ANE – Árvore N-ária de Espalhamento Naturalmente Balanceada

ALEXANDRE GONÇALVES SILVA
ADRIANO FIORESE
ROGÉRIO EDUARDO DA SILVA
GILMÁRIO BARBOSA DOS SANTOS

UDESC – Universidade do Estado de Santa Catarina
DCC – Departamento de Ciência da Computação
Cx Postal 631 – CEP 89223-100 Joinville (SC) – Brazil
(alexandre,fiorese,rsilva,dcc2gbs)@joinville.udesc.br

Resumo. Este trabalho propõe a construção de uma árvore n-ária com critério de busca baseado em uma função de espalhamento adaptativa por nível. Em outras palavras, uma nova estrutura de dados em forma de *hashing* hierárquico, com operações de inserção e remoção, é desenvolvida, pretendendo ter implementação simples e busca eficiente de informação identificada por uma chave primária. Comparações de desempenho entre a estrutura proposta (ANE) e árvores binárias de busca (ABB e AVL), árvores n-árias de busca (ANB e B), tabela de espalhamento usando árvore para colisões (EA), são implementadas. A validação experimental do balanceamento natural da ANE, em relação à altura, é apresentada.

Palavras-Chave: espalhamento hierárquico, estruturas de dados, busca eficiente.

ANE – Naturally Balanced Hashing N-ary Tree

Abstract. This work proposes a construction of a n-ary tree with criterion of searching based on an adaptative hashing function by level. In other words, a new data structure in hierarchical hashing way is developed, aiming to have simple implementation and efficient search of information identified for a primary key. Comparisons of performance between the proposed structure (ANE) and binary search tree (ABB and AVL), n-ary search tree (ANB e B), hash table using tree for collisions (EA), are implemented. The experimental validation of the natural balancing of ANE, in relation to height, is presented.

Keywords: hierarchical hashing, data structure, efficienty search.

(Received July 31, 2006 / Accepted January 03, 2007)

1 Introdução

Uma árvore consiste basicamente em uma estrutura de dados hierárquica, na qual pode-se visitar um ou mais elementos filhos a partir de cada elemento pai. A vantagem para uma estrutura linear advém da possibilidade de existirem várias alternativas de caminhos a seguir adiante. Caso se queira encontrar uma determinada informação, haverá apenas um caminho válido até a mesma, ou seja, a partir de um dado elemento visitado, não contendo este a informação desejada, apenas um de seus filhos é útil na busca. Os demais filhos são descartados e, portanto, é possível chegar até a informação mais rapidamente com um número menor de comparações.

Uma tabela de espalhamento procura reduzir ao máximo este número de comparações. A idéia básica é criar um vetor de ponteiros suficientemente grande e, para cada informação de interesse, mapear o seu conteúdo (identificado por uma chave) em uma destas posições contíguas em memória. Para isto, é necessária a definição de uma função de espalhamento para o mapeamento único da informação em sua posição no vetor ou tabela. Uma colisão ocorre quando duas informações distintas são mapeadas na mesma posição. Idealmente não há colisão se a tabela tiver um tamanho adequado e uma função de espalhamento perfeita [21][16]. Neste caso, nenhuma comparação é necessária para se

detectar uma informação. Outras funções de espalhamento são utilizadas: lineares [9], baseadas no número primo [12], baseadas em operações bit-a-bit [24], entre outras.

Considerando o compromisso entre a quantidade de memória, projeto de uma função de espalhamento ideal e desempenho de busca das informações, é interessante o desenvolvimento de um estrutura híbrida contendo as principais características de árvores e tabelas de espalhamento. Este trabalho propõe uma nova estrutura de dados simples e eficiente derivada da união das duas primeiras.

Algumas aplicações em que é necessária a eficiência de busca e inserção de informações podem ser citadas: redução do custo de transmissão em bases de dados distribuídas [4]; acesso eficiente de estruturas de proteínas em uma base de dados [2]; classificação de endereços IP para filtragem de pacotes no *firewall*, política de roteamento, qualidade do serviço, entre outros serviços da internet [20]; indexação de elementos de multimídia como, por exemplo, base de dados de imagens [14].

A estrutura híbrida pode ser representada inicialmente por uma árvore binária de busca balanceada contendo uma tabela com n_e informações em cada elemento [5]. Ou através de uma melhor organização dos ponteiros entre estes elementos resultando em uma estrutura de acesso à memória residente em tempo real [17]. Estes dois métodos têm algoritmos de percurso que retornam as informações ordenadas por chave, são lineares no pior caso (O(n)), e aproximam-se ao desempenho de uma árvore binária (O($\log n_e$), se $n_e > n$, ou O($\log n$), caso contrário) na prática.

2 Definições Preliminares

Nesta seção, são apresentadas algumas definições úteis à leitura do restante do texto.

2.1 Nomenclatura

Uma árvore pode ser definida como um conjunto finito de elementos (nós) vazio ou particionável em $N+1$ subconjuntos disjuntos onde um deles é unitário e compõe-se apenas de um nó referência denominado *raiz* e N outros subconjuntos conhecidos como *sub-árvores*. Portanto, uma árvore é composta de árvores e pode ser tratada recursivamente. Um *nó* é uma estrutura unitária que armazena, de alguma forma, uma ou mais informações e, além disto, disponibiliza mecanismos de acesso (podem ser ponteiros) a outros nós hierarquicamente inferiores ou *descendentes*. Tais nós ligados a ele são chamados de *filhos*. A *ordem* de uma árvore corresponde à limitação do número máximo de filhos que cada nó pode ter. Em particular, se é estabelecido o máximo de dois filhos para cada nó, então trata-se de uma *árvore binária*, ou seja, uma árvore de ordem 2. Uma árvore com até N filhos por nó é dita *n-ária* (ou multi-via ou ainda multi-direcional). Se o número total de filhos de um nó qualquer for igual a ordem da árvore, então refere-se a um *nó completo*. Um nó com pelo menos uma sub-árvore vazia é definido como *semi-folha*. Se um nó possui todas as sub-árvores vazias, ou seja, não apresenta nenhum filho, então trata-se de uma *folha*. O *nível* (ou *profundidade*) está relacionado a um nó específico. A raiz, como *ancestral* de todos os demais nós, está localizada, por definição, no nível 0, os filhos da raiz, no nível 1, os filhos destes (netos da raiz), no nível 2, e assim sucessivamente. A *altura* de uma árvore (ou sub-árvore) corresponde a máxima profundidade atribuída a um de seus nós. Por definição, a altura de uma árvore vazia é igual a -1. No contexto deste trabalho, o conceito de *balanceamento* está associado à diferença máxima de alturas, $\Delta_{\max}$, entre todas as sub-árvores de um determinado nó. Se todos os nós da árvore têm $\Delta_{\max}$ suficientemente pequeno, então a árvore é considerada balanceada. O critério de altura é adotado por estar relacionado ao número de comparações para se obter uma informação e conseqüentemente, à eficiência de busca. Por outro lado, uma árvore é *degenerada* se estiver relativamente distante da condição de balanceamento, ou seja, se apresentar diferenças consideravelmente grandes de alturas entre as sub-árvores de pelo menos um de seus nós. No caso extremo, a estrutura hierárquica pode degenerar, em sua funcionalidade, para uma estrutura linear.

