# Um Novo Algoritmo de Concorrência para Acesso e Compactação de Árvores-$B$

A. Zisman* V. W. Setzer

Departamento de Ciência da Computação
Instituto de Matemática e Estatística
Universidade de São Paulo
C.P. 20570 - 01452-990 - São Paulo, S.P.
Fone: (011) 8139499 - FAX: 8144135
e-mail: vwsetzer@ime.usp.br

**Resumo**

Esse trabalho apresenta inicialmente uma breve (mas completa) resenha sobre as várias soluções do problema de controle de concorrência de processos percorrendo árvores-$B$. Em seguida é proposta uma nova solução que se caracteriza por empregar índices de tamanho variável, utilizar apenas um único tipo de bloqueio nas operações usuais de acesso à árvore, e empregar desdobramentos prévios, e concatenações e redistribuições adiadas. É apresentado um novo algoritmo de compactação e sua execução concorrente, utilizando um novo tipo de bloqueio.

**Abstract**

This paper presents a brief but fairly exhaustive survey of solutions to the concurrency control problem for B-trees. We then propose a new solution, which is characterized by the use of variable-lenght indices, the employment of a single lock type for the usual access operations and preemptive splits as well as delayed concatenations and distributions. We also introduce a new compactation algorithm and its concurrent execution, using a new lock type.

*Atualmente na RCM Informática; com apoio da FAPESP durante a realização desse trabalho.

# 1 Introdução

A árvore-$B$ introduzida por Bayer e McCreight em 1972 [1] tornou-se a estrutura padrão para implementação de índices em gerenciadores de arquivos e de banco de dados, pela sua eficiência nos acessos sequencial e aleatório por valores do índice. (Uma resenha extensa e atual sobre árvores-$B$ foi desenvolvida por Zisman [27].) O compartilhamento de arquivos em ambientes multiusuários levou à necessidade de se introduzir dispositvos para permitir acesso concorrente aos arquivos de dados e de índices, preservando a integridade e consistência das informações obtidas ou fornecidas pelos usuários.

Neste trabalho apresentamos um novo método de concorrência que se aplica a dois tipos de árvores-$B$ e algumas de suas variações tendo as seguintes características. 1) Reduzido número de nós explicitamente bloqueados (no máximo 2) por cada processo em execução a cada instante, exceto durante compactação. 2) Utilização de 2 tipos de bloqueios, sendo um deles reservado para a operação de compactação a qual, não abordada anteriormente na literatura sobre concorrência em árvores-$B$, pode ser realizada ao mesmo tempo com as operações de leitura, inserção, remoção e atualização. 3) Produz operações adiadas de concatenação de nós e preventivas de redistribuição de elementos de nós e de desdobramentos de nós, permitindo assim um bom fator de ocupação de espaço e evitando a repetição de caminhos já percorridos por um processo em execução.

O restante deste artigo é organizado como se segue. Na seção 2 definimos alguns conceitos necessários para compreensão do texto como um todo e caracterizamos os tipos de árvores que serão utilizadas. Na seção 3 fazemos uma breve resenha sobre todos os trabalhos publicados a respeito de concorrência em árvores-$B$, precedido de definições específicas para a seção. A seção 4 apresenta nosso método com definições específicas, aspectos gerais, algoritmo, problema de deadlock e consistência e uma extensão para valores de índices com tamanho variável. Finalmente na seção 5 resumimos as conclusões obtidas e fazemos propostas para futuras pesquisas.

# 2 Definições Gerais

**D1** Dado um arquivo de dados composto por registros de dados, um *índice* é um campo simples ou composto, de cada um desses registros, cujo *valor* é usado na busca dos mesmos.

**D2** Uma árvore-$B$ de ordem $K$, onde $K$ é um número natural, é uma árvore orientada que obedece as seguintes propriedades. 1) Cada nó da árvore é armazenado em uma e apenas uma página do disco. 2) Cada nó, exceto a raiz, tem $m$ valores de tamanho fixo do índice, onde $K \leq m \leq 2K$, armazenados no nó em ordem não decrescente. 3) A raiz tem entre 1 e $2K$ valores do índice. 4) Todo nó $n$ da árvore ou é uma folha, isto é, não tem descendentes, ou tem $m+1$ filhos; a cada um desses filhos está associado um ponteiro $p$ de tamanho constante armazenado em $n$. 5) A cada valor $i$ de índice em um nó está associado um ponteiro $d$ para o registro de dados que contém $i$. 6) Todos os nós folhas estão no mesmo nível, ou seja, todos os caminhos da raiz até uma folha têm o mesmo comprimento.

**D3** Considere $p_j$ e $p_{j+1}$ $(0 \leq j \leq m-1)$ dois ponteiros adjacentes em um nó $n$ não folha que apontam para $n'$ e $n''$, respectivamente. O valor do índice $i_{j+1}$ é denominado de *separador* de $n'$ e $n''$.

**D4** *Árvore-$B^*$*. Visando diminuir a altura da árvore-$B$ (v. **D2**), Knuth sugeriu em 1973 [11], uma variação desta denominada de árvore-$B^*$[1]. Esta estrutura contém todos os registros de

[1]Na realidade Knuth não atribuiu nenhum nome a esta variação da árvore-$B$. Uma boa parte dos autores

dados nas folhas formando a *região de dados*, as quais são conectadas por ponteiros formando uma lista ligada. Os nós internos contêm apenas valores do índice e ponteiros para os nós filhos, formando a *região de índices*.
**D5** *Árvore-$B^+$*. Variação da árvore-$B^*$ onde os nós folhas contêm ponteiros para os registros de dados e não os próprios registros. Os nós folhas formam também uma lista ligada através de ponteiros para os vizinhos à direita.
**D6** Um nó $n$ é *completo* se $m = 2K$.
**D7** Um nó $n$ torna-se *sobrecarregado* se $m = 2K$ e houver a tentativa de inserção de mais um valor em $n$.
**D8** Um nó $n$ torna-se *subcarregado* se $m = K$ e houver a tentativa de remoção de um valor de $n$.
**D9** O *espaço livre* de um nó é o comprimento (em bytes ou palavras) da parte do nó que não contém valores e ponteiros.

Quando um nó $n'$ torna-se sobrecarregado o mesmo é *desdobrado* ou executa-se uma *repartição* entre seus elementos e os nós vizinhos.
**D10** O *desdobramento* de um nó $n'$ consiste em se criar um novo nó $n''$, irmão adjacente de $n'$, e dividir o conteúdo de $n'$ entre $n'$ e $n''$, de tal forma que $n'$ contenha os $K$ menores elementos, $n''$ os $K$ maiores e o valor mediano entre todos os valores de $n'$ e $n''$ torna-se o separador (v. **D3**) destes nós sendo inserido no nó pai de $n'$, juntamente com um apontador para $n''$.
**D11** Denominamos de *desdobramento de 2 para 3* a técnica introduzida em [11] na qual os elementos de um nó $n'$ sobrecarregado são unidos com os elementos de um irmão $n''$ adjacente e divididos (o mais igualmente possível) entre $n'$, $n''$ e um novo nó $n'''$ irmão adjacente de $n''$.
**D12** A *repartição* dos valores de $n'$ consiste em mover elementos de $n'$ para um de seus irmãos $n''$ adjacente que não está completo, caso exista. O método caracteriza-se por unir os elementos de $n'$ e $n''$, juntamente com o separador destes e com o novo valor a ser incluído e em seguida distribuir esses valores entre $n'$ e $n''$, colocando no seu nó pai o valor mediano como separador, de forma que a quantidade de valores em $n'$ e $n''$ difira no máximo de 1.

