Análise das estruturas de dados verificáveis nas blockchains Ethereum e Neo

Carlos David R. Pasco[1], **Igor M. Coelho**[1]

[1]Instituto de Computação – Universidade Federal Fluminense (UFF)

cpasco@gmail.com, imcoelho@ic.uff.br

Abstract. *Data structures have been one of the main objects of study in computing. Some application scenarios of these structures have requirements related to the security and integrity of their data. Thus, throughout the history of computing, verifiable data structures have been proposed with characteristics that aim to fulfill some security requirements such as immutability and the ability to verify an item of data belonging to these structures. This article analyzes two verifiable data structures, namely Merkle Tree and Merkle PATRICIA Tries, describing their concepts, features and application scenarios in the context of blockchains Ethereum and Neo.*

Resumo. *Estruturas de dados tem sido um dos principais objetos de estudo da computação. Alguns cenários de aplicação dessas estruturas possuem requisitos relacionados à segurança e integridade dos dados nelas contidos. Deste modo, no decorrer da história da computação foram propostas estruturas de dados verificáveis, que apresentam características que visam atender a alguns requisitos de segurança como a garantia de imutabilidade e a verificação de um item de dado pertencente a estas estruturas. Este artigo analisa duas estruturas de dados verificáveis, a saber, a Mekle Tree e a Merkle PATRICIA Trie, descrevendo seus conceitos, características e cenários de aplicação no contexto das blockchains Ethereum e Neo.*

1. Introdução

Estruturas de dados tem sido um dos principais objetos de estudo da computação. Alguns cenários de aplicação dessas estruturas possuem requisitos relacionados à segurança e integridade dos dados nelas contidos. Deste modo, no decorrer da história da computação foram propostas estruturas de dados verificáveis [Eijdenberg et al. 2015], que apresentam características que visam atender a alguns requisitos de segurança como a garantia de imutabilidade e a verificação de um item de dado pertencente a estas estruturas.

Aplicações dessas estruturas vão desde a verificação de logs *append-only* em servidores, transparência de certificados digitais [Laurie 2014], e finalmente, para Blockchain [Nakamoto 2008]. Especialmente no campo das blockchains, devido a sua capacidade inerente de armazenamento de dados em um livro-razão imutável, torna-se fundamental explorar aspectos de verificação rápida e eficiente das informações disponíveis. Neste artigo, são analisadas duas estruturas de dados de natureza verificável: a *Merkle Tree* e a *Merkle PATRICIA Trie* (MPT).

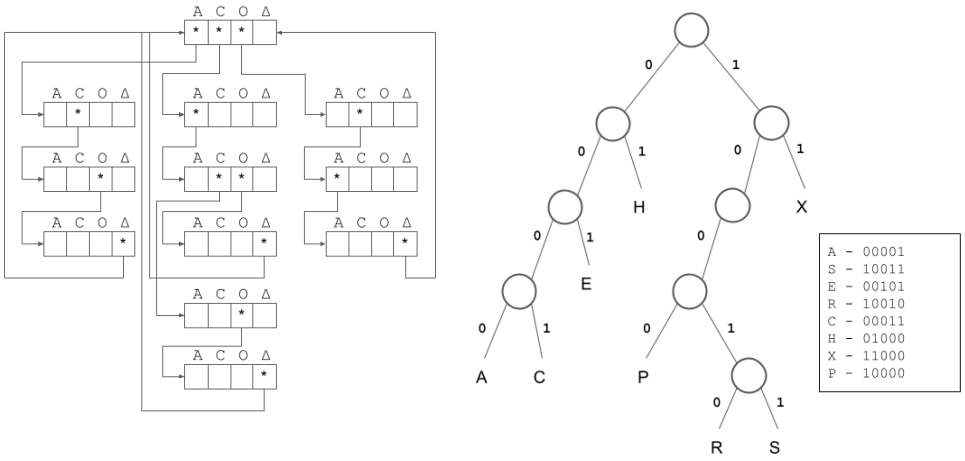
As contribuições deste artigo são: (i) apresentar em detalhes estruturas de dados clássicas para recuperação da informação; (ii) apresentar variações destas estruturas para fins de verificação com foco em blockchain; (iii) explorar aspectos dessas estruturas na implementação de blockchains públicas populares, em especial, Ethereum e Neo.

2. Estruturas de Dados para Recuperação da Informação

2.1. Tries

Uma *Trie*[1] é uma estrutura de dados em árvore otimizada para armazenamento e recuperação de *strings*. Trata-se de uma árvore *M-ária*, cujos nós são vetores de *M* posições, com cada posição correspondendo a um dígito ou caracter. Além disso, cada vetor contém uma posição adicional para representar o fim de uma chave (simbolizado por Δ). Cada nó em um nível *l* representa o conjunto de todas as chaves começadas por uma certa sequência de *l* caracteres chamada de *prefixo* [Fredkin 1960].

