FACULDADE DE EDUCAÇÃO TECNOLÓGICA DO ESTADO DO RIO DE JANEIRO
ANÁLISE E DESENVOLVIMENTO DE SISTEMAS

SILVIA SOARES DE OLIVEIRA
MARCELO MASCARENHAS
FELIPE FERREIRA DE MENEZES
GEKYUME SERNA

ÁRVORES BINÁRIAS:
ÁRVORE MERKLE

RIO DE JANEIRO
2025

SILVIA SOARES DE OLIVEIRA
MARCELO MASCARENHAS
FELIPE FERREIRA DE MENEZES
GEKYUME SERNA

ÁRVORES BINÁRIAS:
ÁRVORE MERKLE

Pesquisa sobre a estrutura da árvore Merkle como avaliação da disciplina apresentada na Faculdade de Educação Tecnológica do Estado do Rio de Janeiro.

Professora: Cláudia Ferlin

RIO DE JANEIRO
2025

SUMÁRIO

1. INTRODUÇÃO

A árvore merkle (também conhecida como árvore hash) é uma estrutura de dados abstrata idealizada e desenvolvida pelo cientista da computação Ralph C. Merkle, sendo fundamental nas áreas de *Blockchain* e Sistemas Distribuídos. A árvore se difere de outras pois cada nó folha contém um hash criptográfico, isso é, uma sequência alfanumérica com tamanho fixo, representando um dado qualquer, enquanto os pais possuem um hash construído através da concatenação do hash de seus filhos, com a raiz contendo um hash concatenado de todos os nós da árvore. Dessa forma a estrutura permite a verificação eficiente de um grande volume de dados, pois para provar que um dado pertence ao conjunto, apenas é preciso comparar os hashes no percurso entre a folha e a raiz utilizando-se do método Merkle Proof. Portanto, a raiz da árvore age como o autenticador dos dados, garantido que quaisquer mudanças nesses sejam detectadas observando o hash da raiz.

O código implementado pode ser encontrado em: https://onlinegdb.com/yDzQ6e4NX

2. ORIGEM DA ÁRVORE MERKLE

A história da árvore Merkle tem suas raízes na evolução da criptografia moderna e no crescimento das redes de computadores durante as décadas de 1960 e 1970, período em que pesquisadores buscavam soluções para garantir autenticidade e integridade de dados em ambientes cada vez mais complexos. Foi nesse cenário que Ralph C. Merkle, então um jovem pesquisador, iniciou estudos relacionados à segurança da informação enquanto desenvolvia sua dissertação de mestrado na Universidade da Califórnia em Berkeley. Na época, o campo da criptografia estava passando por transformações significativas e se afastava gradualmente de métodos militares sigilosos para se tornar uma disciplina acadêmica com fundamentos matemáticos mais claros.

Merkle começou investigar formas de estabelecer comunicação segura entre partes que não haviam trocado previamente uma chave secreta. Seu interesse inicial era a criação de mecanismos criptográficos acessíveis e verificáveis, área que posteriormente o tornaria conhecido também pelo conceito de criptografia de chave pública. Durante esse processo, Merkle passou a observar que a verificação da

integridade de grandes quantidades de dados se tornava cada vez mais difícil conforme sistemas distribuídos e redes remotas se tornavam mais comuns. A transmissão de arquivos entre máquinas exigia métodos confiáveis para confirmar que o conteúdo recebido permanecia idêntico ao conteúdo enviado, e os sistemas disponíveis exigiam comparações diretas e completas, que consumiam muito tempo e recursos computacionais.

Enquanto estudava funções hash, Merkle percebeu que elas poderiam ser utilizadas não apenas como ferramentas matemáticas isoladas, mas como componentes estruturais de um sistema organizado hierarquicamente. As funções hash apresentavam características que interessavam especialmente aos pesquisadores de segurança da época. Produziam valores significativamente menores que os dados originais, eram rápidas de calcular e reagiam de maneira extremamente sensível a alterações, o que significava que até mesmo uma mudança mínima em um bloco de dados modificava completamente o hash resultante. Merkle passou a considerar como essas propriedades poderiam ser aplicadas à verificação de grandes coleções de informações sem a necessidade de reprocessar tudo continuamente.

A partir dessa reflexão, Merkle desenvolveu a ideia de organizar hashes de maneira estruturada, formando uma espécie de árvore na qual cada par de valores seria combinado para gerar um novo hash de nível superior. Esse processo continuaria até que um único valor final representasse todo o conjunto. Em 1979, ele formalizou essa proposta em sua tese, descrevendo o funcionamento da estrutura que viria a ser conhecida como árvore Merkle. Em essência, a estrutura permitia confirmar a integridade de um volume inteiro de dados analisando apenas uma pequena porção deles, algo até então inédito em termos de eficiência para sistemas distribuídos.

