Universidade de Brasília

Blockchain B-tree Indexer

Sistema de Indexação Eficiente para Dados de Blockchain

João Lucas Pinto

Luiza de Araújo Nunes Gomes

Matheus Bezerra

Rafael da Silva Oliveira

Introdução

Implementação de um sistema de indexação para dados de blockchain, utilizando B-trees e uma interface em Streamlit. O objetivo é aplicar estruturas de dados de bancos de dados (inspirado em Du et al. (2021) "EtherH: A Hybrid Index to Support Blockchain Data Query", para solucionar o problema de consulta eficiente em tecnologias de registro distribuído

Contexto e Problema

- Blockchains como o Ethereum usam armazenamento sequencial (LevelDB).
- Isso limita consultas complexas e eficientes por valor ou intervalo.
- A falta de índices prejudica aplicações em finanças e supply chain, conforme apontado por EtherH.

Solução Proposta

- Propõe-se um sistema de indexação com B-trees para dados de blockchain.
- O objetivo é otimizar consultas de O(n) para O(logn).
- O sistema permite indexar atributos específicos.

Introdução

A implementação simula um blockchain com todas as suas características fundamentais, combinado com múltiplos índices B-tree, oferecendo uma interface web intuitiva para demonstração das funcionalidades.

Diferenciais

- O projeto usa Streamlit para interface interativa e visualização em tempo real.
- Prioriza uma B-tree pura para simplicidade didática.
- Difere do EtherH, que usa Geth v1.9 e abordagem híbrida.

Objetivos

- Demonstrar a aplicabilidade de B-trees em blockchains.
- Complementar os resultados do EtherH (que indicou consumo de 700 MB para 4M blocos).
- Oferecer uma ferramenta educacional para o estudo de estruturas de dados em contextos descentralizados.

Blockchain: A Arquitetura de Registros Imutáveis

Blockchains são estruturas de dados distribuídas que armazenam uma lista crescente de blocos. Cada bloco, que contém um hash criptográfico do bloco anterior, um timestamp e dados de transação, é encadeado e protegido por criptografia. Essa arquitetura torna os blockchains resistentes a modificações, garantindo um registro de transações imutável e transparente.

Cada bloco contém:

- Hash criptográfico do bloco anterior;
- Timestamp (marca temporal);
- Dados das transações.

Limitação para consultas:

A estrutura linear apresenta complexidade O(n) para buscas, tornando-se um gargalo em blockchains de grande escala.

B-trees: A Solução Clássica para Indexação de Dados

B-trees são estruturas de dados em árvore, auto-balanceadas, que mantêm os dados ordenados e permitem operações em tempo logarítmico.

> **Vantagens para indexação de blockchain:**
>
> - Redução da complexidade de O(n) para O(log n)
> - Suporte nativo a consultas por intervalo (range queries)
> - Estrutura otimizada para sistemas que leem grandes blocos de dados

Para um blockchain com um milhão de transações, a melhoria pode significar uma redução de até 50.000 vezes no número de operações necessárias.

A Arquitetura de Indexação Múltipla

O sistema implementado utiliza uma estratégia de indexação múltipla, mantendo diferentes índices B-tree para diferentes tipos de consulta.

Indice por ID de Transação:
Para buscas exatas e rápidas de uma transação específica.

Indice por Timestamp:
Para consultas temporais e por intervalo de datas.

Indice por Remetente:
Para análises de fluxo de fundos e histórico de transações.

Indice por Destinatário:
Para rastrear os fundos recebidos por um usuário.

Implementação do Blockchain

O sistema é composto por três classes principais:

Transaction:

- Representa uma transação individual
- Contém sender, receiver, amount e timestamp
- Gera transaction_id único usando hash SHA-256

Block:

- Unidade fundamental do blockchain
- Contém index, timestamp, transactions, previous_hash
- Implementa algoritmo de Proof of Work simplificado

Blockchain:

- Gerencia a cadeia completa de blocos
- Adiciona transações pendentes e minera novos blocos
- Valida a integridade da cadeia

Implementação do Blockchain

TRANSACTION

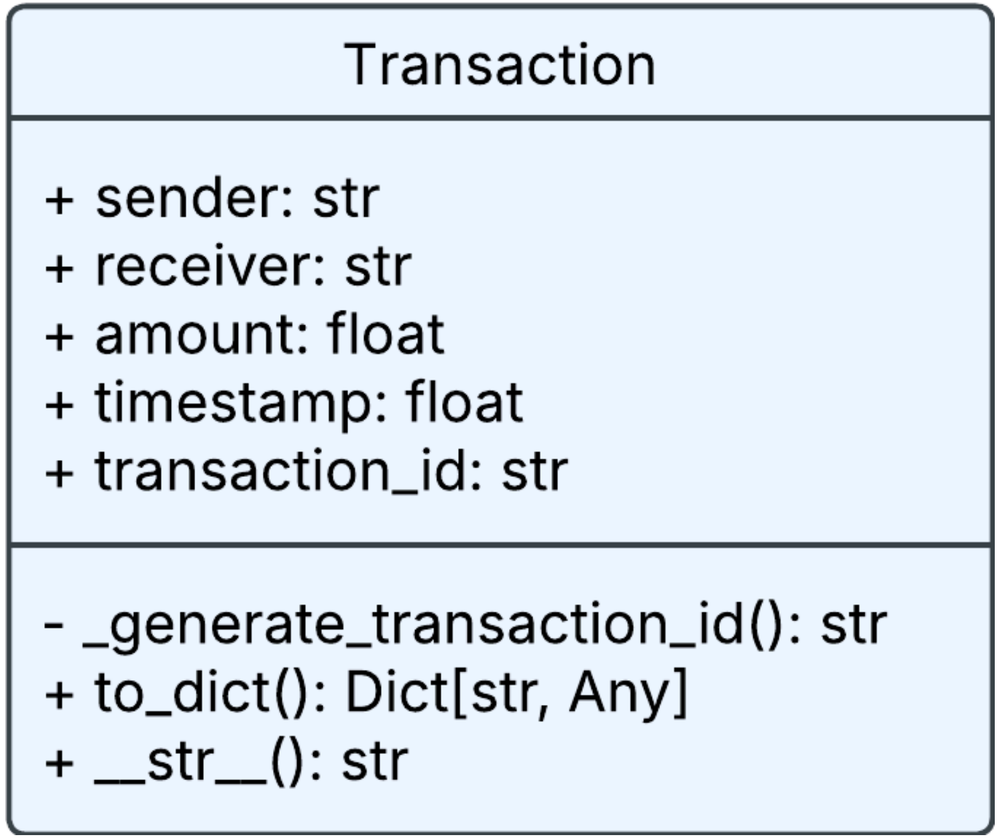
Finalidade: Representar uma transferência de valor entre duas partes

