Algoritmos e Estrutura de Dados II

Projeto 1ºVA

Aluno: Guilherme Rutemberg Felix de Lima

UFAPE - Garanhuns
2022

Introdução

Um algoritmo pode ser visto como uma ferramenta para resolver um problema computacional bem especificado. Árvore é uma das mais importantes estruturas de dados não lineares na computação. Nas árvores os dados estão dispostos de forma hierárquica, os seus elementos se encontram "acima" ou "abaixo" de outros elementos da árvore. Toda a árvore possui o elemento chamado raiz, que possui ligações para outros elementos denominados ramos ou filhos. Estes ramos podem estar ligados a outros elementos que também podem possuir outros ramos. O elemento que não possui ramos é conhecido como nó folha, e levam $O(\log n)$ em seus piores casos de execução, onde n é o número de nós na árvore antes da operação.

Metodologia

O algoritmo foi executado como prioridade, apenas o terminal e os recursos do sistema operacional sendo executados.

Especificações da máquina:

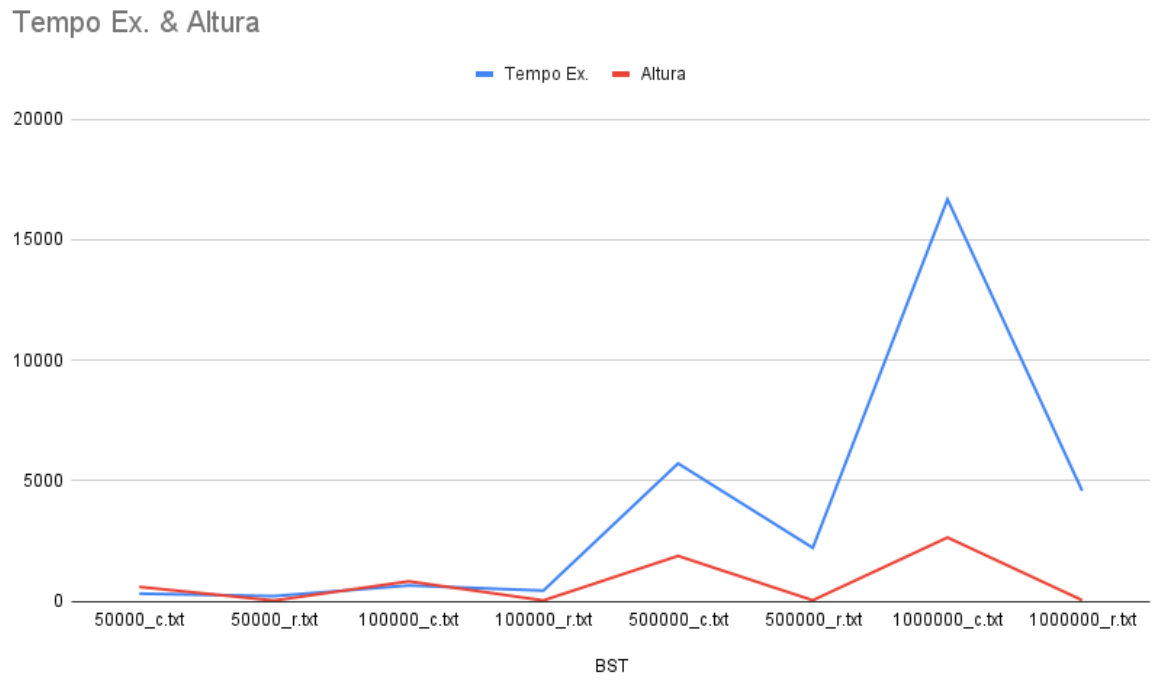
CPU: Ryzen 3 3200G 3.60Ghz (stock).
RAM: 8x2(16GB) 2666Mhz.
SSD: Kingston A400 480GB Sata III, Leitura 500MBs Gravação 450MBs.
GPU: GTX 1650 4GB SUPER DDR6.
SO: Linux Mint 21 Cinnamon, Versão 5.4.12(Dual boot).

Resultados

1. Árvores binárias de busca(BST)

Uma árvore binária de busca, é uma árvore de dados e estrutura binária enraizada com a chave de cada nó interno sendo maior que todas as chaves no nó respectivo nó da subárvore esquerda e menos que os nós da subárvore direita. A complexidade do tempo das operações na árvore binária é diretamente proporcional à altura da árvore.

