ESTUDO EMPÍRICO DE UMA NOVA IMPLEMENTAÇÃO PARA O ALGORITMO DE CONSTRUÇÃO DE ÁRVORE PATRICIA

L. C. A. ALBUQUERQUE[1], N. ZIVIANI[2]

SUMÁRIO

PATRICIA é um algoritmo eficiente para casamento de cadeias. Este trabalho tem por objetivo apresentar um algoritmo melhorado para construção de árvores Patricia. O algoritmo proposto é comparado empiricamente com os algoritmos mais conhecidos de construção de árvores Patricia, obtendo resultados bem favoráveis ao novo algoritmo.

ABSTRACT

PATRICIA is an efficient algorithm for string matching. The objective of this paper is to present a better algorithm for Patricia tree construction. The proposed algorithm is compared empirically to the well known algorithms for Patricia tree construction, and favorable results are obtained.

[1] Bel. em Matemática (UFV, 1974), Aluno de Pós-Graduação em Ciência da Computação (UFMG); estruturas de dados, linguagens e sistemas de programação; Departamento de Matemática; UFV, 36570 Viçosa, MG. Tel.: (031) 891-1790, r. 368.

[2] Engº Mecânico (UFMG, 1971), Mestre em Informática (PUC/RJ, 1976), Ph.D. em Ciência da Computação (Univ. Waterloo, 1982); engenharia de software, estruturas de dados, análise de algoritmos; Departamento de Ciência da Computação, ICEx, UFMG, Caixa Postal 702, 30000 Belo Horizonte, MG. Tel.: (031) 443-4088.

1. INTRODUÇÃO

PATRICIA é a abreviatura de *Pratical Algorithm To Retrieve Information Coded In Alphanumeric* (Algoritmo Prático para Recuperar Informação Codificada em Alfanumérico). Este algoritmo foi originalmente criado por MORRISON [1968] num trabalho de casamento de cadeias (*string matching*), aplicado em recuperação de informação em arquivos de grande porte. KNUTH [1973] deu um novo tratamento ao algoritmo, reapresentando-o de forma mais clara como um caso particular de pesquisa digital, essencialmente, um caso de árvore *trie* binária.

Além disso KNUTH [1973] apresentou explicitamente um algoritmo de pesquisa e propôs um algoritmo de inserção (construção de árvore Patricia) como exercício, dando sugestão de como elaborá-lo na parte de respostas aos exercícios. Ele apresentou também um estudo analítico do algoritmo de pesquisa, obtendo o número de inspeções de bits para uma pesquisa aleatória com sucesso e para uma pesquisa sem sucesso; esses valores são, respectivamente, $\log_2 N + 0.33275$ e $\log_2 N - 0.31875$, onde N é o número de chaves armazenadas na árvore.

SEDGEWICK [1983] apresentou novos algoritmos de pesquisa e de inserção baseados nos algoritmos propostos por Knuth. GONNET [1984a] propôs também outros algoritmos de pesquisa e de inserção usando recursividade e uma estrutura de dados ligeiramente diferente para representar a árvore.

O objetivo deste trabalho é o de apresentar uma melhoria no algoritmo de inserção cuja característica principal é a de conseguir construir a árvore Patricia utilizando o menor número de inspeções de bits dentre os algoritmos mais conhecidos. Para mostrar esta característica do algoritmo, realizou-se um estudo experimental, onde foram comparados quatro algoritmos de inserção:

(1) Algoritmo de Sedgewick (SEDGEWICK [1983], p. 222),
(2) Algoritmo de Gonnet (GONNET [1984a], pp. 110-111),
(3) Algoritmo Melhorado de Gonnet (GONNET [1984b]),
(4) Algoritmo Proposto.

As medidas de complexidade consideradas foram:

I(N) - o número total de inspeções de bits necessário para construir uma árvore com N chaves;
C(N) - o número de comparações entre bits de chaves necessário para construir uma árvore com N chaves;
W(N) - o número de inspeções de bits necessário para traçar o caminho de pesquisa e inserção durante a construção de uma árvore com N chaves;
E(N) - o número de nodos de uma árvore com N chaves.