Quando um nó $n'$ torna-se subcarregado o mesmo é *concatenado* com um de seus irmãos adjacentes $n''$ (processo inverso ao desdobramento), ou faz-se uma *redistribuição* entre os elementos de $n'$ e $n''$.
**D13** A *concatenação* consiste em se unir os elementos de $n'$ com os de $n''$ e com o separador destes, quando o número total desses elementos não ultrapassar $2K$. Nesse caso, os elementos da união são colocados em um dos nós e elimina-se o outro. No caso em que a cardinalidade da união for maior do que $2K$, executa-se uma *redistribuição* entre $n'$ e $n''$ de forma análoga à operação de repartição.
**D14** Um processo que percorre a árvore-$B$ pode ser classificado em *leitor*, o qual executa a operação de busca de um determinado valor $i$ do índice, ou *atualizador*, responsável por uma operação de inserção (ou de remoção) de um valor. Um processo atualizador é dividido em 2 sub-processos: o *atualizador-leitor* que executa a busca do nó onde (de onde) será inserido (removido) um valor, e *atualizador-gravador* que realiza a inserção (remoção) com a possível reestruturação da árvore.
**D15** Os nós da árvore visitados por um processo $P$ formam o *caminho de acesso* de $P$.

denominam-na de árvore-$B^*$. Por outro lado, Comer [5], Korth [12], Boswell e Tharp [4] classificaram a árvore-$B^*$ de Knuth de árvore-$B^+$.

**D16** Um processo *compactador* de uma árvore-$B$ percorre toda árvore reorganizando-a, afim de diminuir o espaço livre (v. **D9**) dos nós.
**D17** Um processo produz um *bloqueio* em um nó $n$ quando ele associa a $n$ uma marca indicando alguma restrição de acesso a $n$ por parte de outros processos. Um processo *desbloqueia* $n$ quando ele retira de $n$ um bloqueio anterior por ele efetuado.
**D18** Diz-se que um nó $n$ contém um bloqueio *exclusivo* produzido por um processo $P$, quando $n$ não pode ser visitado por nenhum outro processo até que $P$ o libere. Quando um processo $P$ está na situação imediatamente anterior à realização de uma alteração em um nó $n$, $P$ deve realizar um bloqueio exclusivo em $n$. Quando um processo encontra um nó $n$ com bloqueio exclusivo ele fica em estado de espera, em uma fila associada a $n$, até que $n$ seja desbloqueado.
**D19** Um nó $n$ é chamado de *seguro* quando uma operação de inserção ou remoção não afeta nenhum dos seus antecedentes; caso contrário, será chamado de *não seguro*. Resulta que para uma inserção um nó é seguro quando ele contém menos do que $2K$ valores do índice; para uma remoção, um nó está seguro quando seu número de valores é maior do que $K$.

# 3 Soluções de Concorrência

Para permitir a utilização das árvores-$B$, juntamente com suas variações, nas aplicações de multiprocessamento e multiprogramação surgiram várias soluções para o problema de concorrência. Essas soluções serão apresentadas em ordem cronológica de aparecimento na literatura.

## 3.1 Definições Específicas

Definimos adiante algumas técnicas de bloqueios (cujos nomes foram dados por Kwong e Wood [13, 14, 15] ), que são utilizadas nas soluções a serem apresentadas.
**D20** *Lock-Coupling.* Durante a execução de um processo leitor (v. **D14**) um nó é desbloqueado apenas quando seu filho for bloqueado. Durante a execução de um processo atualizador todos os nós do seu caminho de acesso (v. **D15**) são bloqueados até que um nó $n$ seguro (v. **D19**) seja visitado. Neste instante, liberam-se os bloqueios dos ancestrais de $n$ pertencentes ao caminho.
**D21** *Driving-off.* Um processo atualizador $P$ interrompe a execução de outro processo fazendo bloqueios exclusivos (v. **D18**) nos nós visitados por $P$.
**D22** *Side-Branching.* Quando um nó $n$ torna-se sobrecarregado (subcarregado) ele não é desdobrado (concatenado). Copia-se uma metade apropriada dos seus elementos em um novo nó, adicionando-se o elemento a ser inserido nessa nova metade, deixando o original intacto (une-se a seus elementos os elementos do irmão adjacente, retirando-se o elemento a ser removido). Continuando o percurso, ao encontrar-se um nó seguro, removem-se, de cima para baixo, as metades apropriadas de cada nó (os nós redundantes) do caminho de acesso.
**D23** *Árvore-$B^L$.* Introduzida por Lehman e Yao em 1981 [17], a árvore-$B^L$ (do original árvore-$B^{Ligada}$) é uma variação da árvore-$B^+$ (v. **D5**). Cada nó da árvore contém um par (valor, ponteiro) a mais, isto é, para todo nó $n$ existe um par $(i_{m+1}, p_{m+1})$, $K \leq m < 2K$, onde o conteúdo de $i_{m+1}$ é o maior valor existente na sub-árvore enraizada por $p_m$ em $n$, e $p_{m+1}$, chamado de *ponteiro de ligação*, aponta para o nó vizinho à direita de $n$.
**D24** *Árvore-2-3.* Introduzida por John Hopcroft em 1970, (não publicado, referido por [5, 11]) é uma árvore balanceada onde cada nó não folha possui no mínimo 2 e no máximo 3 filhos. Seu uso é adequado para dispositivos de memória primária.

Em todos os métodos cria-se uma fila de processos para cada nó que sofreu um pedido de bloqueio não realizado (por já está bloqueado). Os pedidos são posteriormente executados pela ordem de entrada na fila.

## 3.2 Histórico

A primeira solução para o problema de concorrência foi introduzida por Samadi em 1976 [24] e por Parr em 1977 [22] , na qual são válidas apenas as operações de bloquear e desbloquear um nó. Essas operações são executadas através da técnica padrão de *semáforos* (Dijkstra [7] ) fazendo uso de um único tipo de bloqueio. Na realidade, os processos leitores e atualizadores utilizam a técnica de lock-coupling (v. **D20**).