O desenvolvimento da árvore Merkle ocorreu em um período decisivo para a tecnologia da informação. Pesquisadores de várias partes do mundo exploravam redes experimentais que evoluiriam para a Internet, e a preocupação com a confiabilidade das transmissões era crescente. A circulação de dados entre máquinas diferentes, muitas vezes geograficamente distantes, trazia riscos constantes de corrupção acidental ou adulteração intencional durante o tráfego. A proposta de Merkle forneceu uma ferramenta robusta para enfrentar esses desafios porque qualquer alteração em um único bloco de dados propagava modificações

pelos níveis superiores da árvore, facilitando a identificação rápida de inconsistências.

A recepção ao trabalho de Merkle foi gradual. Embora sua ideia fosse inovadora, muitos sistemas computacionais da época ainda não possuíam escala suficiente para necessitar de estruturas muito sofisticadas de verificação. Entretanto, conforme a Internet se popularizou e sistemas distribuídos começaram a lidar com volumes massivos de dados, pesquisadores passaram a revisitar a proposta original e perceber seu potencial. Projetos como redes peer to peer, sistemas de controle de versões e plataformas de armazenamento remoto adotaram a estrutura para garantir integridade durante transmissões e sincronizações de arquivos.

Com o avanço dos estudos sobre registros distribuídos nas décadas seguintes, especialmente na virada dos anos 2000, o conceito da árvore Merkle adquiriu novos significados. A necessidade de representar grandes grupos de informações em redes descentralizadas sem comprometer a segurança ou o desempenho coincidiu diretamente com a capacidade da estrutura proposta por Merkle. Isso se tornou especialmente evidente quando tecnologias baseadas em blockchain passaram a utilizar a raiz de Merkle como elemento central na verificação de blocos de transações, destacando o valor histórico e a importância do trabalho desenvolvido por Merkle ainda na década de 1970.

3. APLICAÇÕES:

As aplicações da árvore Merkle começaram a se consolidar à medida que sistemas distribuídos se tornavam indispensáveis para o funcionamento de redes modernas. Com o crescimento do volume de dados transmitidos entre máquinas e a necessidade de verificar a integridade de informações em ambientes amplos, diversos pesquisadores e desenvolvedores passaram a adotar estruturas baseadas em funções hash hierarquizadas. A característica fundamental de permitir a verificação rápida e confiável de grandes conjuntos de dados, sem recorrer à análise completa do conteúdo, transformou a árvore Merkle em uma solução versátil e apropriada para vários cenários tecnológicos.

Uma das áreas em que a estrutura encontrou lugar com maior naturalidade foi o armazenamento distribuído. Plataformas que dependiam da fragmentação de arquivos entre diferentes servidores, muitas vezes geograficamente separados,

utilizavam a árvore Merkle para assegurar que cada porção de dados permanecia íntegra mesmo diante de falhas no sistema ou interrupções na transmissão. A raiz de Merkle, ao representar todo o conjunto de dados, oferecia uma maneira eficiente de detectar rapidamente inconsistências ou corrupções, permitindo ao sistema recuperar apenas os blocos alterados em vez de substituir arquivos inteiros.

Além disso, redes peer to peer passaram a adotar a estrutura como forma de validar informações recebidas de diferentes fontes. Quando um arquivo era distribuído entre inúmeros usuários, cada trecho podia ser verificado de maneira independente, reduzindo riscos de adulteração e garantindo que o conteúdo reconstruído fosse idêntico ao original. Essa mesma lógica foi incorporada em sistemas de controle de versões, protocolos de comunicação e ferramentas de sincronização remota, que encontraram na árvore Merkle uma solução ideal para lidar com múltiplas versões de arquivos e com a necessidade de identificar mudanças rapidamente.

Com o tempo, a estrutura se tornou especialmente importante em tecnologias que dependem de segurança criptográfica e descentralização. Em sistemas de blockchain, por exemplo, a organização das transações dentro de blocos utiliza a árvore Merkle para permitir que cada transação seja verificada sem que o usuário tenha de acessar o bloco integral.