Dentre os algoritmos estudados, verificou-se que os algoritmos (1) e (4) acima são os que apresentam melhor desempenho quanto ao número total I(N) de inspeções de bits, com o al-

goritmo (4) apresentando ainda uma redução de mais de 20% sobre o algoritmo (1) no número I(N). Cabe observar que a inspeção de bits é a operação mais cara dentre as operações dos algoritmos considerados.

2. PESQUISA DIGITAL

A pesquisa digital é baseada na representação das chaves como uma seqüência de caracteres ou de dígitos. A grosso modo, o método de pesquisa digital é realizado da mesma forma que uma pesquisa em dicionários que possuem aqueles "índices de dedo". Com a primeira letra da palavra são determinadas todas as páginas que contêm as palavras iniciadas por aquela letra.

Os métodos de pesquisa digital são particularmente vantajosos quando as chaves são grandes e de tamanho variável. No problema de casamento de cadeias trabalha-se com chaves semi-infinitas, isto é, sem limitação explícita quanto ao tamanho delas. Um aspecto interessante quanto aos métodos de pesquisa digital é a possibilidade de localizar todas as ocorrências de uma determinada cadeia em um texto, com tempo de resposta logarítmico em relação ao tamanho do texto.

Este aspecto é particularmente importante em aplicações que utilizam grandes bancos de dados, tais como os utilizados em sistemas legislativos e judiciários, jornalismo, informações técnico-científicas, informações policiais, informações mercadológicas, bibliotecas e centros de informação em geral. Nesses tipos de arquivos apenas adições são realizadas, e as consultas são imprevisíveis. Como exemplo, alguém poderia necessitar de uma relação dos artigos de periódicos científicos que tratam do assunto "bancos de dados não estruturados", ou então a relação dos acidentes de tráfego causados por embriaguez em regiões urbanas durante os fins de semana. A apresentação de algoritmos eficientes para recuperação de informação utilizando pesquisa digital em bancos de dados não estruturados é tratada em detalhes em ALBUQUERQUE, GONNET e ZIVIANI [1985].

A seguir apresentam-se os métodos de pesquisa digital mais utilizados.

2.1. *Trie*

Uma *trie* é uma árvore M-ária cujos nodos são vetores de M componentes com campos correspondentes aos dígitos ou caracteres que formam as chaves. Cada nodo no nível i representa o conjunto de todas as chaves que começam com a mesma seqüência de i dígitos ou caracteres. Este nodo especifica uma ramificação com M caminhos dependendo do (i+1)-ésimo dígito ou carácter de uma chave.

Supondo as chaves como seqüência de bits, isto é, M = 2, o algoritmo de pesquisa digital em *trie* é semelhante ao de pesquisa em árvore binária, exceto que, ao invés de se caminhar na árvore de acordo com o resultado de comparação entre chaves, caminha-

se de acordo com os bits da chave. Em 1960, Fredkin chamou este método de *trie*, que corresponde a um pedaço da palavra *retrieval* (KNUTH [1973], p. 481).

A Figura 2.1.1. mostra uma *trie* construída a partir das seguintes chaves de 6 bits:

B = 010010
C = 010011
H = 011000
J = 100001
Q = 101000

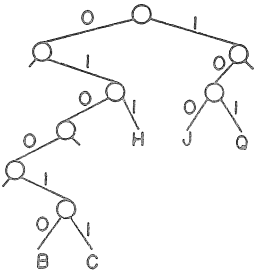

Fig. 2.1.1

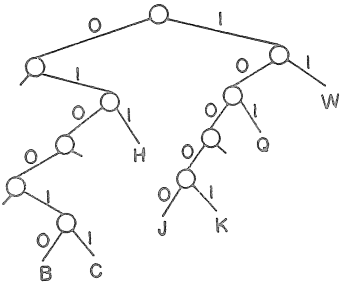

Fig. 2.1.2

Para construir uma *trie* faz-se uma pesquisa na árvore com a chave a ser inserida. Se o nodo externo em que a pesquisa terminar for vazio, cria-se um novo nodo externo nesse ponto contendo a nova chave, como ilustra a inserção da chave W = 110110 na Figura 2.1.2. Se o nodo externo contiver uma chave, cria-se um ou mais nodos internos cujos descendentes conterão a chave já existente e a nova chave. A Figura 2.1.2 ilustra a inserção da chave K = 100010 que envolve repor J por um novo nodo interno cuja sub-árvore esquerda é outro novo nodo interno cujos filhos são J e K, porque estas chaves possuem os mesmos bits até a quinta posição.