Pode-se entender *hash* ou *espalhamento* como um sistema que identifica uma informação associada a uma chave com exclusividade e eficiência. O método mais comum, em termos de classificação de dados, é a transformação de uma chave em uma posição de uma *tabela de espalhamento* através de uma *função de espalhamento*. A determinação desta função não é trivial, pois posições idênticas na tabela podem ser mapeadas para chaves distintas. Este efeito é denominado *colisão*.

Neste artigo, uma tabela com função de espalhamento diferente é implementada em cada nó de uma árvore. Valores de chaves assumem, portanto, posições específicas na árvore a partir da adaptação de uma função de espalhamento proposta, facilitando, desta forma, a busca de informações. Vale lembrar que são consideradas apenas chaves primárias ou exclusivas. A seguir, são apresentadas estruturas de dados usuais na computação para posterior comparação com a árvore de espalhamento proposta (ANE).

2.2 Árvores Binárias de Busca

Uma árvore binária de busca (ABB) [13][6] apresenta ordem 2 e critério de ordenação das chaves. Para que um nova informação possa ser inserida, sua chave é comparada à chave da informação já armazenada e, caso seja maior, a inserção deve ocorrer à direita do nó existente. Se menor, é inserida à esquerda deste mesmo nó. O novo nó se posiciona, desta forma, como filho de um nó folha. Cada nó da ABB apresenta, portanto, sub-árvore esquerda com chaves menores e sub-árvore direita com chaves maiores que a sua própria chave. Existem várias derivações da ABB. Dentre elas, a AVL [1] que também é binária de busca, porém mantida de forma balanceada pelo critério da diferença entre as alturas das sub-árvores dos nós (deve ser de no máximo 1). A cada inserção na AVL, caso ocorra desbalanceamento de um nó, pode haver uma rotação (religação de ponteiros) para manutenção deste critério, implicando que a árvore jamais se degenere. Esta ação promove a redução da quantidade de comparações de chaves para a busca de uma informação.

2.3 Árvores N-árias de Busca

Uma árvore n-ária é tal que em cada nó da mesma há um número arbitrário, limitado pela ordem da árvore, de filhos [7]. Normalmente uma árvore n-ária é entendida como um superconjunto de árvores binárias. É o caso da árvore n-ária de busca (ANB) [23] como superconjunto da árvore binária de busca (ABB). Pode-se também citar a árvore B [3] como uma versão, neste formato, bastante utilizada em sistemas de arquivos que, para ordem $N = 2d + 1$, sendo d um número natural, deve ser vazia ou satisfazer as seguintes condições: a raiz é uma folha ou tem no mínimo dois filhos; cada nó diferente da raiz e das folhas possui no mínimo $d+1$ filhos; cada nó tem no máximo $2d+1$ filhos. Cada nó em uma árvore B é chamado de página e pode armazenar várias chaves (com suas informações ou registros) [19]. É importante salientar que, para este tipo de árvore, a complexidade computacional é O$(\log n)$, no pior caso, em relação ao número de chaves. Uma redução ainda maior do número de comparações ou acessos (crítico em memória secundária), pode ser obtida pelo aumento do número de chaves por nó.

2.4 Tabela de Espalhamento

Uma tabela de espalhamento [23][13] é uma página de informações mapeada por uma função de espalhamento de chaves. Muitos aspectos sobre esta estrutura já foram apresentados. Para efeito de comparação com a ANE, este trabalho considera apenas um vetor, de tamanho N, de informações mapeadas pela função resto da divisão inteira da chave por N. Para resolução de colisão, é considerada a existência de árvores binárias de busca (ABB) em cada posição do vetor. Esta versão é denominada espalhamento usando árvores para armazenar as colisões (EA).

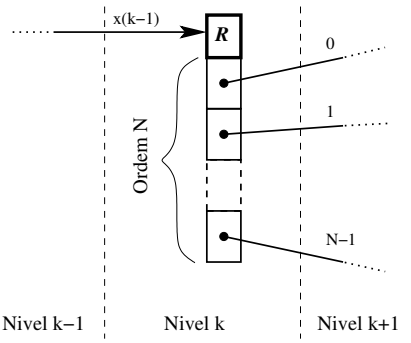


Figura 1: Estrutura de um nó da ANE e método de espalhamento.

3 Estrutura de busca proposta

A estrutura de dados proposta contém características tanto de árvores de busca como de tabelas de espalhamento. O objetivo é o de desenvolver um método híbrido de busca extremamente eficiente sem a necessidade de manipulação de ponteiros ou cópia de dados para a manutenção do balanceamento. Porém, não será possível um percurso em tal estrutura que determine a ordenação das chaves primárias das informações. Deste ponto em diante, um "elemento" da árvore será denominado "nó", termo este mais adequado na descrição da estrutura interna para efeito de implementação. A Figura 1 exibe, portanto, os campos de um nó da Árvore N-ária de Espalhamento (ANE) de ordem N localizado em um nível k. Por se tratar de uma estrutura hierárquica, algum nó no nível $k-1$ deve ter uma sub-árvore $x(k-1)$, onde $x(k-1) \in [0, N-1]$, que aponta para este nó que, por sua vez, apresenta um registro R (contendo uma chave ch que o identifique) e N outros ponteiros para o nível seguinte $(k+1)$. A inserção de um novo registro sempre ocorre em uma sub-árvore vazia de uma semi-folha. O processo de busca por esta posição é baseado na função de espalhamento "resto da divisão inteira" (o símbolo "$\%$" será usado para representar este operador). A contribuição deste trabalho está no fato deste espalhamento, que ocorre em cada nó no processo de busca, ser diferenciado para cada nível da ANE. Observa-se, através de experimentação, que esta configuração proporciona um certo balanceamento natural da árvore.

