9</b>
	<b>REFERÊNCIAS</b>	<b>13</b>



## 1 Introdução

O trabalho consiste na implementação de um índice Árvore B\* de ordem  $m$  para os registros criados pelo programa implementado na primeira parte do projeto. Foram inseridas novas funcionalidades ao código para permitir a utilização do índice. O programa, implementado em linguagem C, ao inserir um registro no arquivo de dados insere a chave de busca na Árvore B\*, junto de seu *offset*, explicaremos nesta documentação as decisões tomadas e como as funções foram implementadas. Foi utilizado como referência o Manual do Linux ([EATON, 1990](#)).

O programa foi compilado e testado no sistema operacional Linux (Ubuntu 16.04 lts), utilizando um arquivo Makefile com as diretivas all, run e debug.

- make - comando de compilação.
- make debug - compila com as flags de debug.
- make clean - apaga os arquivos de dados .bin, .idx, .header e o executável main.
- make run - comando de execução.
- make test1 - executa utilizando o arquivo teste1.in como entrada (testa a parte 1).
- make test2 - executa utilizando o arquivo teste2.in como entrada (testa a parte 2).
- make fullrun - executa com valgrind e com suas flags.
- make fulltest1 - executa com o valgrind utilizando o arquivo teste1.in como entrada.
- make fulltest2 - executa com o valgrind utilizando o arquivo teste2.in como entrada.
- make zip - cria um arquivo zip com todos arquivos da pasta raíz.

## 2 Macros

Optamos por utilizar macros para melhorar a leitura e interpretação do código, além de garantir sua generalização. Seguem descritas abaixo:

- NIL - *Not in List*, representa um elemento que não está na lista. É representado por um inteiro com -1.
- PAGESIZE - representa o tamanho da página de disco, utilizaremos 64 bytes.
- M - representa a ordem da árvore, descrito na [seção 6](#).
- SHORT - tamanho de um short, utilizado somente nas diretivas de compilação.
- INT - tamanho de um int, utilizado somente nas diretivas de compilação.

### 3 Modificações em relação a primeira parte do projeto

Em relação a primeira parte foram realizadas algumas modificações em nosso código. Optamos por mudar:

- a) Tipo do ID de uma série: unsigned int para int, para poder assumir o valor NIL(-1). Apesar de mudarmos para int, os IDs continuam assumindo apenas valores maiores ou iguais a zero (com exceção do NIL) .
- b) Operação de inserção no arquivo de dados: retorna o offset em que o novo dado é inserido.
- c) Buscas: passam a utilizar o índice Árvore B\*.
- d) ID único: a verificação de IDs repetidos passa a ser feita na Árvore B\*.
- e) Gerar séries: adicionada interação com o usuário, para que ele possa escolher a quantidade de séries a serem geradas.

### 4 Definição de Árvore B\*

Árvore B\* é uma variação da Árvore B proposta por Knuth em 1973, tem as seguintes propriedades:

1. Cada página possui no máximo  $m$  descendentes, e no máximo  $m - 1$  chaves.
2. Cada página, exeto a raiz e as folhas, possui no mínimo  $\frac{(2m-1)}{3}$  descendentes.
3. A raiz possui no mínimo 2 descendentes, a menos que seja um nó folha.
4. Todas as folhas aparecem no mesmo nível.
5. Uma página interna com  $k$  descendentes contém  $k - 1$  chaves.
6. Uma folha possui no mínimo

$$\lfloor \frac{(2m - 1)}{3} \rfloor \quad (1)$$

chaves.

Essas características causam em média, para árvores grandes, um índice de ocupação de 69%. Podemos melhorar o índice de ocupação implementando a redistribuição na inserção, podendo chegar até 86%, que será feita neste projeto.

## 5 Estrutura de Dados

Foram criadas novas estruturas de dados para implementar a Árvore B\*:

- BTree: contém os ponteiros para os arquivos de índice e cabeçalho, e também uma referência para o cabeçalho em memória.
- Header: contém a altura da árvore, a ordem da árvore, a quantidade máxima de chaves em um nó, a taxa mínima de ocupação de um nó intermediário e a taxa de mínima de ocupação de um nó folha, o RRN do nó raiz, e o número total de elementos na árvore.
- Node: contém um vetor de M-1 inteiros de 4 *bytes* que representam as chaves primárias de busca, um vetor M-1 inteiros de 4 *bytes* que representam o *offset* de cada registro no arquivo de dados, e um vetor de M inteiros de 2 *bytes* que representa o RRN dos nós filhos de cada chave.