Em 1977, Bayer e Schkolnick [2] apresentaram 4 soluções para as árvores-$B^*$ (v. **D4**). As soluções fazem uso de 3 tipos diferentes de bloqueios: *leitor* ($\rho$-lock), *alternativo* ($\alpha$-lock) e *exclusivo* ($\varepsilon$-lock). Em todas as soluções os processos leitores executam $\rho$-locks nos nós percorridos, através da técnica de lock-coupling específica para este tipo de processo. Já os processos atualizadores são executados da seguinte forma.

**Solução 1.** Realiza bloqueios através da técnica de lock-coupling usando $\varepsilon$-locks nos nós visitados.

**Solução 2.** O sub-processo atualizador-leitor é executado da mesma forma que um processo leitor. Ao se atingir um nó folha $n$ realiza-se um $\varepsilon$-lock em $n$. Se $n$ não for um nó seguro, despreza-se toda análise feita, liberam-se os bloqueios dos nós percorridos e utiliza-se o método da solução 1.

**Solução 3.** O sub-processo atualizador-leitor é executado através da técnica de lock-coupling usando $\alpha$-locks nos nós visitados. Quando um nó folha é atingido realizam-se, de cima para baixo, conversões dos nós com $\alpha$-locks para $\varepsilon$-locks.

**Solução 4.** Esta solução consiste de uma combinação generalizada das outras 3 soluções. Dependendo de certos parâmetros, em determinados trechos da árvore será usada a solução 1, 2 ou 3.

Em 1978 Miller e Snyder [18] introduziram outra solução para as árvores-$B$, que se caracteriza por bloquear uma porção menor da árvore, quando comparada com as soluções anteriores de Bayer e Schkolnick.

Ellis propôs em 1980 [8] uma solução para as árvores-2-3 (v. **D24**). Esta solução baseia-se na solução de Bayer e Schkolnick e na idéia de Lamport [16] de permitir que processos leitores e atualizadores examinem o mesmo nó, simultaneamente, em direções opostas. Ou seja, os processos leitores examinam os valores em um nó da esquerda para direita e os atualizadores da direita para esquerda.

Lehman e Yao definiram em 1981 [17] um método de concorrência aplicado às árvores-$B^L$ (v. **D23**). Este método utiliza um único tipo de bloqueio e executa um número pequeno (constante) de bloqueios nos nós para cada processo em execução. Os processos leitores e os sub-processos atualizadores-leitores não executam nenhum tipo de bloqueio. No momento em que um nó folha $n$ for detetado ele é bloqueado. Quando houver necessidade de se percorrer o vizinho à direita de $n$ o mesmo é bloqueado, $n$ desbloqueado e o vizinho passa a ser o nó corrente. A solução apresentada não faz uso das técnicas de concatenação e redistribuição (v. **D13**) e permite a existência de nós subcarregados (v. **D8**).

Em 1986, Sagiv [23] aperfeçoou a solução acima garantindo o bloqueio em apenas um único nó por cada processo em execução. A solução faz uso de um "bloco especial" que armazena a altura da árvore e um vetor de ponteiros para cada nó mais à esquerda de cada nível.

Objetivando melhorar as soluções de Bayer e Schkolnick e de Ellis, Kwong e Wood sugeriram de 1980 a 1982 [13, 14, 15] uma nova solução para árvores-$B^*$ que faz uso das 3 técnicas de bloqueios (v. **D20, D21, D22**). O processo leitor é executado como nas soluções de Bayer e Schkolnick. Já o sub-processo atualizador-leitor é executado como na solução 3 introduzida por estes autores. No entanto, quando um nó folha precisa ser desdobrado (v. **D10**) (concatenado) utiliza-se a técnica de side-branching, de baixo para cima, até atingir o nó seguro mais próximo do caminho percorrido pelo processo. Neste instante, atualizam-se os nós, de cima para baixo, fazendo uso da técnica de driving-off.

Mond e Raz propuseram em 1985 [20] uma solução de concorrência para as árvores-$B^*$ baseando-se na modificação do algoritmo para processos atualizadores dada por Guibas e Sedgewick em 1978 [9]. Essa solução caracteriza-se por efetuar desdobramentos e concatenações nos nós percorridos por sub-processos atualizadores-leitores, deixando esses nós prontos para futuras inserções e remoções. A solução faz uso de 2 tipos de bloqueios que dependem do processo sendo executado e garantem que no máximo 2 nós são bloqueados simultaneamente por cada processo em execução. Os bloqueios são realizados em pares, no nó corrente e no seu respectivo pai. Antes de se bloquear o filho apropriado do nó corrente libera-se o bloqueio do nó pai.

Em 1988 Keller e Wiederhold [10] afirmaram que o método de Mond e Raz não se aplica para valores de índices com tamanho variável e introduziram a técnica de *promoção vizinha*. Esta técnica caracteriza-se por criar um nó vizinho a um nó completo, colocando metade das informações do nó completo no novo nó e uma "marca" no nó pai, indicando a existência do vizinho.

Finalmente, Souza e Carvalho propuseram em 1992 [25] um novo método de concorrência para árvores-$B^*$, no qual o caminho de acesso de cada processo em execução fica residente na memória. Ao encontrar um nó bloqueado, um processo passa a percorrer sucessivamente o caminho até poder continuar. Esse método permite também o uso da técnica de repartição quando um nó se encontra subcarregado.

# 4 O Método

O método de concorrência a ser apresentado aplica-se as árvores-$B^*$ e as árvores-$B^+$ (v. **D4, D5**), daqui por diante chamadas simplesmente de *árvores*.

## 4.1 Definições Específicas

**D25** O *fator de ocupação de um nível* de uma árvore é dado por $F = \frac{I}{2KN}$, onde $I$ é o número de valores de índices do nível, $N$ o número de nós do nível e $K$ a ordem dos nós deste nível. Note-se que nas árvores-$B^*$ e $B^+$ a ordem $K$ dos nós folhas é diferente da ordem dos nós internos, pois eles contêm tipos distintos de informações.

**D26** *Vetor de fator de ocupação* de uma árvore é um vetor que contém o fator de ocupação de cada nível dessa árvore.

**D27** Um *bloqueio-f* em um nó consiste de um bloqueio exclusivo realizado por um processo leitor ou atualizador. Um *boqueio-c* em um nó é um bloqueio realizado por um processo de compactação (v. **D16**).

**D28** *Indicador de compactação* é uma variável booleana cujo valor indica se uma árvore está (valor *verdadeiro*) ou não (valor *falso*) sendo compactada. Este indicador é consultado antes da execução de um processo leitor ou atualizador.

**D29** *Indicador de filho subcarregado* é uma variável booleana associada a cada valor $i_j$ ($1 \leq j \leq m$) de índice de um nó $n$ não folha. Esta variável indica se o filho de $n$ apontado por $p_j$ está subcarregado (valor *verdadeiro*) ou não (valor *falso*), evitando uma leitura desnecessária de algum nó filho. (Pode-se implementar essa variável como sendo o bit mais à esquerda de cada ponteiro, ou através de um *vetor de bits* guardado em $n$.)
**D30** *Fila de compactação* é uma estrutura que armazena pedidos de todos os processos que foram interrompidos em função da execução de um processo de compactação na árvore. Após o término do processo de compactação, os pedidos da fila são disparados automaticamente e os processos recomeçam a partir da raiz da árvore.

