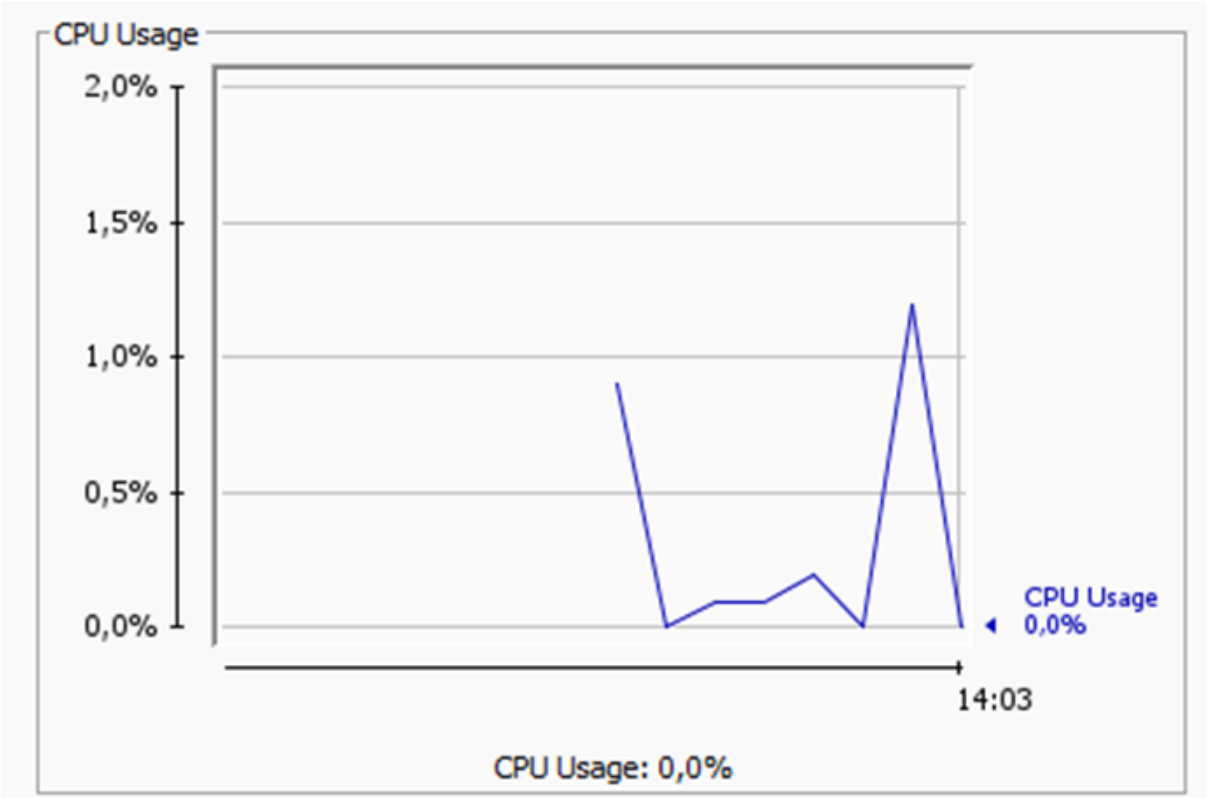
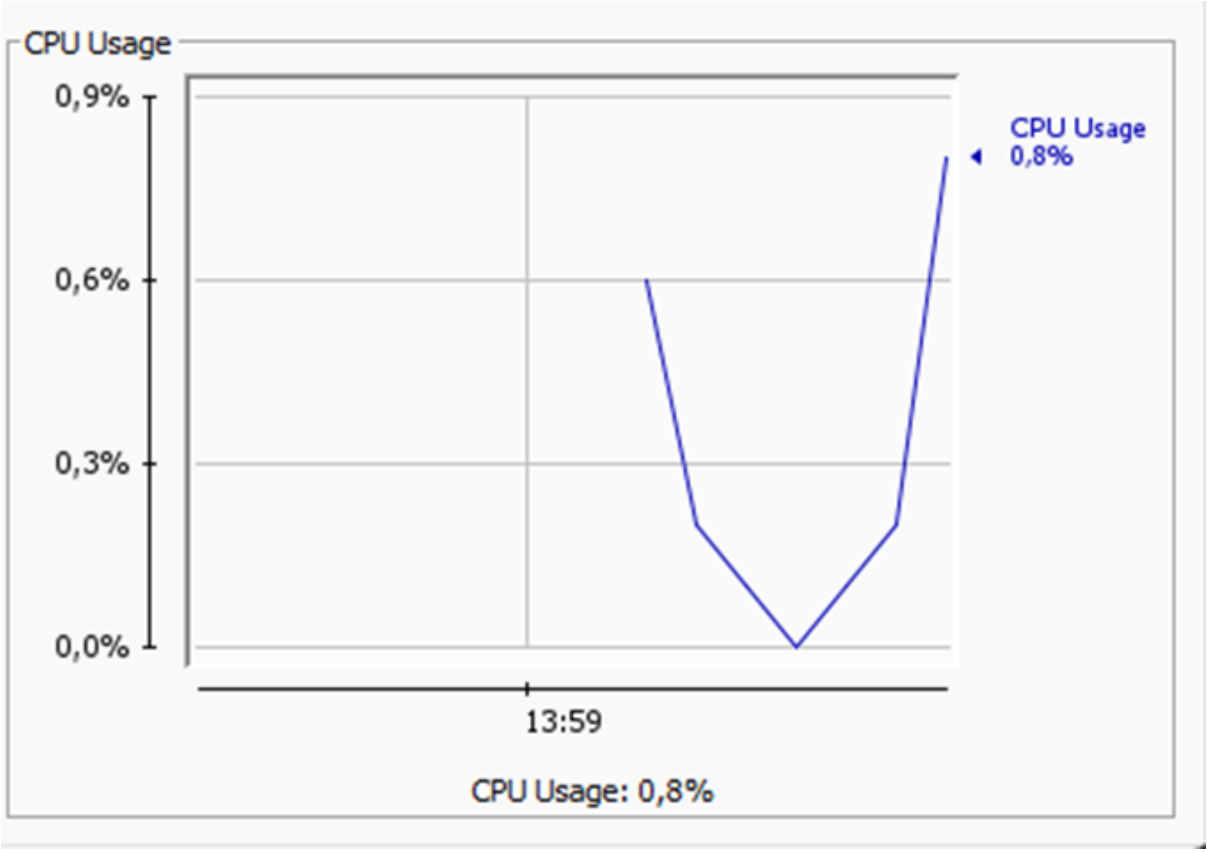
Relatório árvore de busca e AVL
Estudante: João Pedro Igeski Morais

