Estrutura de dados PATRICIA

Alfredo Gabriel de Sousa Oliveira[1], **Kalil Saldanha Kaliffe**[1], **Marcus Maciel Oliveira**[1]

[1]Instituto de Ciências Exatas e Naturais - Faculdade de Computação
Universidade Federal do Pará (UFPA) - Belém, PA - Brasil

{alfredo.oliveira, kalil.kaliffe, marcus.oliveira}@icen.ufpa.br

Abstract. *This article describes the functionality of the PATRICIA data structure, compressed binary tree for words storage from an alphanumeric system in an efficient way, dealing with relevant issues within a Trie data structure while minimizing the number of comparisons in its operations. This work is made for the discipline of Data Structures II of the Bachelor's Degree in Computer Science, taught by teacher Reginaldo Cordeiro dos Santos Filho.*

Resumo. *Este artigo descreve o funcionamento da estrutura de dados PATRICIA, árvore binária de compressão para armazenamento de palavras de um sistema alfanumérico de forma eficiente, lidando com problemas pertinentes à estrutura de dados Trie enquanto minimizando o número de comparações em suas operações. Este trabalho é feito para a disciplina de Projeto de Algoritmos II do curso de bacharelado da Ciência da Computação, ministrado pelo docente Reginaldo Cordeiro dos Santos Filho.*

1. Introdução

Desde o século XX, a área da computação busca estruturas de dados que consigam encapsular operações e otimizar buscas em uma menor quantidade de tempo possível enquanto evitando utilizar uma quantidade exacerbada de memória. Nesse contexto, várias estruturas de dados foram desenvolvidas, como pilhas, listas encadeadas e árvores; e múltiplas aplicações utilizam essas estruturas, como o encadeamento recursivo de funções, representações de grafos computacionalmente e a estrutura de arquivos de um sistema operacional. Este trabalho objetiva descrever uma das estruturas que permeiam as literaturas no que tange às derivações das árvores, que é a árvore PATRICIA, estrutura especializada da estrutura de dados Trie, que é uma árvore em que nós com apenas um filho são agrupados em um nó pai, além de descrever implementações que utilizam essa estrutura como base.

Nesse sentido, este trabalho será dividido nas seguintes seções: a Seção 2, que descreverá alguns tópicos relacionados à estrutura de dados PATRICIA, considerando que a própria estrutura é uma evolução de determinadas tecnologias; a Seção 3, que descreverá o que é a estrutura de dados PATRICIA, abrangendo as operações fundamentais da estrutura, como inserção, remoção e busca de um elemento, e a complexidade de tempo de cada operação; a Seção 4, que demonstra aplicações reais que utilizam da estrutura de dados PATRICIA como base, descrevendo as principais características dessa estrutura nessas aplicações; e a Seção 5, que enfatiza as considerações finais obtidas no desenvolvimento do trabalho.

2. Tópicos relacionados

2.1. Árvore Binária

Uma árvore binária é um caso particular de uma árvore n-ária em que cada nó contido no grafo possui dois filhos no máximo, sendo estes designado de filho à esquerda e filho à direita, com estrita diferença de ordenação entre ambos. Por exemplo, a Figura 1 mostra duas árvores binárias que aparentam ser equivalentes entre si, apenas invertendo a ordem de representação; contudo, pelo fato de a árvore binária requerer uma diferenciação estrita entre filho à esquerda e filho à direita, são árvores diferentes.