O formato das *tries*, diferentemente das árvores binárias comuns, não depende da ordem em que as chaves são inseridas e sim da estrutura das chaves através da distribuição de seus bits. Uma grande desvantagem das *tries* é a formação de caminhos de uma só direção para chaves com um grande número de bits em comum. Por exemplo, se duas chaves diferirem somente no último bit, elas formarão um caminho cujo comprimento é igual ao tamanho delas, não importando quantas chaves existem na árvore. Veja o caminho gerado pelas chaves B e C na Figura 2.1.1.

2.2. Patricia

O algoritmo para construção de árvore Patricia (MORRISON [1968]) é baseado no método de pesquisa digital, mas sem apresentar o inconveniente citado para o caso das *tries*. O problema de caminhos de uma só direção é eliminado por meio de uma solução simples e elegante: cada nodo interno da árvore contém o índice do bit a ser testado para decidir qual rumo tomar. A Figura 2.2.1 apresenta a árvore Patricia gerada a partir das chaves dadas acima.

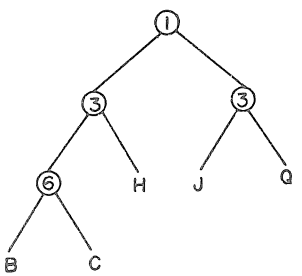

Fig. 2.2.1

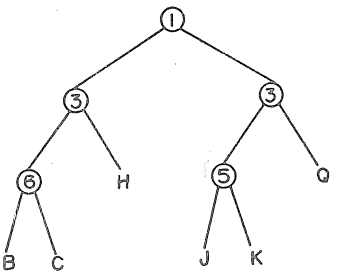

Fig. 2.2.2

Para inserir a chave K = 100010 na árvore da Figura 2.2.1, a pesquisa inicia pela raiz e termina quando se chega ao nodo externo contendo J. Os índices dos bits nas chaves estão ordenados da esquerda para a direita. Assim, o bit de índice 1 de K é 1, indicando a sub-árvore direita, e o bit de índice 3 indica a sub-árvore esquerda que, neste caso, é um nodo externo. Isto significa que as chaves J e K mantêm o padrão de bits 1x0xxx, assim como qualquer outra chave que seguir este caminho de pesquisa. Um novo nodo interno repõe o nodo J, e este juntamente com o nodo K serão os nodos externos descendentes. O índice do novo nodo interno é dado pelo primeiro bit diferente das duas chaves em questão, que é o bit de índice 5. Para determinar qual será o descendente esquerdo e o direito, é só verificar o valor do bit 5 de ambas as chaves, conforme mostrado na Figura 2.2.2.

A inserção da chave W = 110110 ilustra um outro aspecto. A pesquisa sem sucesso na árvore da Figura 2.2.2 é realizada de maneira análoga. Os bits das chaves K e W são comparados a partir do primeiro para determinar em qual índice eles diferem, sendo, neste caso, os de índice 2. Portanto o ponto de inserção agora será no caminho de pesquisa entre os nodos internos de índice 1 e 3. Cria-se aí um novo nodo interno de índice 2, cujo descendente direito é um nodo externo contendo W e cujo descendente

esquerdo é a sub-árvore de raiz de índice 3, conforme ilustra a Figura 2.2.3.

