## Árvore B+ em Python

Este projeto apresenta uma implementação detalhada de um sistema fakerational que simula comandos básicos de um terminal Unix, tendo sua implementação feita através de uma Árvore B+ (BPlus Tree) em Python. A estrutura suporta operações de inserção, busca, remoção e consulta por intervalo de forma eficiente, utilizando uma arquitetura de nós internos e nós folha conectados entre si, formando um lista duplamente encadeada neste caso.

## Conceitos-Chave

- **Árvore B +** : estrutura de dados balanceada amplamente usada em sistemas de arquivos e bancos de dados.

- **Nós internos** : armazenam apenas chaves e apontam para outros nós (internos ou folhas).

- **Nós folha** : armazenam as chaves reais e os valores associados, conectados entre si para facilitar buscas por intervalo.

Comandos do terminal :

● ls

Lista os arquivos e diretórios presentes no diretório atual, em ordem lexicográfica.

● cd <nome>

Acessa o diretório <nome>, se ele existir.

● cd ..

Retorna ao diretório pai. Caso esteja na raiz, permanece nela.

● mkdir <nome>

Cria um novo diretório com o nome especificado.

● touch <nome>

Cria um novo arquivo com o nome especificado.

● rm <nome>

Remove um arquivo ou diretório existente (não é necessário implementar remoção recursiva).

## Estrutura das Classes

**LeafNode :**

#Nó folha da árvore, armazena pares chave-valor.

- **is\_cheio()**: Verifica se o nó está cheio com base na ordem da árvore.

- **vazio()**: Retorna se o nó está vazio.

- **proximo/anterior**: Ponteiros para facilitar varredura ordenada entre folhas.

**InternalNode**

#Nó interno da árvore, armazena chaves para navegação e ponteiros para filhos.

- **is\_cheio()**: Verifica se há overflow de chaves.

**BPlusTree**

#Controla a estrutura da árvore. Principais funções:

- **\_\_init\_\_(ordem) :**

#Inicializa a árvore com a ordem especificada.

- **insert(chave, valor) :**

**“””**

Insere um par chave-valor na árvore.

Localiza a folha correta.

Insere de forma ordenada.

Se o nó estiver cheio, divide a folha.

Propaga divisões para o pai se necessário.

**””’**

- **insert\_leaf(leaf, chave, valor) :**

#Insere ordenadamente dentro de uma folha, sobrescrevendo se a chave já existir.

**insert\_pai(node\_atual, chave, novo\_node) :**

**“””**

Trata a inserção de uma nova chave no pai após uma divisão:

Cria nova raiz se necessário.

Propaga recursivamente o split para cima se o pai ficar cheio.

**“””**

**buscar(chave):**

#Navega do nó raiz até a folha correta para a chave.

**buscar\_folha(chave)**

#Busca e retorna o valor associado à chave, se existir.

**remover(chave)**

**“””**

Remove uma chave da árvore:

- Reorganiza a folha.

- Atualiza separadores nos pais.

- Rebalanceia a árvore se necessário com fusão ou redistribuição de nós.

**“””**

**remover\_entrada(node) :**

**“””**

Gerencia a remoção recursiva e rebalanceamento após underflow:

Redistribui chaves com irmãos.

Faz fusão de nós quando necessário.

Pode remover a raiz se ela se tornar redundante.

**“””**

**fixar\_chave\_pai(node):**

#Atualiza as chaves de separação do nó pai se a primeira chave da folha for alterada.

**pegar\_esquerda()` / `pegar\_direita():**

#Redistribuem chaves entre irmãos à esquerda ou à direita durante rebalanceamento.

**alterna\_nodes(esquerda, direita, pai, sep\_index)**

#Funde dois nós irmãos e remove a chave separadora do pai.

**minimo\_chaves() , minimo\_filho\_no() , minimo\_chavesdentro() :**

#Calculam o número mínimo de chaves que um nó pode ter após uma remoção, com base na #ordem da árvore.