3.1. Git

O Git emprega de forma direta a lógica estrutural da árvore de Merkle para organizar e verificar todos os elementos que compõem um repositório. Embora o sistema tenha sido criado décadas após a proposta original de Ralph Merkle, a solução encontrada por Linus Torvalds, criador do Git e do Kernel Linux, para garantir integridade, rastreabilidade e consistência em um ambiente distribuído baseou-se justamente nos mesmos princípios que caracterizam essa árvore. No Git, cada dado armazenado é representado por um objeto cujo identificador é calculado por meio de uma função hash. Esse identificador depende exclusivamente do conteúdo do próprio objeto, o que significa que qualquer alteração, mesmo mínima, resulta em um novo hash completamente diferente.

A estrutura interna do Git é formada por três tipos principais de objetos. Os blobs representam o conteúdo dos arquivos, as árvores representam diretórios contendo referências a vários blobs e outras árvores, e os commits representam

estados completos do repositório em determinado momento. Cada blob possui um hash calculado a partir do conteúdo que armazena. As árvores, por sua vez, combinam os hashes de todos os objetos que contêm, produzindo um novo hash que representa aquele diretório como um todo. Finalmente, os commits armazenam o hash da árvore raiz correspondente àquele estado do projeto, além de metadados como autor, mensagem e hora do registro. Esses valores também são combinados para formar um único hash final do commit.

Essa organização cria uma cadeia hierárquica formada por hashes em diferentes níveis. Quando um arquivo é modificado, o hash do blob muda, o que altera o hash da árvore que contém esse blob, propagando a modificação até alcançar o commit correspondente. Esse comportamento reflete exatamente a lógica da árvore de Merkle descrita em 1979, na qual cada alteração em um nó inferior repercute nos níveis superiores. No Git, essa característica garante que cada commit represente de forma íntegra e verificável o estado completo do repositório naquele momento específico.

A utilização dessa estrutura permite que o Git identifique de maneira extremamente rápida se dois repositórios possuem estados equivalentes. Comparar repositórios distribuídos torna-se simples, pois basta comparar os hashes dos commits mais recentes. Caso esses valores sejam idênticos, o Git sabe que todas as árvores e blobs abaixo deles também são idênticos, reproduzindo o mesmo princípio da raiz de Merkle, que representa todo um conjunto de dados a partir de um único hash final. Essa abordagem tornou possível distribuir repositórios completos entre diversos colaboradores sem comprometer a segurança do histórico.

Além disso, o uso da lógica da árvore Merkle protege o repositório contra adulterações. Como cada objeto é definido estritamente por seu hash, qualquer tentativa de modificação não autorizada em um arquivo ou diretório gera imediatamente novos hashes, quebrando a ligação com os commits anteriores. Essa característica impede a alteração silenciosa do histórico e reforça o papel do Git como um sistema confiável para desenvolvimento colaborativo. Dessa forma, ao estruturar seus dados internos por meio de um encadeamento de funções hash que refletem a lógica da árvore Merkle, o Git incorporou um dos conceitos mais importantes da segurança da informação moderna e o aplicou de maneira prática ao controle de versões distribuído.

3.2. Protocolos P2P e armazenamento distribuído

A utilização de árvores de Merkle também se tornou central na estruturação de protocolos peer-to-peer e sistemas de armazenamento distribuído, especialmente aqueles projetados para operar sem autoridade central e que dependem de transferência fragmentada de dados. O avanço desses modelos ganhou força a partir do início dos anos 2000, quando redes P2P precisavam garantir integridade durante a distribuição descentralizada de arquivos, muitas vezes compartilhados entre milhares de nós heterogêneos. Nesse contexto, as árvores de Merkle forneceram uma solução elegante para o problema de verificar partes individuais de um conteúdo sem a necessidade de baixar o arquivo completo ou confiar no emissor original.

Um dos casos mais emblemáticos dessa aplicação é o BitTorrent, um dos primeiros protocolos P2P amplamente difundidos a incorporar estruturas de hashing hierárquicas. Nas versões modernas do protocolo, cada arquivo é segmentado em blocos e cada bloco tem seu hash calculado. Esses hashes são organizados em uma árvore Merkle que permite que um cliente valide imediatamente qualquer pedaço recebido de um par desconhecido. Isso reduz o risco de corrupção intencional ou acidental, ao mesmo tempo em que otimiza a distribuição, já que o cliente pode baixar partes diferentes do arquivo de múltiplas fontes e verificar cada uma independentemente. A adoção dessa estrutura fortaleceu a resiliência do protocolo e contribuiu para sua longevidade como uma das principais tecnologias P2P ainda em utilização.