Inserção de elementos
100 Elementos



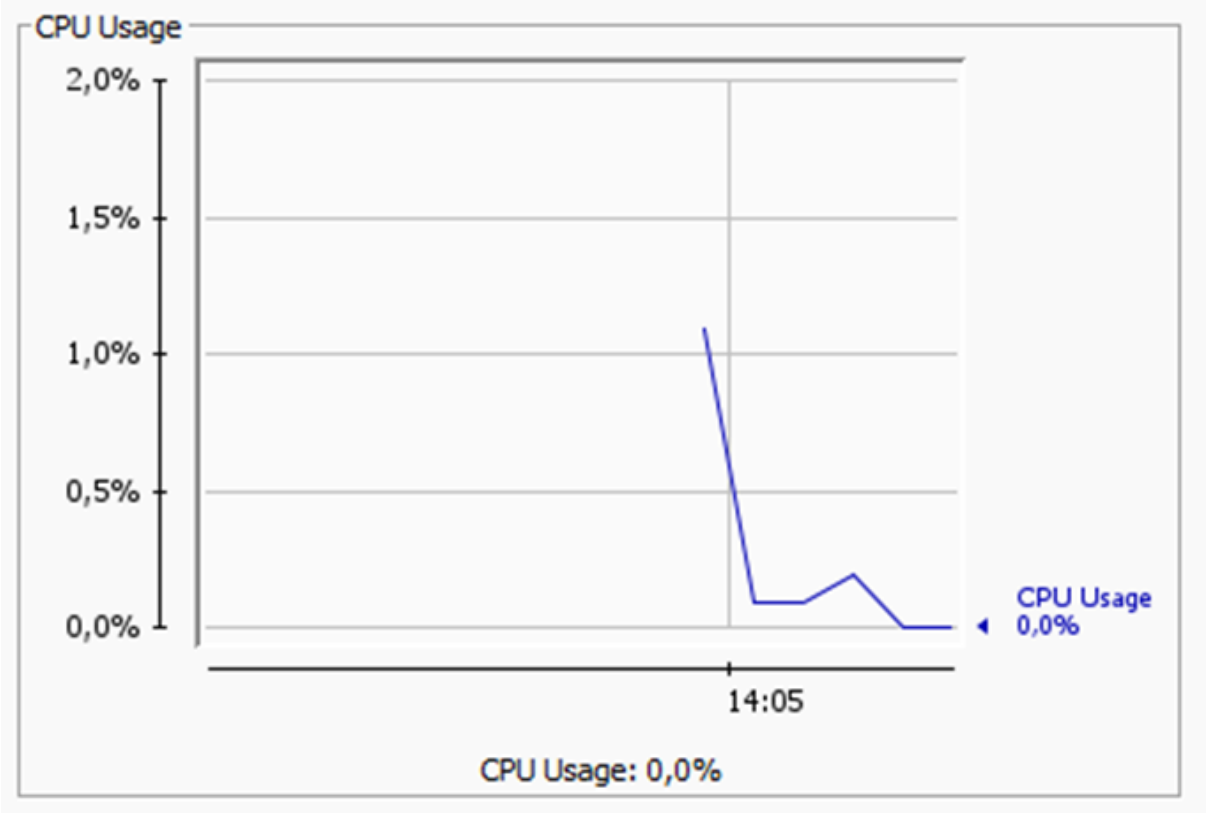
Resultado: A árvore de busca exigiu mais da CPU do que a árvore AVL. Sendo que o primeiro pico corresponde a inserção do conjunto na AVL e o segundo pico na árvore de busca.