## 4.2 Descrição Geral

O método usa a técnica de manter os nós da árvore seguros tanto para inserção como para remoção baseando-se na idéia de Mond e Raz [20] , que aplicaram para as árvores-$B$ a técnica introduzida por Guibas e Sedgewick [9] . Estes introduziram desdobramentos prévios em estruturas balanceadas, como as árvores-2-3 e sua extensão, árvores-2-3-4.

Diferentemente da técnica de Mond e Raz o método aqui exposto executa além dos desdobramentos prévios, repartições prévias, e concatenações e redistribuições adiadas, com o intuito de evitar propagações na reestruturação da árvore; além disso, simplifica os procedimentos, pois executa um único tipo de bloqueio.

Nas operações de leitura, inserção e remoção realizam-se bloqueios do tipo *bloqueio-f* (v. **D27**) no nó corrente $n$ (sendo visitado) e em seu respectivo pai. Antes de se visitar um filho apropriado $n'$ de $n$, libera-se o bloqueio do pai de $n$. Com isso, garante-se que para cada processo em execução apenas 2 nós ($n$ e seu pai) estão explicitamente bloqueados a cada instante. Note-se que seus descendentes estão implicitamente bloqueados para processos que ainda não atingiram o pai de $n$. No entanto, alguns desses descendentes podem conter bloqueios-$f$ devido a outros processos que já passaram por $n'$. Seja $P$ o processo em execução e $n$ seu nó corrente não folha.

Quando $P$ for um processo leitor ou sub-processo atualizador-leitor e atingir $n$ não-bloqueado e completo (v. **D6**), são executadas concatenações em nós filhos de $n$ adjacentes subcarregados, ou uma repartição (v. **D12**) entre $n$ e alguns dos seus irmãos, ou um desdobramento de $n$, nesta ordem, afim de permitir que $n$ passe a conter espaço livre para futuras inserções e possa provocar desbloqueio de seu pai. Como veremos, a escolha de uma repartição ou desdobramento depende do fator de ocupação do nível (v. **D25**) do pai de $n$, guardado no vetor de fatores de ocupação (v. **D26**).

O procedimento de concatenação é executado quando se verifica a existência de 2 filhos subcarregados de $n$, a uma "distância" de no máximo, por exemplo, 3 ponteiros adjacentes de $n$. Essa informação é facilmente obtida através dos indicadores de filho subcarregado (v. **D29**). No caso em que existem nós filhos subcarregados e estes não estão bloqueados por outro processo, concatenam-se esses filhos - através de redistribuições, quando esses não forem adjacentes - eliminando algum separador de $n$. Entretanto, se os nós filhos subcarregados estiverem bloqueados por outro processo, $P$ fica em estado de espera até que esses bloqueios sejam liberados. Se não for possível executar a concatenação, então, dependendo do caso, sugerimos a aplicação da repartição ou de desdobramento.

Note-se que a subcarga de 3 ou mais nós consecutivos filhos de $n$ poderia ser tal que esses nós pudessem ser concatenados em um único nó. Devido às possibilidades de inserções seguidas à concatenação e para diminuir a leitura de nós, optamos por unir apenas 2 nós. Algum processo poderá posteriormente efetuar a concatenação com o terceiro nó.

O procedimento de repartição é realizado quando $n$ está completo e o fator de ocupação do nível de seu pai for maior do que um valor $f_d$ pré-estabelecido, que denominaremos de *fator limite de desdobramento* (por exemplo, $f_d = 85\%$). Esse fator assinala a situação em que é preferível repartir ao invés de desdobrar, pois esta última ação acarreta a inserção de um novo separador no pai de $n$. O sistema tenta conseguir espaço livre em $n$ através da seguinte técnica. Se $n$ tem pelo menos 2 irmãos à esquerda e/ou à direita, o procedimento tentará fazer repartição entre os elementos de $n$ e irmãos, examinando primeiro os irmãos da esquerda e depois os da direita; ou seja, a repartição é executada examinando-se no máximo 4 irmãos de $n$ ou um outro número pré-estabelecido de nós irmãos consecutivos de $n$. Se algum desses irmãos estiver bloqueado por um outro processo, $P$ fica esperando até que ele seja liberado. No caso em que $n$ não possui 2 irmãos para algum dos lados, então examina-se para este lado a quantidade existente. Quando não for possível empregar essa técnica para algum desses irmãos de $n$, faz-se um desdobramento em $n$.

No caso em que $n$ está completo e o fator de ocupação do nível do pai de $n$ for menor ou igual a $f_d$, ou for maior do que $f_d$, mas não foi possível realizar repartição entre seus irmãos, então faz-se um desdobramento de 2 para 3 (v. **D11**) em torno de $n$. Se um dos irmãos encontra-se bloqueado por outro processo, então $P$ aguarda até que ele seja liberado.

Note-se que o desdobramento não acarreta uma propagação de novos desdobramentos, no sentido de baixo para cima, pois por construção o nó pai de $n$ está bloqueado e contém espaço suficiente para inserção de um novo elemento. Esse é um ponto essencial do método, pois com isso evita-se a necessidade de cada processo bloquear um grande número de nós.

Antes de se fazer concatenações nos filhos de $n$, não há necessidade de se bloquear esses filhos, pois neste instante nenhum outro processo que ainda não passou por $n$ poderá bloqueá-los, uma vez que $n$ encontra-se bloqueado. O mesmo se aplica para os casos de repartições e desdobramentos de 2 para 3, no que diz respeito aos irmãos e filhos de $n$, pois o pai de $n$ também encontra-se bloqueado.

Contrariamente às soluções apresentadas na seção 3, o nosso método não faz distinção entre os processos leitores e atualizadores, isto é, estes processos se comportam da mesma maneira.

Note-se que nesse método podem ocorrer nós subcarregados, que assim permanecem até que sejam realizadas concatenações (com eventuais redistribuições), quando um pai tornar-se completo. Considerando que na prática o número de inserções tende a ser assintoticamente maior ou igual do que o de remoções (pois, caso contrário, o arquivo poderia vir a desaparecer), é conveniente adiar aquelas operações. (Lehman e Yao [17] também apontam para o fato do número de inserções superar o de remoções.)

Quando a quantidade de nós subcarregados for elevada o fator de ocupação tende a diminuir. Para contornar essa situação deve-se disparar automática ou manualmente um processo de compactação, o qual pode ser executado concorrentemente com os demais processos de leitura e de atualização. Note-se que isso não se encontra na literatura.

