ED2EP2

Willian Miura Mori
NUSP: 12542219

May 26, 2022

1 Compilação e dependências

Dependências: rustc

Compilação: No diretório do EP, digite o comando 'make'. O executável será *exec*

2 Estrutura do programa

- A interação com o usuário (stdin e stdout) está em $main.rs$
- A interface da tabela de símbolos é definida em $symbol_table.rs$
- As implementações estão no diretório $symbol_table$

3 Implementações

3.1 Vetor Ordenado

Os pares chave valor estão armazenados linearmente de modo ordenado utilizando um $Vec < (Key, Item) >$ (array de tamanho dinâmico que guarda pares chave valor)

- Inserção: $O(N)$, pois como a memória é linear podemos deslocar $O(N)$ elementos para inserir um novo. No caso de precisarmos inserir no começo do vetor, devemos deslocar todos os outros elementos
- Busca por chave: $O(logN)$, com uma busca binária (pois o vetor está ordenado pela chave)
- Busca por rank: $O(1)$, pois podemos apenas indexar pelo rank

3.2 Árvore de Busca Binária

Para um par chave valor descemos a árvore e adicionamos um nó com esse par de modo que ele seja uma folha. Não há rebalanceamento.

As operações dependem da altura da árvore, que pode ser $O(N)$

3.3 Treap

Uma ABB com a invariante que os filhos de um nó devem ter altura menor que a desse nó. Como as alturas são aleatórias, a altura da árvore é por um argumento probabilístico de ordem $O(logN)$.

Implementada similarmente à ABB, mas com um rebalanceamento feito por meio de rotações.

3.4 Árvore 2-3

Uma ABB em que os nós podem ser de dois tipos: 2-nó ou 3-nó. É mantida a invariante que otodos os caminhos do nó até uma folha tem o mesmo comprimento. Assim, a altura da árvore é de ordem $O(logN)$.

Implementamos o nó com a ajuda de sum-types/tagged unions, e o rebalanceamento é feito por meio da transformação de um 3-nó para um 4-nó temporário, que é transformado para um 2-nó com 2 filhos na raíz (modelado no código como um retorno de um par chave e valor — que será levado para cima na árvore — e dois filhos que deverão ser adicionados ao pai).

Não podemos copiar e colar as operações da ABB, por causa da existência do 3-nó.

3.5 Árvore Rubro-Negra

Uma ABB em que os nós podem ser pretos ou vermelhos, com a invariante que todos caminhos da raíz até um NIL tem a mesma quantidade de nós pretos, e todos nós vermelhos tem como filhos NIL ou nós pretos. Isso garante que a altura da árvore é da ordem de $O(logN)$.

Como precisamos para todo nó guardar um ponteiro para o pai, não podemos implementar essa estrutura de um modo elegante em *Rust*, precisando utilizar ponteiros normais (que não são smart pointers, e devem ser dealocados manualmente).

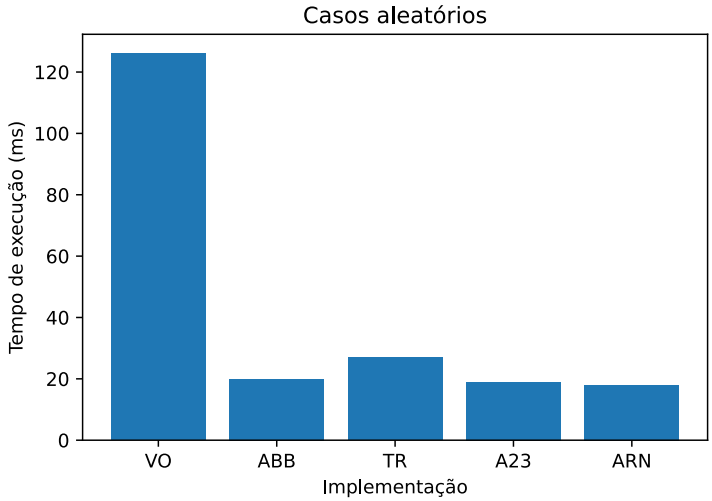
O rebalanceamento é feito por rotações e recolorações.

4 Testes

Para os testes, faremos sempre N operações de cada tipo $(value, rank, select)$ após inserir N elementos.