Inserir 500 elementos

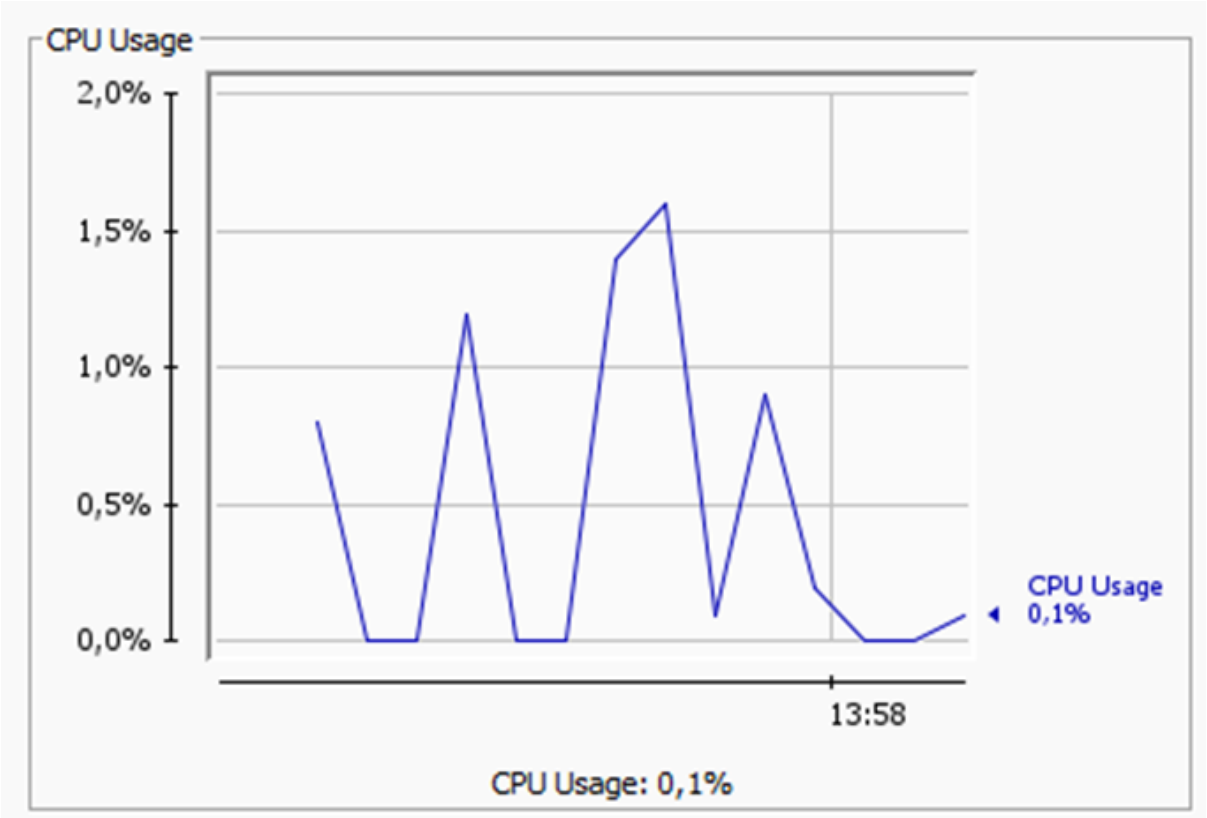


Resultado: A árvore de busca exigiu mais da CPU do que a árvore AVL. Sendo que o primeiro pico corresponde a inserção do conjunto na AVL e o segundo pico na árvore de busca.