O processo de compactação ativa, no início de sua execução, o indicador de compactação (v. **D28**), o qual é desativado apenas no final de sua execução. Ele utiliza um bloqueio especial do tipo *bloqueio-c* (v. **D27**) realizado apenas nos nós folhas. Um nó $n$ com bloqueio-$c$ pode ser lido por outros processos. Os nós folhas à direita de um nó com bloqueio-$c$ podem ser lidos e atualizados por outros processos.

Baseando-se no algoritmo de compactação de Miller et al. [19], o processo de compactação constrói a árvore compactada em uma nova área do disco, isto é, a árvore não é montada em cima da original. Após a construção, a nova árvore é copiada para a área que contém a antiga. No final, libera-se o espaço onde a árvore foi construída. Miller utilizou uma compactação

baseada em busca em profundidade. Nosso método parte das folhas construindo um nível de cada vez, o que favorece a contigüidade física da árvore a ser construída e a concorrência.

O processo consiste em se visitar o nó folha $n_1$ mais à esquerda, fazendo um bloqueio-c neste nó. Em seguida percorre-se $n_1$ colocando-se seus elementos em uma nova página $P_{G_1}$ do disco. Através do ponteiro para direita (v. **D4**, **D5**) atinge-se o vizinho imediato $n_2$ de $n_1$, executa-se um bloqueio-c em $n_2$ e inserem-se elementos de $n_2$ em $P_{G_1}$, quando couber, mantendo-se um fator de ocupação pré-estabelecido para os nós ou, se não couber, em uma nova página $P_{G_2}$, fazendo $P_{G_1}$ apontar para $P_{G_2}$. Esse processo continua até que seja atingido o último nó folha, produzindo a última página $P_{G_m}$. As páginas $P_{G_1}, \cdots, P_{G_m}$ passam a ser os nós folhas da nova árvore sendo construída. A partir desse momento o procedimento constrói os nós do nível dos pais de $P_{G_1}, \cdots, P_{G_m}$, percorrendo cada uma destas páginas e colocando o último elemento de $P_{G_j}$ como separador de $P_{G_j}$ e $P_{G_{j+1}}$ $(1 \leq j \leq m-1)$. O processo continua da mesma forma para os demais níveis até que se construa a raiz da nova árvore. Após a montagem dessa raiz a nova árvore é copiada para a área da antiga e todos os processos passam a utilizá-la. Observemos que os nós folhas ficam fisicamente à esquerda dos seus pais, e assim sucessivamente, permitindo uma certa contigüidade física para aumentar a eficiência da busca. Notemos que durante a construção dos nós internos da nova árvore por um processo de compactação só são executados, por outros processos, procedimentos de leitura na árvore antiga, pois nesta situação todos os nós folhas desta contêm bloqueios-c.

Quando um processo atualizador detetar um nó folha contendo um bloqueio-c, ele deve ser interrompido e colocado na fila de compactação (v. **D30**). No momento em que o processo de compactação terminar todos os pedidos da fila são disparados automaticamente e os processos recomeçados a partir da raiz da nova árvore, tentando efetuar a operação desejada.

A existência de um processo de compacatação deve provocar alterações nos processos leitores e atualizadores. Ambos devem verificar inicialmente o indicador de compactação. Se este estiver ativado os processos não devem executar concatenações, redistribuições, desdobramentos e repartições, uma vez que a árvore será compactada.

No caso de inserções estas só devem ser realizadas nos nós folhas que não estão completos ou naquelas folhas que contêm nós pais não completos. Qualquer outro tipo de modificação na árvore devido a inserção, acarreta a interrupção do processo atualizador que deverá ser colocado na fila de compactação.

## 4.3 Processos e Algoritmos

Os diferentes processos descritos a seguir utilizam algoritmos que são apresentados em forma de tabelas de decisão (Nagayma [21]), onde as condições e ações são expressas em "macros" definidas depois das tabelas. As macros usadas em um algoritmo e já referidas em algoritmo anterior não são repetidas.

### 4.3.1 Processo leitor ou sub-processo atualizador-leitor

Chama sucessivamente o algoritmo de busca para cada nó $n$, da raiz até uma folha.

### 4.3.2 Processo atualizador

Chama o algoritmo de busca até determinar o nó folha $n$ apropriado. Retorna $n$ juntamente com a informação de que *achou* ou *não achou* o valor do índice sendo procurado, bem como

a posição correta dentro de $n$ de inserção ou remoção. Em seguida, executa o algoritmo de inserção ou de remoção, dependendo do tipo de processo.

### 4.3.3 Processo de Compactação

Chama o algoritmo de compactação e percorre os nós folhas da árvore a ser compactada, a partir do nó mais à esquerda. Isto é feito com auxílio dos ponteiros da direita que referenciam os nós vizinhos (lista encadeada).

### 4.3.4 Algoritmo de Busca

| indicador-compactação? | N | N | N | N | N | N | N | N | N | N | S | S | S |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| completo? | N | S | S | S | S | S | S | S | - | - | - | - | - |
| há-filhos-subcarregados? | - | S | S | S | S | N | N | N | - | - | - | - | - |
| pode-concatenar-filhos? | - | S | N | N | N | - | - | - | - | - | - | - | - |
| ocup-nível-pai-maior? | - | - | S | S | N | S | S | N | - | - | - | - | - |
| pode-repartir? | - | - | S | N | - | S | N | - | - | - | - | - | - |
| folha? | N | N | N | N | N | N | N | N | S | S | N | S | S |
| achou? | - | - | - | - | - | - | - | - | S | N | - | S | N |
| concatene-filhos | | 1 | | | | | | | | | | | |
| reparta | | | 1 | | | 1 | | | | | | | |
| desdobre | | | | 1 | 1 | | 1 | 1 | | | | | |
| segue | 1 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | | | 1 | | |
| retorne-achou | | | | | | | | | 1 | | | 1 | |
| retorne-não-achou | | | | | | | | | | 1 | | | 1 |

**Condições:**
**indicador-compactação:** o indicador de compactação contém valor *verdadeiro*;
**completo:** o nó $n$ é completo;
**há-filhos-subcarregados:** o nó $n$ contém filhos $n'$ e $n''$ subcarregados com distância entre $n'$ e $n''$ de no máximo 2 ponteiros;
**pode-concatenar-filhos:** é possível concatenar 2 nós desde $n'$ até $n''$ com eventual redistribuição dos nós entre eles;
**ocup-nível-pai-maior:** o fator de ocupação do nível do pai de $n$ é maior do que $f_d$;
**pode-repartir:** é possível repartir $n$ entre até 2 irmãos adjacentes de cada lado;
**folha:** $n$ é folha;
**achou:** se achou o valor do índice, este foi encontrado no nó $n$ na posição $j$; se não achou, o valor do índice deverá ser inserido na posição $(j, a)$ ou $(j, d)$, onde $a$ indica que se trata da posição imediatamente anterior a $i_j$ e $d$ da imediatamente depois de $i_j$ $(1 \leq j \leq m)$.