## 6 Ordem da Árvore B\*

Para definir a ordem da árvore precisamos encontrar um valor de  $m$  que respeite as propriedades da Árvore B\* e que permita que cada nó caiba em uma página de disco (e que não tenha muito desperdício de memória), ou seja:

$$2(\text{sizeof}(\text{int})(m-1)) + \text{sizeof}(\text{short})(m+1) \leq \text{PAGESIZE} \quad (2)$$

que pode ser obtida a partir da estrutura de dados descrita na [seção 5](#).

O valor  $m$  deve ser escolhido de tal forma que ao dividir 2 nós cheios em 3 nós, nenhum dos nós resultantes possua menos elementos do que a taxa de ocupação mínima (1). Tal relação é dada por:

$$\lfloor \frac{2(m-1)}{3} \rfloor \geq \lfloor \frac{2m-1}{3} \rfloor \quad (3)$$

Dessa relação obtemos que  $m = \{3k - 1, \forall k \in \mathbb{N}\}$ .

Como a ordem deve respeitar (2) e (3), podemos escolher no máximo  $m = 7$ , valor que será usado no projeto. Nosso programa foi feito para se adaptar de acordo com o tamanho da página definida, utilizando diretivas de compilação. Basta alterar a macro PAGESIZE para que uma nova ordem  $m$  seja gerada em etapa de pré-compilação. As condições (2) e (3) são verificadas em pré-compilação também, utilizando as macros INT e SHORT, garantindo que  $m$  seja sempre um valor válido. A macro PAGESIZE deve ser definida com um valor maior ou igual a 24 bytes, que seria o equivalente a ordem 3, pois a ordem 2 é inválida pela relação (3) e ordens menores não fazem sentido.

## 6.1 Problema

Ao realizarmos o *split 1-to-2* na raiz, seus nós resultantes desrespeitarão as restrições 2 e 6 da definição de Árvore B\* (seção 4) para a maior parte das ordens  $m$  escolhidas. Para a ordem 7 apenas a restrição 6 é desrespeitada, pois o *split* gerará dois nós com 3 elementos cada, lembrando que a taxa mínima de ocupação de um nó folha é 4 (seção 4.6), e de um nó intermediário é 3 (seção 4.2). Isso ocorre apenas no primeiro *split 1-to-2* da raiz, resultando a seguinte disposição de Árvore (Figura 1):

```
Ordem: 7 / Tamanho da pagina de disco: 64 / Quantidade de registros no arquivo: 7

-----Nível 1-----

RRN 000:
(001) 38424 (002) NIL ( * ) NIL ( * ) NIL ( * ) NIL ( * ) NIL ( * )

-----Nível 2-----

RRN 001:
( * ) 03113 ( * ) 10782 ( * ) 13354 ( * ) NIL ( * ) NIL ( * ) NIL ( * )

RRN 002:
( * ) 63310 ( * ) 79606 ( * ) 86685 ( * ) NIL ( * ) NIL ( * ) NIL ( * )

-----
```

Figura 1: Árvore B\* de ordem 7 após a 7ª inserção.

No melhor caso (no qual as próximas 2 chaves inseridas vão uma para cada nó filho), as restrições da Árvore B\* são desrespeitadas para uma quantidade de registros no intervalo [7, 8].

No pior caso (no qual as próximas 3 chaves inseridas vão todas para o mesmo nó filho), as restrições da Árvore B\* são desrespeitadas para uma quantidade de registros no intervalo [7, 10].

Optamos por manter esse "problema" na implementação final, pois sua solução (dobrar o tamanho da raiz) se mostra menos eficiente, devido a necessidade de 2 acessos a disco para ler a raiz, e o intervalo onde ocorre o problema no pior caso é insignificante (de 7 a 10 chaves).



## 7 Implementação

Para podermos implementar as funções da Árvore B\*, precisamos esclarecer algumas decisões de implementação, são elas:

- Por questão de simplicidade escolhemos utilizar um arquivo separado para o cabeçalho do índice, caso não exista é criado um novo cabeçalho vazio. A validade do conteúdo do cabeçalho não é verificada, pois assumimos que a execução do programa não será interrompida. O cabeçalho é gravado em disco ao final do programa.
- A chave primária de busca é o ID único de cada série.
- Para podermos realizar o split na raiz, utilizamos o *split 1-to-2*. Como a raiz contém no máximo 6 chaves, o *split 1-to-2* gera 3 chaves para cada filho e uma chave é promovida, porém o mínimo dos nós folhas é 4. Isso desrespeita a taxa de ocupação mínima dos nós folha, ocorre apenas na primeira vez que o split da raiz é chamado. Esta taxa de ocupação é corrigida quando mais chaves são inseridas, pois no próximo *split 1-to-2* os nós filhos gerados a partir são intermediários, portanto sua taxa mínima de ocupação é 3 (seção 4.2).