Inserir 1000 elementos

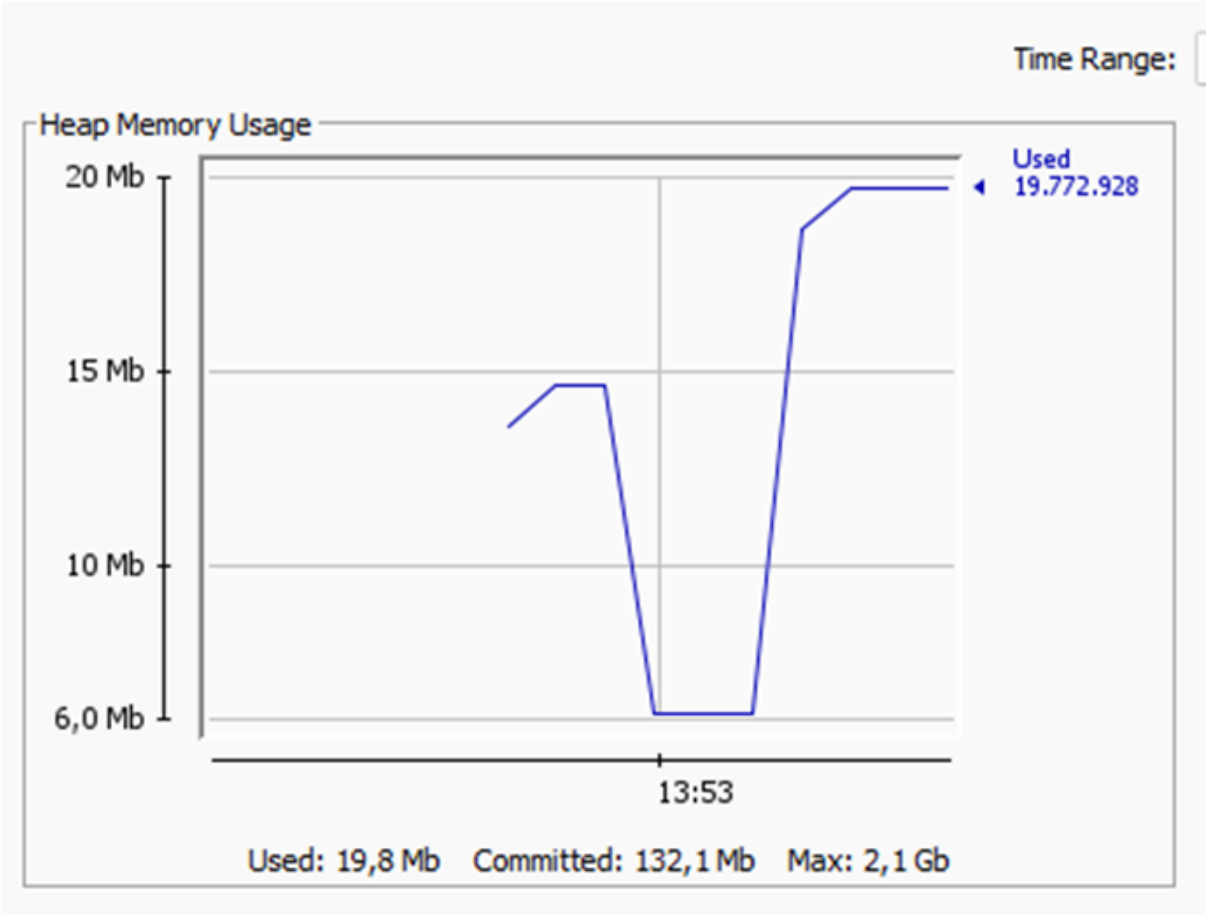


Inserir 10000 elementos



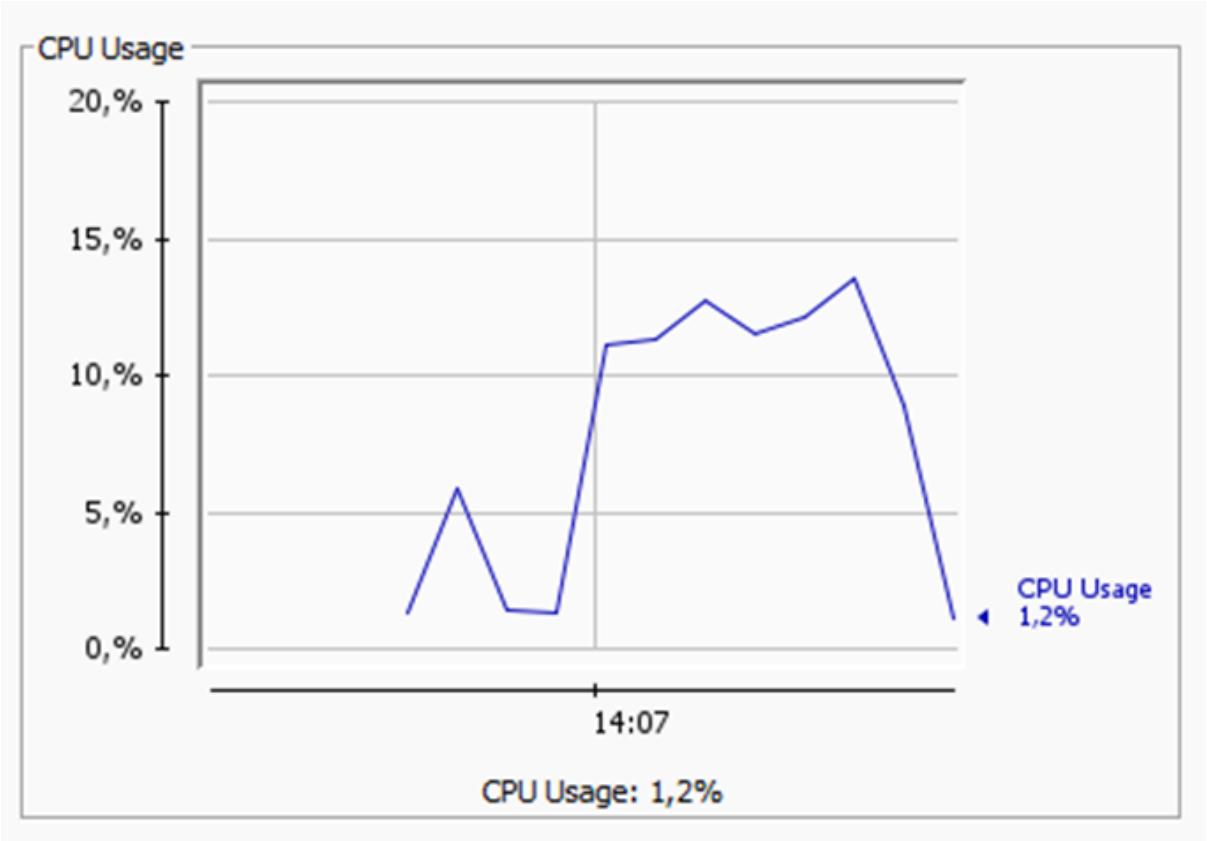
Novamente, a árvore AVL (Primeiro pico) exigiu menos recursos do que a árvore de busca (Segundo pico)

