

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

```
C:\Users\Kauã Nunes\.jdk\openjdk-20.0.2\bin\java.exe
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 fill up in nanoseconds: 8831000  
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  
Binary tree is valid
```

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  
Tree 1: true
```

- 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

```
70880 70509 73257 83365 76302 76195 84265 95509 91332 90916 94373 96875 95513 98792 97301 99856
3 14737 14384 14205 16979 21550 20184 19970 18014 21394 20651 21716 21639 21761 22550 34771 29259 25350 23936 23863 23929 24178 24073 24216 26868 26541 25415 26600 28551 27
18 7073 8158 7838 9675 8836 8819 8625 8534 8670 8824 9046 8931 9466 9471 10606 10187 9998 10242 10219 10361 11625 10662 11953 20651 17149 14737 14008 13383 12563 12776 13522
7 918 908 903 911 967 938 946 973 1080 1051 1026 988 1003 1044 1077 1071 1056 1073 1078 1154 1109 1085 1115 1203 1174 1209 1435 1371 1335 1292 1366 1346 1390 1388 1379 1389
584 581 587 621 602 622 714 673 662 642 639 653 670 691 674 707 816 724 716 745 859 862 1270 1039 940 916 908 889 910 919 918 929 979 967 942 1003 994 988 1000 1016 1006 10
```

```
94939 94925 95434 97280 96126 96114 96208 98178 97341 97385 99007 98792 99050
88 46382 46608 46908 46866 47006 48244 47643 47604 47998 48528 48338 48672 55524 53217 51197 50113 49817 49858 50768 50261 51153 52559 52255 51901 51294 52368 52382
2 5140 5193 5178 8424 7035 6089 5603 5395 5362 5343 5306 5333 5350 5391 5379 5394 5407 5456 5396 5446 5482 5578 5526 5573 5581 5727 5637 5631 5612 5661 5948 5818 57
3419 3399 3479 3471 3473 3513 3491 3659 3615 3550 3537 3575 3554 3634 3626 3623 3631 3640 3647 3681 3675 3668 3678 3724 3706 3943 3865 3807 3789 3787 3793 3843 3822
```

```
301 97341 97398 98493 97646 97563 97960 99050 98760 99192 99857
0 11068 11133 11240 11210 11254 11250 11276 11272 11403 11354 11314 11307 11331 11376 11393 11470 11428 11420 11406 1
562 6590 6618 6602 6593 6606 6624 6627 6850 6791 6735 6686 6647 6643 6701 6782 6763 6808 6800 6792 6807 6844 6816 68
```

```
734 99785 99786 99902 99824 99812 99799 99813 99876 99856 99879 99978 99933 99925 99945 99946 99987 99979 99994 99989
```

BST AVL

```
0 85460 96875 95509 90750 88431 85773 92479 94373 97301
42 15021 15303 16238 16979 19614 17515 19593 18111 18014 17917 18556 21550 20011 19830 20651 21491 20823 21394 22746 21761 22223
0 8625 12513 10662 9998 9125 9129 9466 9402 10159 10507 10444 10187 10222 10652 10606 11953 11625 11403 11826 12712 14503 13191
52 1435 1371 1292 1389 1379 1382 1411 1511 1500 1455 1458 1552 1537 1524 1564 2104 1974 1698 1653 1878 1868 1727 1798 1751 1754
724 777 756 792 780 832 816 862 859 946 916 908 901 903 911 912 938 929 918 919 942 967 1088 1056 1016 1003 994 1000 1026 1023 1
```

```
94939 94892 94893 98911 97301 96247 96208 96209 97341 99857 99050
8931 48672 48321 48528 48338 50474 50261 49858 48958 49021 49093 50698 52255 51197 51153 51294 52559 52382 52810 52991 52946 531
5781 5791 5799 5864 5844 5818 5948 5892 5987 5960 5978 6046 6089 6128 6126 6092 6450 6222 6202 6206 6313 6229 6347 6468 6561 65
3623 3634 3659 3643 3639 3647 3663 3859 3837 3787 3764 3780 3786 3807 3789 3824 3825 3853 3843 4059 3923 3865 3921 3876 3891 391
```

```
98 98178 97854 97960 98418 98787 98493 99857 99374
443 12421 12365 12287 12290 12376 12400 12434 12625 12556 12527 12646 12678 12776 12760 12764 1
7010 7089 7065 7052 7047 7056 7070 7141 7124 7262 7160 7227 7206 7173 7216 7208 7222 7258 9046
```

```
99657 99646 99957 99933 99995 99979 99989
22 56025 55987 55986 55993 56008 56016 56030 56235 56036 56125 56049 56071 56056 56054 56050 56067 5616
```

- 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 $\log N$, 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 $\log N$. Porém dependendo da situação uma árvore binária não balanceada é uma opção válida.