Diferentemente das árvores binárias de busca, os nós da Trie não armazenam a chave a ele associada. Antes, a posição do nó na Trie define a chave com a qual ele está associado. Isto faz com que o valor de cada chave se distribua na estrutura de dados, o que significa que nem todo nó tem necessariamente um valor associado [Fredkin 1960]. Cada chave é armazenada do nó raiz para o nó folha na Trie, com base em seu prefixo. Todos os prefixos de comprimento 1 são armazenados até o nível 1, todos os prefixos de comprimento 2 são classificados até o nível 2 e assim por diante.



(a) Uma *Trie* com chaves *ACO, CAO, OCA* e *CACO*

(b) Uma Radix Trie

Figure 1. Tries e Radix Tries

Tries permitem que strings sejam recuperadas de forma eficiente e que consultas que envolvam prefixos sejam respondidas rapidamente. No entanto, Tries são especialmente ineficientes em espaço, uma vez que nem todos os ponteiros de um nó são utilizados. Em um sistema de 64 bits, cada ponteiro ocupa cerca de 8 bytes. Em uma Trie que

[1]Pronuncia-se */tri/*, como em *reTRIEval*, de recuperação em inglês, segundo o autor do termo Edward Fredkin, embora muitos pronunciem */trai/* (`https://xlinux.nist.gov/dads/HTML/trie.html`).

suporte 26 caracteres do nosso alfabeto, temos 27 ponteiros, contando o ponteiro de fim da string. Neste caso, cada nó ocupará pelo menos 216 bytes. Em um nó que use apenas um ponteiro, teremos 208 bytes de ponteiros nulos.

2.2. Radix Tries

Radix Trie é uma estrutura de dados que permite a execução de *Busca Radix*[2]: um método de busca que examina as chaves de busca n bits por vez, onde a quantidade n de bits verificados é determinada pela sua base r (*radix*), que é definida por $r = 2^n$. Ao invés de usar comparações entre as chaves em cada passo, a radix trie examina a sequência de bits que define a chave para recuperar o seu valor na trie. Esta forma de comparação tira proveito de recursos de operações binárias nativas da máquina [Sedgewick 1983].

A chave na radix trie não fica armazenada em todos os nós da árvore, mas apenas em seus nós folha. Temos, portanto dois tipos de nós: nós internos que contém apenas o link para outros nós, e nós externos, que contém apenas a chave. Cada chave é armazenada em um nó externo no ramo da árvore definido pelo padrão de bits da chave [Sedgewick 1983].

Se as chaves compartilharem posições adicionais de bits, faz-se necessário adicionar nós externos que não correspondam a nenhuma chave existente na árvore. A Figura 1(b) mostra um exemplo de uma radix trie.

Uma característica indesejável das radix tries são os "ramos de um só caminho" gerados quando existem chaves com uma grande quantidade de bits em comum. Isto faz com que as Radix Tries apresentem uma alta complexidade de espaço dado o grande número de nós com ponteiros nulos gerados em casos como o descrito acima [Sedgewick 1983].

2.3. PATRICIA Tries

Uma *PATRICIA Tries* é uma Radix Trie com $r = 2$, o que significa que cada bit da chave é comparado individualmente, com cada nó possuindo dois ramos, um esquerdo, representando o bit 0 e outro direito, representando o bit 1 [Morrison 1968].

PATRICIA[3] é um algoritmo proposto por Donald R. Morrison em 1968. PATRICIA Trie é uma Trie resultante da aplicação deste algoritmo [Morrison 1968]. Embora PATRICIA Tries conceitualmente se refiram a Radix Tries com $r = 2$, na prática, observa-se que o termo é utilizado para referenciar Radix Tries com $r \geq 2$ que utilizem um algoritmo de armazenamento e recuperação semelhantes ao do artigo original.

Uma das formas de se aumentar a eficiência de espaço das Radix Tries é abreviando os "ramos de um caminho só". PATRICIA Tries fazem isso agregando a parte da chave de cada nó filho único ao seu pai. Isto é possível pelo fato de que cada nó contém o índice do bit a ser testado. Este índice é incrementado até que haja chaves com um bit divergente naquela posição. Deste modo, não há nenhum nó interno com um único filho, reduzindo o espaço utilizado pela estrutura de dados [Sedgewick 1983].

---

[2]Do inglês *Radix Search*

[3]Acrônimo de *Practical Algorithm To Retrieve Information Coded in Alphanumeric*

3. Árvores Verificáveis para Blockchain

No contexto de recuperação e verificação da informação, estruturas de árvore tem sido propostas e adaptadas com foco em verificação através de hashes criptográficos seguros [Katz and Lindell 2014], especialmente com foco em blockchain.