4.1 Chaves aleatórias

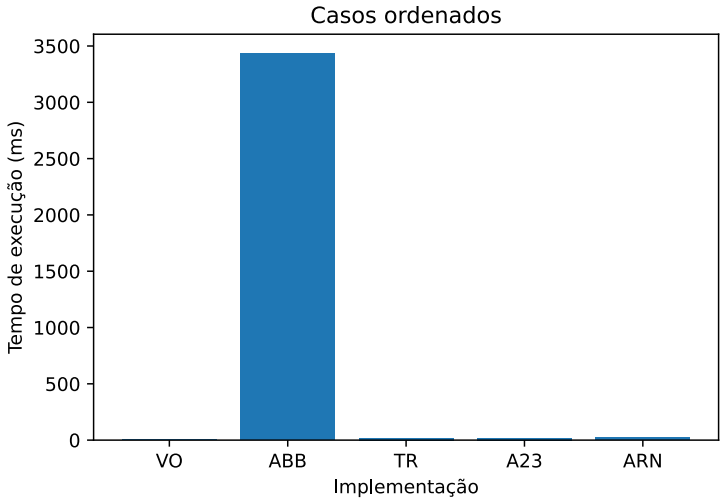
A complexidade esperada da inserção de um elemento no vetor ordenado é $O(N)$, pois é esperado que precisemos inserir um par chave valor no meio do vetor, deslocando em média $\frac{N}{2}$ elementos. Como as chaves são aleatórias, podemos afirmar que a ABB terá uma altura da ordem de $O(logN)$. As outras implementações terão altura da ordem de $O(logN)$ em todos casos de teste.



4.2 Chaves ordenadas

A inserção no vetor ordenado terá complexidade $O(1)$, pois sempre iremso inserir o elemento no fim do vetor. É esperado que essa seja a melhor estrutura nesse caso de teste.

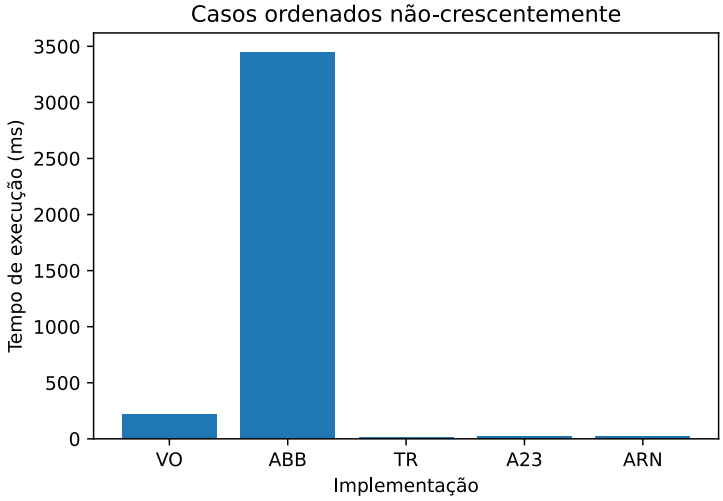
A ABB, em contrapartida, terá complexidade $O(N)$ por inserção pois sua altura será de exatamente N elementos, a qual é a pior possível.



4.3 Chaves ordenadas não-crescentemente

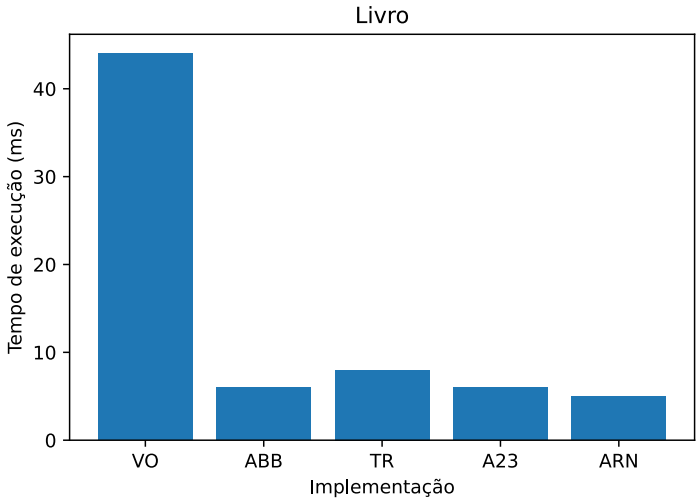
A inserção no vetor ordenado nesse caso será o pior caso, em que devemos sempre colocar o elemento no começo do vetor. Teremos uma complexidade $O(N)$ por inserção.

A ABB, similarmente ao caso anterior, terá altura N.



4.4 eBook of Tea and the effects of tea drinking by W. Scott Tebb

Intuitivamente, podemos inferir que o livro se comportará como um caso com chaves aleatórias



5 Conclusões

O vetor ordenado se comportou como esperado, mas podiamos evitar o pior caso com as chaves ordenadas não-crescentemente utilizando uma estrutura de dados como o *deque* (em que a inserção no começo também é $O(1)$). No caso das chaves aleatórias a complexidade por adição continuaria de ordem linear no entanto.

Nos casos ordenados, a ABB apresentou um resultado muito pior que o VO, mesmo comparado ao pior caso deste. É possível inferir que isto se deve ao custo de iterar sobre uma memória linear (cache-friendly) em comparação a iterar sobre nós dispostos aleatoriamente na memória, além da complexidade adicional de uma ABB em relação a um simples vetor.

A árvore rubro-negra e a árvore 2-3 apresentaram resultados parecidos, enquanto a Treap teve um resulado ligeiramente pior, presumidamente devido à altura esperada da árvore ter uma consta que a altura das outras duas árvores.