**Ações:**
**concatene-filhos:** concatene os filhos subcarregados de $n$ com eventual redistribuição;
**reparta:** reparta os elementos de $n$ entre até 2 irmãos adjacentes de cada lado;
**desdobre:** desdobre $n$ de 2 para 3;
**segue:** desbloqueie o pai de $n$, determine o filho apropriado de $n$, bloqueie este filho e chame-o de $n$;
**retorne-achou:** indica que *achou* o valor do índice sendo procurado e retorna a posição $j$;

retorne-não-achou: indica que *não achou* o valor do índice sendo procurado e retorna $(j, a)$ ou $(j, d)$.

#### 4.3.5 Algoritmo de Inserção

| indicador-compactação? | - | N | N | N | N | S | S | S | - |
|---|---|---|---|---|---|---|---|---|---|
| achou? | S | N | N | N | N | N | N | N | - |
| seguro-inserção? | - | S | N | N | N | S | N | N | - |
| pai-seguro-inserção? | - | - | - | - | - | - | S | N | - |
| ocup-nível-pai-maior? | - | - | S | S | N | - | - | - | - |
| pode-repartir-$n$? | - | - | S | N | - | - | - | - | - |
| há-bloqueio-$c$? | - | - | - | - | - | N | N | N | S |
| insira-valor | | 1 | 1 | 1 | 1 | 1 | 1 | | |
| reparta | | | 2 | | | | | | |
| desdobre | | | | 2 | 2 | | 2 | | |
| mensagem-1 | 1 | | | | | | | | |
| ins-fila-compactação | | | | | | | | 1 | 1 |
| desbloqueios | | 2 | 3 | 3 | 3 | 2 | 3 | 2 | |

**Condições:**
seguro-inserção: o nó folha $n$ está seguro para inserção, isto é, $n$ contém menos do que $2K$ valores;
pai-seguro-inserção: o pai do nó folha $n$ está seguro para inserção;
há-bloqueio-$c$: o nó $n$ contém um bloqueio-$c$.

**Ações:**
insira-valor: insere o valor do índice no nó folha $n$ na posição $(j, a)$ ou $(j, d)$;
mensagem-1: emite a mensagem: *o valor do índice já existe na árvore*;
ins-fila-compactação: insere o processo na fila de compactação;
desbloqueios: desbloqueia o pai de $n$ e o nó $n$, nesta ordem.

#### 4.3.6 Algoritmo de Remoção

| indicador-compactação? | - | N | N | S |
|---|---|---|---|---|
| achou? | N | S | S | - |
| seguro-remoção? | - | S | N | - |
| há-bloqueio-$c$? | - | N | N | S |
| remove-valor | | 1 | 1 | |
| indicador-subcarregado | | | 2 | |
| mensagem-2 | 1 | | | |
| ins-fila-compactação | | | | 1 |
| desbloqueios | | 2 | 3 | |

**Condições:**
seguro-remoção: o nó folha $n$ está seguro para remoção, isto é, contém mais do que $K$ valores.

**Ações:**

remove-valor: retira o valor do índice do nó folha $n$ da posição $j$;
indicador-subcarregado: atualiza o indicador de filho subcarregado, indicando que o nó folha $n$ tornou-se subcarregado;
mensagem-2: emite a mensagem: *o valor do índice não existe na árvore.*

### 4.3.7 Algoritmo de Compactação

| pont-dir-nulo? | N | S |
|---|---|---|
| bloqueia-$c$ | 1 | |
| constrói-folha-nova | 2 | |
| atualiza-pont-dir | 3 | |
| volta-tabela | 4 | |
| constrói-nós-internos | | 1 |
| copia-nova-árvore | | 2 |
| atualiza-ind-compactação | | 3 |
| inicia-fila-compactação | | 4 |

**Condições:**
pont-dir-nulo : o ponteiro para o nó vizinho à direita de $n$ é nulo.

**Ações:**
bloqueia-$c$: executa um bloqueio-$c$ no nó folha $n$;
constrói-folha-nova: coloca as informações de $n$ em uma página da árvore sendo construída;
atualiza-pont-dir: verifica o ponteiro da direita pertencente a $n$;
volta-tabela: recomeçã tabela de decisão;
constrói-nós-internos: constrói os nós internos da nova árvore;
copia-nova-árvore: copia a nova árvore compactada em cima da antiga;
atualiza-ind-compactação: atualiza o indicador de compactação com o valor *falso*;
inicia-fila-compactação: inicializa a execução dos processos armazenados na fila de compactação.

## 4.4 Deadlock e Consistência

Em virtude dos tipos de bloqueios e da forma com que os mesmos são realizados, isto é, sempre aos pares e seguindo uma determinada ordenação correspondente à estrutura da árvore (de cima para baixo), podemos garantir que o método não ocasiona deadlock.

Considere a situação da figura 1 que ilustra um trecho de uma árvore com o processo $P_1$ em execução, onde $n$ é o nó corrente, $n_p$ o seu pai e $f(P_1)$ o bloqueio-$f$ realizado por $P_1$. Se existir um processo $P_2$ bloqueando o nó $n'$, onde $n'$ é um filho de $n$, então $n$ não é o nó corrente em $P_2$ pois, no caso contrário, $n$ estaria bloqueado por $P_2$. Seja $n''$, filho de $n'$, o nó corrente em $P_2$. Afirmamos que as alterações executadas por $P_2$ em $n''$ e consequentemente em $n'$, não se propagam para $n$, pois por construção $n'$ contém espaço livre para futuras inserções ($P_2$ já visitou $n'$), evitando a situação de deadlock.

Por outro lado, na figura 2 ilustramos o caso de uma tentativa de redistribuição ou desdobramento pelo processo $P_1$ onde um dos irmãos de $n$, por exemplo $n_1$, encontra-se bloqueado por um processo $P_2$. Garantimos que o nó corrente em $P_2$ é um filho $n'_1$ de $n_1$ pois, no caso contrário, o pai $n_p$ de $n$ não estaria bloqueado por $P_1$. Nesse caso, $P_1$ deve esperar até que

$P_2$ libere o bloqueio de $n_1$. Notemos que também não há problema de deadlock, uma vez que $n_1$ contém espaço livre para inserção de um novo elemento ($P_2$ já visitou $n_1$) e assim não há propagação de inserção até $n_p$ (que está bloqueado por $P_1$).

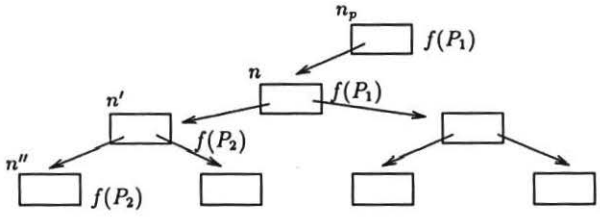


Figura 1: Exemplo dos processos $P_1$ e $P_2$ em execução.