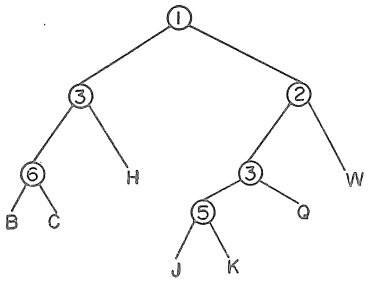

Fig. 2.2.3

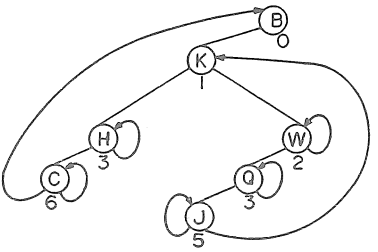

Fig. 2.2.4

O método delineado acima, assim como o método da *trie*, usa dois tipos distintos de nodos. Os nodos internos contêm o índice do bit a ser inspecionado e os nodos externos contêm somente as chaves. Ao invés de armazenar as chaves nos nodos externos, existe uma outra opção que é a de manter somente um apontador para outra estrutura que contenha as chaves fisicamente. Isto é interessante principalmente no caso de as chaves serem longas cadeias de caracteres.

Em algumas linguagens de programação pode ser complicado definir uma estrutura de dados com dois tipos de nodos diferentes. Para contornar este problema KNUTH [1973] sugere a criação de mais um campo nos nodos internos para conter as chaves. Os nodos externos são embutidos nos nodos internos, fazendo com que todos tenham a mesma estrutura. Neste caso as chaves nos nodos não serão usadas ao longo do caminho de pesquisa. Ao chegar a um ponto de um nodo externo, a chave é localizada através de um apontador dirigido para cima, conforme ilustra a Figura 2.2.4. Embora o tamanho de cada nodo aumente, isto reduz à metade o número de nodos da árvore, pois os nodos externos não existem mais explicitamente. Observe na Figura 2.2.4 que as chaves estão dentro dos nodos e os índices estão abaixo. Além disso há um nodo-cabeça cujo índice foi considerado como 0 (zero).

Agora aparece um pequeno problema: como diferenciar um apontador puro para um nodo interno de um apontador para cima simulando a ocorrência de um nodo externo? KNUTH [1973] utiliza mais dois campos extras de 1 bit cada para indicar se o respectivo apontador para a a raiz da sub-árvore está apontando para um nodo interno ou um nodo externo (para cima). E esta é a única solução razoável que poderia ser tomada por Knuth, pois o campo de índice formulado por ele não contém o real valor do índice, e sim o incremento que deve ser dado ao valor corrente do índice para obter o novo índice.

Isto tem a vantagem de serem necessários menos bits no campo de índice, compensando, assim, os dois bits adicionais para discriminar os apontadores das sub-árvores.

SEDGEWICK [1983] adotou outra solução a partir da observação de que os índices de uma mesma sub-árvore aparecem ordenadamente em relação ao nível do nodo. Neste caso o índice de um nodo de nível i é menor do que o de um nodo de nível j, desde que i < j. Logo, testando os índices dos nodos do caminho de pesquisa, é possível saber se se chegou a um "nodo externo" ou não.

3. ALGORITMOS PATRICIA

A seguir são apresentados dois algoritmos conhecidos para construção de árvore Patricia, uma proposta de melhoria de um deles, e uma nova proposta de algoritmo que é mais eficiente que os três primeiros.

3.1. Algoritmo de Sedgewick

O Algoritmo de Sedgewick (SEDGEWICK [1983], p. 222) usa a estrutura de dados para representar a árvore com os nodos externos implícitos. Neste caso a medida de espaço é E(N) = N, onde N é o número de chaves da árvore.

A árvore é inicializada com um nodo-cabeça contendo a primeira chave a ser inserida, o campo de índice igual a 0 (zero) e o apontador de sub-árvore esquerda dirigido para o próprio nodo. Considerando uma árvore com n chaves, a inserção de uma nova chave k segue os passos abaixo:

1. Desce-se pela árvore da maneira já descrita na Seção 2.2 até chegar a uma posição de um "nodo externo", o qual não existe explicitamente na árvore, e sim embutido em algum nodo acima.
2. Os bits da chave k' do nodo determinado no passo 1 são comparados com os respectivos bits da chave k, a partir do primeiro, até chegar a um índice cujos bits difiram. Se todos os bits forem iguais, a chave k já se encontra na árvore e o algoritmo termina.
3. Desce-se novamente pela árvore a partir da raiz até chegar a um nodo cujo campo de índice seja maior que o índice i encontrado no passo 2, ou até chegar a uma posição de "nodo externo". Este é o ponto de inserção do novo nodo contendo a chave k e o campo de índice igual a i. A inserção propriamente é feita como descrito na Seção 2.2.