Atributos:

- **sender:** remetente
- **receiver:** destinatário
- **amount:** valor enviado
- **timestamp:** momento da criação
- **transaction_id:** hash único (SHA-256, 16 primeiros caracteres)

Métodos importantes:

- **_generate_transaction_id():** gera ID única de 16 caracteres para fins de simplificação. Utilizando como base:
  - index;
  - timestamp;
  - transactions;
  - previous_hash;
  - nonce



Implementação do Blockchain

BLOCK

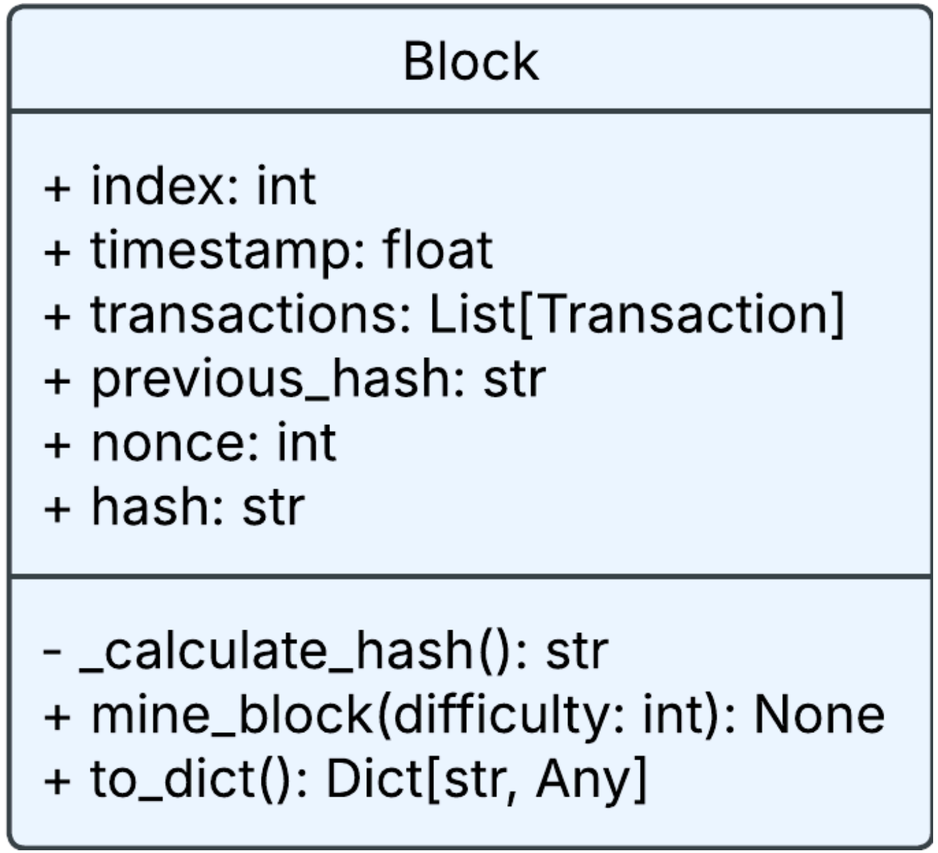
Finalidade: Representar um grupo de transações encadeado à cadeia

Atributos:

- **index:** posição do bloco
- **timestamp:** horário de criação
- **transactions:** lista de transações
- **previous_hash:** hash do bloco anterior
- **nonce:** usado no proof of work
- **hash:** hash do bloco

Métodos importantes:

- **_calculate_hash():** gera hash com base nos dados
- **mine_block(difficulty):** faz proof of work (busca hash com N zeros)



Implementação do Blockchain

BLOCKCHAIN

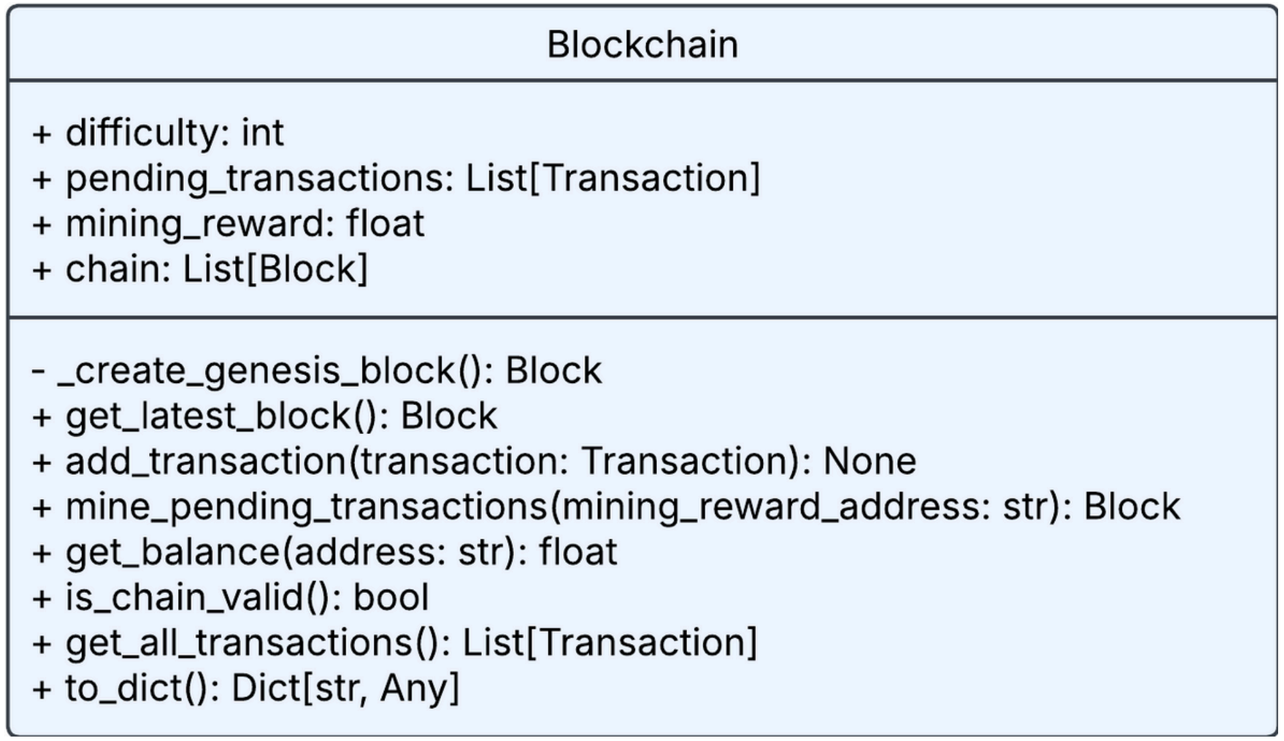
Finalidade: Controlar os blocos, transações, mineração e integridade