3.1. Merkle Tree (*Append Only*)

Uma *Merkle Tree*, também chamada de *hash tree* e *binary hash tree*, é uma estrutura de dados em árvore, onde os nós folha contém os hashes de blocos de dados a eles relacionados, e cada nó interno contém o hash do conteúdo de seus nós filhos. Informações são incluídas como um log crescente em modo *Append Only* (sem exclusão). A Merkle Tree foi originalmente proposta em 1979 por Ralph Charles Merkle, a quem deve seu nome [Merkle 1989].

Uma das propriedades da Merkle Tree é possibilitar a autenticação de um item de dado a ela pertencente conhecendo apenas alguns nós da árvore. Tentativas de adulteração do conteúdo da Merkle Tree podem ser rapidamente identificadas, uma vez que qualquer alteração nos dados de um bloco irá alterar o hash do nó folha associado e de todos os nós ascendentes. Qualquer alteração em um dos nós terá como resultado a alteração do hash do nó raiz, também chamado de *Merkle root*. Isto inclui tentativas de inclusão e exclusão de nós na árvore. Além disso, a estrutura de árvore permite a rápida identificação do item alterado [Merkle 1989].

3.2. Merkle PATRICIA Trie

Quando analisada do ponto de vista de eficiência quanto ao armazenamento e recuperação de dados, outras estruturas de dados se mostram mais adequadas que a Merkle Tree.

Conforme visto na Seção 2, PATRICIA Tries são estruturas otimizadas para recuperação e armazenamento de dados, resolvendo o problema de complexidade de espaço apresentada pela Radix Trie. Em [Wood 2014], é proposta a *modified Merkle PATRICIA Trie —MPT—*, uma PATRICIA Trie modificada que incorpora os princípios de verificação de dados presentes na Merkle Tree. De forma similar a uma radix trie, cada caminho do nó raiz ao nó folha corresponde a um único par chave-valor. A chave é verificada durante o percurso do nó raiz à folha, com um nibble sendo obtido a cada *nó ramo* encontrado no caminho, como em uma radix trie de $r = 16$.

Com caminhos compostos por 64 caracteres na MPT, não raro haverá casos onde não haverá divergência em boa parte do caminho. Porém, diferentemente de uma radix trie, quando chaves diferentes compartilham um mesmo prefixo ou quando uma chave tem um sufixo único, dois nós de otimização são fornecidos: os *nós folha* e o *nó de extensão* [Wood 2014]. Deste modo, existem três tipos de nó na trie: *Folha*, que é composto por dois elementos onde o primeiro é a parte restante da chave, que não foi avaliada anteriormente nos nós do percurso, e o segundo é o valor; *Extensão*, que é composto por dois elementos onde o primeiro é uma sequência de nibbles que é compartilhada por mais de uma chave e o segunto é a chave para o nó ramo seguinte; e *Ramo*, que é uma estrutura com 17 elementos, onde os primeiros 16 correspondem a cada um dos 16 valores de nibble possíveis para as chaves neste ponto de seu percurso. O 17.º elemento é usado no caso de ser este um nó final e, desta forma, com a chave terminando o seu percurso neste ponto.

A verificação, a parte "Merkle" da estrutura, é possível devido ao fato de o hash do nó ser usado como sua chave, onde $chave == hash(encoding(valor))$. Além disso, o root hash é obtido de forma semelhante à da Merkle tree, com o hash de um nó sendo calculado por meio do hash dos seus nós filhos.

Se o root hash de uma MPT for publicamente conhecido, um nó da rede que possua toda a MPT pode fornecer uma prova de que a MPT possui um dado valor em um caminho específico, apresentando os nós que compõem o caminho. Isto faz com que seja impossível para um atacante fornecer um par (caminho, valor) que não exista, uma vez que qualquer alteração na MPT irá alterar o root hash [Ethereum Wiki 2021].

3.3. Estruturas Verificáveis no Ethereum

No Ethereum, há quatro cenários onde a MPT é utilizada: estado, armazenamento, transação e recibo.

Uma *trie de estado* armazena o estado global e é atualizada com o passar do tempo. Nesta MPT, o caminho é definido por $sha3(enderecoEthereum)$ e o valor por $rlp(contaEthereum)$. Para cada bloco há uma *trie de transações*. O caminho nesta MPT é definido por $rlp(indiceTransacao)$. O índice da transação é o seu índice dentro do bloco minerado. Após minerado, a trie de transações nunca é alterada [Ethereum Wiki 2021].

De forma a codificar informações de uma transação, o Ethereum cria um recibo contendo informações de sua execução [Wood 2014]. Cada recibo é armazenado em uma *trie de recibos*, que é instanciada para cada bloco. O caminho nesta MPT é dado por $rlp(indiceTransacao)$, onde o índice da transação é o seu índice dentro do bloco minerado [Ethereum Wiki 2021].