É bom notar que, em todos estes três passos, são realizados inspeções de bits. E, além disso, fazem-se duas descidas pela árvore. Para efeito de estruturação do Algoritmo de Sedgewick, considerou-se que cada chave possui um bit 0 virtual na posição 0, ou seja, à esquerda do bit de índice 1.

3.2. Algoritmo de Gonnet

GONNET [1984a, pp. 110-111] formula seu algoritmo de forma recursiva e utiliza os nodos externos explicitamente. Neste caso a medida de espaço é E(N) = 2N - 1, onde N é o número de chaves armazenadas na árvore.

A árvore é inicializada vazia. Em seguida, qualquer chave k é inserida, partindo da raiz, segundo o esquema:

1. Se a sub-árvore em questão é vazia, é criado um novo nodo externo contendo a chave k (isto ocorre somente na inserção da primeira chave) e o algoritmo termina. Caso contrário, é obtida uma chave k' de qualquer nodo externo da sub-árvore (por exemplo, o nodo externo mais à esquerda).
2. Os bits da chave k' obtida no passo 1 são comparados, a partir do primeiro, com os bits correspondentes da chave k até chegar a um índice i cujos bits difiram. Se todos forem iguais, a chave k já se encontra na árvore e o algoritmo termina.
3. Se a sub-árvore em questão for um nodo externo ou o índice da sua raiz for maior que o índice i encontrado no passo 2, inserem-se, neste ponto, um novo nodo interno com índice igual a i e um novo nodo externo contendo a chave k, da forma descrita na Seção 2.2.
4. Caso contrário, executam-se recursivamente os passos 1, 2 e 3 novamente, considerando a sub-árvore esquerda ou a direita, de acordo com o bit da chave k de índice indicado pelo nodo da sub-árvore corrente.

Neste caso desce-se somente uma vez pela árvore. Mas, em toda instância, tem-se que acessar um nodo externo qualquer da sub-árvore corrente (passo 1), havendo inspeções de bits (passo 2), além da inspeção usual que é realizada para descer pela árvore.

3.3. Algoritmo Melhorado de Gonnet

No passo 2 do Algoritmo de Gonnet, observou-se que não seria necessário inciar a comparação dos bits a partir do primeiro, a cada ativação da chamada recursiva. Isso porque os bits já foram comparados até o índice i da chamada recursiva anterior (GONNET [1984b]). Assim sendo, basta iniciar a comparação a partir desse índice na ativação corrente. É digno de nota que, para cada comparação de bits, são necessárias duas inspeções de bits.

3.4. Algoritmo Proposto

O algoritmo proposto também é recursivo e utiliza os nodos externos explicitamente. Entretanto, com pequenas alterações é possível implementá-lo com os nodos externos embutidos. A árvore é inicializada com um nodo-cabeça interno contendo 0 (zero) no campo de índice e o apontador de sub-árvore esquerda nulo. Neste caso a medida de espaço é E(N) = 2N, sendo N o número de chaves da árvore.