Atributos:

- **difficulty:** dificuldade do proof of work (ex: 4 zeros)
- **pending_transactions:** transações a serem mineradas
- **mining_reward:** recompensa ao minerador
- **chain:** lista de blocos encadeados

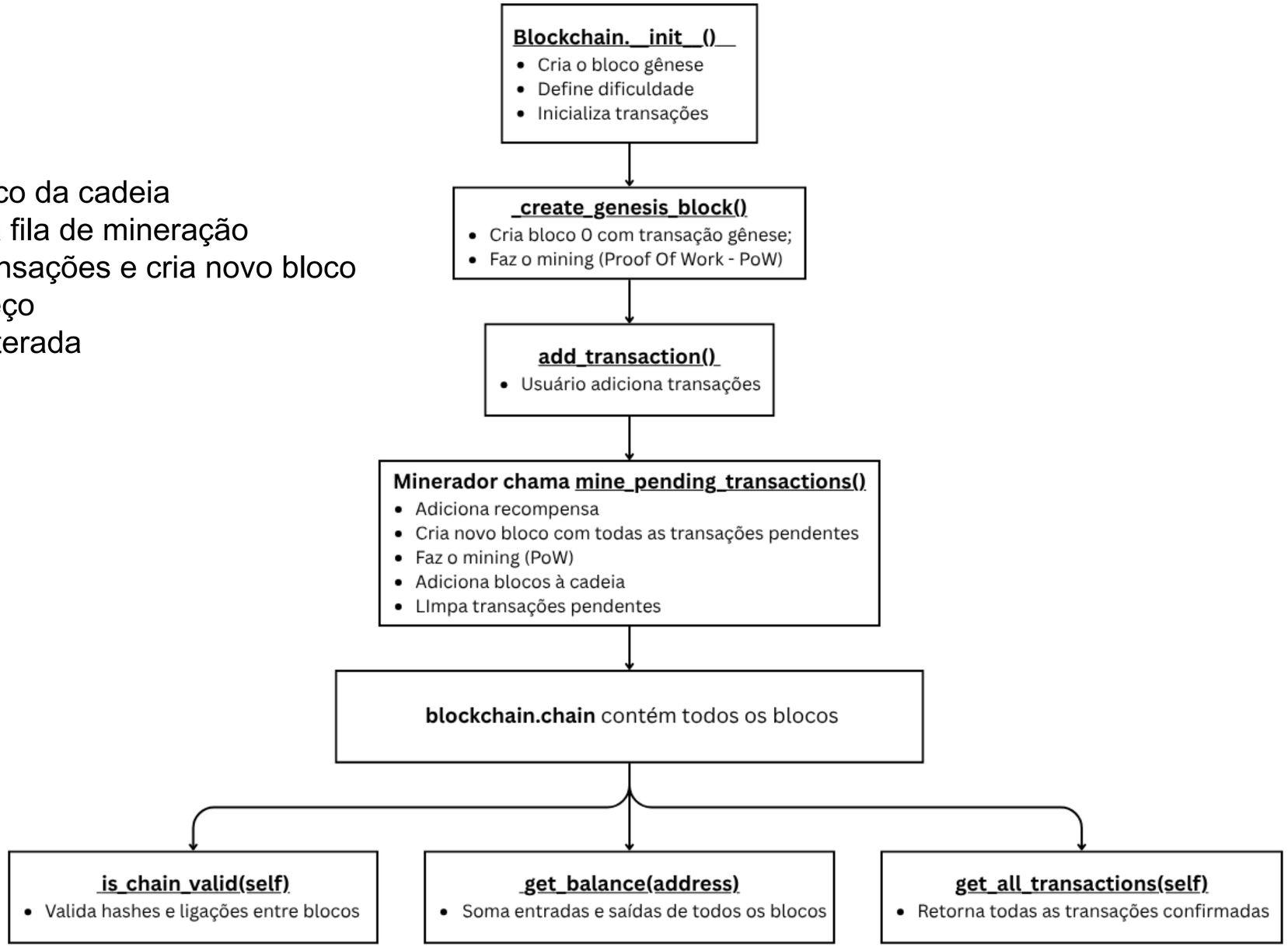
Métodos:

- **_create_genesis_block():** cria o bloco inicial com dados fictícios
- **add_transaction():** adiciona transações pendentes
- **mine_pending_transactions():** cria novo bloco, realiza mineração e adiciona à cadeia
- **get_balance(addr):** retorna o saldo total de um endereço
- **is_chain_valid():** verifica a integridade da blockchain (hashes e ligações)
- **get_all_transactions():** retorna todas as transações da cadeia



Classe Blockchain

__init__ – Inicializa a blockchain
_create_genesis_block() – Cria o primeiro bloco da cadeia
add_transaction() – Adiciona uma transação à fila de mineração
mine_pending_transactions() – Minera as transações e cria novo bloco
get_balance() – Retorna o saldo de um endereço
is_chain_valid() – Verifica se a cadeia foi adulterada



Implementação do Sistema de Indexação B-tree

O módulo **btree.py** contém duas classes principais:

BTreeNode:

- Representa um nó individual na estrutura da B-tree
- Mantém listas de keys, values e children
- Implementa inserção ordenada e operação de split

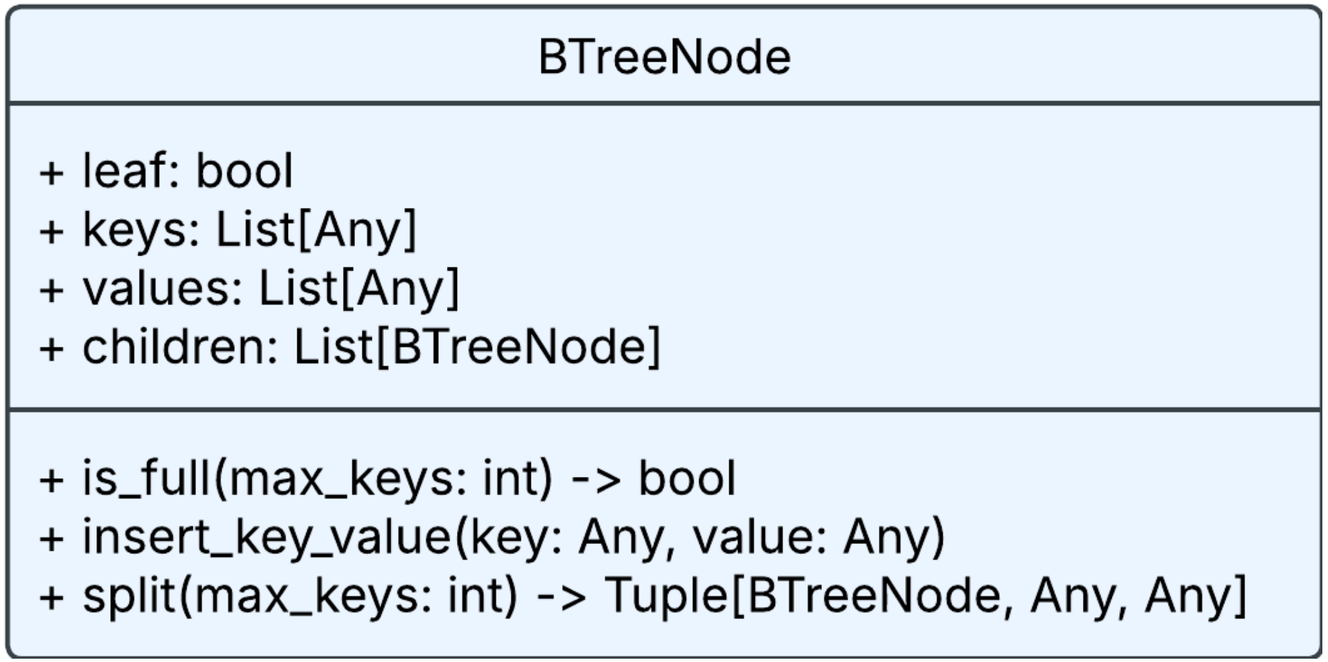
BTree:

- Implementa a lógica completa da B-tree
- Gerencia a raiz e coordena operações
- Suporta chaves duplicadas e consultas por intervalo

Implementação do Sistema de Indexação B-tree

BTreeNode

Finalidade: Representar um nó da B-tree, que armazena chaves, valores e referências para seus filhos.



Atributos:

- **leaf:** indica se o nó é folha (True/False)
- **keys:** lista de chaves armazenadas no nó
- **values:** lista de valores associados às chaves
- **children:** lista de nós filhos (subárvores)

Métodos:

- **__init__(leaf: bool = False):** inicializa um nó, definindo se é folha
- **is_full(max_keys: int):** verifica se o nó está cheio, comparando o número de chaves com o máximo permitido
- **insert_key_value(key: Any, value: Any):** insere uma chave e valor no nó, mantendo a ordem das chaves
- **split(max_keys: int):** divide o nó cheio em dois, retornando o novo nó direito e a chave/valor mediano

Implementação do Sistema de Indexação B-tree

BTree

Finalidade: Implementar uma árvore B para indexação eficiente de dados, suportando inserção, busca e busca por intervalo.

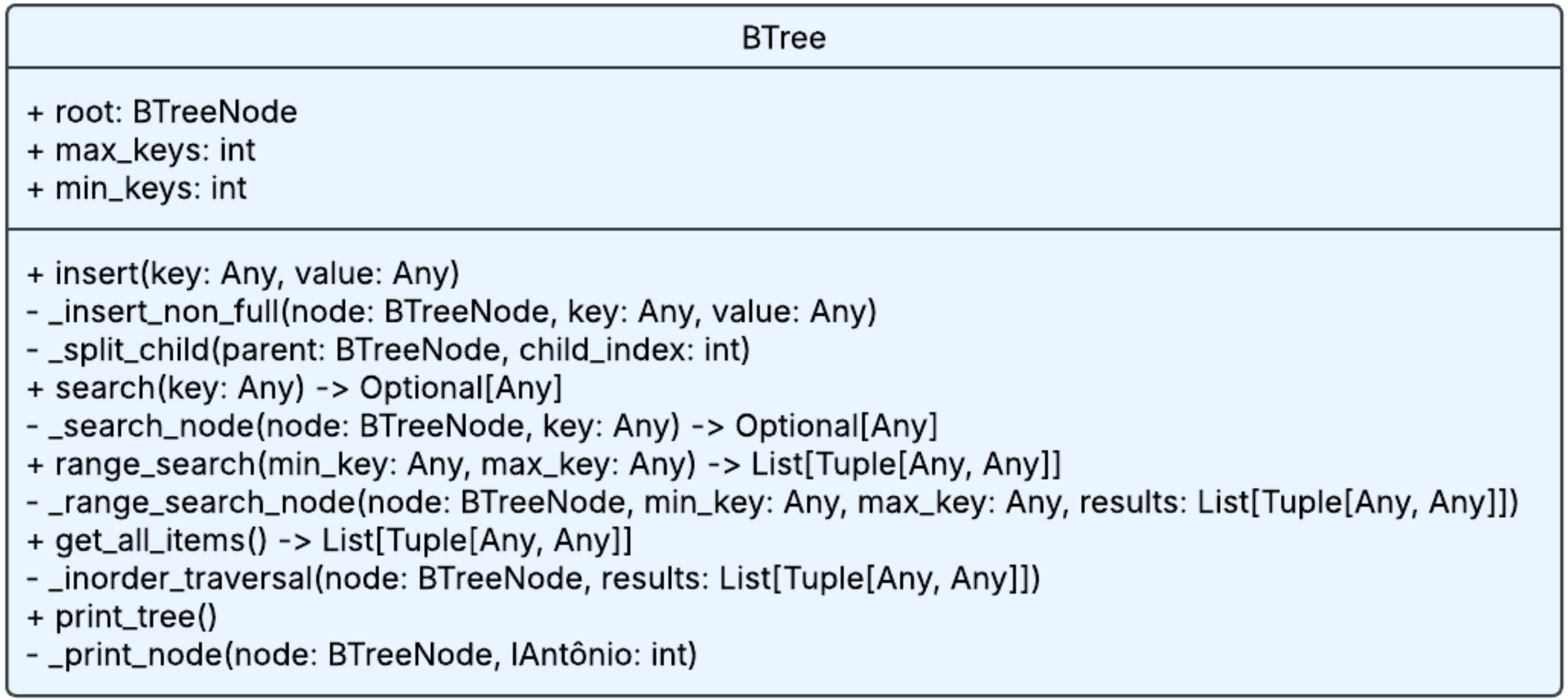
Atributos:

- **root:** nó raiz da árvore (tipo BTreeNode)
- **max_keys:** número máximo de chaves por nó (ordem da árvore)
- **min_keys:** número mínimo de chaves por nó (para balanceamento)

Métodos Principais:

- **insert(key: Any, value: Any)**: insere uma chave e valor na árvore, tratando divisão da raiz se necessário
- **search(key: Any)**: busca uma chave na árvore e retorna seu valor, ou None se não encontrada
- **range_search(min_key: Any, max_key: Any)**: busca todas as chaves e valores dentro de um intervalo
- **get_all_items()**: retorna todos os pares chave-valor da árvore em ordem crescente
- **print_tree():** imprime a estrutura da árvore para fins de depuração