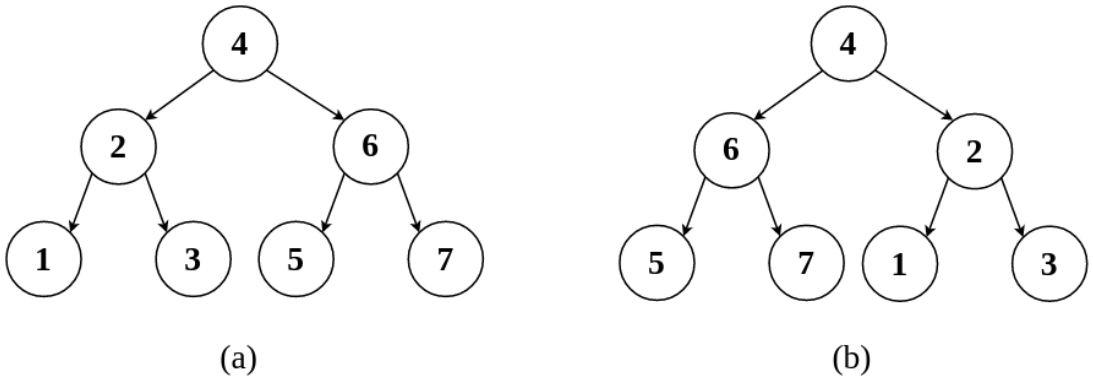

Figura 1. Exemplo de árvores binárias diferentes. (a) Uma árvore binária em que a subárvore do filho esquerdo da raiz possui elementos com valores menores que o mesmo, e a subárvore do filho direito da raiz possui valores maiores que o mesmo. (b) Uma árvore binária em que a subárvore do filho esquerdo da raiz possui elementos com valores maiores que o mesmo, e a subárvore do filho direito da raiz possui valores menores que o mesmo. Fonte: acervo próprio.

2.2. Trie

Uma árvore Trie (Information Re**Trie**val), também conhecida como árvore digital, é uma estrutura de dados especializada para a realização de uma busca digital, isto é, uma busca em uma estrutura heterogênea de dados em que o conteúdo dos nós podem possuir diferentes tipos e tamanhos, possivelmente grandes o suficiente para exceder o limite de tamanho de uma palavra de computador, tornando-se necessário utilizar várias palavras pra representar uma única palavra; premissa diferente do que é considerado no estudo de uma árvore n-ária, em que os nós possuíam o mesmo tipo de dado e o mesmo tamanho de objetos que armazenam. Esta árvore é uma árvore n-ária; caso se deseje utilizar a mesma para armazenamento de strings do alfabeto latino-romano, então é uma árvore 26-ária, ou seja, cada nó pode conter no máximo até 26 filhos, o que permite abranger todas as palavras do alfabeto.

Nessa estrutura, cada nó da árvore possui uma palavra, um identificador de um símbolo terminal e ponteiros para sucessivos nós que utilizam da palavra do nó em questão, com exceção do nó raiz, que possui apenas ponteiros para o início de palavras. A Figura 2 mostra um exemplo de uma árvore Trie 26-ária, em que a palavra do nó em

questão é formada pelos nós anteriores ao mesmo, o identificador de um símbolo terminal é um símbolo #, e os ponteiros para os sucessivos nós são representados por arestas. Por exemplo, para identificar a palavra ”SER”, se passa pelo nó com a palavra ”S”, pelo nó com a palavra ”E”e pelo nó com a palavra ”R”, onde termina com sucesso por possuir um identificador de símbolo terminal, formando a palavra ”SER”.