Cada chave k é inserida de acordo com os passos abaixo, partindo da raiz:

1. Se a sub-árvore corrente é vazia, então é criado um nodo externo contendo a chave k (isto ocorre somente na inserção da primeira chave) e o algoritmo termina.
2. Se a sub-árvore corrente é simplesmente um nodo externo, os bits da chave k são comparados, a partir do primeiro, com os bits correspondentes da chave k' deste nodo externo até encontrar um índice i cujos bits difiram. (Se todos forem iguais, a chave já se encontra na árvore e o algoritmo termina). Depois são criados um nodo interno e um nodo externo: o primeiro contendo o índice i e o segundo, a chave k. O nodo interno é ligado ao externo pelo apontador de sub-árvore esquerda ou direita, dependendo se o bit de índice i da chave k seja 0 ou 1 respectivamente.
3. Caso contrário, ou seja, se a raiz da sub-árvore corrente for um nodo interno, vai-se para a sub-árvore indicada pelo bit da chave k de índice do nodo corrente, de forma recursiva.
4. O caminho de inserção é percorrido novamente de baixo para cima, subindo com o par de nodos criados no passo 2 até chegar a um nodo interno cujo índice seja menor que o índice determinado também no passo 2. Este é o ponto de inserção e o par de nodos é inserido de acordo com o que foi visto na Seção 2.2.

Este algoritmo utiliza o menor número de operações de inspeção de bits em relação aos demais, por causa do esquema utilizado no passo 4, que evita percorrer a árvore uma segunda vez, aproveitando o mecanismo da recursividade. Da mesma forma que o Algoritmo de Sedgewick, considerou-se que cada chave tem um bit 0 virtual à esquerda do bit de índice 1, para efeito de estruturação do algoritmo. Para maiores detalhes, veja o Algoritmo Proposto descrito em Pascal no Apêndice.

4. RESULTADOS EXPERIMENTAIS

O desempenho dos algoritmos de construção de árvore Patrícia foi determinado por meio do seguinte experimento: foram geradas N chaves distintas e aleatórias, cada uma com 32 bits uniformemente distribuídos. Utilizou-se um gerador de números aleatórios baseado no método de congruência multiplicativo. As chaves foram inseridas uma a uma, em uma árvore inicialmente vazia. Durante as inserções foram feitos os cálculos das medidas de complexidade I(N), C(N) e W(N) definidas na Seção 1. O mesmo conjunto de N chaves foi submetido a todos os quatro algoritmos. O experimento foi repetido um número suficiente de vezes para tamanhos diferentes de N.

As Tabelas 4.1, 4.2, 4.3 e 4.4 apresentam os resultados médios obtidos para árvores contendo diversos números de chaves. Os intervalos de confiança para as médias foram ao nível de 95% de probabilidade.

Tabela 4.1. Algoritmo de Sedgewick

N	I(N)	C(N)	W(N)
50	1025 ± 4	277 ± 2	472 ± 2
100	2451 ± 6	654 ± 3	1143 ± 3
500	16895 ± 13	4430 ± 5	8034 ± 7
1000	37792 ± 22	9863 ± 8	18066 ± 14
5000	235282 ± 78	60883 ± 34	113516 ± 38
10000	510495 ± 119	131724 ± 45	247048 ± 58

Tabela 4.2. Algoritmo de Gonnet

N	I(N)	C(N)	W(N)
50	2049 ± 17	935 ± 9	178 ± 1
100	5507 ± 29	2526 ± 14	455 ± 2
500	47913 ± 89	22241 ± 43	3431 ± 5
1000	116488 ± 180	54314 ± 87	7861 ± 8
5000	861190 ± 689	405153 ± 338	50884 ± 23
10000	1995768 ± 888	941991 ± 432	111786 ± 35

Tabela 4.3. Algoritmo Melhorado de Gonnet

N	I(N)	C(N)	W(N)
50	1088 ± 5	455 ± 2	178 ± 1
100	2673 ± 8	1109 ± 3	455 ± 2
500	19155 ± 17	7862 ± 7	3431 ± 5
1000	43307 ± 30	17723 ± 12	7861 ± 8
5000	274418 ± 90	111767 ± 39	50884 ± 23
10000	598806 ± 149	243510 ± 62	111786 ± 35

Tabela 4.4. Algoritmo Proposto

N	I(N)	C(N)	W(N)
50	799 ± 4	277 ± 2	246 ± 1
100	1898 ± 6	654 ± 3	590 ± 1
500	12965 ± 11	4430 ± 5	4105 ± 3
1000	28934 ± 18	9863 ± 8	9208 ± 6
5000	179400 ± 74	60883 ± 34	57634 ± 17
10000	388711 ±101	131724 ± 45	125263 ± 26

O Algoritmo de Gonnet realiza o menor número W(N) de inspeções de bits para dirigir o caminhamento pela árvore. Isto porque ele faz somente uma descida pela árvore, parando o caminhamento já no ponto de inserção. Porém, a cada nível do caminho, são realizadas comparações de bits como mostram os altos valores de C(N). O Algoritmo Melhorado de Gonnet diminui os valores de C(N), utilizando o mecanismo descrito na Seção 3.3.

