Avaliação RA02

Kauã Nunes, Eduarda Dallagrana, André Thiago;

- Aqui podemos ver duas imagens da performance para inserir 100, 500, 1000, 10000, e 20000 elementos nas árvores binárias. No geral a árvore binária de busca é mais direta para inserir novos elementos. Por ter uma lógica mais simples para verificar a inserção, enquanto a BST AVL (Binary Search Tree AVL) checa para o balanceamento e movimentação de sub árvores.

BST

```
Tree width: 100
Time to fill up in nanoseconds: 3828599
Tree width: 500
Time to fill up in nanoseconds: 235001
Tree width: 1000
Time to fill up in nanoseconds: 283399
Tree width: 10000
Time to fill up in nanoseconds: 3071000
Tree width: 20000
Time to fill up in nanoseconds: 6651600
```

BST AVL

```
Tree width: 100
Time to fill up in nanoseconds: 484600
Tree width: 500
Time to fill up in nanoseconds: 440000
Tree width: 1000
Time to fill up in nanoseconds: 479400
Tree width: 10000
Time to fill up in nanoseconds: 4482100
Tree width: 20000
Time to fill up in nanoseconds: 9041900
```

- No caso da remoção de elementos a árvore binária não balanceada pode ser mais rápida dependendo se a árvore não é uma árvore ordenada e não é necessário passar por todos os nodes pais. Enquanto a árvore binária AVL por ter um fator de balanceamento entre -1, 0, +1. Ela é mais otimizada para busca e remoção mais complexas, por mais que precise fazer o balanceamento da árvore após a remoção.

BST

```
Time to remove 50 elements in nanoseconds: 94400
Time to remove 250 elements in nanoseconds: 91900
Time to remove 500 elements in nanoseconds: 171300
Time to remove 5000 elements in nanoseconds: 1755300
Time to remove 10000 elements in nanoseconds: 3878899
```

BST AVL

```
Time to remove 50 elements in nanoseconds: 94699
Time to remove 250 elements in nanoseconds: 192499
Time to remove 500 elements in nanoseconds: 323500
Time to remove 5000 elements in nanoseconds: 4328799
Time to remove 10000 elements in nanoseconds: 10702401
```

- Checa a lógica do código e da árvore. E retorna se é uma árvore válida, em ambos os casos foi uma questão de teste durante a implementação do código.

BST

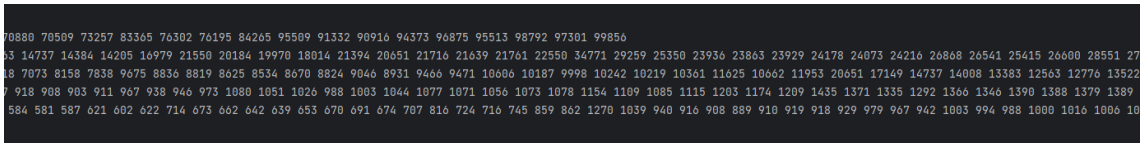
```
Binary tree is valid
Binary tree is valid
Binary tree is valid
Binary tree is valid
Binary tree is valid
```

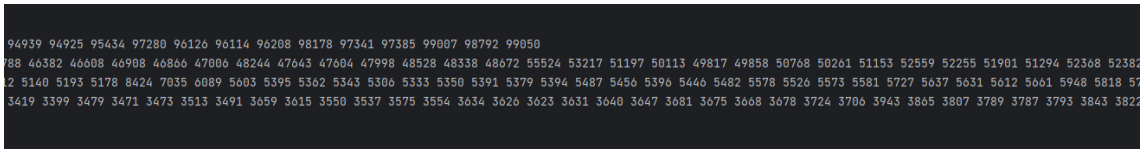
BST AVL

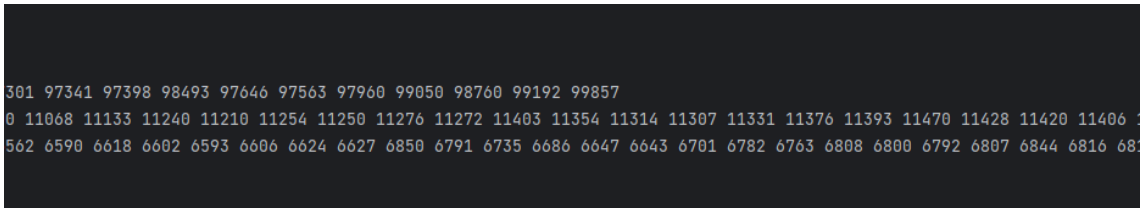
```
Tree 1: true
Is sorted: true
Tree 2: true
Is sorted: true
Tree 3: true
Is sorted: true
Tree 4: true
Is sorted: true
Tree 5: true
Is sorted: true
```

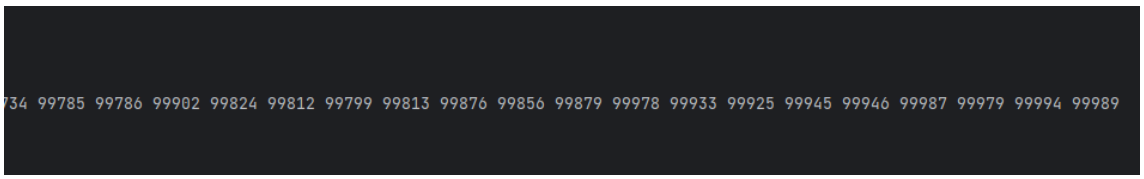
- Print das árvores binárias em *Preorder*. Percorre as árvores começando pela raiz, vai para a árvore esquerda, depois a direita recursivamente e imprime todos os elementos da árvore depois da remoção de metade dos elementos da árvore.