Implementação do Sistema de Indexação B-tree



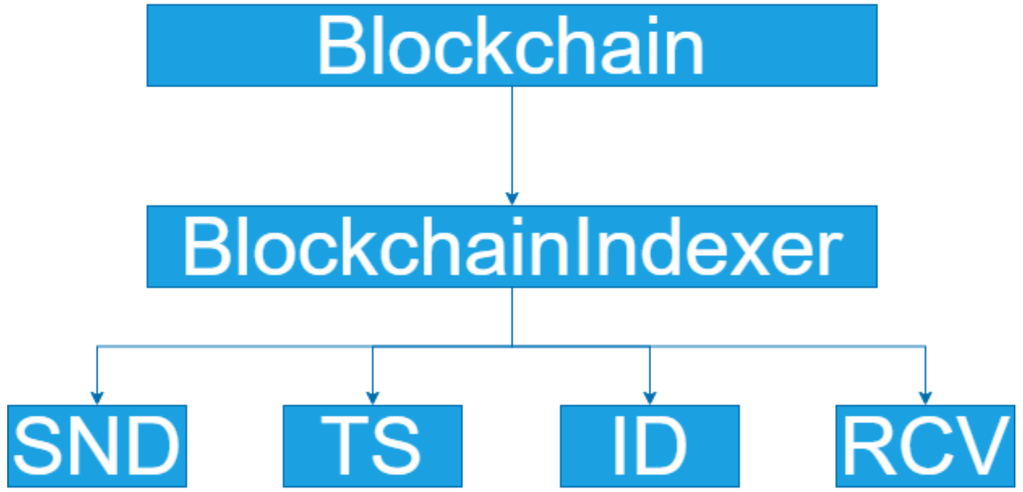
Integração Blockchain-Indexer

A classe **BlockchainIndexer** é o elo crucial que conecta o blockchain com o sistema de indexação B-tree:

Inicialização:
Cria uma instância de Blockchain e inicializa quatro B-trees separadas, cada uma com uma finalidade específica.

Indexação Automática: A função `_index_block(block)` é chamada automaticamente sempre que um novo bloco é minerado e adicionado à cadeia.

Métodos de Consulta:
Expõe métodos de alto nível para a aplicação Streamlit realizar consultas, como:
`get_transaction_by_id`
`get_transactions_by_sender`



Esta integração garante que os índices estejam sempre atualizados e prontos para consultas eficientes.

Integração Blockchain-Indexer

Arquitetura dos Índices

Estratégia de Indexação Múltipla:

- **ID**: Busca exata por *transaction_id*
- **Timestamp**: Consultas por intervalo de tempo
- **Sender**: Análise de fluxo de saída de fundos
- **Receiver**: Rastreamento de recebimentos

Cada índice é uma *B-tree* independente com complexidade O(log n).

Índice	**Descrição**	**Caso de Uso**
ID	Busca única	Verificar transação específica
Timestamp	Intervalo temporal	Relatórios históricos
Sender	Agrupamento por origem	Análise de envio
Receiver	Agrupamento por destino	Análise de recebimento

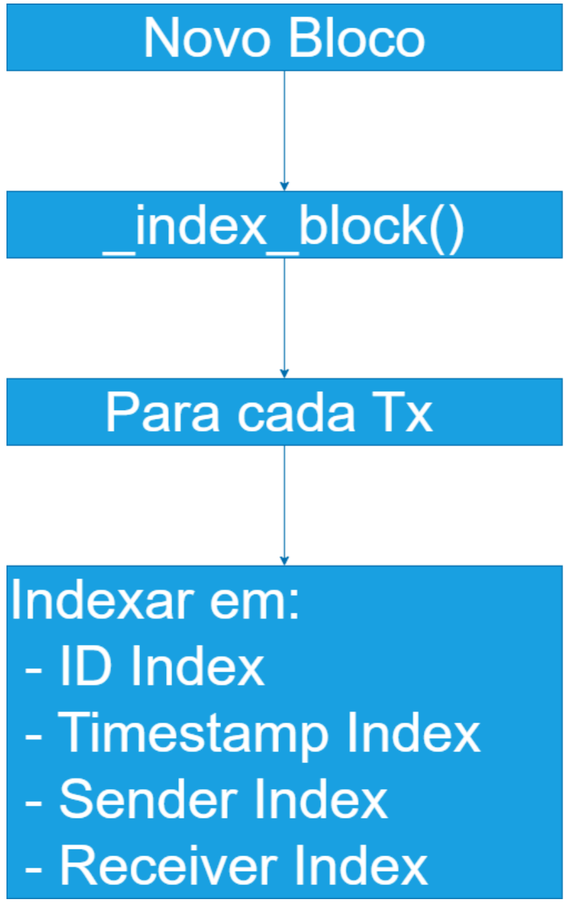
Integração Blockchain-Indexer

Sistema de Indexação de Blocos

Como funciona a indexação:

1. Bloco minerado (**mine_block**)
2. Método **_index_block** percorre cada transação
3. Insere simultaneamente nos 4 índices *B-tree*

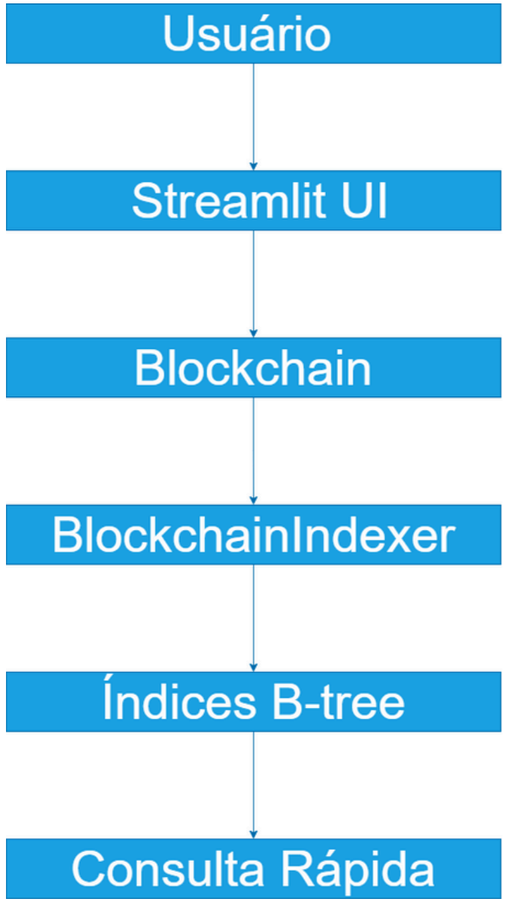
Vantagem: Indexação automática, sem intervenção do usuário.