3.1 Espalhamento adaptativo

Cada nó da ANE apresenta um registro R (contendo uma chave ch) e N caminhos para seguir adiante. Poder-se-ia imaginar que a busca pudesse simplesmente utilizar o resto da divisão do valor da chave ch do registro pela ordem N. No entanto, esta abordagem gera árvores tão degeneradas quando maior for o número de colisões. Para uma estrutura de ordem 2, por exemplo, temos metade das chaves optando pela primeira sub-árvore (se $ch\%2$ igual a 0) e a outra metade, pela segunda (se $ch\%2$ igual a 1). Algo semelhante ao que ocorre com a ABB definida na Seção 2. Portanto, é preciso tornar a função de espalhamento adaptativa, ou seja, diferente para cada nível k da árvore. As Equações 1 e 2 descrevem a estratégia de decisão por uma subárvore $x(k)$, a partir do nível k, no processo de busca por uma chave em uma árvore de altura h.

$$t(k,i) = (ch + i)\%(N + k - i) \tag{1}$$

$$x(k) = \begin{cases} t(k,i) & \text{se } t(k,i) < N \\ t(k,i+1) & \text{caso contrário} \end{cases} \tag{2}$$

$$\text{onde } i \in [0,k], \text{ e } k \in [0,h]$$

Caso a chave ch procurada não esteja em um nível k qualquer, é necessário buscá-la no nível seguinte $(k+1)$. A escolha da sub-árvore $x(k)$, dentre as N possíveis (referente a ordem escolhida para a ANE), é feita inicialmente a partir do resultado $ch\%(N + k)$. Caso este valor não esteja no intervalo $[0, N - 1]$, então a nova tentativa é $(ch+1)\%(N+k-1)$. A terceira tentativa é $(ch+2)\%(N+k-2)$. Isto continua até que se encontre um valor de sub-árvore $(ch + i)\%(N + k - i)$ válido, onde $i \in [0, k]$. O último teste possível é $(ch + k)\%N$ resultando em um valor $x(k)$ obrigatoriamente pertencente ao conjunto $[0, N - 1]$.

3.2 Busca e inserção

O algoritmo de busca da ANE deve retornar uma referência (ponteiro) para um nó, cujo registro R contém a chave ch desejada, ou a posição de inserção (ponteiro nulo) caso ch ainda não exista. Por outro lado, se a busca concluir que ch já existe, a inserção deve resultar fracasso já que apenas chaves primárias (sem repetição) são consideradas. Alguns exemplos são construídos a seguir com a intenção de facilitar a compreensão da estrutura proposta. De modo a simplificar a explicação, apenas a chave ch (e não o registro R como um todo)

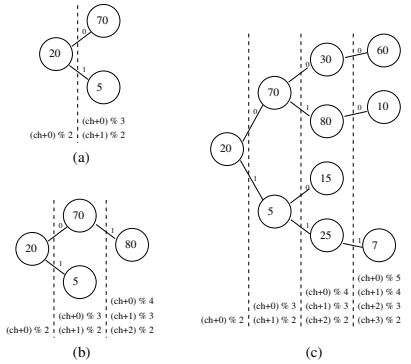


Figura 2: ANE de ordem 2. (a) Três primeiras inserções. (b) Quarta inserção. (c) Demais inserções.

é considerada em cada nó da árvore. Para a seqüência aleatória de inserção: 20, 5, 70, 80, 15, 30, 25, 7, 10, 60; a Figura 2 ilustra três momentos da formação de uma ANE de ordem 2. Em (a), é visualizada a árvore após as três primeiras inserções. A primeira chave (20 neste caso) corresponde sempre a raiz (localizada no nível 0). A inserção da segunda chave (5) consiste na comparação primeiramente com a raiz (se for o mesmo valor, a inserção fracassa). Caso seja diferente, há dois caminhos a decidir, prosseguindo a busca da posição de inserção: sub-árvore 0 ou sub-árvore 1. Esta escolha é feita, conforme a Equação 1, calculando o valor $(5 + 0)\%(2 + 0 - 0)$, que neste caso é 1. Como a sub-árvore 1 da raiz é vazia, a chave 5 é inserida nesta posição. O procedimento se repete para a terceira inserção (70). Esta chave é inserida na sub-árvore vazia $(70 + 0)\%(2 + 0 - 0)$ da raiz, ou seja, sub-árvore 0. Em (b), ocorre a inserção da próxima chave (80). Esta é comparada com a raiz. Sendo diferente, deve ir ao próximo nível pelo caminho $(80 + 0)\%(2 + 0 - 0)$, ou seja, 0. Agora no nível 1, é comparada com a chave já inserida 70. Sendo diferente, deve ir ao próximo nível pelo caminho $(80 + 0)\%(2 + 1 - 0)$, ou seja, 2. No entanto, este caminho não existe e, conforme a Equação 2, um novo $x(1)$ deve ser testado para $i = 1$: $(80 + 1)\%(2 + 1 - 1)$. Portanto, 80 é inserido na sub-árvore 1 da chave 70. Em (c), as demais chaves consideradas são inseridas seguindo a mesma idéia.

Considerando agora uma seqüência ordenada (mesmas chaves da Figura 2) de inserção: 5, 7, 10, 15, 20, 25, 30, 60, 70, 80; a Figura 3 mostra a construção da ANE, em (a), de ordem 2 ($N = 2$) e, em (b), de ordem 4 ($N = 4$). Observa-se que a distribuição das chaves pouco interfere na distribuição "física" dos nós na árvore, mantendo o balanceamento. Ou seja, a diferença entre as alturas de quaisquer duas sub-árvores de um nó qualquer, não difere demasiadamente na prática (ba-

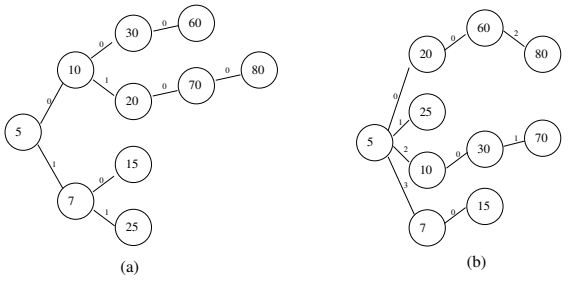


Figura 3: ANE com inserção ordenada. (a) Ordem 2. (b) Ordem 4.

lanceamento natural). Ressalta-se que tal processo dispensa cópias ou manipulação de ponteiros. Não há, no entanto, um percurso trivial que resulte na classificação em ordem crescente (ou decrescente) das chaves.