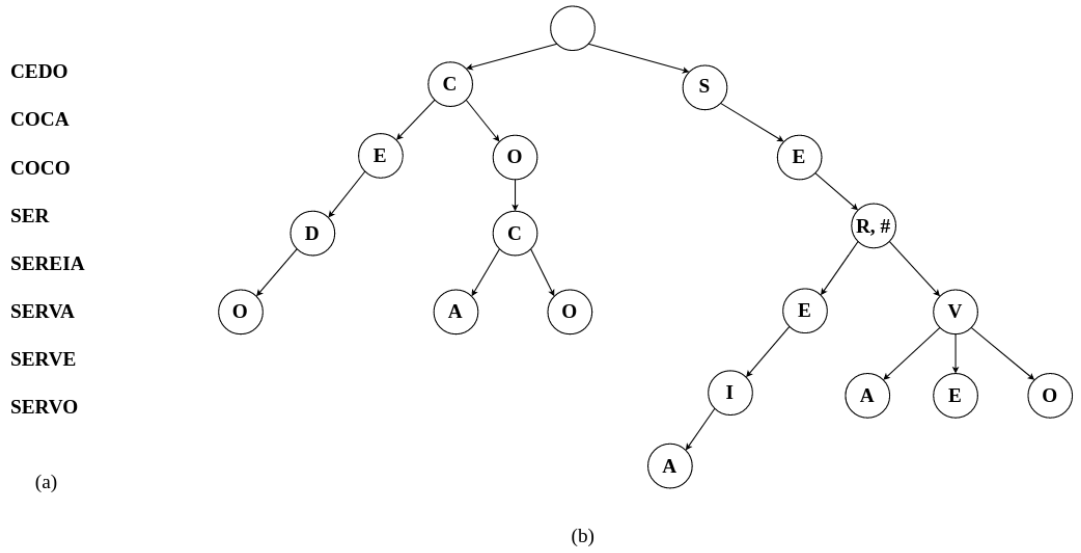

Figura 2. Exemplo de árvore Trie. (a) As palavras que a árvore Trie em questão armazena. (b) Uma árvore Trie. Fonte: acervo próprio.

3. Patricia

PATRICIA (Practical Algorithm To Retrieve Information Coded in Alphanumeric) é um algoritmo proposto por Donald R. Morrison para busca digital, simulando o comportamento de uma biblioteca física no computador, em que as informações binárias do conteúdo da palavra ficam armazenados na aresta entre nós invés de estarem armazenadas em um nó pelo fato de que, em conformidade com o teorema de Euler, o número de arestas em uma árvore binária é igual ao número de vértices menos um, reduzindo a quantidade de memória necessária para armazenamento de conteúdo [Morrison 1968]. Foi consolidado por Donald E. Knuth como uma estrutura eficiente para a pesquisa digital, uma vez que requer um número menor de comparações em suas operações, consome menos memória comparado à estrutura Trie e permite armazenar várias palavras em uma mesma, apesar de ser mais complexa a implementação e o tempo resultante de cada comparação difira do tempo resultante de cada comparação da Trie [Knuth 1998].

Essa estrutura é uma especialização da árvore 2-ária Trie, ou seja, cada nó possui no máximo até dois filhos, e as operações realizadas na estrutura são otimizadas para utilização de um alfabeto binário, como 0’s e 1’s ou palavras em que a posição do caractere possui no máximo 2 caracteres diferentes dependendo do nível da árvore em que está, além da otimização na quantidade de tempo de busca, uma vez que, por ser uma árvore de compressão, a estrutura PATRICIA se livra dos zigue-zagues presentes nas estruturas Trie comprimindo os nós que possuem apenas um único filho em uma única aresta, guardando no nó imediato acima da aresta o número da posição que a aresta em questão deve suceder em relação à parte da palavra que antecede a mesma. A Figura 3 mostra um exemplo de

uma árvore 2-ária Trie em que a estrutura possui um zigue-zague, e a árvore PATRICIA equivalente à árvore Trie. É possível perceber que, enquanto a árvore Trie representada (b) teve de utilizar 9 vértices e 8 arestas para representar as três palavras representadas pelo (a), a árvore PATRICIA representada por (c) necessitou apenas de 6 vértices e 5 arestas, além de não necessitar percorrer vários nós que possuem apenas um caminho para encontrar determinadas palavras. Para valores assintóticos, isso é extremamente relevante, uma vez que as operações na estrutura Trie poderão têr de percorrer todos esses nós, enquanto a estrutura PATRICIA necessitará apenas fazer a comparação entre os bits da aresta em questão, pulando os bits dos nós anteriores pelo fato de que o conteúdo do nó armazena a posição em que se começa a comparação.