Inserir 20000 elementos



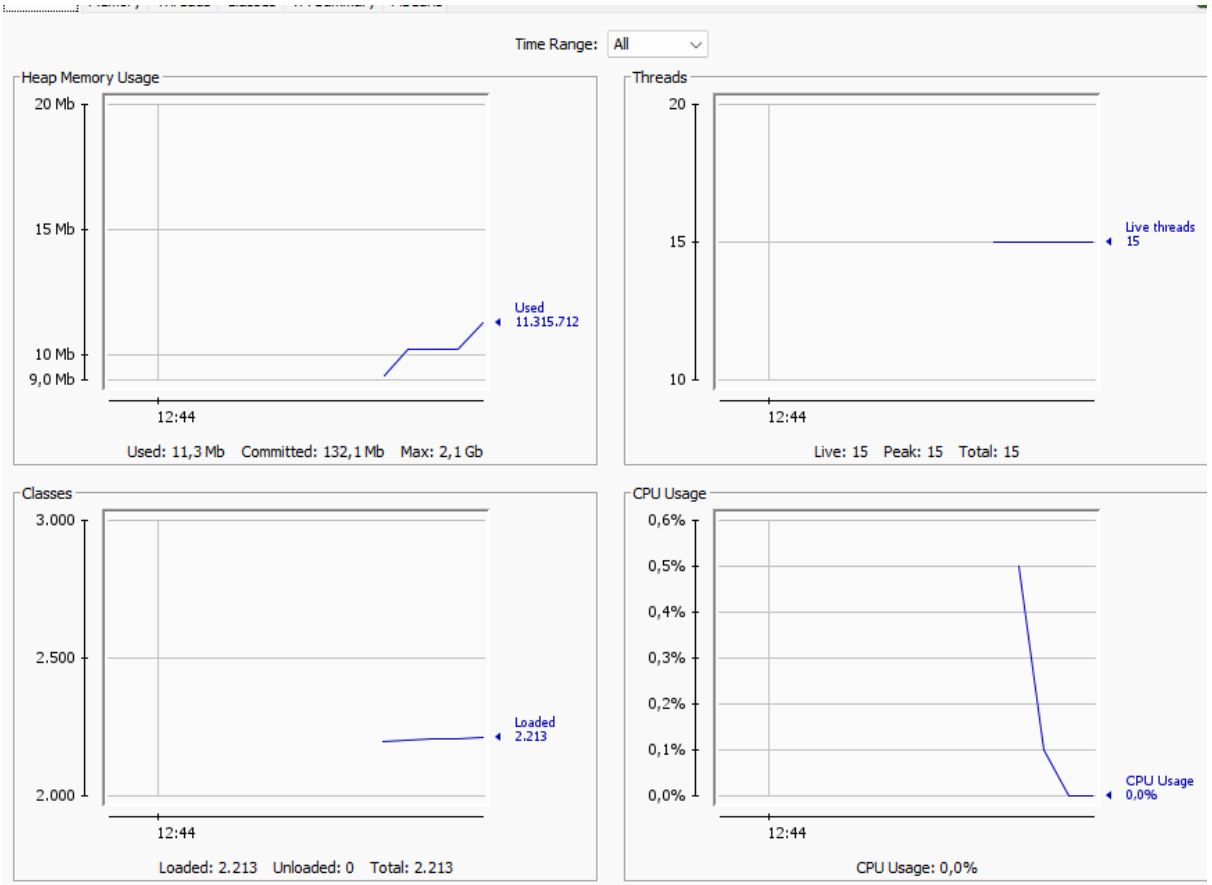
Desta vez analisei o consumo de memória para inserção. O primeiro pico (menos de 15mb) é o que foi consumido para inserir na AVl. Já o segundo pico (cerca de 20mb), é o que foi consumido para inserir na árvore de busca.