O Algoritmo de Sedgewick e o Proposto realizam o mesmo número de C(N) de comparações de bits. Ambos fazem esta operação somente quando chegam a um nodo externo, evitando ter que fazê-lo a cada nível do caminho de descida. O Algoritmo Proposto reduz ainda mais o número total I(N) de inspeções de bits em relação ao Algoritmo de Sedgewick, pois este desce uma segunda vez pela árvore e o Algoritmo Proposto aproveita o caminho já traçado pela descida inicial. Isto é refletido pela redução dos valores de W(N).

Para todos os algoritmos estudados vale a relação:

$$I(N) = 2C(N) + W(N).$$

A operação de inspeção de bits cuja medida é dada por I(N) constitui a operação mais cara dos algoritmos Patricia. Nas aplicações práticas desses algoritmos, a rotina responsável pelas inspeções de bits é que deverá fazer os acessos às chaves, as quais certamente estarão armazenadas em memória secundária. Portanto, o fator predominante de custo será a operação de inspeção de bits. Dos quatro algoritmos estudados o que apresenta o menor I(N) é o Algoritmo Proposto. Daí ser ele o mais eficiente em tempo.

5. CONCLUSÕES

As árvores Patricia são adequadas quando se trabalha com chaves de tamanho muito grande e variável. Chaves desse tipo aparecem em problemas de casamento de cadeias, os quais têm aplicação prática em grandes bancos de dados não estruturados sob a forma de informações textuais.

O número esperado de inspeções de bits para pesquisa em árvores Patricia foi determinado

analiticamente por KNUTH [1973], obtendo valores $O(\log_2 N)$. Por outro lado, o custo de construção de uma árvore com N chaves nunca foi estudado analiticamente. Este trabalho propõe mais um algoritmo de inserção. O algoritmo proposto foi estudado empiricamente junto com outros dois algoritmos conhecidos. Os resultados experimentais mostraram que o algoritmo proposto é o que faz menos inspeções de bits, sendo esta a operação crucial durante a construção de uma árvore Patricia.

6. BIBLIOGRAFIA

ALBUQUERQUE, L. C. A.; GONNET, G. H. e ZIVIANI, N. [1985]. "Recuperação Eficiente de Informação em Bancos de Dados Não Estruturados". Série de Monografias do Departamento de Ciência da Computação, UFMG, Belo Horizonte (a ser publicado).

GONNET, G. H. [1984a]. *Handbook of Algorithms and Data Structures*. Addison-Wesley, Reading, Massachusetts.

GONNET, G. H. [1984b]. Comunicação pessoal.

KNUTH, D. E. [1973]. *The Art of Computer Programming; vol. 3: Sorting and Searching*. Addison-Wesley, Reading, Massachusetts.

MORRISON, D.R. [1968]. "PATRICIA - Pratical Algorithm to Retrieve Information Coded in Alphanumeric". *Journal of the ACM*, 15:4 (October), 514-534.

SEDGEWICK, R. [1983]. *Algorithms*. Addison-Wesley, Reading, Massachusetts.