**range\_query(comeco, final) :**

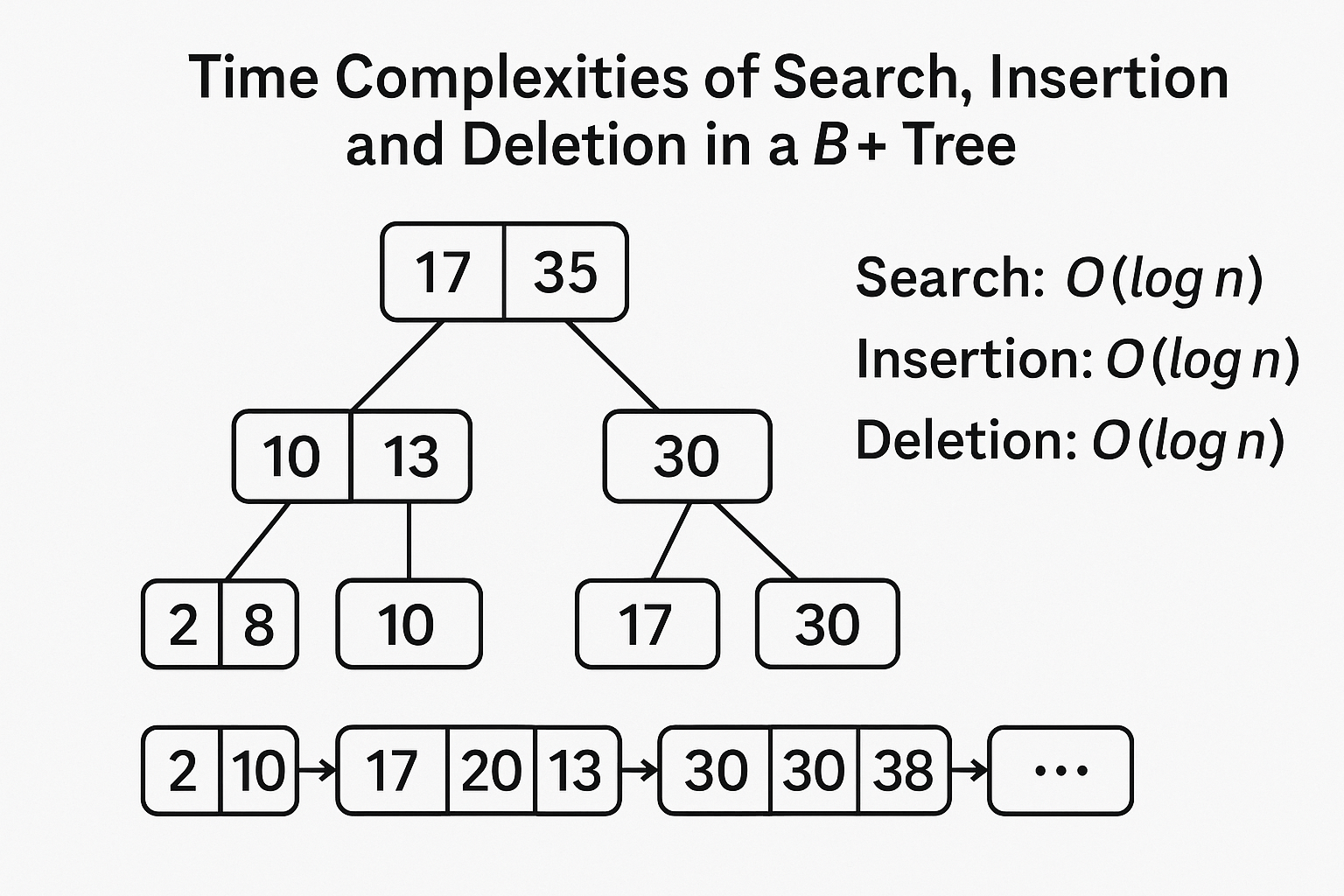
#Retorna todos os pares chave-valor dentro de um intervalo. Utiliza os ponteiros entre folhas #para varredura eficiente.