3.3 Remoção

O procedimento de remoção de uma chave ch da ANE é simples. No processo de busca, se ch não for encontrada, ocorre fracasso. Caso contrário, ch deve ser retirada da estrutura. Há dois casos possíveis: remoção de nó folha que é simplesmente desalocado; remoção de nó não-folha que deve ser desalocado e substituído pela folha mais profunda entre todas as suas sub-árvores. A Figura 4 exemplifica duas remoções. A motivação desta maneira de proceder a substituição está na tentativa de manutenção do balanceamento natural da árvore. Em (a), a chave 10 é removida. A folha mais profunda entre suas sub-árvores, 0 e 1, apresenta $ch = 80$ e esta a substitui. Em (b), a chave 5 é removida. As folhas mais profundas entre suas sub-árvores, 0, 1, 2 e 3, apresentam $ch = 80$ (sub-árvore 0) e $ch = 70$ (sub-árvore 2). Escolhe-se qualquer uma destas na substituição. Quando há mais de uma candidata, pode-se, por exemplo, escolher a chave substituta localizada na última sub-árvore visitada como na Figura 4(b).

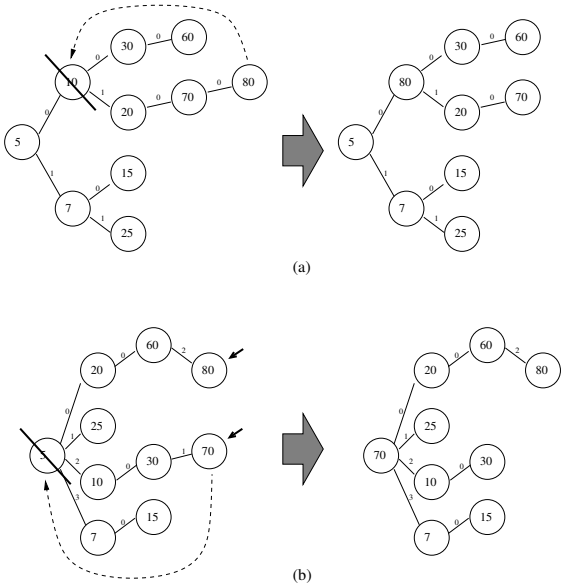


Figura 4: Remoção na ANE. (a) Remoção da chave 10 em uma ANE de ordem 2. (b) Remoção da chave 5 em uma ANE de ordem 4.

4 Experimentação e resultados

Nesta seção, o desempenho no processo de busca ou posicionamento de uma informação na ANE é apresentado. Os gráficos são elaborados considerando o número médio de comparações (ou logaritmos, na base 10, deste número) necessário na obtenção de uma chave existente (já inserida) ou determinação da posição de inserção de uma nova chave, em relação ao número de chaves possíveis. São estabelecidos 28 conjuntos com as seguintes quantidades de chaves: 10^2, $2 \cdot 10^2$, $3 \cdot 10^2$, $4 \cdot 10^2$, $5 \cdot 10^2$, $6 \cdot 10^2$, $7 \cdot 10^2$, $8 \cdot 10^2$, $9 \cdot 10^2$, 10^3, $2 \cdot 10^3$, $3 \cdot 10^3$, $4 \cdot 10^3$, $5 \cdot 10^3$, $6 \cdot 10^3$, $7 \cdot 10^3$, $8 \cdot 10^3$, $9 \cdot 10^3$, 10^4, $2 \cdot 10^4$, $3 \cdot 10^4$, $4 \cdot 10^4$, $5 \cdot 10^4$, $6 \cdot 10^4$, $7 \cdot 10^4$, $8 \cdot 10^4$, $9 \cdot 10^4$, 10^5. Tais conjuntos podem conter chaves organizadas de forma aleatória ou em ordem crescente. Espera-se que o desempenho de cada ANE, criada para cada conjunto, seja aproximadamente o mesmo independente da distribuição das chaves.

A Figura 5 exibe a comparação entre ABB, AVL e ANE (ordem 2) para todos os conjuntos de chaves. Em (a), as mesmas foram inseridas aleatoriamente e, neste caso, para 10^5 chaves são necessárias aproximadamente 16 comparações, em média, na AVL, e 21 comparações, na ABB e ANE. Estes valores são relativamente reduzidos demonstrando eficiência nas três árvores. Em (b), por outro lado, são feitas inserções de chaves ordenadas, sendo o pior caso para a ABB que se torna completamente degenerada (eficiência de busca idêntica a de uma estrutura linear), enquanto para a eficiência da AVL e da ANE mantém-se praticamente inalterada. Estes primeiros gráficos revelam o balanceamento natural da ANE, resultado da excelente distribuição de chaves proporcionada pelo espalhamento adaptativo. Através do ajuste de curva por regressão logarítmica [10], conforme a Figura 6 (com desvio padrão de 0.11 e coeficiente de correlação de 1.00), obtém-se a Equação 3 para a ANE de ordem 2. Onde x é o número de chaves e y, o número médio de comparações. Como a correlação é alta e o erro de ajuste relativamente reduzido, conclui-se que a ANE de ordem 2 é $O(\log n)$ em relação ao número de chaves. Ao incrementar a ordem da ANE, espera-se um desempenho ainda melhor. Esta expectativa é comprovada nas experimentações ilustradas na

Figura 7.

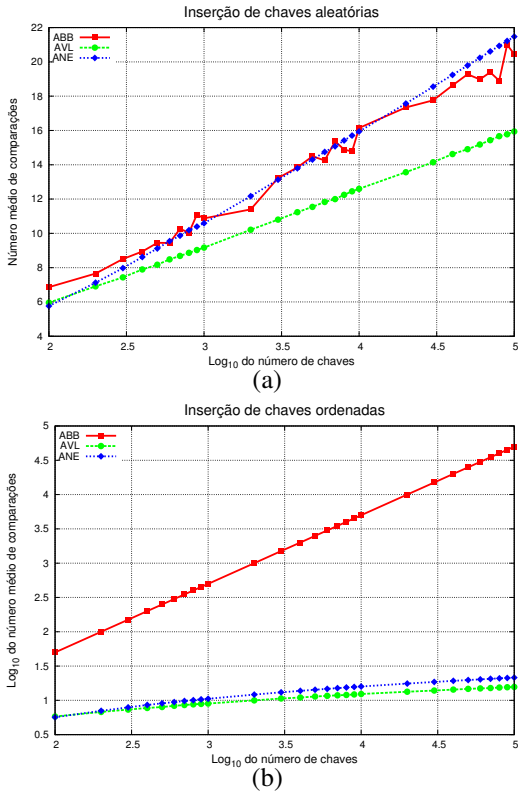
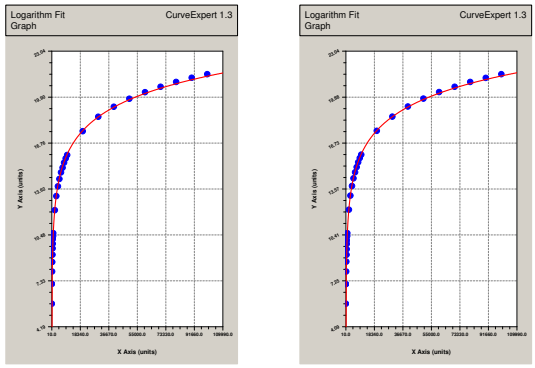