APÊNDICE

ALGORITMO PROPOSTO

1. Definição da Estrutura de Dados

```
const D = "nº de bits de cada chave";  {depende de ChaveTipo}
type ChaveTipo = "a definir; depende da aplicação";
     IndexAmp = 0..D;
     NodoTipo = (interno, externo);
     Arvore = ↑PatNodo;
     PatNodo = record
                  case nt : NodoTipo of
                  interno :
                     (index : IndexAmp;
                      esq, dir : Arvore);
                  externo :
                     (chave : ChaveTipo)
               end;
     Dib = 0..1;
```

2. Funções e Procedimentos Auxiliares

```
function Bit (i : IndexAmp; k : ChaveTipo) : Dib;
{Retorna o i-ésimo bit, a partir da esquerda, da chave k.
 A implementação abaixo é dada a título de exemplo. Uma
 implementação mais eficiente para uso em programas produto
 deve ser feita em linguagem simbólica (assembler), dependendo
 assim da máquina hospedeira.}
var c, j : integer;
begin
   if i = 0 then Bit := 0 else
   begin
      c := ord(k);
      for j := 1 to D-i do c := c div 2;
      Bit := c mod 2
   end
end {Bit};

function EData (p : Arvore) : boolean;
{Verifica se p↑ é nodo externo.}
begin
   EData := p↑.nt = externo
end {EData};

procedure CrieNodos (k, ka : ChaveTipo; var v : Arvore;
                     var h : boolean);
{Cria um par de nodos: um interno e outro externo. O nodo
 interno conterá o valor do índice do bit em que as chaves
 k e ka diferem. O nodo externo conterá a chave k a ser
 inserida. O nodo interno é ligado ao nodo externo de acordo
 com o bit em que as chaves diferem. O endereço do nodo
 interno retorna em v. h indica se a inserção já foi feita.}
var i : integer; p, q : Arvore;
    b : boolean;
begin {CrieNodos}
   i := 1; b := true;
   while b and (i ≤ D) do
      if Bit(i,k) = Bit(i,ka) then i := i + 1
      else b := false;
   if i > D then
   begin {k já se encontra na árvore}
      h := true; erro
   end else
   begin
      h := false;
      new (p, interno); new (q, externo);
      with p↑ do
      begin nt := interno; index := i;
         if Bit(i,k) = 0 then esq := q
                         else dir := q
      end;
      with q↑ do
      begin nt := externo; chave := k
      end;
      v := p
   end
end {CrieNodos};
```

3. Algoritmo de Pesquisa

```
procedure Pesquise (k : ChaveTipo; t : Arvore);

begin
   if EData(t) then
      if k = t↑.chave then Encontrado(t)
                      else NaoEncontrado(k)
   else
      if Bit (t↑.index, k) = 0 then Pesquise (k, t↑.esq)
                               else Pesquise (k, t↑.dir)
end {Pesquise};
```

4. Algoritmo de Inserção

```
procedure Insira (k : ChaveTipo; var t, u : Arvore;
                  var h : boolean);

{k é a chave a ser inserida;
 t contém o endereço da raiz da sub-árvore corrente; na
   primeira chamada t = raiz;
 u contém o endereço do nodo interno ligado a um nodo externo
   contendo a chave k a ser inserida. Este par de nodos sobe
   pelo caminho de pesquisa até chegar ao ponto de inserção.
 h indica se já foi realizada a inserção ou não do par de
   nodos; se h = true, a inserção já se realizou.}

begin {Insira}
   if t = nil then {insere a primeira chave}
   begin
      h := true;
      new (t, externo);
      with t↑ do
      begin nt := externo; chave := k
      end
   end else
   if EData(t) then CrieNodos (k, t↑.chave, u, h) else
   if Bit (t↑.index, k) = 0 then
      begin
         Insira (k, t↑.esq, u, h);
         if not h then
            if t↑.index < u↑.index then
            begin {insere o par de nodos cujo endereço é u}
               h := true;
               if Bit(i,k) = 0 then u↑.dir := t↑.esq
                               else u↑.esq := t↑.esq;
               t↑.esq := u
            end
      end
   else
      begin
         Insira (k, t↑.dir, u, h);
         if not h then
            if t↑.index < u↑.index then
            begin {insere o par de nodos cujo endereço é u}
               h := true;
               if Bit(i,k) = 0 then u↑.dir := t↑.dir
                               else u↑.esq := t↑.dir;
               t↑.dir := u
            end
      end
end {Insira};
```