Outra tecnologia que utiliza intensivamente árvores de Merkle é o IPFS (InterPlanetary File System), um sistema de armazenamento descentralizado que organiza dados em uma estrutura denominada Merkle-DAG. Nesse modelo, cada arquivo é fragmentado em blocos que recebem identificadores derivados de hashes criptográficos. Esses blocos são conectados por meio de relações de referência que formam um grafo acíclico direcionado com propriedades equivalentes às das árvores de Merkle. Essa arquitetura garante que qualquer modificação em um bloco produza automaticamente um novo identificador, tornando impossível alterar dados sem alterar toda a cadeia de dependências. Isso dá ao IPFS propriedades como versionamento automático, deduplicação de conteúdo e verificação intrínseca de

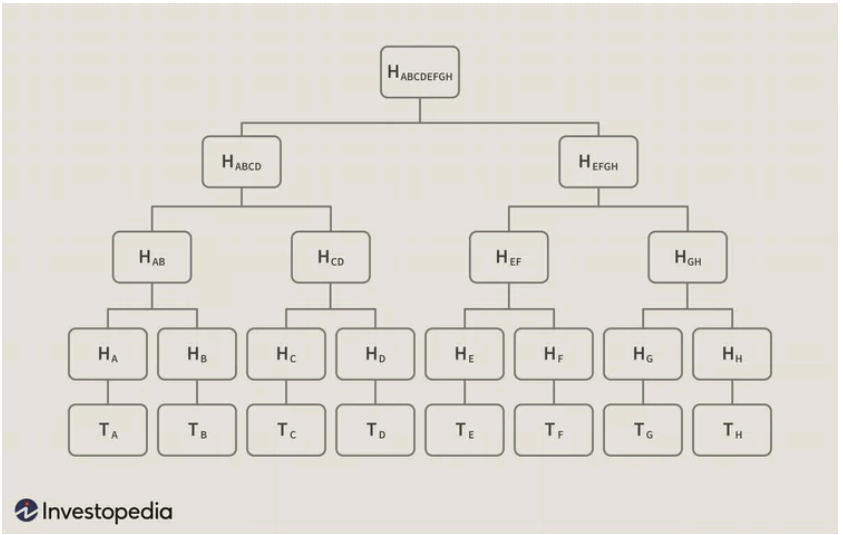
integridade, características fundamentais para aplicações distribuídas e resistentes a falhas.

3.3 Blockchain

A *Blockchain* é uma tecnologia que surgiu como resposta ao problema de transações duplicadas presentes no sistema P2P, permitindo a implementação de sistemas descentralizados a rede sem que haja a necessidade de um terceiro nó de comunicação responsável por validar as ações realizadas, na blockchain cada par consegue realizar as validações por si mesmos.

As *Blockchain* não precisam usar árvores merkle para cumprirem sua função, porém muitas usam por conta de sua eficiência na verificação de dados e a frequência das transações, sendo o Bitcoin a mais famoso entre aquelas que utilizam. Usando a criptomoeda como exemplo: O software criptografa cada transação individualmente ao invés de todas ao mesmo tempo, criando um hash, que irá criar ser atribuído a outro criando assim um par, que criptografado novamente e assim o software irá criptografar as transações até que sobre apenas um único hash, como apresentado na figura abaixo, onde Ts representação transações e Hs representam hashes.

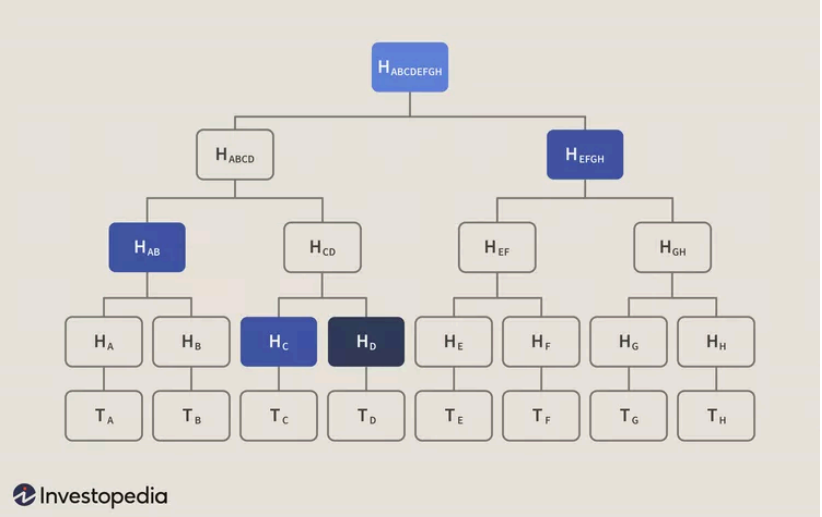
Figura 1 - Entendendo Merkle Tree



Fonte: Investopedia (2025)