Integração Blockchain-Indexer

Fluxo Completo de Indexação

Fluxo End-to-End:
O usuário adiciona uma transação pela interface **Streamlit**, minera o bloco, o **BlockchainIndexer** indexa tudo automaticamente, e as consultas são respondidas em O(log n).



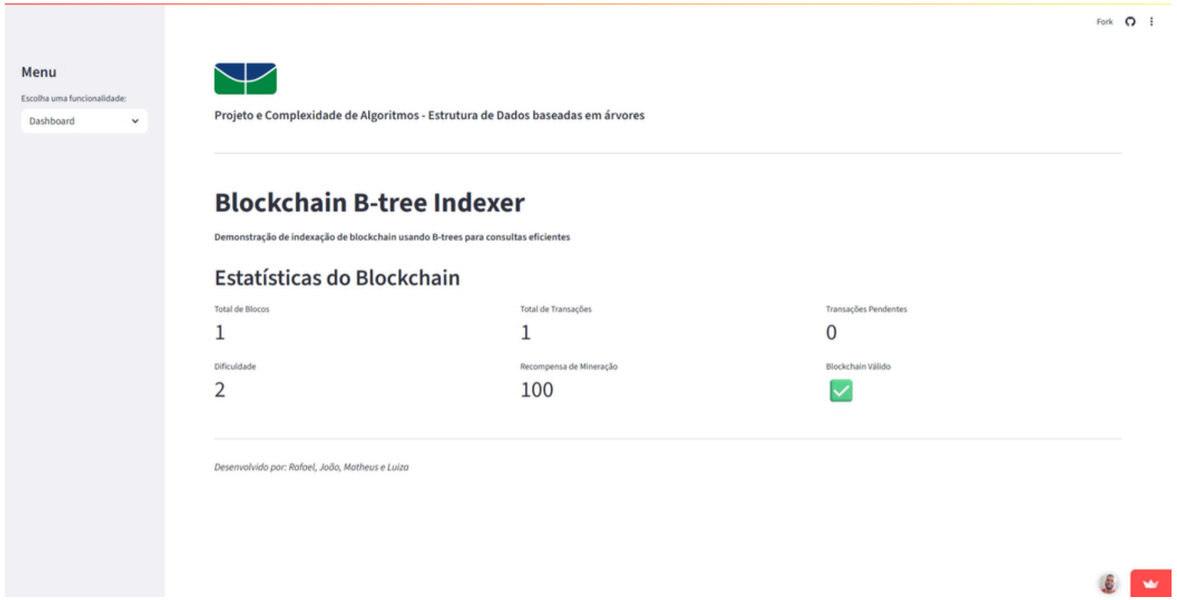
Interface Streamlit

A aplicação **app.py**

É a interface do usuário construída com Streamlit, que permite a interação visual com o blockchain e seus índices B-tree:

As seções de consulta (por ID, remetente, destinatário, período) mostram a eficiência das B-trees na recuperação rápida de dados.

O Streamlit atua como meio pelo qual o usuário pode observar e interagir com a atuação conjunta do blockchain e seus índices B-tree.

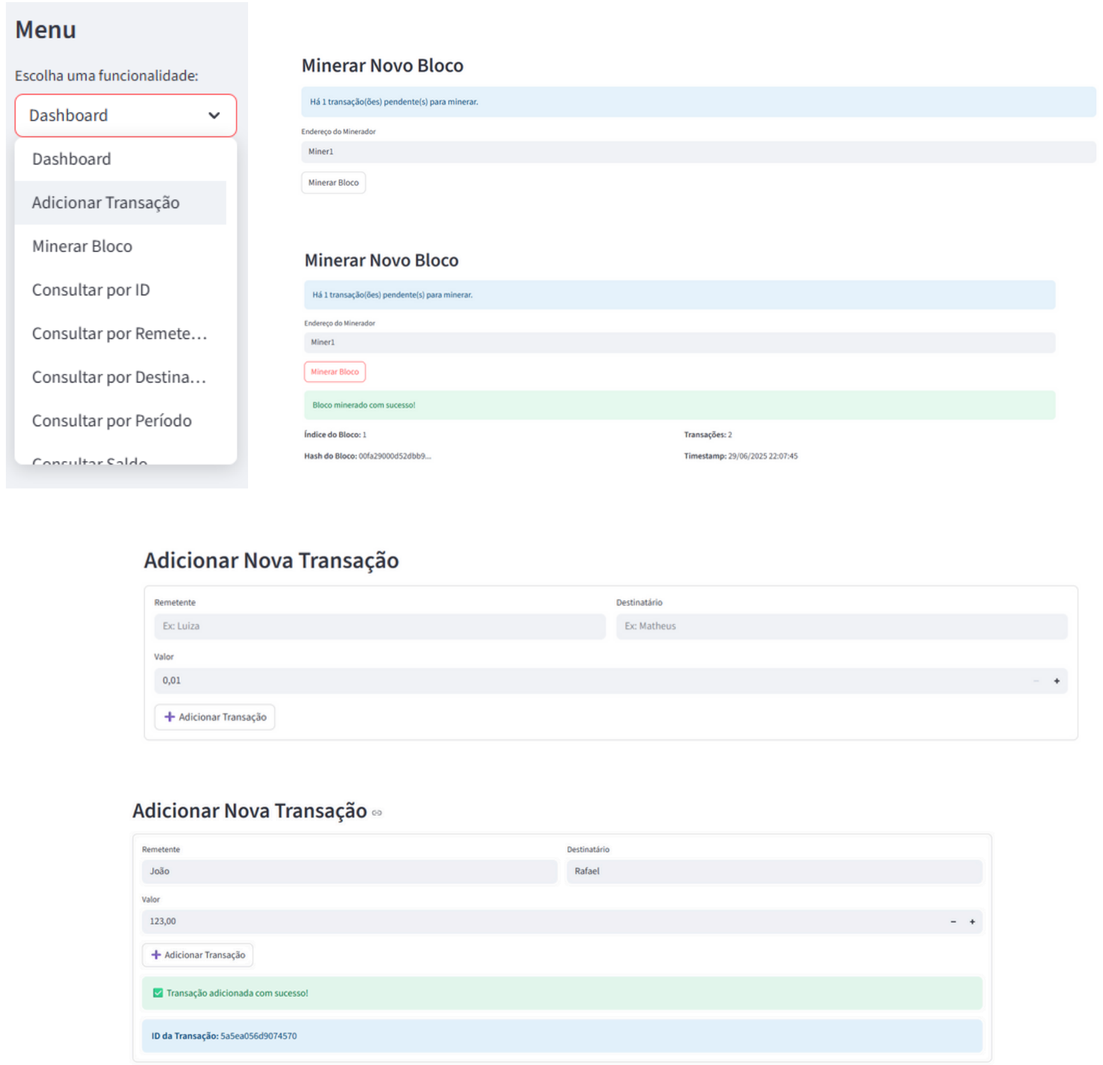


Interface Streamlit

Componentes Interativos

Na Interface (app.py):

- **Menu lateral** (st.sidebar) com páginas: Dashboard, Adicionar, Minerar, Consultar.
- **Formulários dinâmicos** com campos validados (Remetente, Destinatário, Valor).
- **Botões de ação** claros para minerar blocos e consultar.
- Resultados exibidos com **st.expander** para expandir detalhes de forma organizada.