Segundo [Wood 2014], das quatro MPTs citadas, o Ethereum utiliza o Merkle root de três delas como parte do cabeçalho de cada bloco: *stateRoot*, *transactionsRoot* e *receiptsRoot*. Devido ao Proof-of-Work competitivo do Ethereum, o estado (*World State*) já é atualizado assim que o bloco é minerado, e colocado no cabeçalho do bloco (faz parte do hash do bloco e é imutável). O fato de ser imutável, pode levar a problemas, como no DAO Hack em 2016, que separou a rede em Ethereum Classic e Ethereum.

3.4. Estruturas Verificáveis no Neo

Estudos das estruturas verificáveis foram feitos na plataforma NeoCompiler Eco [Coelho and Coelho 2021], desenvolvida de acordo com padrões da Neo Blockchain versão 2.x, uma blockchain Turing-completa idealizada em 2015 por Erik Zhang e Da Hongfei. Sua estrutura se assemelha à blockchain Ethereum. Ambas as redes se utilizam do conceito de GAS como forma de pagar pelo gasto computacional das operações com contratos inteligentes, sendo esse GAS disponibilizado gratuitamente para todos os usuários da plataforma de experimentação NeoCompiler Eco.

O Neo Blockchain utiliza uma Merkle Tree para armazenar as transações realizadas. O Merkle root das transações é uma das partes que compõem o cabeçalho de um bloco e é utilizada no cálculo do hash do bloco. A forma como a Merkle tree é utilizada no Neo é similar à utilizada no Bitcoin.

Além da Merkle Tree, o Neo utiliza uma MPT para o armazenamento de dados das transações, equivalente a trie de armazenamento do Ethereum. Sua implementação

segue a mesma estratégia da MPT proposta pelo Ethereum. Porém, diferentemente deste, o Neo não associa o Merkle root desta MPT ao cabeçalho do bloco, o que faz com que alterações nesta trie não influenciem no hash dos blocos. Uma visualização da MPT do NEO pode ser encontrada na documentação da comunidade Neo News Today[4].

4. Conclusão

Foram apresentadas estruturas de dados em árvore, como a Radix Trie e PATRICIA Trie, onde foram brevemente abordadas suas características, potenciais e limitações. Além disso, foi apresentada a Merkle Tree, as propriedades que a tornam uma estrutura de dados simples e eficientemente verificável, e algumas de suas aplicações. Todas estas estruturas de dados foram fundamentais para a concepção da Merkle PATRICIA Trie.

Embora tanto a Merkle Tree quanto a Merkle PATRICIA Trie sejam estruturas de dados verificáveis, a Merkle Tree teve sua gênese orientada à verificação do dado por ela referenciado, enquanto que a Merkle PATRICIA Trie buscou atender não somente aos requisitos de segurança como também aos requisitos relacionados à eficiência de espaço e tempo no armazenamento e recuperação de dados.

References

Coelho, I. M. and Coelho, V. N. (2021). Neocompiler eco: experimentação de consenso em blockchain e contratos inteligentes. In *Anais do VI Workshop do testbed FIBRE*, pages 57–67. SBC.

Eijdenberg, A., Laurie, B., and Cutter, A. (2015). Verifiable data structures. Google Research, Tech. Rep, `https://continusec.com/static/VerifiableDataStructures.pdf`.

Ethereum Wiki (2021). Patricia tree. `https://eth.wiki/fundamentals/patricia-tree`. [Online; acessado em 12/06/2021].

Fredkin, E. (1960). Trie memory. *Communications ACM*, 3(9):490–499.

Katz, J. and Lindell, Y. (2014). *Introduction to Modern Cryptography, Second Edition*. Chapman & Hall/CRC, 2nd edition.

Laurie, B. (2014). Certificate transparency: Public, verifiable, append-only logs. *Queue*, 12(8):10–19.

Merkle, R. C. (1989). A certified digital signature. In *Conference on the Theory and Application of Cryptology*, pages 218–238. Springer.

Morrison, D. (1968). PATRICIA - Practical Algorithm to Retrieve Information Coded in Alphanumeric. *Journal of the ACM*, 15(4):514–534.

Nakamoto, S. (2008). Bitcoin: A peer-to-peer electronic cash system. Tech. Report, 31st October 2008, `https://bitcoin.org/bitcoin.pdf`.

Sedgewick, R. (1983). *Algorithms*. Addison-Wesley, 1st edition.

Wood, G. (2014). Ethereum: A secure decentralised generalised transaction ledger. Technical report, Solidity.

[4]https://neonewstoday.com/development/road-to-neo3-merkle-patricia-trie-implementation/