Nesse cenário a árvore merkle se prova útil pois permite que usuário verifique a validade de determinada transação sem baixar toda a blockchain, que pode conter até milhares de transações distintas e centenas de gigabytes em tamanho. Tomemos como exemplo a transação T_D, presente na imagem anterior, caso possua o hash da raiz é possível requisitar para o software sua validade de H_D, o sistema então com poucas pesquisas é capaz de confirmar a validade da transação, com os computadores atuais sendo capazes de realizar o processos em milissegundos.

Figura 2 - Entendendo Merkle Tree



Fonte: Investopedia (2025)

3.4. Vantagens e Desvantagens

A árvore Merkle oferece diversas vantagens importantes, sobretudo em sistemas que exigem segurança, integridade e verificação eficiente de dados. Sua principal força está na capacidade de validar grandes volumes de informações utilizando apenas hashes, o que reduz drasticamente o espaço necessário para armazenamento e permite verificações rápidas sem acessar todo o conjunto de dados original. A estrutura hierárquica garante alta confiabilidade: qualquer alteração em um único bloco de dados modifica seu hash e, consequentemente, toda a cadeia acima dele, tornando praticamente impossível adulterar informações sem que isso seja detectado. Isso faz das árvores Merkle uma tecnologia fundamental em blockchains, sistemas distribuídos e protocolos de autenticação.

Por outro lado, a árvore Merkle também apresenta algumas desvantagens. O processo inicial de construção, envolvendo o cálculo de múltiplos hashes, pode ser computacionalmente custoso, especialmente em bases de dados muito grandes. Além disso, qualquer atualização em um ponto da árvore exige recalcular vários hashes ascendentes, o que pode aumentar a complexidade e o tempo de processamento. Há ainda a dependência total de funções de hash seguras: caso a função utilizada seja comprometida, toda a estrutura perde sua confiabilidade. Assim, embora altamente eficiente e segura, a árvore Merkle exige cuidados técnicos e computacionais que podem limitar sua adoção em certos contextos.

4. ESTRUTURA DA ÁRVORE MERKLE

A Árvore Merkle se difere das outras árvores pois seu objetivo não é armazenamento e organização de dados convencionais, sua estrutura é construída a fim de modelar uma criptografia complexa e garantir a integridade dos dados. Sua principal responsabilidade é a validação das informações da árvore de forma rápida e eficiente através da comparação dos hashes de seus nós, operando com uma complexidade O(log n). Cada nó folha da árvore possui um hash que por sua vez representa determinado dado ou bloco de dados, o nós internos(pais), entretanto, são compostos da seguinte forma: o hashes dos filhos é concatenado e então é criado um novo hash a partir dessa concatenação que será atribuído ao pai, caso em algum nível da árvore haja um número ímpar de nós, o último nó ficará sem par e terá de ser duplicado e concatenado com si mesmo para então ser criado o hash, isso é feito para manter a árvore completa durante todo o processo. Esse processo será repetido recursivamente até que seja formado um único hash final, chamado de *Merkle Root,* representando todo o conjunto de dados da árvore. Por padrão a estrutura não utiliza de buffers, fazendo com que qualquer alteração em uma folha resulte em todo o processo sendo refeito.

Como os nós internos não possuem valores reais, apenas hashes que levam até a folha, a árvore merkle não permite o rebalanceamento dos nós, dessa forma, a profundidade da árvore irá depender exclusivamente da quantidade de folhas que esta possui, esta altura é definida por $log2\,n\ +\ 1$, onde log2 n são os níveis acima da folha até a raiz, n é o número de folhas e o ‘+1’ é o próprio nível da folha.

Portanto, uma árvore com 4 folhas teria nível 3(pois log2 4 + 1 = 2 + 1 = 3) mas uma árvore com 8 folhas teria nível 4 (log2 8 +1 = 3 + 1), mesmo dobrando as folhas, a altura só aumentou em 1.

4.1. Inserção

O processo de inserção em uma Árvore Merkle ocorre essencialmente durante a construção da estrutura. Isso significa que, sempre que um novo elemento precisa ser adicionado, a árvore deve ser reconstruída por completo, de modo a preservar a integridade criptográfica. A construção inicia-se pela geração do hash de cada folha, também denominada chunk, a partir dos dados brutos. Esses hashes individuais constituem o nível mais baixo da árvore e servem como base para a posterior combinação hierárquica que formará os nós internos e, ao final, a Merkle Root.