Figura 5: Comparação de desempenho entre ABB, AVL e ANE de ordem 2. (a) Chaves aleatórias. (b) Chaves ordenadas.

$$y \approx \begin{cases} 2.3068 \ln x - 5.2249 & \text{se aleatório} \\ 2.3218 \ln x - 5.3896 & \text{se ordenado} \end{cases} \quad (3)$$

O ajuste logarítmico das curvas na Figura 7 são: para ANE de ordem 4, $y \approx 1.5230 \ln x - 4.6578$ (desvio padrão: 0.32, coeficiente de correlação: 1.00); e para ANE de ordem 8, $y \approx 0.7756 \ln x - 1.7121$ (desvio padrão: 0.16, coeficiente de correlação: 1.00). Observa-se que, a medida que a ordem da ANE aumenta, o coeficiente multiplicativo do logaritmo neperiano diminui, ou seja, a constante de complexidade é reduzida. Se o grau da ANE é maior que 2, seu desempenho, como era de se esperar, supera o da AVL (lembrando que um nó desta última pode ter, no máximo, dois filhos).

Na Figura 7, foram comparadas estruturas com ordens (número de filhos por nó) distintas. A AVL é binária, mas outras estruturas como a ANB e B podem ter ordens maiores que 2. A Figura 8 exibe um melhor desempenho da ANE, em relação a estas duas últimas estruturas. As três árvores são comparadas em igualdade de condições, ou seja, com a mesma ordem



(a) (b)

Figura 6: Ajuste de curvas [10] para os resultados experimentais da ANE de ordem 2. (a) Chaves aleatórias. (b) Chaves ordenadas.

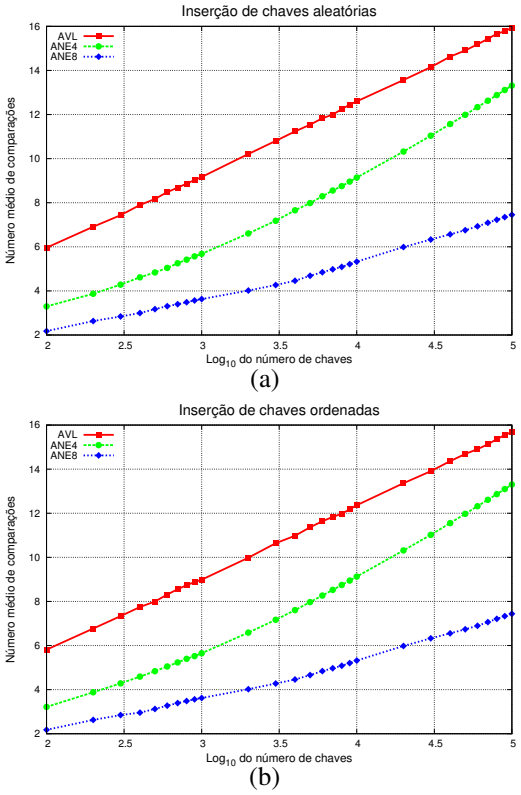


Figura 7: Comparação de desempenho entre AVL, ANE de ordem 4 e ANE de ordem 8. (a) Chaves aleatórias. (b) Chaves ordenadas.

(8 ou 1000) e mesmo conjunto de chaves (aleatórias ou ordenadas). A ANB não é balanceada e torna-se degenerada se as chaves forem inseridas de maneira ordenada. A árvore B demonstra-se comportada (configuração linear nos gráficos log-log apresentados), assim como pôde ser verificado também com a AVL. E a ANE, proposta deste trabalho, apresenta dois comportamentos diferentes. É $O(\log n)$ para grau 8, conforme verificado anteriormente, e apresenta uma curva diferenciada para grau 1000. Considerando os resíduos mínimos de aproximação para um conjunto com mais de 10^3 chaves, a função que melhor se ajusta aos dados obtidos para a ANE de ordem 1000 é $y \approx \frac{1.9629x}{478.1015+x}$ (desvio padrão: 0.19, coeficiente de correlação: 0.91). No limite, $x \leftarrow \infty$, esta função se aproxima de 1.9629, ou seja, torna-se uma constante ou um espalhamento perfeito $O(1)$. No entanto, isto não é possível na prática. Uma ANE de ordem 1000 certamente não oferece espalhamento perfeito para um conjunto de chaves suficientemente grande. Esta análise de dependência entre o grau da ANE e a quantidade de chaves consideradas é feita adiante.

A Figura 9 ilustra a última comparação feita, desta vez entre o EA e a ANE, para ordens 10 e 1000 (considerando apenas chaves aleatórias). Ambos espalhamentos se mantém perfeito até 10^3 chaves. A ANE supera o EA rapidamente com o aumento do número de chaves.

De modo a demonstrar o balanceamento natural da estrutura proposta, uma vez construída a ANE para um dado conjunto de chaves (quantidade igual a 10^2, 10^3, 10^4 e 10^5, aleatórias e ordenadas), as alturas de todas as sub-árvores da raiz foram calculadas. As alturas mínima e máxima detectadas são apresentadas na Tabela 1 para cada árvore (de ordem 2, 4, 8, 16, 50, 100 e 320). Conclui-se que todas as sub-árvores, em relação à raiz, têm alturas praticamente iguais. A máxima diferença encontrada, por exemplo, foi igual a 4 para a ANE, de ordem 4, criada a partir da inserção de 10^4 chaves ordenadas. Na grande maioria dos casos, no entanto, esta diferença é 0 ou 1. Este resultado é semelhante ao critério de balanceamento da AVL, mas foi gerado naturalmente e é preservado em qualquer ordem da ANE. O valor 320 escolhido para ordem foi um limiar encontrado nos testes realizados para um espalhamento perfeito na ANE, considerando até 10^5 chaves.

A Figura 10 apresenta experimentações realizadas com quatro conjuntos – 10^2 chaves, 10^3 chaves, 10^4 chaves e 10^5 chaves – para ANE com ordem variando entre 2 e 230. Nota-se que há um expressivo decaimento do número médio de comparações com o aumento da ordem da ANE.