**pegar\_todas\_chaves() :**

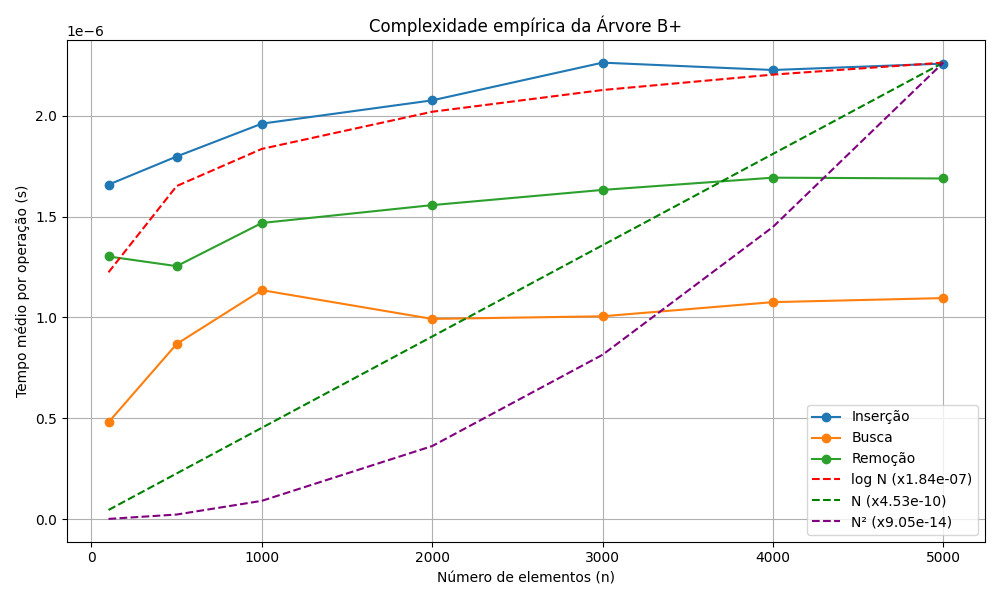
#Retorna todas as chaves da árvore em ordem crescente, começando da folha mais à #esquerda.

## Complexidade das Operações

**Complexidades Esperadas :**

****

**Complexidade Real (Testada)**



Referências :

1. <https://www.geeksforgeeks.org/dsa/b-tree-in-python-2/#key-characteristics-of-b-trees>
2. <https://www.programiz.com/dsa/b-plus-tree>
3. [B+ tree index](https://www.teach.cs.toronto.edu/~csc443h/fall/posted_practice/btree-index.html)
4. [B+ Tree](https://www.programiz.com/dsa/b-plus-tree)
5. [b-tree-java/BTreeNode.java at main · angelozero/b-tree-java](https://github.com/angelozero/b-tree-java/blob/main/BTreeNode.java)
6. [B-tree Data Structure | Baeldung on Computer Science](https://www.baeldung.com/cs/b-tree-data-structure)
7. [Introduction of B+ Tree - GeeksforGeeks](https://www.geeksforgeeks.org/dbms/introduction-of-b-tree/)
8. [B-trees - CS Notes](https://notes.eddyerburgh.me/data-structures-and-algorithms/data-structures/b-trees)
9. [Mastering the B-Tree Data Structure: An In-Depth Practical Guide - Guru Software](https://www.gurusoftware.com/mastering-the-b-tree-data-structure-an-in-depth-practical-guide/)
10. <https://www.bing.com/ck/a?!&&p=3c666c1d8267edfc366632bb3ab7e6ba48c13e6befb9d4fbe93cb5cc01e4a070JmltdHM9MTc1MDg5NjAwMA&ptn=3&ver=2&hsh=4&fclid=3a13766c-0c88-698d-1f24-63850d886898&psq=How+do+I+simulate+an+interactive+terminal+like+the+prompt&u=a1aHR0cHM6Ly9naXRodWIuY29tL215cXVpdGUvaW50ZXJhY3RpdmUtdGVybWluYWw&ntb=1>
11. [B Trees and B+ Trees. When it comes to indexing and efficient… | by AkashSDas | Medium](https://medium.com/@akashsdas_dev/b-trees-and-b-trees-682d363df1f7)