Figura 3 - Merkle Tree, a simple explanation and implementation

Fonte: Medium (2022)

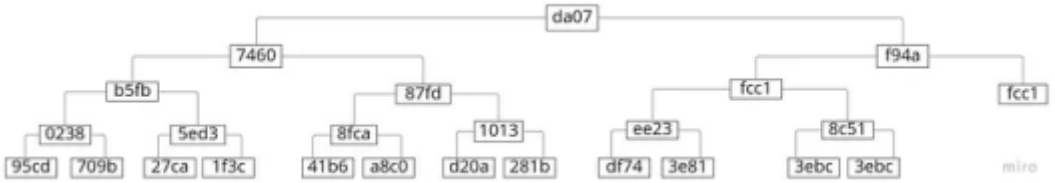
Em seguida, usar os hashes para gerar os pais concatenando os pares de hash folhas e gerando um novo hash para cada pai.

Figura 4 - Merkle Tree, a simple explanation and implementation

Fonte: Medium (2022)

Neste exemplo, há 11 hash folha, então o último gera um pai com uma cópia de si mesmo. Deve-se repetir o processo de gerar os pais até que só reste um hash, que vai ser a raiz da árvore

Figura 5 - Merkle Tree, a simple explanation and implementation



Fonte: Medium (2022)

Além disso, é importante destacar que o processo de construção de uma Árvore Merkle segue uma lógica determinística, garantindo que qualquer alteração mínima em um único dado bruto altere não apenas o hash de sua folha correspondente, mas também todos os nós ao longo do caminho até a raiz. Isso assegura a integridade global da estrutura, pois a Merkle Root funciona como uma representação compacta e imutável de todo o conjunto de dados.

Outro ponto relevante ocorre quando o número de hashes folha é ímpar. Em implementações tradicionais, como no caso do Bitcoin, o último hash é duplicado para formar um par, permitindo que todos os nós internos continuem sendo binários. Esse procedimento mantém a árvore balanceada e simplifica o cálculo dos níveis superiores. A duplicação não compromete a segurança, pois a árvore resultante permanece criptograficamente consistente e qualquer alteração em qualquer folha ainda refletirá na raiz.

A partir desse emparelhamento, inicia-se a geração dos nós internos: cada nó pai é calculado a partir da concatenação dos dois hashes filhos, seguida da aplicação de uma função hash criptográfica. Esse processo é repetido camada por camada, reduzindo gradualmente o número de nós até que reste apenas um a Merkle Root. Esse valor final resume todo o conjunto de dados de forma eficiente, possibilitando verificações rápidas e provas de integridade, mesmo em estruturas com grande volume de informação, como blockchains e sistemas distribuídos.

4.2. Remoção

A remoção de um elemento em uma Árvore Merkle envolve uma sequência de operações que afetam não apenas a folha onde o dado estava armazenado, mas também diversos níveis superiores da estrutura. Isso acontece porque a árvore é organizada de modo que cada nó interno armazena o hash resultante da concatenação dos hashes de seus dois nós filhos. Dessa forma, qualquer

modificação, por menor que seja, em uma das folhas, altera imediatamente todos os nós intermediários até o topo. A remoção tem, portanto, um efeito cascata que percorre vários níveis sucessivos da árvore.

O processo começa no nível das folhas, onde o elemento alvo é localizado. Ao remover essa folha, o hash correspondente deixa de existir e, com isso, o par que ela formava com sua folha irmã torna-se incompleto. Se antes havia dois hashes combinados para formar o hash do nó pai, agora existe apenas um. Esse desbalanceamento exige que a estrutura decida como proceder com o hash remanescente: ele pode ser reagrupado com outra folha, pode ser promovido para ocupar o lugar do nó pai ou pode exigir a compactação do nível como um todo, dependendo da forma como a árvore foi organizada antes da remoção. Como cada nível da árvore depende da existência de pares simétricos de nós, qualquer quebra dessa simetria leva à necessidade de reorganizar as folhas remanescentes.

Após o ajuste inicial nas folhas, o impacto começa a subir camada por camada. O nó pai da folha removida perde a validade, pois seu valor anterior era resultado da combinação dos dois hashes filhos originais. Sem um dos filhos, o cálculo do hash do pai precisa ser refeito. Em muitos casos, esse nó pai deixa até mesmo de fazer sentido estruturalmente, pois pode não ter mais a função de combinar dois elementos. Isso pode levar à eliminação desse nó e à realocação do hash remanescente diretamente para o nível superior, eliminando etapas intermediárias e fazendo com que a árvore se reorganize em uma configuração mais compacta.