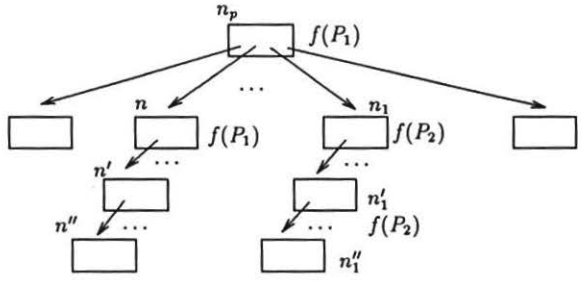


Figura 2: Exemplo dos processos $P_1$ e $P_2$ em execução.

Notemos que a tentativa de redistribuição e de desdobramento foi reduzida ao exame dos dois irmãos imediatamente precedentes ou subsequentes a $n$ com o intuito de evitar uma eventual possibilidade de deadlock, fato que aconteceria se fossem usados nós "primos" de $n$.

Por sua vez, o processo de compactação executado simultaneamente com os processos leitores e atualizadores também não acarreta deadlock. Isso é garantido em função da utilização do bloqueio-$c$ e da construção da árvore compactada em uma nova área.

## 4.5 Extensão para Valores com Tamanho Variável

Para diminuir a altura das árvores utilizam-se, dentro dos nós, valores de índices com tamanho variável através do uso de técnicas de compressão frontal e posterior (Wagner [26]) e árvores-$B$ Pré-fixada (Bayer e Unterauer [3]).

Keller e Wiederhold [10] afirmaram que o método de Mond e Raz [20] não pode ser utilizado com valores de índices com tamanho variável. Nesse caso, o espaço livre existente em cada nó da árvore pode não ser suficiente para comportar um novo valor a ser inserido. Ou seja, há uma dificuldade em caracterizar se um nó está ou não completo.

Nosso método pode ser extendido para comportar esse tipo de valores. Para isso propomos que o usuário especifique o tamanho máximo ($T_m$) em bytes (ou palavras) dos valores do índice. Vamos supor que $T_m$ seja menor do que a metade do tamanho de um nó. É necessário introduzir modificações no método apresentado.

Os limites $K$ e $2K$ do número de valores para ocupação de um nó não são mais caracterizados, devendo ser substituídos pela ocupação da metade do tamanho de um nó, até o seu tamanho total. Com isso é necessário redefinir a noção de nó completo.

**D31** Em uma árvore com valores de tamanho variável e tamanho máximo de valor $T_m$, um nó é *completo* quando seu espaço livre é menor do que $T_m$.
**D32** Em uma árvore com valores de tamanho variável um nó é *subcarregado* quando seu espaço livre é maior do que a metade do seu tamanho.

As operações de concatenação, repartição e desdobramento devem ser realizadas de forma a manter nós não completos.

No presente caso, quando um nó $n$ está completo e contém filhos subcarregados, com distância entre estes de no máximo, por exemplo, de 3 ponteiros, deve-se iniciar pela tentativa se de efetuar uma concatenação. Na impossibilidade desta, ou se após a mesma o nó continuar completo, tenta-se executar uma redistribuição entre os filhos subcarregados e eventuais irmãos que estejam entre eles. Com isso pode-se tentar mudar os atuais separadores para outros de menor tamanho, tornando $n$ não completo.

# 5 Comparações, Conclusões e Pesquisas Futuras

As soluções de Bayer e Schkolnick [2], Miller e Snyder [18] e Kwong e Wood [13, 14, 15], caracterizam-se por utilizar vários tipos de bloqueios com conversões entre estes. O método aqui apresentado utiliza apenas um único tipo de bloqueio para realizar as operações de leitura, inserção, remoção e atualização. Introduz, também, um outro tipo de bloqueio para executar um processo de compactação; o problema de concorrência com compactação ainda não havia sido abordado na literatura. Contrariamente às soluções de Bayer e Schkolnick, Lehman e Yao [17], Sagiv [23] e Kwong e Wood [13, 14, 15], Souza e Carvalho [25], no nosso método cada processo em execução percorre a árvore somente uma única vez, no sentido de cima para baixo, exceto no caso de compactação.

A solução proposta extende o método de Mond e Raz [20] permitindo concatenações, desdobramentos e, item não abordado por eles, redistribuições, sem provocar propagações na árvore. Um item não permitido na solução deles, presente na nossa, consiste do uso de valores de índice com tamanho variável. O método garante um bom fator de ocupação da árvore, utilizando um novo parâmetro, que consiste do uso do fator de ocupação de cada nível da árvore, e complementado ainda com o uso concorrente de técnica de compactação com algoritmo original. Esta técnica garante uma contigüidade física dos nós por nível e de níveis consecutivos.

O método de concorrência sem compactação pode ser também aplicado às árvores-$B$.

Este trabalho abre os seguintes tópicos para pesquisas futuras: 1) simulações do método para

determinar fatores limites de desdobramento ideal, de ocupação dos nós após uma compactação e para disparo automático da compactação; 2) extensão do método para evitar desdobramentos prévios e concatenações adiadas nos nós localizados próximos à raiz, uma vez que o número destas operações nestes nós superiores é muito baixa em relação aos nós próximos às folhas; 3) simulações para comparação com outros métodos.

## Referências

[1] R. Bayer and E. McCreight. Organization and maintenance of large ordered indexes. *Acta Informatica*, 1:173–189, 1972.
Primeiro artigo a abordar a estrutura de *árvore-B*, definindo-a, propondo e analisando os algoritmos de busca, inserção e remoção para um valor de índice e mostrando alguns resultados experimentais.

[2] R. Bayer and M. Schkolnick. Concurrency of operations on B-trees. *Acta Informatica*, 9(1):1–21, 1977.
Sugere 4 soluções para o problema de concorrência nas *árvores-B**, as quais fazem uso de 3 tipos de bloqueios.

[3] R. Bayer and K. Unterauer. Prefix B-tree. *ACM Transactions on Database Systems*, 2(1):12–26, March 1977.
Apresenta o conceito de *árvore-B* com pré-fixação simples e não simples, juntamente com os algoritmos para estas estruturas.

[4] W. Boswell and A.L. Tharp. *Advances in Computing and Information*, volume 468 of *Lecture Notes in Computer Science*, chapter Alternatives to The B$^+$-Tree, pages 266–274. Springer, May 1990.
Exemplifica 2 estruturas de arquivos para aplicações que requerem acesso sequencial e direto: *Bounded Disorder* e *Adaptive Hashing*. Faz uma análise comparativa entre estas estruturas e a *árvore-B*$^+$.

[5] D. Comer. The Ubiquitous B-tree. *Computing Surveys*, 11(2):121–137, 1979.
Este artigo é uma resenha (survey) abordando de forma sucinta muitos dos tópicos relacionados com o tema *árvore-B*.