### 7.1 Inicialização

Quando o programa é iniciado, ele verifica se o arquivo de cabeçalho já existe, caso exista ele verifica se o tamanho da página de disco descrita nele confere com o valor definido pelo usuário na macro PAGESIZE, isso garante o funcionamento correto do programa para índices já gerados. Se não existir, um cabeçalho vazio é criado e a partir dele será gerado um novo índice.

### 7.2 Impressão

A função de impressão da Árvore B\* é feita a partir de uma BFS (*Breadth-first search*), no qual cada nó é retirado de uma fila, impresso e seus filhos são adicionados a fila. Cada nó é representado como descrito na Figura 2, os níveis são separados e indicados para facilitar a visualização da árvore.

```
-----Nível 1-----
RRN 000:
(009) 49105 (010)  NIL  ( * )  NIL  ( * )  NIL  ( * )  NIL  ( * )  NIL  ( * )
-----
```

Figura 2: Representação de um nível impresso. Asteriscos representam nós descendentes.

### 7.3 Busca

A busca de uma chave é feita a partir do índice Árvore B\*. Foi utilizado um algoritmo iterativo, pois não há necessidade de salvar as chamadas anteriores durante a busca. Iniciando no nó raiz a chave é buscada em ordem crescente no nó (enquanto a chave for menor que o elemento da lista). Caso seja encontrada retorna seu *offset* associado no arquivo de dados, caso não seja encontrada a busca é reiniciada no filho apontado pelo contador, se o filho existir. Se a chave não for encontrada, a busca retorna NIL.

### 7.4 Inserção

A inserção foi implementada de forma recursiva, pois nos casos de propagação de overflow é necessário acessar nós predecessores ao nó que gerou overflow. Foi feita com base nos passos descritos abaixo:

- Inserção em uma árvore vazia - É criado um novo nó raiz, no qual a chave é inserida.
- Inserção no nó raiz - As chaves são inseridas ordenadamente, até que o nó fique cheio.
- Inserção no nó raiz, overflow - Quando o nó raiz está cheio, é necessário realizar o *split 1-to-2*, aumentando assim da altura da árvore, e gerando dois novos nós. Vale ressaltar que o RRN do nó raiz é não se altera.
- Inserção em nós folhas não cheios - As chaves são inseridas ordenadamente, enquanto houver espaço.
- Inserção em nós folhas cheios, redistribuição - Caso o nó esteja cheio, tenta-se realizar a redistribuição com o nó da esquerda, se não for possível tenta-se com o da direita. Optamos por redistribuir as chaves igualmente entre os nós, caso o número de chaves seja ímpar o nó da direita recebe uma chave a mais.
- Inserção em nós folhas cheios, split - Caso o nó esteja cheio, e não seja possível realizar a redistribuição, o *split 2-to-3* é realizado. O nó que gerou o overflow é retornado para que ele possa ser tratado pela chamada anterior da recursão. No *split 2-to-3* pode ocorrer overflow no nó predecessor, caso isso aconteça tentamos redistribuir ou então realizar novamente o *split*.

## 8 Interface

A interface do programa é feita via terminal, o menu (Figura 3) apresenta as opções disponíveis para o usuário. As funcionalidades são:

1. Gerar uma quantidade, fornecida pelo usuário, de no máximo 250 séries.
2. Insere uma série manualmente, pela entrada padrão.
3. Busca uma chave no índice Árvore B\*.
4. Imprime todas as séries percorrendo o arquivo sequencialmente.
5. Imprime a árvore gerada pelas inserções no índice, como descrito em 7.2.

```
joey@Joey:~/Documents/Arquivos/Final$ make run
Nome do arquivo de dados: Series

1 - Gerar series.
2 - Inserir serie.
3 - Buscar serie.
4 - Imprimir series.
5 - Imprimir Arvore B*.
0 - Sair.
```

Figura 3: Interface principal do programa.

## 9 Testes

Foi realizada uma bateria de testes para demonstrar o funcionamento do nosso programa. Nesta seção demonstraremos o passo-a-passo do teste realizado, explicando sucintamente cada passo.

1. [Figura 4](#) - Menu principal com o arquivo de dados carregado (inicialmente vazio).
2. [Figura 5](#) - São geradas automaticamente 30 séries e inseridas no arquivo de dados e no índice Árvore B\*.
3. [Figura 6](#) - Impressão da árvore de chaves primárias com 30 itens inseridos.
4. [Figura 7](#) - Saíndo do programa, fazendo com que o arquivo de cabeçalho seja escrito em disco.
5. [Figura 8](#) - Carregando o arquivo de índices, o arquivo de cabeçalho e também o arquivo de dados (gerados anteriormente).
6. [Figura 9](#) - Inserindo mais 15 séries geradas automaticamente.
7. [Figura 10](#) - Realizando nova impressão da árvore, com as novas chaves inseridas.

```
joey@Joey:~/Documents/Arquivos/Final$ make run
Nome do arquivo de dados: Series

1 - Gerar series.
2 - Inserir serie.
3 - Buscar serie.
4 - Imprimir series.
5 - Imprimir Arvore B*.
0 - Sair.
```