O conjunto de Equações 4 consiste no ajuste das

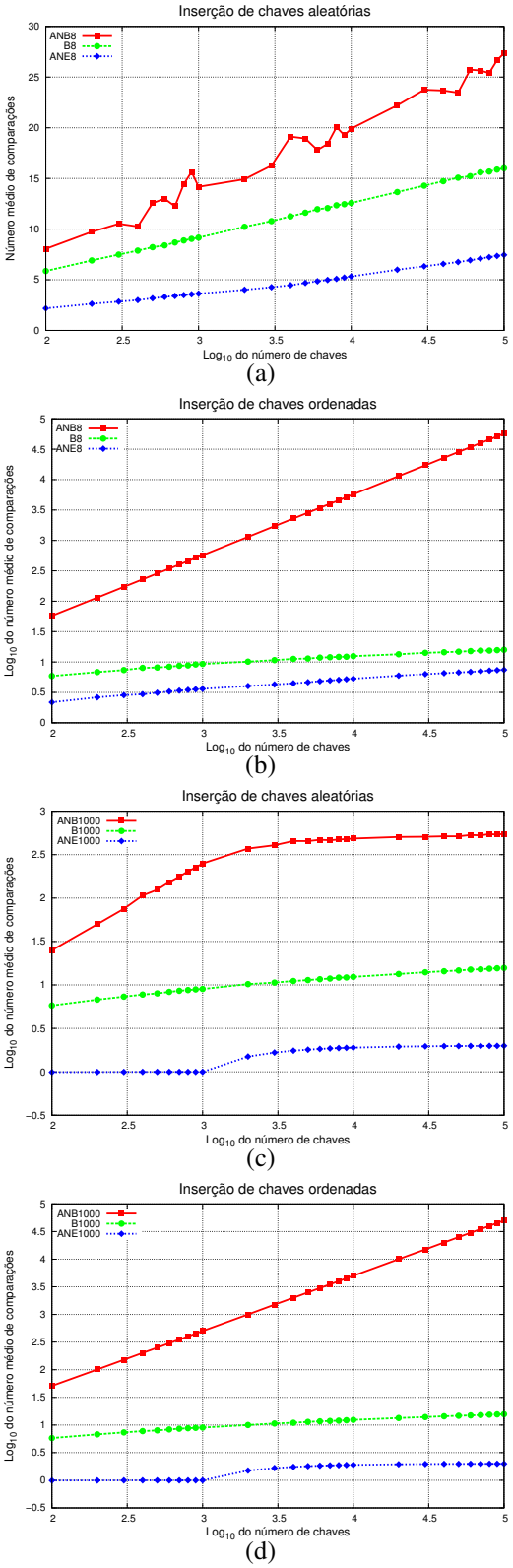
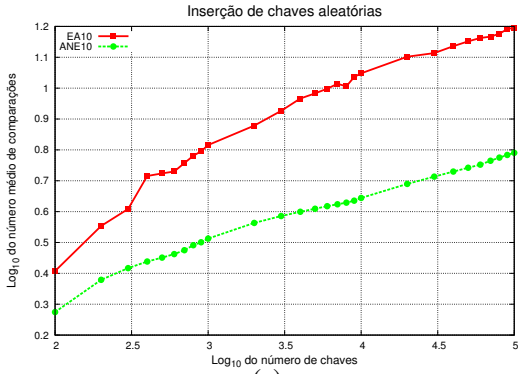
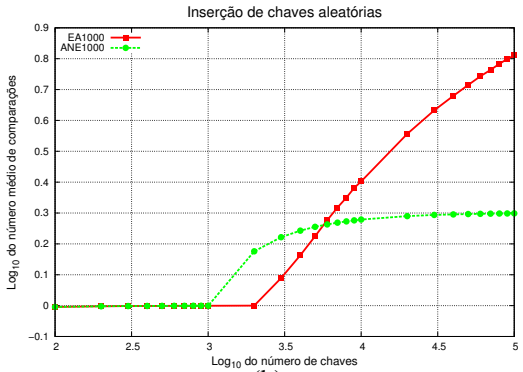


Figura 8: Comparação de desempenho entre ANB, B e ANE. (a) Ordem 8 com chaves aleatórias. (b) Ordem 8 com chaves ordenadas. (c) Ordem 1000 com chaves aleatórias. (d) Ordem 1000 com chaves ordenadas.



(a)



(b)

Figura 9: Comparação de desempenho entre EA e ANE. (a) Ordem 10. (b) Ordem 1000.

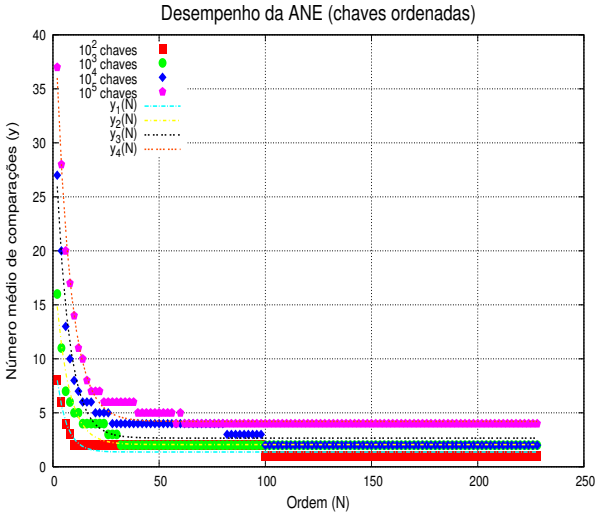


Figura 10: Desempenho da ANE em relação a sua ordem.

Tabela 1: Alturas máxima e mínima entre todas as sub-árvores do nó raiz, para cada conjunto de chaves (inserção aleatória ou ordenada), considerando ANE de ordem 2, 4, 8, 100 e 320.

Alturas máxima e mínima da estrutura proposta	Número de chaves			
	10^2	10^3	10^4	10^5
Mínima (ordem 2, aleatórias)	8	15	25	35
Máxima (ordem 2, aleatórias)	10	15	26	37
Mínima (ordem 2, ordenadas)	7	15	25	35
Mínima (ordem 2, ordenadas)	7	15	26	36
Mínima (ordem 4, aleatórias)	4	9	15	25
Máxima (ordem 4, aleatórias)	5	10	18	27
Mínima (ordem 4, ordenadas)	4	8	15	24
Mínima (ordem 4, ordenadas)	5	10	19	27
Mínima (ordem 8, aleatórias)	2	4	8	14
Máxima (ordem 8, aleatórias)	2	5	9	15
Mínima (ordem 8, ordenadas)	2	4	8	14
Mínima (ordem 8, ordenadas)	2	5	9	16
Mínima (ordem 16, aleatórias)	1	2	4	6
Máxima (ordem 16, aleatórias)	1	3	4	7
Mínima (ordem 16, ordenadas)	1	2	4	6
Mínima (ordem 16, ordenadas)	1	3	5	7
Mínima (ordem 50, aleatórias)	0	1	2	3
Máxima (ordem 50, aleatórias)	1	1	3	4
Mínima (ordem 50, ordenadas)	0	1	2	3
Mínima (ordem 50, ordenadas)	1	1	3	4
Mínima (ordem 100, aleatórias)	-1	1	1	2
Máxima (ordem 100, aleatórias)	0	1	1	3
Mínima (ordem 100, ordenadas)	-1	1	1	2
Mínima (ordem 100, ordenadas)	0	1	1	3
Mínima (ordem 320, aleatórias)	-1	1	1	1
Máxima (ordem 320, aleatórias)	0	1	1	1
Mínima (ordem 320, ordenadas)	-1	1	1	1
Mínima (ordem 320, ordenadas)	0	1	1	1

curvas da Figura 10. Entre parênteses, são relatados o desvio padrão (s) e o coeficiente de correlação (r) de cada ajuste.