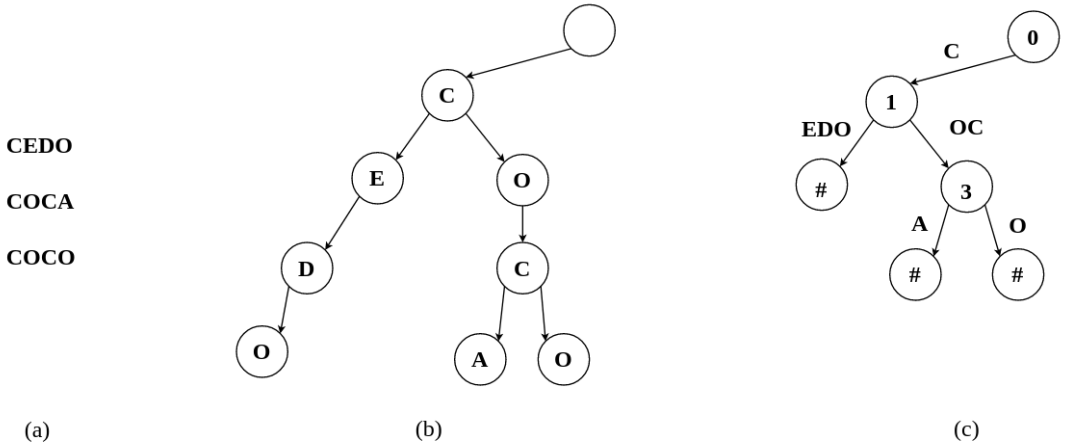

Figura 3. Exemplo de árvore Trie. (a) As palavras que a árvore Trie e PATRICIA em questão armazenam. (b) Uma árvore Trie 2-ária com zigue-zagues. (c) Uma árvore PATRICIA equivalente à árvore (b). Fonte: acervo próprio.

Para efeitos de implementação, é fundamental considerar que, todos os nós internos da estrutura armazenam quatro variáveis: o ponteiro para o filho esquerdo, o ponteiro para o filho direito, uma *flag* para identificar se é um símbolo terminal e um valor inteiro que representa qual posição da palavra a ser inserida, buscada ou removida que deve ser comparada com a palavra presente na árvore. Os nós externos, nesse caso, devem possuir apenas a *flag* para identificar se é um símbolo terminal, uma vez que não possui filhos; e, pois, não necessita armazenar ponteiros para os filhos nem de um valor inteiro da posição à ser comparado. As arestas entre os nós devem possuir o conteúdo comprimido.

3.1. Inserção

Listing 1. Algoritmo de busca em árvore PATRICIA. Fonte: [Szwarcfiter]

```
algoritmo 1: Inclusão em árvore Patrícia
    se ptraiz = λ então
        ocupar(pt)
        pt↑.esq :== pt↑.dir:=λ
        ptraiz := pt
    senão pty' := ptraiz
        buscapat(x, pty', a)
        enquanto ptz↑.esq ≠ λ e descer faça pty':=pty'↑.esq
```

```
l:=0
enquanto l < min{k, c} e d[l+1] = d'[l+1] faça
    l:=l+1
se l ≠ min{k, c} então
    ocupar(ptv); ocupar(ptw)
    ptv↑.r := l+1; ptw↑.r := x
    ptw↑.esq := ptw↑.dir := λ
    ptz := ptraiz; descer := V
    enquanto ptz↑.esq ≠ λ e descer faça
        se ptz↑.r ≤ l + 1 então
            ptq := ptz
            se d[ptz↑.r] = 0 então
                ptz := ptz↑.esq; esquerdo := V
            senão ptz := ptz↑.dir; esquerdo := F
        senão descer := F
    senão d[l+1] = U então
        ptv↑.esq := ptw;
        ptv↑.dir := ptz;
    senão ptv↑.esq := ptz
        ptv↑.dir := ptw
    se ptz = ptraiz então ptraiz := ptv
    senão se esquerdo então
            ptq↑.esq := ptv
        senão ptq↑.dir := ptv
    senão "inclusão inválida"
```