Figura 4: Criando arquivo de dados.

```
1 - Gerar series.
2 - Inserir serie.
3 - Buscar serie.
4 - Imprimir series.
5 - Imprimir Arvore B*.
0 - Sair.
1

Quantas series gostaria de gerar? (Maximo de 250): 30
Series geradas com sucesso.
```

Figura 5: Gerando 30 séries.

```
Ordem: 7 / Tamanho da pagina de disco: 64 / Quantidade de registros no arquivo: 30
-----Nivel 1-----
RRN 000:
(001) 13975 (002) 32385 (005) 51444 (004) 81476 (003) NIL ( * ) NIL ( * )
-----
-----Nivel 2-----
RRN 001:
( * ) 02794 ( * ) 03833 ( * ) 06083 ( * ) 06649 ( * ) NIL ( * ) NIL ( * )
RRN 002:
( * ) 14825 ( * ) 20680 ( * ) 26143 ( * ) 28003 ( * ) 28371 ( * ) NIL ( * )
RRN 005:
( * ) 41519 ( * ) 42068 ( * ) 44086 ( * ) 46683 ( * ) 49105 ( * ) NIL ( * )
RRN 004:
( * ) 58200 ( * ) 63696 ( * ) 65865 ( * ) 72716 ( * ) 76332 ( * ) 77939 ( * )
RRN 003:
( * ) 84440 ( * ) 86280 ( * ) 86694 ( * ) 87166 ( * ) 88728 ( * ) 91136 ( * )
-----
```

Figura 6: Impressão da Árvore B\* com 30 itens.

```
1 - Gerar series.  
2 - Inserir serie.  
3 - Buscar serie.  
4 - Imprimir series.  
5 - Imprimir Arvore B*.  
0 - Sair.  
0
```

```
joey@Joey:~/Documents/Arquivos/Final$
```

Figura 7: Saíndo do programa.

```
joey@Joey:~/Documents/Arquivos/Final$ make run  
Nome do arquivo de dados: Series  
Arquivo carregado com sucesso!  
  
1 - Gerar series.  
2 - Inserir serie.  
3 - Buscar serie.  
4 - Imprimir series.  
5 - Imprimir Arvore B*.  
0 - Sair.
```

Figura 8: Carregando arquivo de dados já existente.

```
Quantas series gostaria de gerar? (Maximo de 250): 15  
Series geradas com sucesso.
```

Figura 9: Gerando 15 séries adicionais.

Ordem: 7 / Tamanho da pagina de disco: 64 / Quantidade de registros no arquivo: 45

```

-----Nivel 1-----
RRN 000:
(009) 49105 (010) NIL ( * ) NIL ( * ) NIL ( * ) NIL ( * ) NIL ( * )
-----
-----Nivel 2-----
RRN 009:
(001) 11331 (002) 25332 (007) 37465 (005) NIL ( * ) NIL ( * ) NIL ( * )
RRN 010:
(004) 65865 (006) 77939 (003) 86694 (008) NIL ( * ) NIL ( * ) NIL ( * )
-----
-----Nivel 3-----
RRN 001:
( * ) 02648 ( * ) 02794 ( * ) 03833 ( * ) 06083 ( * ) 06649 ( * ) NIL ( * )
RRN 002:
( * ) 12952 ( * ) 13975 ( * ) 14825 ( * ) 20680 ( * ) 24882 ( * ) NIL ( * )
RRN 007:
( * ) 26143 ( * ) 28003 ( * ) 28371 ( * ) 30530 ( * ) 32385 ( * ) NIL ( * )
RRN 005:
( * ) 41277 ( * ) 41519 ( * ) 42068 ( * ) 42326 ( * ) 44086 ( * ) 46683 ( * )
RRN 004:
( * ) 51444 ( * ) 56878 ( * ) 58200 ( * ) 59323 ( * ) 63696 ( * ) NIL ( * )
RRN 006:
( * ) 70500 ( * ) 72716 ( * ) 76288 ( * ) 76332 ( * ) NIL ( * ) NIL ( * )
RRN 003:
( * ) 79724 ( * ) 81476 ( * ) 84440 ( * ) 86280 ( * ) NIL ( * ) NIL ( * )
RRN 008:
( * ) 87166 ( * ) 88728 ( * ) 91136 ( * ) 98901 ( * ) NIL ( * ) NIL ( * )
-----

```

Figura 10: Impressão da Árvore B\* com 45 itens.

# Referências

EATON, J. W. *Linux User's Manual*. 2.7.5. ed. [S.l.], 1990. Citado na página [3](#).