$$\begin{aligned} y_1 &\approx 9.1828\frac{1}{N^{0.4236}} \quad (s=0.39, r=0.91) \\ y_2 &\approx 19.7471\frac{1}{N^{0.5085}} \quad (s=0.63, r=0.93) \\ y_3 &\approx 39.7066\frac{1}{N^{0.6057}} \quad (s=0.68, r=0.98) \\ y_4 &\approx 52.7194\frac{1}{N^{0.5664}} \quad (s=1.25, r=0.96) \end{aligned} \tag{4}$$

Os numeradores $9.1828, 19.7471, 39.7066$ e 52.7194 das razões das Equações 4 definem, por sua vez, uma reta e são ajustadas com desvio padrão relativamente baixo (2.73) e coeficiente de correlação bastante alto (0.99). Portanto, a ANE com diversas ordens e para 10^p chaves segue aproximadamente a Equação 5, sendo

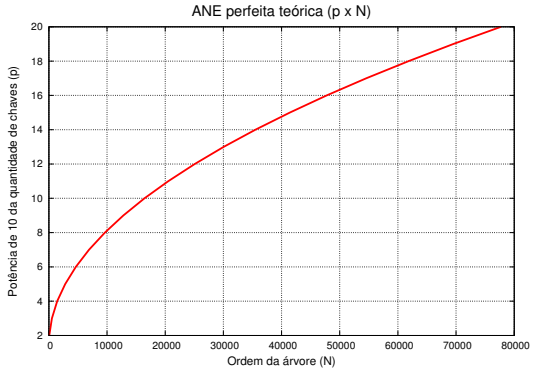


Figura 11: Quantidade de chaves (dado p, há 10^p chaves) em relação à ordem (N) da ANE para uma única comparação, em média, na busca.

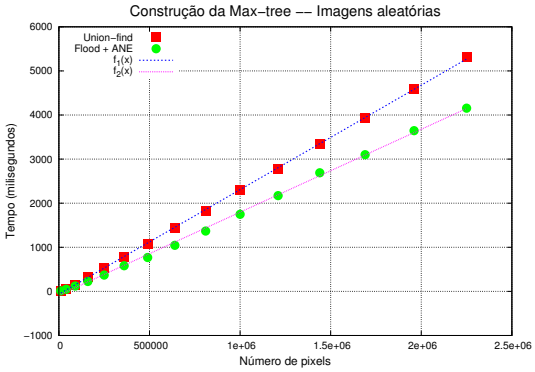


Figura 12: Comparação entre os algoritmos Max-tree de Najman e Couprie ("Union-find") e Salembier modificado ("Flood + ANE").

$\alpha \approx 0.5$, para determinação do número médio de comparações y.

$$y \approx (15.0569p - 22.3603)\,\frac{1}{N^{\alpha}} \quad \text{para } 0.0 < \alpha < 1.0 \tag{5}$$

O projeto de uma ANE com apenas uma única comparação, em média, pode ser derivado da Equação 5, considerando $y = 1$. Desta forma, e supondo $\alpha = 0.5$, obtém-se a Equação 6 para um espalhamento teórico perfeito. A Figura 11 ilustra o gráfico, a partir da Equação 6, dos valores necessários de p e de N para uma suposta ANE perfeita. Para 10^{100} chaves ($p = 100$), por exemplo, é necessário que a ANE tenha ordem aproximadamente igual a $N \approx 2200267$ para um espalhamento perfeito.

$$N \approx (15.0569p - 22.3603)^2 \tag{6}$$

5 Aplicação de Processamento de Imagens

Uma árvore para representação de imagens em níveis de cinza e desenvolvimento de algoritmos eficientes de processamento de imagens, denominada Max-tree [8] [11], foi beneficiada com a utilização da ANE. Najman e Couprie [15] propuseram recentemente um algoritmo rápido para a construção da Max-tree, usando o método *Union-find* de Tarjan [22]. Salembier et al. [18] havia proposto anteriormente um método de inundação hierárquico (*Flood*). Este necessita estabelecer relações pai-filho entre nós dinamicamente e a ANE se adapta bem à situação, promovendo ganho de desempenho. A Figura 12 compara os tempos de execução dos algoritmos com "Union-find" e a "Flood + ANE". A inclinação da reta $f_1(x)$ (novo algoritmo usando a ANE de ordem 11) é 20.63% menor que a $f_2(x)$ (ajustadas aos testes com imagens quadradas aleatórias).

6 Considerações Finais

Este trabalho propõe um nova estrutura de dados para busca e inserção eficiente de informações com implementação simples baseada nas características de busca de duas outras estruturas clássicas da computação: árvore e tabela de espalhamento. A análise experimental demonstra que a árvore n-ária de espalhamento (ANE) proposta, em termos do número médio de comparações, é de $\mathrm{O}(\log n)$ se sua ordem for relativamente baixa e de $\mathrm{O}(1)$, se suficientemente alta, em relação ao número de chaves considerado. Embora o desempenho da ANE, para ordem 2, seja inferior a de uma AVL, os dados coletados indicam tendência semelhante no gráfico entre número de comparações e quantidade de chaves. Para ordens maiores, a ANE supera a AVL. Já diante das estruturas hierárquicas, como a árvore B, considerando a mesma quantidade máxima de filhos por nó, a ANE sempre se mostrou melhor. Assim como foi melhor em relação ao espalhamento simples (não hierárquico) usando ABB para resolução de colisões (EA). Por outro lado, ABB, AVL, ANB, B são estruturas que, a partir de um determinado percurso, podem resultar na seqüência ordenada das chaves. Funcionalidade esta não suportada pela EA e ANE que carecem, neste caso, de um método de classificação separado. A ANE é, portanto, um mecanismo essencialmente de busca e deve ser executada, como exibida neste trabalho, em memória principal. No entanto, pode perfeitamente ser implementada em memória secundária através do acréscimo do número de chaves (registros) armazenadas em cada nó, nos moldes das árvores n-árias tradicionais. Neste caso, as chaves seriam mantidas ordenadas em cada nó e poderia ser feita uma busca binária simples para determi-