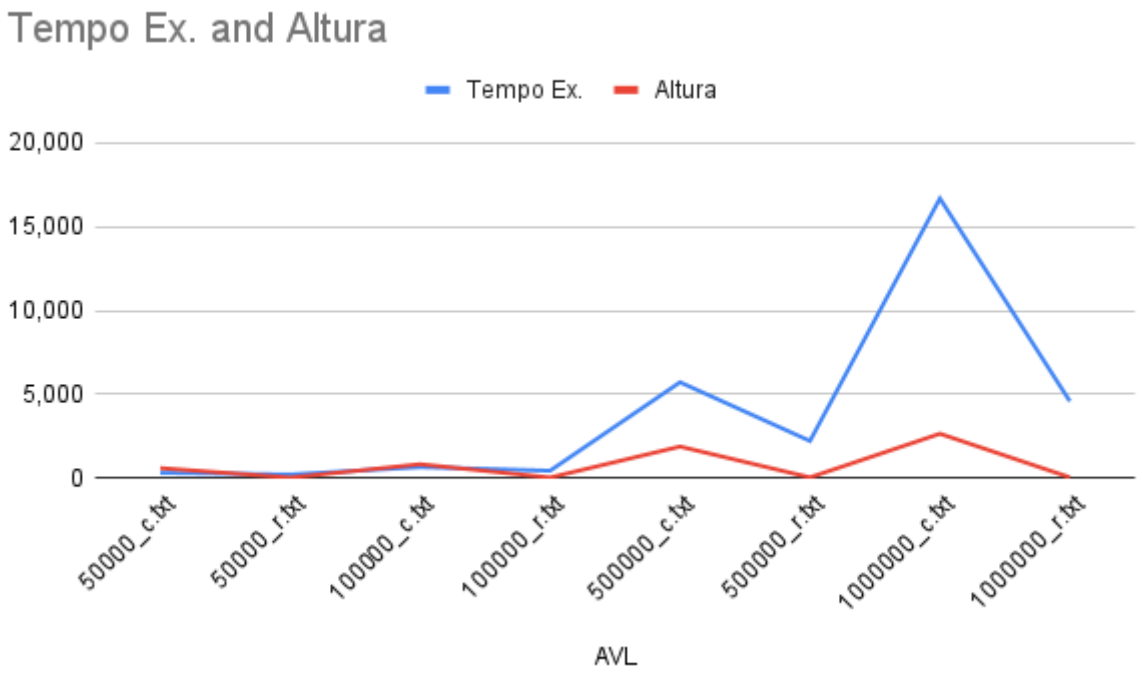
Árvores binárias de busca permitem adições, buscas e remoções mais rápidas, comparadas às árvores **AVLs** e **rubro-negras**. Árvores binárias podem ser usadas na implementação de tipos de dados abstratos, e usadas em algoritmos de ordenação.



2. *Adelson-Velsky and Landis*(AVL)

A árvore AVL é uma árvore de busca binária auto-equilibrada. Na árvore AVL, as alturas das duas subárvores filhas de qualquer nó difere em no máximo um, se em algum momento a altura for mais que um, o rebalanceamento é realizado para restaurar o equilíbrio de no máximo um de altura entre as subárvores.

As funções de inserir, remover e buscar todas levam O(log *n*) em ambos medianos e piores casos, onde *n* é o número de nós na árvore antes da operação. Em algumas situações ao inserir ou remover podem requisitar que a árvore seja rebalanceada uma ou mais vezes, executando rotações na árvore. Para aplicações de busca intensiva, as árvores AVL se sobressaem em velocidade sobre árvores rubro-negras, por serem mais rigorosamente equilibradas, e são equilibradas por sua altura.



3. **Rubro-negra(RB)**

A árvore vermelho-preto é um tipo de árvore de busca binária auto-equilibrada. Onde o nó armazena um bit extra representando "cor" ("vermelho" ou "preto"), essa cor é usada para garantir que a árvore permaneça equilibrada durante as operações de inserir e remover.

Quando a árvore é modificada, a nova árvore é reorganizada e "repintada" para restaurar as propriedades de coloração que restringem o quão desequilibrada a árvore pode se tornar no pior dos casos. As propriedades são projetadas de modo que essa reorganização e recoloração possam ser realizadas de forma eficiente.

O rebalanceamento não é perfeito, mas garante a busca em tempo O(log n), onde n é o número de entradas. As operações de inserir e remover, juntamente com o ajuste e a recoloração da árvore com o reajuste, também são executadas em tempo O(log n)tempo. Mantendo as propriedades a árvore consegue manter seu equilíbrio de folhas pretas.

Considerações Finais

Este experimento me ensinou que de fato há várias maneiras de se implementar algoritmos para realizar uma determinada função e resolver determinado problema. A complexidade de um algoritmo de cada árvore de dados varia conforme o tamanho de condições e propriedades para o equilíbrio das árvores, neste experimento ficou claro que quando se trata de um problema por partes e de maneira que tenha condições suficientes implementadas, para o melhor ou pior caso que venha a acontecer, a implementação das árvores é essencial para algoritmos robustos.

Aprender sobre as árvores foi bem interessante, pois aprendi bastante sobre como desenvolver um algoritmo pode ser frustrante e revigorante ao mesmo tempo. A maneira com que as árvores binárias de busca podem ser úteis para diversas aplicações, pois métodos de ordenação como Heap e Quick sort eram muito efetivas, mas árvores *BST* e *AVL*, utilizam além de uma estrutura mais equilibrada para inserção de dados, também possuem bastante vantagens quanto ao tempo de execução de suas operações, obtendo ótimos resultados se implementadas em aplicações que utilizam diversos tipos de dados.

A árvore rubro-negra foi de longe a mais difícil de implementar, realmente tive bastante contato com a famigerada “falha de segmentação”, graças a operação de remoção em árvores muito extensas não consegui o resultado que esperava, pois não consegui fazer com que meu algoritmo conseguisse realizar todas as operações sem a falha de segmentação. Porém, estudá-la foi um ótimo aprendizado, e em um curto prazo irei conseguir realizar a proeza de conseguir o resultado esperado.

Portanto, digo que foi um ótimo experimento para meu conhecimento e também bom para minha prática de programação, compreendi o quão importante é buscar soluções que executem de maneira rápida e eficiente.