3.2. Remoção

```
funcao inserir
     caso conteúdo do nó pai seja igual a nulo e ambos os nó filhos são
        nó guarda o valor do contador de posição igual à 0
        cria um nó no filho à direita do nó
        crie uma aresta entre o nó e o nó direito
        atribua a palavra a ser inserida na aresta criada
        atribua o símbolo terminal ao nó direito
        pare a execução da função


 caso possua
    para cada letra na palavra do no, compare com cada letra da palavra
caso todas as letras sejam iguais (contador=ponteiro pra length)
    retorne que elemento ja existe
senao
    remova a palavra
    coloque o contador
    coloque o numero de caracteres da palavra
    crie um ponteiro
```

3.3. Busca

Listing 2. Algoritmo de busca em árvore PATRICIA. Fonte: [Szwarcfiter]

```
algoritmo 3: Busca em árvore Patrícia
procedimento buscapat(x, pt, a)
    se pt↑.esq = λ então a:=1
    senão se k < pt↑.r então a:=2
        senão se d[ptup.r] = 0 então
            pt:=pt↑.esq
            buscapat(x, pt, a)
        senão pt:=pt↑.dir
            buscapat(x, pt, a)
```

4. Implementação

4.1. Correspondência de prefixo comum mais longa

Quando dois *hosts* necessitam se comunicar entre si, a camada de transporte estabelece uma conexão fim-a-fim do emissor ao receptor, e a camada de redes seleciona qual algoritmo de roteamento será utilizado pro envio da mensagem. Nesse contexto, quando os pacotes são encaminhados ao roteador, esses dispositivos devem ser capazes de, a partir de uma tabela que contêm os endereços de comutadores interligados, escolher rotas adequadas para envio. Um dos algoritmos criados com tal propósito foi o algoritmo de Correspondência de prefixo comum, isto é, algoritmo que almeja encontrar o prefixo de maior comprimento e de maior valor, fazendo com que as tabelas de roteamento guardem os prefixos (endereços de rede) e o comprimento do prefixo, e comparem o prefixo do pacote com o comprimento do prefixo contido na tabela de roteamento, selecionando como rota o endereço que possui o maior prefixo comum entre ambos mais longo. [ccn]

De acordo com George Vaghese, esse algoritmo foi implementado erroneamente com a estrutura de dados PATRICIA, uma vez que essa estrutura foi designada para buscas de palavras digitalmente, e não para busca de prefixos entre palavras, o que fazia com que, no pior caso do algoritmo, a compressão de dados retornasse um valor errôneo de prefixo entre o valor do pacote e do roteador, se tornando necessário fazer um *backtracking* para um estado em que os prefixos coincidem, o que poderia dobrar a quantidade de acessos à memória. Contudo, foi argumentado também que uma mudança no algoritmo para guardar o valor de bits que foram pulados pela compressão poderia evitar o backtracking, uma vez que somente uma parte dos bits comprimidos por formar zigue-zagues se torna necessária nesse caso, evitando ter de retornar recursivamente à nós anteriores várias vezes e aumentar a probabilidade de a melhor rota ser escolhida. [Varghese 2005]

4.2. Ethereum trie

- State Trie
- Storage Trie
- Transactions Trie

A tecnologia descentralizada Ethereum é uma máquina de estados, que são um par de chave-valor e, em geral, é dividida entre uma camada de consenso e uma camada de execução. A camada de execução dessa tecnologia possui cabeçalhos de bloco que possui o nó-raiz da estrutura de estado, da estrutura de transações e da estrutura de recibos. Essas

três estruturas são implementadas pela estrutura de dados árvore Modified Merkle Patricia Trie, estrutura caracterizada pela mesclagem entre a estrutura PATRICIA e a estrutura árvore Merkle, isto é, uma árvore de hashes em que nós internos armazenam a soma do hashes de nós-filhos e os nós-folhas armazenam dados. Ademais, a estrutura Modified Merkle Patricia Trie inclui características específicas da aplicação do Ethereum. [[kiy]]