nação da posição de inserção ou da posição em que se encontra a informação dentro do nó (como ocorre, por exemplo, na árvore B usada nas comparações). Uma vez demonstrado experimentalmente, neste trabalho, o balanceamento natural da ANE, pode-se definir a posição de interesse (busca ou inserção) mais rapidamente, ou seja, com menor número de comparações. Isto em memória secundária, como disco rígido, significa uma quantidade menor de re-posicionamentos do cabeçote de leitura, processo físico demasiadamente lento (em relação à memória RAM). A exigência de um número mínimo de chaves por nó na árvore B, bastante utilizada para disco, não reflete um desempenho melhor quando comparada a ANE nas experimentações feitas. O balanceamento, em relação à altura de sub-árvores, proporcionado naturalmente pela ANE é um resultado bastante expressivo e útil em aplicações onde é crítica a busca incessante por informações, sejam elas de qualquer natureza, desde que indexadas por um chave inteira.

Como trabalhos futuros, outros parâmetros para análise de desempenho devem ser considerados como, por exemplo, uso de memória e tempo de execução. Além disto, a melhor estratégia de implementação da proposta de remoção de informações deve ser estudada e implementada para posterior análise de desempenho considerando também esta operação.

Na aplicação citada (seção 5), cada par pai-filho de elementos necessários à construção da Max-tree antes buscados linearmente em um conjunto de dados, são agora localizados pela ANE. Em testes preliminares, com uma ANE de ordem 11, a redução do tempo de execução foi de 34.90 vezes para uma imagem aleatória de 300×300 pixels e de 31.61 vezes para uma imagem aleatória de 500×500 pixels. No entanto, ao aumentar a ordem da ANE, o tempo não necessariamente diminuiu provavelmente devido ao custo computacional de alocação. Novos testes, auxiliado pela análise de memória ocupada, devem ser elaborados para esclarecer este efeito. Outras aplicações devem ser consideradas.

Referências

[1] Adelson-Velskii, G. and Landis, E. M. An algorithm for the organization of information. *Doklady Akademii Nauk SSSR*, 146:263–266, 1962.

[2] Akutsu, T., Onizuka, K., and Ishikawa, M. New hashing techniques and their application to a protein structure database system. In *Proceedings of the 28th Hawaii International Conference on System Sciences*, page 197. IEEE, 1995.

[3] Bayer, R. Binary b-trees for virtual memory. In *ACM-SIGFIDET Workshop*, pages 219–235, San Diego, CA, 1971.

[4] Chen, T., Chen, A., and Yang, W. Hash-semijoin: a new technique for minimizing distributed query time. In *Proceedings of the Third Workshop on Future Trends of Distributed Computing Systems*, pages 325–330, 1992.

[5] Choi, K. and Kim, K. T*-tree: a main memory database index structure for real time applications. In *Proceedings of Third International Workshop on Real-Time Computing Systems and Applications*, pages 81–88, 1996.

[6] Cormen, T. H., Leiserson, C. E., Rivest, R. L., and Stein, C. *Introduction to Algorithms*. MIT Press and McGraw-Hill, second edition, 2001.

[7] Esakov, J. and Weiss, T. *Data structures an advanced approach using C*. Prentice-Hall, Inc., 1989.

[8] Garrido, L. O. *Hierarchical region based processing of images and video sequences: application to filtering, segmentation and information retrieval*. PhD thesis, Department of Signal Theory and Communications - Universitat Politècnica de Catalunya, Barcelona, April 2002.

[9] Grossman, J. and Jakab, L. Using the bch construction to generate robust linear hash functions. In *Information Theory Workshop. IEEE*, pages 250–253, 2004.

[10] Hyams, D. G. *CurveExpert 1.3, a comprehensive curve fitting package for Windows*, 2005. http://curveexpert.webhop.biz (visitado em 07/2006).

[11] J. Mattes, R. M. and Demongeot, J. Tree Representation for Image Matching and Object Recognition. In 8^{th} *International Conference on Discrete Geometry for Computer Imagery*, pages 298–312, Marne-la-Vallee, France, March 1999.

[12] Kharbutli, M., Solihin, Y., and Lee, J. Eliminating conflict misses using prime number-based cache indexing. *IEEE Trans. Comput.*, 54(5):573–586, 2005.

[13] Knuth, D. *The Art of Computer Programming*. Addison-Wesley, third edition, 1997.

[14] McCarthy, E., Balado, F., Slvestre, G., and Hurley, N. A framework for soft hashing and its application to robust image hashing. In *International Conference on Image Processing*, pages 397–400, 2004.

[15] Najman, L. and Couprie, M. Building the Component Tree in Quasi-Linear Time. *IEEE Transactions on Image Processing*, 15(11):3531–3539, 2006.

[16] Ramakrishna, M. V. A simple perfect hashing method for static sets. In *Proceedings of the Fourth International Conference on Computing and Information*, pages 401–404. IEEE, 1992.

[17] Ryu, C., Song, E., Jun, B., Kim, Y., and Jin, S. Hybrid-th: a hybrid access mechanism for real-time memory-resident database systems. In *Proceedings of Fifth International Conference on Real-Time Computing Systems and Applications*, pages 303–310, 1998.

[18] Salembier, P., Oliveras, A., and Garrido, L. Antiextensive Connected Operators for Image and Sequence Processing. *IEEE Transactions on Image Processing*, 7(4):555–570, 1998.

[19] Scwarcfiter, J. L. and Markenzon, L. *Estruturas de Dados e Seus Algoritmos*. Ed. LTC., 1994.

[20] Shang, F. and Pan, Y. An absolute non-collision hash algorithm. In *Proceedings of 2004 International Conference on Machine Learning and Cybernetics*, pages 26–29, 2004.

[21] Sprugnoli, R. J. Perfect hashing functions: a single probe retrieving method for static sets. *Communications of the ACM*, 20:841–850, 1977.

[22] Tarjan, R. E. Efficiency of a good but not linear set union algorithm. *Journal of the ACM*, 22:215–225, 1975.

[23] Tenenbaum, A. M., Langsam, Y., and Augenstein, M. J. *Data Structures Using C*. McGraw-Hill, 1989.

[24] Vandierendonck, H. and Bosschere, K. Xor-based hash functions. *IEEE Trans. Comput.*, 54(7):800–812, 2005.