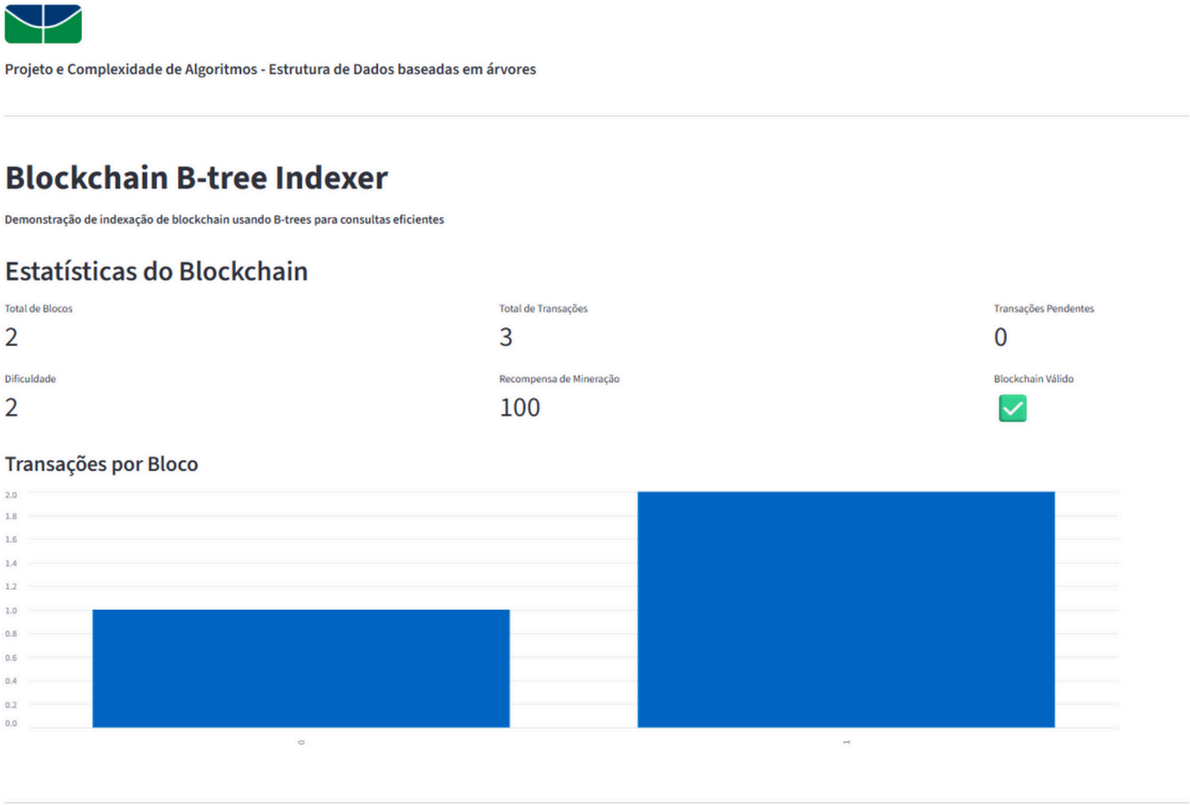
Interface Streamlit

Visualização de Dados

Dashboard ao vivo:

- **st.metric mostra**: Total de Blocos, Transações, Pendências, Dificuldade, Blockchain Válido.
- **st.bar_chart** exibe gráfico de transações por bloco.
- **display_transaction**: formato limpo para cada transação buscada.

Vantagem: Acompanha o funcionamento em tempo real.

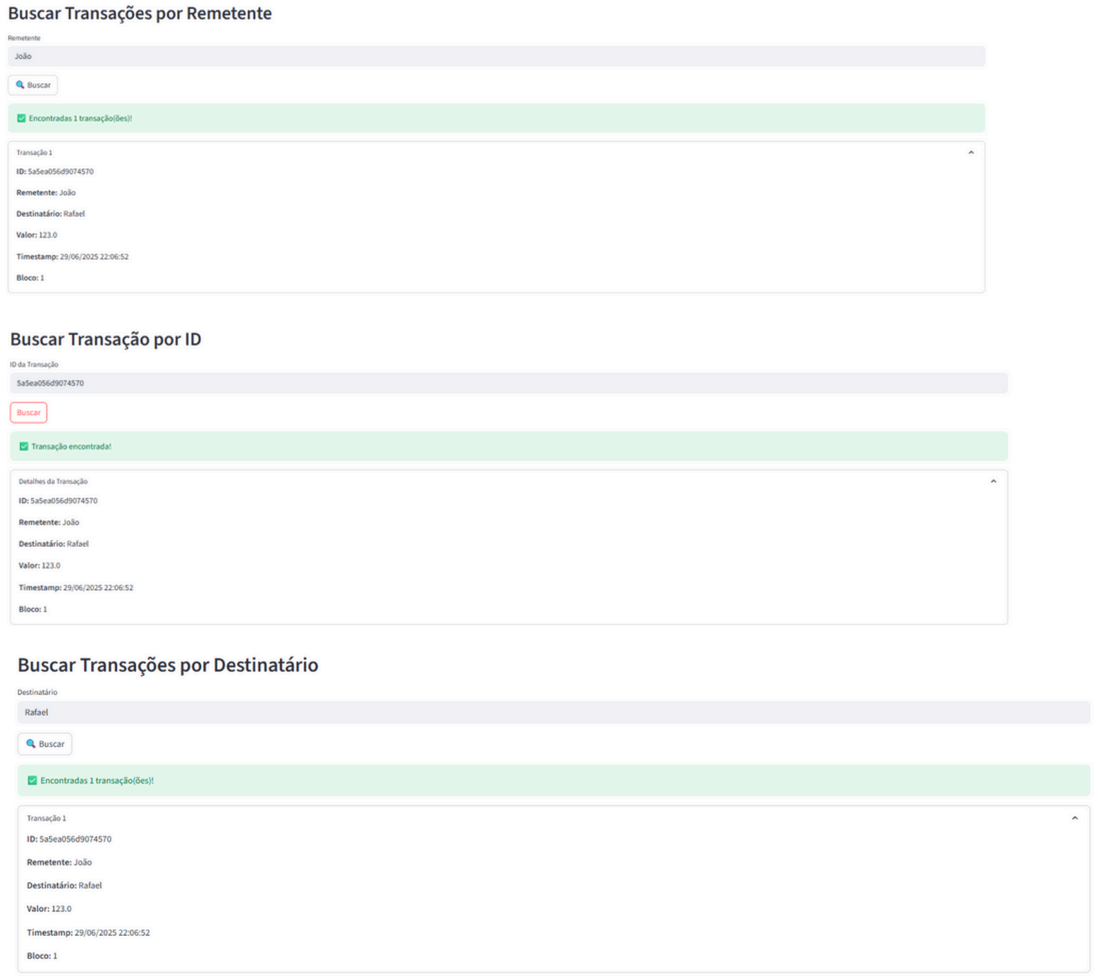


Interface Streamlit

Demonstração de Consultas

Consultas Eficientes:

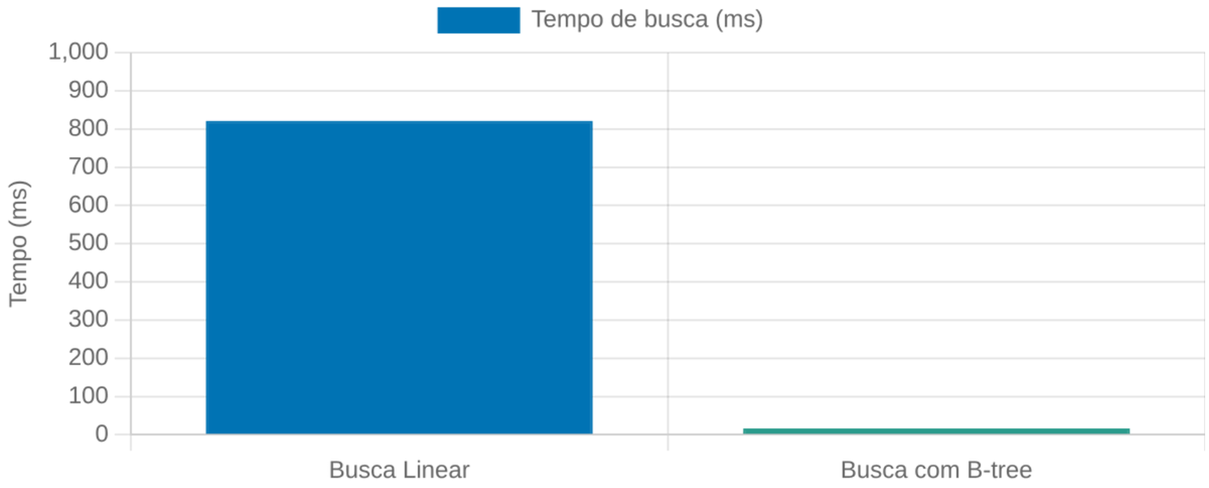
- Por **ID de transação** (get_transaction_by_id)
- Por **Remetente** (get_transactions_by_sender)
- Por **Destinatário** (get_transactions_by_receiver)
- Por **Período** (get_transactions_by_time_range)



Análise de Performance

A implementação de índices B-tree oferece melhorias dramáticas de performance:

Método	Complexidade	Descrição
Sem Índice	O(n)	Varredura linear em toda a blockchain
Com B-tree	O(log n) + O(k)	k = número de transações do remetente



Comparação de tempo de busca (ms) com 1 milhão de transações

Otimizações de Memória:

- Referências aos dados originais em vez de cópias completas
- Cache para nós frequentemente acessados
- Indexação seletiva apenas dos campos relevantes

Casos de Uso e Aplicações Práticas

Auditoria e Compliance Financeiro

Rastreamento rápido de fluxo de fundos e análise de atividades durante períodos específicos. Essencial para investigações financeiras e verificação de origem de transações.

Geração de Relatórios Regulatórios

Organizações financeiras podem gerar rapidamente relatórios sobre atividades específicas, cumprindo requisitos regulatórios sem necessidade de varreduras completas da cadeia.

Análise de Dados e Business Intelligence

Extração de insights valiosos sobre padrões de uso e comportamento da rede. Identificação de tendências, picos de atividade e análise de grafos de transação.

Aplicações de Pagamento e Carteiras Digitais

Recuperação rápida do histórico de transações de um usuário, busca avançada em carteiras e reconciliação automática de saldos e histórico de transações.

O sistema de indexação de blockchain com B-trees abre um leque de possibilidades para aplicações práticas, resolvendo problemas de performance que são inerentes à natureza sequencial dos blockchains.

Conclusão

Principais conclusões do projeto:

- A integração de B-trees com blockchain resolve o problema crítico de consultas eficientes, reduzindo a complexidade de O(n) para O(log n).
- A estratégia de indexação múltipla permite otimizar diferentes tipos de consulta sem comprometer a performance geral.
- O overhead de manutenção dos índices é mínimo comparado aos ganhos de performance nas consultas subsequentes.
- A interface Streamlit demonstra de forma prática como a indexação transforma a experiência do usuário em aplicações blockchain.

Principais Referências

ALEXANDRE, R. **Bitcoin Core Documentation**. 2023. Disponível em: https://bitcoincore.org/en/doc/. Acesso em: 12 jun. 2025.

BAKA, B. **Python Data Structures and Algorithms**. 1. ed. Birmingham: Packt Publishing, 2023. Acesso em: 05 jun. 2025.

BASHIR, I. **Mastering Blockchain: A Deep Dive into Distributed Ledgers**. 3. ed. Birmingham: Packt Publishing, 2022. Acesso em: 18 jun. 2025.

COMER, D. **B-Trees and Relational Database Systems**. Berkeley: University of California Press, 1981. Acesso em: 03 jun. 2025.

CORMEN, T. H. et al. **Introduction to Algorithms.** 4. ed. Cambridge: MIT Press, 2022. Acesso em: 15 jun. 2025.

DU, Pengting et al. **EtherH: A Hybrid Index to Support Blockchain Data Query.** In: ACM TURING CELEBRATION CONFERENCE – CHINA (ACM TURC '21), 30 jul.–1 ago. 2021, Hefei, China. Anais [...]. New York: ACM, 2021. p. 1-5.Disponível em: https://doi.org/10.1145/3472634.3472653. Acesso em: 10 jun. 2025.

ETHEREUM. **Ethereum Yellow Paper**. 2023. Disponível em: https://ethereum.github.io/yellowpaper/paper.pdf. Acesso em: 20 jun. 2025.

GRAEFE, G. **Modern B-Tree Techniques**. Hanover: Now Publishers, 2011. Acesso em: 01 jun. 2025.

Projeto Blockchain B-tree Indexer (Streamlit)