Esse fenômeno se repete de maneira progressiva em todos os níveis superiores. Em cada camada, o nó afetado precisa ser reconstruído com base na nova estrutura de seus descendentes. Se a remoção gerou um número ímpar de nós em um nível específico, a árvore deve contornar essa assimetria. Isso pode envolver operações como redistribuição de nós irmãos, reagrupamento de pares, ou eventual eliminação de nós internos que tenham perdido sua função original. A árvore precisa sempre reconstruir as relações que garantem que cada nível seja composto por pares de nós, o que às vezes exige operações adicionais além do simples recálculo de hashes.

Esse processo pode provocar um efeito dominó, no qual a atualização em um único ponto se propaga por vários ramos da estrutura. Por exemplo, a remoção de uma folha pode alterar a quantidade total de nós em uma camada intermediária, o

que exige uma reorganização dessa camada, que por sua vez afeta a formação dos pares superiores, alterando novamente a forma como o próximo nível calcula seus hashes. Dependendo da profundidade e do número de elementos da árvore, esse encadeamento pode resultar em uma reconstrução completa de vários níveis ou até mesmo de quase toda a árvore, exceto pelas folhas que não foram afetadas diretamente.

À medida que a atualização progride para cima, cada nó recalculado produz um novo hash que substitui o valor anterior. Esse recalculo envolve a concatenação dos novos hashes filhos e a aplicação da função hash utilizada na construção da árvore. Caso um nó tenha sido eliminado devido ao desaparecimento de seu par, o nó acima dele também deve ser ajustado, pois agora ele recebe como entrada um conjunto diferente do conjunto original. Essa interdependência torna o processo de remoção complexo e sensível, uma vez que cada mudança precisa refletir exatamente a nova distribuição das folhas restantes.

Em alguns casos, a remoção pode exigir a recomputação de todos os nós de um lado inteiro da árvore, mesmo que apenas uma folha tenha sido removida. Isso ocorre quando a estrutura original era altamente compacta ou quando a remoção altera diretamente uma sequência de pares que dependiam uns dos outros. Em outros cenários, apenas um ramo específico pode ser afetado, mas ainda assim é necessário recalcular os nós desde a base até o topo desse ramo. Em ambos os casos, o que define o custo da remoção é a posição da folha removida e a forma como a árvore se ajusta internamente para manter sua organização.

4.3. Busca

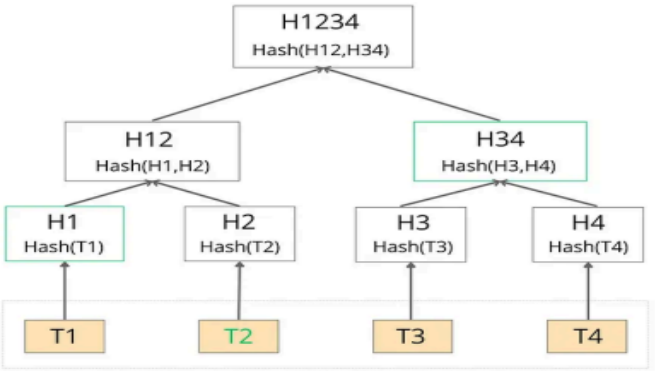
A verificação de pertencimento de um elemento é realizada por meio da técnica denominada Merkle Proof, ou prova de inclusão. Essa técnica permite confirmar se um dado específico integra a estrutura sem a necessidade de acessar todas as folhas, garantindo eficiência e preservando a integridade criptográfica.

O procedimento consiste em iniciar a partir do hash da transação de interesse e, então, reconstruir o caminho até a raiz da árvore. Para isso, utilizam-se os hashes

dos nós irmãos correspondentes, fornecidos como parte da prova. A cada etapa, combina-se o hash atual com o hash do nó irmão indicado, seguindo exatamente a mesma função utilizada na construção original da árvore. Se o hash resultante ao final desse processo coincidir com a Merkle Root conhecida, como aquela registrada no cabeçalho de um bloco, no contexto de blockchain, a presença do dado é confirmada.

Na Figura 6, observa-se um exemplo de Merkle Proof para a transação T2. O hash correspondente é combinado sequencialmente com seus nós irmãos (H1 e H34) para reconstruir a raiz (H1234). Caso esse resultado seja idêntico à Merkle Root armazenada, conclui-se que a transação é válida e realmente faz parte do conjunto representado pela árvore.