BST

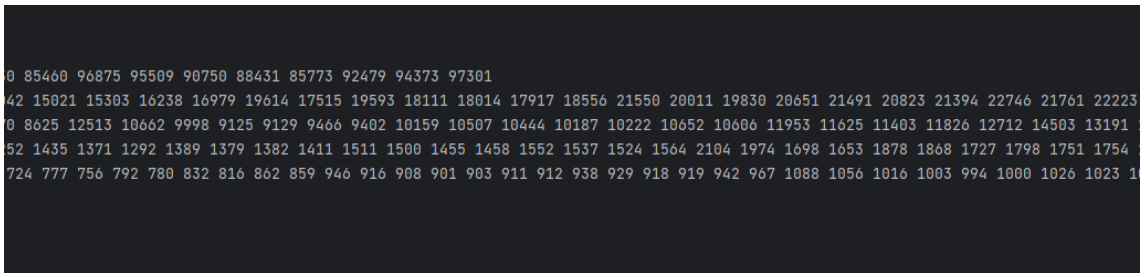


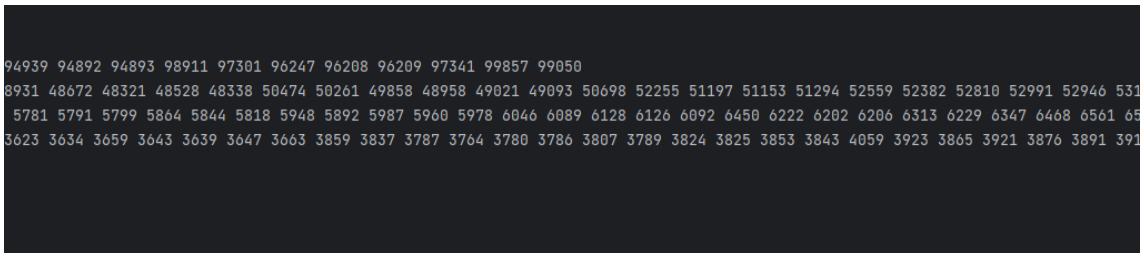


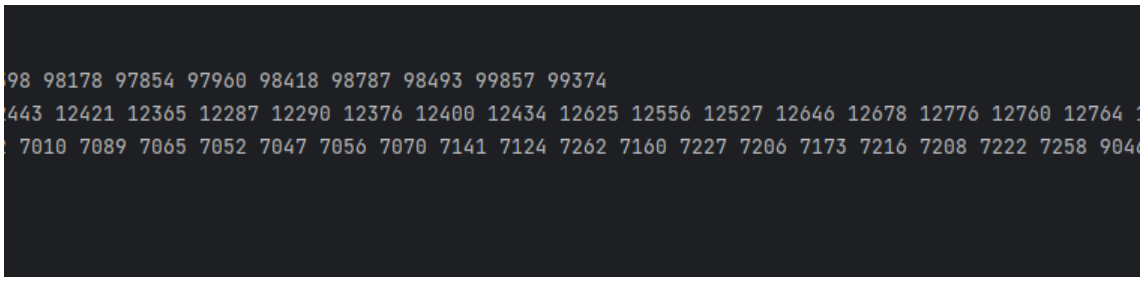


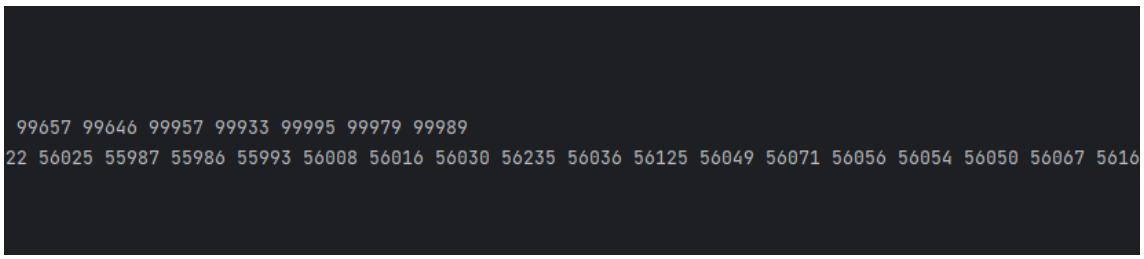


BST AVL









- Para a implementação dos códigos de árvore binária. Por mais que a árvore binária AVL tenha uma lógica um pouco mais complicada na questão de balanceamento e rotação, depois de dominada, a implementação não foi tão diferente, é uma forma mais complexa do que uma árvore binária não balanceada, e pode trazer mais benefícios na questão de busca de elementos, árvores binárias AVL nunca tem a altura maior que LogN, N sendo o número de nodes, enquanto como já mencionei a árvore binária não balanceada, se formada de forma ordenada, pode ter a altura muito maior que LogN. Porém dependendo da situação uma árvore binária não balanceada é uma opção válida.