A Figura 4

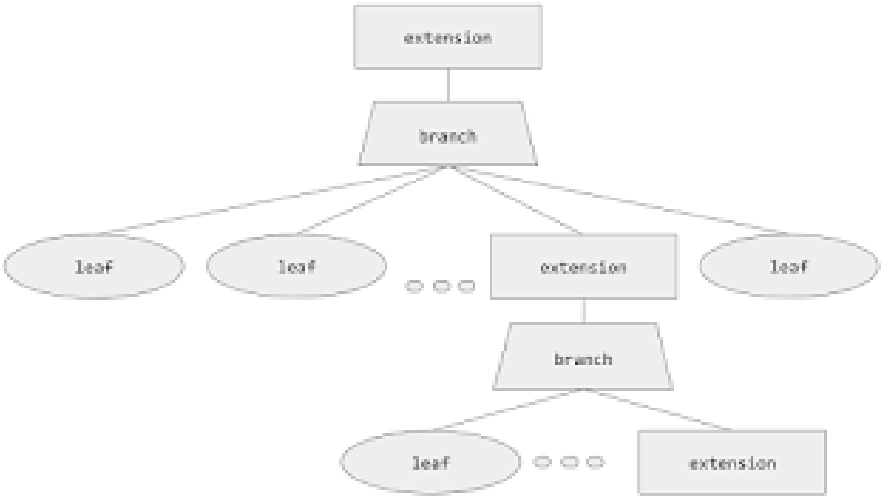

Figura 4. Modelo da árvore Modified Merkle Patricia Trie. Fonte: [kiy]

[fca]

[kiy]

5. Conclusão

Portanto, tendo em vista os fatos apresentados, é possível inferir que a estrutura de dados de árvore PATRICIA é um algoritmo eficiente na complexidade de tempo, considerando que a compressão reduz consideravelmente o número de nós que necessita percorrer em casos de zigue-zagues em árvore e reduz o número de comparações ao guardar o índice de início de comparações, e de espaço, considerando que reduz demasiadamente o número de nós e arestas criadas para armazenamento de uma palavra. Ademais, mediante análise de implementações que utilizam da PATRICIA, foi possível perceber que, quando utilizado fora do contexto de sua criação, problemas de otimização relevantes surgem, apesar que a estrutura é flexível para certas mudanças, e combinado à outras estruturas, como a árvore Merkle, pode produzir aplicações extremamente robustas. A codificação da estrutura, incluindo suas operações, estão disponíveis no repositório @Tsumioshino/patricia$_tree$

Referências

Conceitos de Roteamento. [Acesso em 2022 set 24] Disponível em: https://ccna.network/conceitos-roteamento/.

Modified Merkle Patricia Trie — How Ethereum saves a state. [Acesso em 2022 set 18] Disponível em: https://medium.com/codechain/modified-merkle-patricia-trie-how-ethereum-saves-a-state-e6d7555078dd.

Patricia Merkle Trees. [Acesso em 2022 set 18] Disponível em: https://ethereum.org/en/developers/docs/data-structures-and-encoding/patricia-merkle-trie/.

Knuth, D. E. (1998). The art of computer programming, 2nd edn. sorting and searching, vol. 3.

Morrison, D. R. (1968). Patricia—practical algorithm to retrieve information coded in alphanumeric. *Journal of the ACM (JACM)*, 15(4):514–534.

Szwarcfiter, J. L. *Estruturas de Dados e seus Algoritmos*, volume 2.

Varghese, G. (2005). *Network Algorithmics: an interdisciplinary approach to designing fast networked devices*. Morgan Kaufmann.

Apêndice