[6] T.H. Cormen, C.E. Leiserson, and R.L. Reivest. *Introduction to Algorithms*. The MIT Press and McGraw Hill Book Company, New York, 1990.
Consultados em aspectos gerais sobre tipos de árvores de busca.

[7] E.W. Dijkstra. The structure of the multi-programming system. *Communications of the ACM*, 11(5):341–346, May 1968.
Introduz a noção de *semáforos* para manipular os problemas de concorrência.

[8] C.S. Ellis. Concurrent search and insertion in 2-3-trees. *Acta Informatica*, 14(1):63–86, 1980.
Apresenta 2 algoritmos que permitem um alto grau de concorrência nas *árvores-2-3* sem ocasionar deadlock (Bloqueio-2-3 e Pipeline-2-3).

[9] L.J. Guibas and R. Sedgewick. A dichromatic framework for balanced trees. In *Proc. 19th Annual Symposium Foundation of Computer Science*, pages 8–21, 1978.
Aborda uma técnica de coloração para binarizar árvores e torná-las sempre balanceadas. Propõe também um novo algoritmo de atualização para *árvores-B*, o qual executa pré-desdobramentos nos nós durante a fase de busca.

[10] M.A. Keller and G. Wiederhold. Concurrent use of B-trees with variable-length entries. *Sigmod Records*, 17(2):89–90, June 1988.
Propõe uma solução para o problema de concorrência em *árvores-B* com valores de tamanho variável.

[11] D.E. Knuth. *The Art Computer Programming: Sorting and Searching*, volume 3. Addison-Wesley Publishing Company, Reading, Massachusetts, 1973.
Aborda as *árvores-B* e introduz a $B^*$, bem como os *desdobramentos de 2 para 3*.

[12] H.F. Korth and A. Silberchatz. *Database System Concepts*. Mc. Graw Hill Inc., New York, 1986.
Consultado em aspectos de organização, implementação, projeto e seleção de arquivos utilizados em banco de dados, em particular para os arquivos indexados.

[13] Y.S. Kwong and D. Wood. *In Proc. MFCS*, volume 88 of *Lecture Notes in Computer Science*, chapter Approaches to Concurrency in B-Trees, pages 402–413. Springer, 1980.
Expõe os problemas de concorrência com *árvores-B* e analizam algumas soluções existentes.

[14] Y.S. Kwong and D. Wood. *In Proc. 4th International Symposium Programming*, volume 83 of *Lecture Notes in Computer Science*, chapter Concurrent Operations in Large Ordered Indexes, pages 208–221. Springer, 1980.
Apresenta soluções para o problema de concorrência nas *árvores-B** e propõem uma nova estrutura denominada de *árvore-T* (uma árvore de árvores).

[15] Y.S Kwong and D. Wood. A new method of concurrency in B-trees. *IEEE Transactions on Software Engineering*, SE-8(3):211–222, May 1982.
Propõe uma solução para o problema de concorrência em *árvores-B**, fazem uma comparação das várias soluções existentes e tratam o caso de concorrência dentro de um nó.

[16] L. Lamport. Concurrent reading and writing. *Communications of the ACM*, 20(11):806–811, November 1977.
Exibe vários algoritmos para o problema de concorrência. Todos estes algoritmos permitem o compartilhamento dos dados, pois fazem uso da técnica de ler os dados em um sentido e escrevê-los em outro.

[17] P.L. Lehman and S.B. Yao. Efficient locking for concurrent operations on B-tree. *ACM Transactions on Database Systems*, 6(4):650–670, December 1981.
Propõe uma solução de concorrência para as *árvores-B** fazendo uma ligeira modificação na estrutura e garantindo o bloqueio de um único nó em cada instante de um processo em execução.

[18] E.R. Miller and L. Snyder. Multiple access to B-trees. In *Proc. Conference Information Sciences and Systems*, pages 400–407, John Hopkins University - Baltimore, March 1978.
Trata do problema de concorrência em *árvores-B* sugerindo uma solução na qual os nós bloqueados estão a uma distância pequena daquele que sofrerá modificação.

[19] E.R. Miller, N. Pippenger, A.L. Rosemberg, and L. Snyder. Optimal 2-3 trees. *Siam Journal of Computing*, 8(1):42–59, February 1979.
Propõe um algoritmo de tempo linear para construir *árvores-2-3* de espaço ótimo, isto é, baixas, a partir de um conjunto de itens e da determinação de um perfil para este conjunto. O algoritmo é facilmente adaptável para as *árvores-B*.

[20] Y. Mond and Y. Raz. Concurrency control in $B^+$-trees databases using preparatory operations. In *Proc. of the* $11^{th}$ *International Conference on Very Large Database*, pages 331–334, Stockholm, 1985.
Propõe um controle de concorrência para as *árvores-B** evitando um número elevado de nós bloqueados por cada processo em execução, através do uso de operações preparatórias que garantem a segurança dos nós para inserções e remoções.

[21] S. Nagayama. *Tabelas de Decisão e Implementação de Gerador I-M-E.* Dissertação de Mestrado, Instituto de Matemática e Estatística - USP, São Paulo, 1990.
Contém extensa resenha sobre tabelas de decisão, bem como a implementação do gerador de aplicações I-M-E.

[22] J.R. Parr. An access method for concurrently sharing a B-tree index. Technical Report 36, University of Western Ontario, Departament of Computer Science, April 1977.
Artigo não localizado (citado em [23]).

[23] Y. Sagiv. Concurrent operations on $B^*$-trees with overtacking. *Journal of Computer and Systems Science*, 33:275–296, 1986.
Aperfeiçoou a solução proposta por Lehman e Yao [17] utilizando uma técnica de redistribuição denominada de *compressão* que pode ser utilizada concorrentemente com outras operações.

[24] B. Samadi. B-trees in a system with multiple users. *Inf. Processing Letters*, 5(4):107–112, October 1976.
Sugere uma solução para o problema de concorrência nas *árvores-B* fazendo uso da técnica padrão de *semáforos*.

[25] R.M.F. Souza and O.S.F. Carvalho. Controle de concorrência em *árvores-b*. *IV- Simpósio Brasileiro de Arquitetura de Computadores - Processamento de Alto Desempenho (IV S-BAC PAD)*, pages 397–412, 1992.
Apresentam algoritmos de busca e inserção para o problema de concorrência nas *árvores-B* com elementos de tamanho variável. Estes algoritmos permitem o uso da técnica de overflow.

[26] R.E. Wagner. Indexing design considerations. *IBM System Journal*, (4):351–367, 1973.
Define os tipos de índices, as operações de inserção, remoção e atualização em arquivos de índices e propõe técnicas de compressão de caracteres para os valores dos índices e de enderaçamento relativo.

[27] A. Zisman. *A árvore-B e Uma Proposta de Implementação.* Dissertação de Mestrado, Instituto de Matemática e Estatística - USP, São Paulo, 1993.
Contém extensa resenha sobre *árvores-B* e uma proposta de implementação, com um novo algoritmo de concorrência e de compactação.