Inserir 1 milhão de elementos

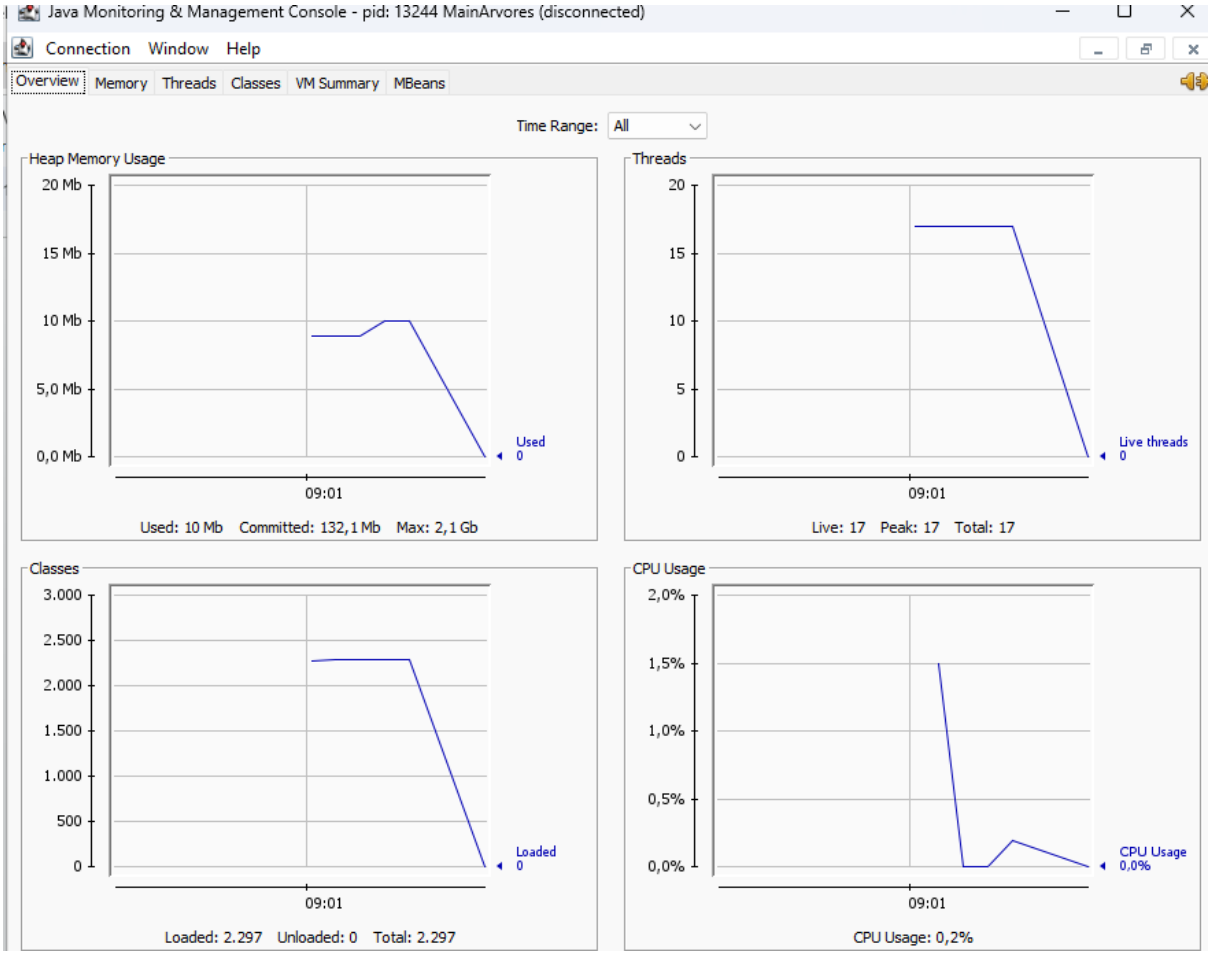


Fiz este teste mais extremo, que pode simular um sistema real. No caso, para inserir 1 milhão de elementos na árvore AVL exigiu um uso de cerca de 6% da CPU. Já a árvore de busca, exigiu cerca de 13% da CPU

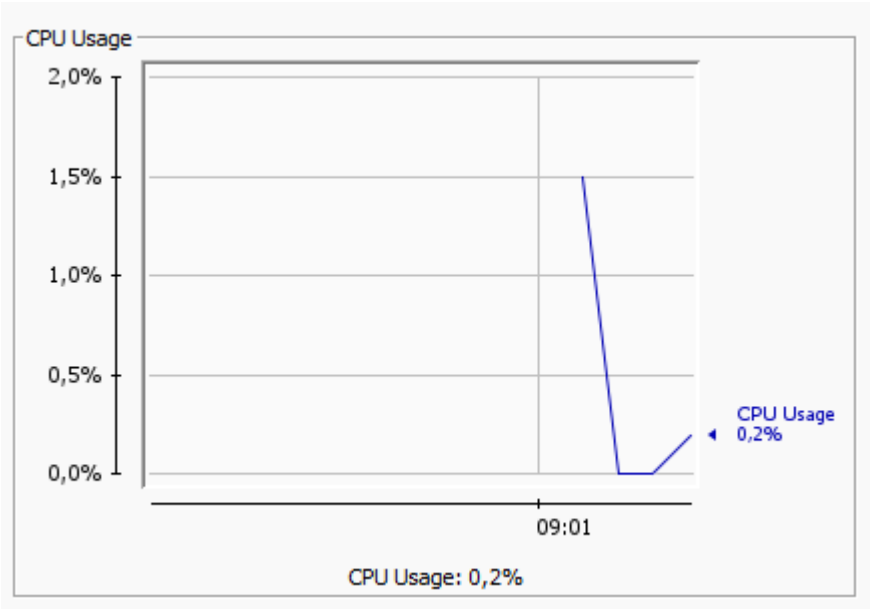
Buscar elemento em AVL



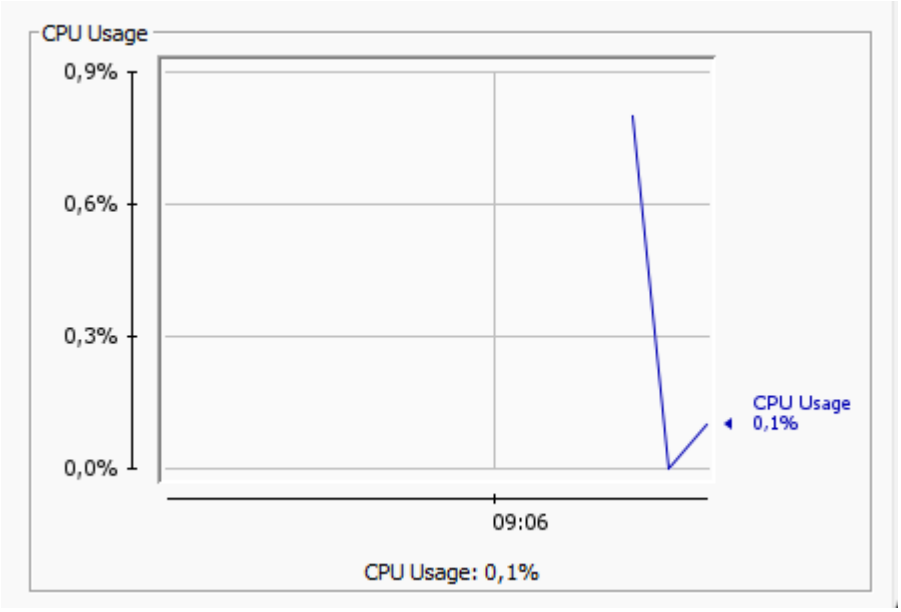
Busca elemento árvore de busca



Resultados para busca em árvores de 20000 Elementos
Para busca na AVL foi usado 0.2% da CPU
Para busca na árvore de busca foi usado 1.5% da CPU