Figura 6: Merkle Proof



Fonte: Medium (2024)

5. CONCLUSÃO

A árvore Merkle, ao longo de todo o trabalho, mostrou ser uma estrutura de dados que vai muito além de um conceito estudado em disciplinas de estrutura de dados. Ela representa uma solução prática para um problema real: garantir integridade, verificar dados rapidamente e possibilitar auditorias eficientes mesmo em sistemas enormes. Isso acontece porque sua lógica baseada em hashes encadeados impede que qualquer modificação passe despercebida, já que qualquer alteração, por menor que seja, se propaga até a Merkle Root. Essa característica simples, porém extremamente poderosa, é o que faz essa estrutura continuar sendo usada décadas depois de criada.

Também ficou claro que a estrutura não trouxe impacto apenas no campo

teórico, mas redefiniu como sistemas modernos lidam com confiança em ambientes distribuídos. Quando analisamos suas aplicações, percebemos que ela está presente em tecnologias essenciais do nosso dia a dia digital. No Git, por exemplo, ela mantém o histórico de versões confiável e garante que cada commit possa ser rastreado e verificado. Em blockchains, ela permite validar transações sem carregar a cadeia inteira, tornando redes como Bitcoin e Ethereum muito mais leves e escaláveis. Já nos sistemas P2P e no armazenamento distribuído, como o BitTorrent e o IPFS, ela resolve o desafio de conferir rapidamente se cada pedaço de arquivo transferido não foi corrompido ou adulterado.

Ao observar esses diferentes contextos, fica evidente que a relevância da árvore Merkle não está apenas em sua eficiência, mas na forma como ela possibilita que sistemas desconfiem menos e verifiquem mais. Em um cenário onde dados são constantemente transmitidos, replicados e manipulados, ter uma estrutura capaz de garantir integridade sem exigir processamento exagerado é essencial. Isso explica porque ela foi adotada por tantas tecnologias e continua sendo um elemento central em sistemas distribuídos.

Portanto, ela se destaca por unir simplicidade, velocidade, segurança e aplicabilidade real. Ela se mantém como uma das estruturas de dados mais importantes para sistemas que precisam lidar com grandes volumes de informação, verificações rápidas e ambientes descentralizados. A partir do estudo realizado, fica claro que essa estrutura não apenas facilitou o desenvolvimento de diversas tecnologias modernas, mas também abriu caminho para novas soluções em segurança, verificação e confiabilidade de dados. E justamente por ser tão versátil e eficiente, sua utilização deve continuar crescendo, acompanhando a evolução dos sistemas distribuídos e das novas demandas de integridade no mundo digital.

6. REFERÊNCIAS

DA SILVA RODRIGUES, Carlo Kleber. **Uma análise simples de eficiência e segurança da tecnologia Blockchain.** Revista de Sistemas e Computação-RSC, v. 7, n. 2, 2017.

GOMES, Washington**. O que é uma Árvore Merkle?** Bit2MeAcademy, 2023. Disponível em: https://academy.bit2me.com/pt/que-es-un-arbol-merkle/. Acesso em: 23 nov. 2025.

LUNARDI, Roben Castagna et al**. Introdução a Blockchain: Visão Geral e Conceitos Básicos.**

TEAM, Investopia. **Understanding Merkle Trees: Enhancing Blockchain Efficiency and Security.** Investopia, 2025. Disponível em: https://www.investopedia.com/terms/m/merkle-tree.asp#toc-application-of-merkle-trees-in-blockchain-networksformatica.pt/blockchain-and-merkle-tree/. Acesso em: 25 nov. 2025.

TAVARES, Pedro. **Blockchain and Merkle Tree.** Segurança Informática, 2017. Disponível em: https://seguranca-informatica.pt/blockchain-and-merkle-tree/. Acesso em: 25 nov. 2025.

THEN, Jeremy. **Merkle Tree: A simple explanation and implementation.** Medium, 2021. Disponível em: https://medium.com/coinmonks/merkle-tree-a-simple-explanation-and-implementation-48903442bc08. Acesso em: 23 nov. 2025.

WITTER, Maurício; DE VIT, A. Rodrigo. **Blockchain e Sistemas Distribuídos: conceitos básicos e implicações.** arXiv preprint arXiv:2403.14854, 2024.

YADAV, S. **Merkle proofs explained - Swastika Yadav - Medium.** Disponível em: https://medium.com/@swastika0015/merkle-proofs-explained-208a72971a50. Acesso em: 23 nov. 2025.