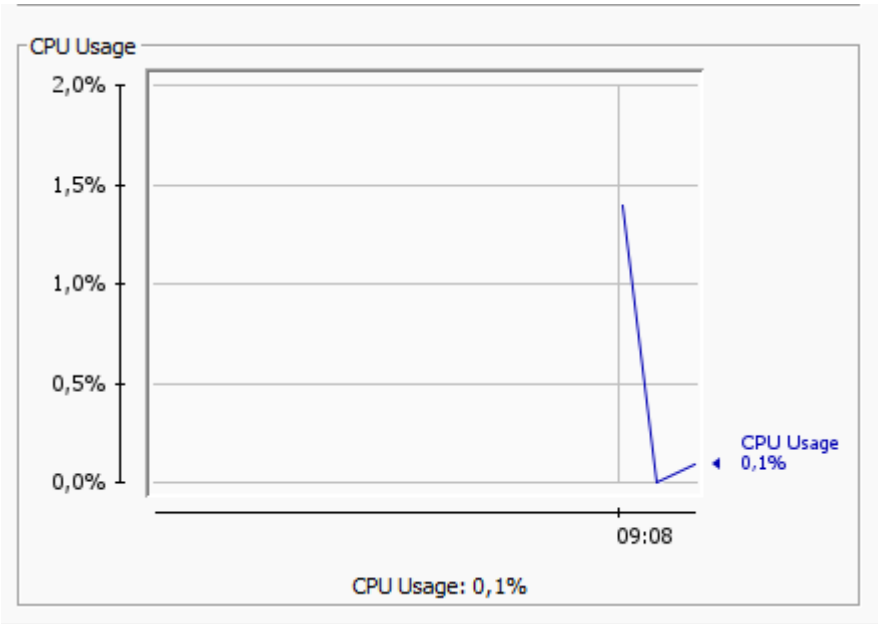


Para 1000 elementos o resultado também foi parecido

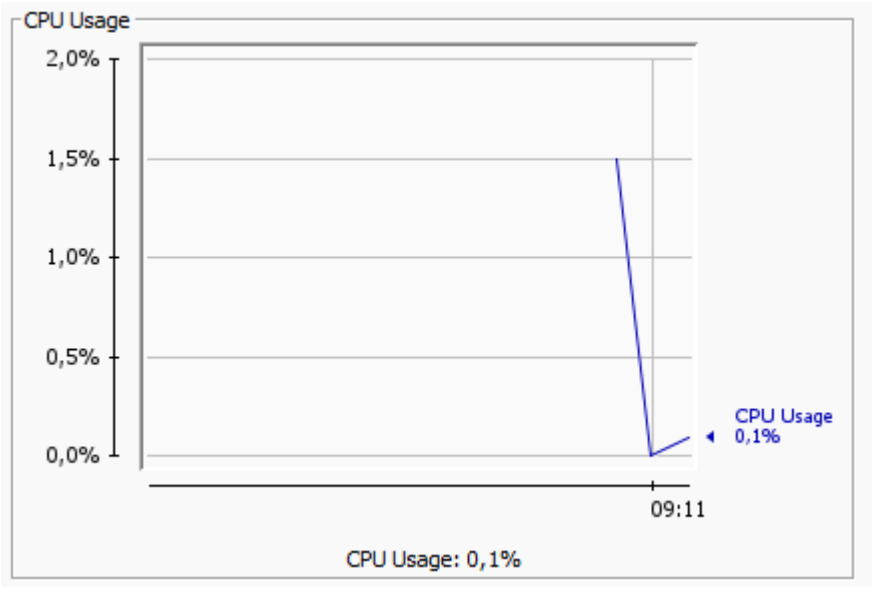


Neste caso, quando buscamos na árvore de busca, precisamos usar cerca de 0.8% da CPU. Enquanto isso na AVL, usamos apenas 0.1%

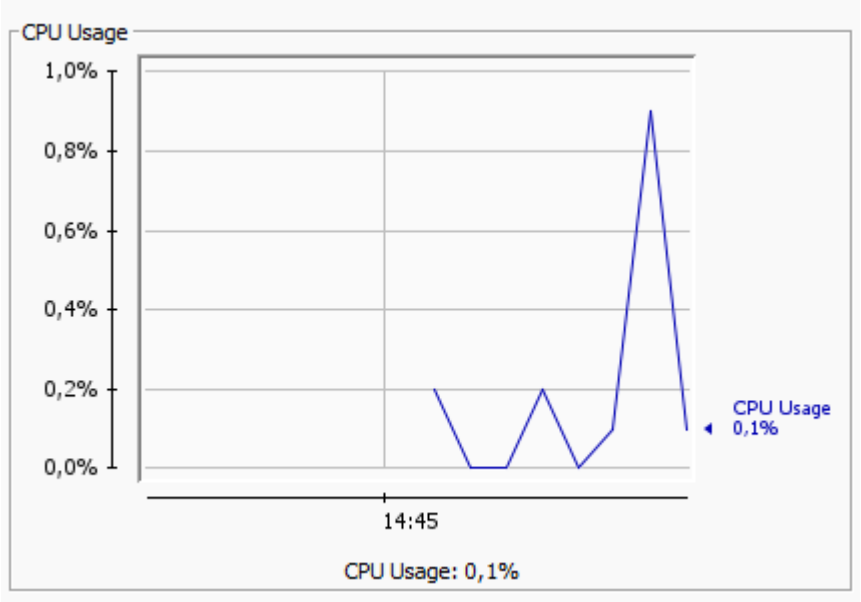
Para 500 elementos o resultado também foi parecido



Para 1 milhão de elementos, a árvore AVL foi mais performática também, ou seja, cenário parecido

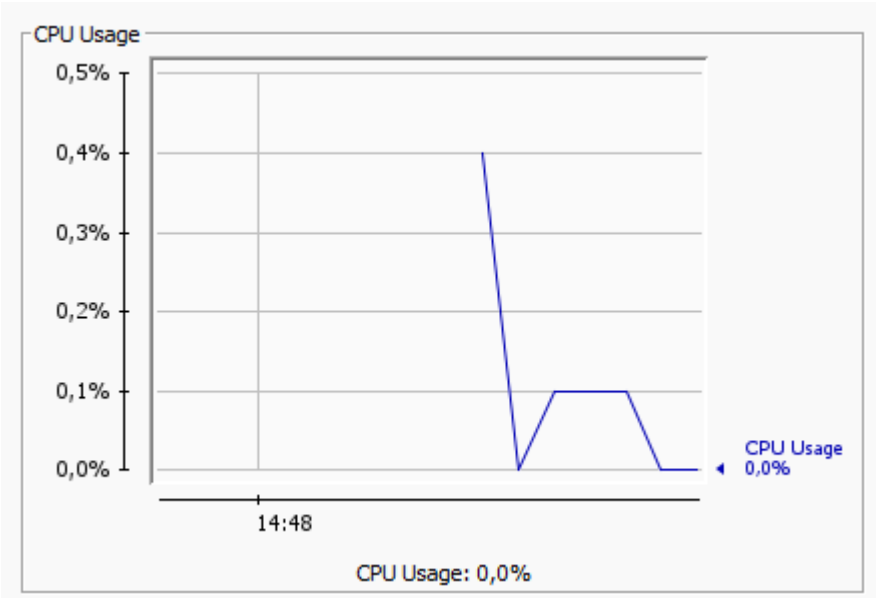


Remover um elemento
Total de elementos nas árvores: 20000



As duas árvores possuíam os mesmos elementos, logo, o mesmo elemento seria removido das duas. Primeiro foi removido o elementos da AVL (que usou apenas 0.2% da CPU). Já para remoção na árvore busca, tivemos que usar quase 5x mais (0.9% da CPU)

Para remover em uma árvore menor (100 elementos) não houve tanta diferença. A árvore AVL gastou mais recursos, pois precisou rebalancear a árvore. A diferença foi de apenas 0.3%.



Código - Busca

INSERIR

Função iterativa para inserir um nó. Se a raiz for nula, define a raiz como o novo nó. Senão, usa um nó auxiliar para percorrer a árvore e encontrar o local adequado para o novo nó, realizando comparação.

BUSCA

Função que retorna o nó da árvore com base em um valor. Para isso verifica se é a raiz, se não for, itera sobre os nós para encontrar o nó correspondente na árvore

BUSCA PAI

Retorna o pai de um nó

REMOVER

Remove um nó da árvore. Para isso, verifica se é raiz, se é nó folha, se tem apenas 1 filho ou se tem 2 filhos. Para cada cenário, ele exclui o nó e adapta a árvore se necessário (caso dos 2 filhos).

Código - AVL

INSERIR
Função iterativa para inserir um nó. Se a raiz for nula, define a raiz como o novo nó. Senão, usa um nó auxiliar para percorrer a árvore e encontrar o local adequado para o novo nó, realizando comparação. Após inserir, percorre a árvore para verificar se os nós estão balanceados.

BUSCA
Função que retorna o nó da árvore com base em um valor. Para isso verifica se é a raiz, se não for, itera sobre os nós para encontrar o nó correspondente na árvore

BUSCA PAI
Retorna o pai de um nó

REMOVER
Remove um nó da árvore. Para isso, verifica se é raiz, se é nó folha, se tem apenas 1 filho ou se tem 2 filhos. Para cada cenário, ele exclui o nó e adapta a árvore se necessário (caso dos 2 filhos). Depois de remover, percorre a árvore para verificar se está balanceada

CALCULAR FB
Retorna o fator de balanceamento de um nó com base em sua altura (alt. sub-árvore esq - alt. sub-árvore dir)

ALTURA
Retorna a altura de um nó

BALANCEAR
Realiza as rotações na árvore em caso de desbalanceamento. Se estiver desequilibrado pra esquerda, e se sim, se está desequilibrado por causa de muitos nós à esquerda. Para desequilíbrio à direita, são realizadas verificações similares.

Conclusão

As árvores AVL, que são balanceadas, vão ser melhor aproveitadas em cenários que exigem manipulação de uma grande massa de dados. Isso ocorre porque a árvore vai estar balanceada, ou seja, em sua estrutura, não terão sub-árvores degeneradas (que são praticamente listas encadeadas). Com isso, será necessário um número menor de iterações, o que resulta em um consumo menor de recursos computacionais.

Por outro lado, as árvores AVL podem ser desnecessárias em cenários que se trabalha com uma quantidade de dados menor, por conta de que os recursos utilizados por ela e a árvore de busca, pelo mesmo cenário, não terão diferenças significativas.

Ou seja, para definir qual vai ser melhor, depende do contexto. Se você tiver uma grande quantidade de informações, use a árvore AVL. Se você for trabalhar com quantidades menores de dados, em um ambiente acadêmico, por exemplo, pode ser mais interessante usar a árvore de busca, já que os recursos expendidos serão os mesmos e é mais fácil de implementar.

Síntese dos resultados obtidos

Operação INSERIR					
Árvore de busca			Árvore AVL		
número de elementos	CPU usage	Heap Memory Usage	número de elementos	Heap Memory Usage	CPU usage
100	1.7	12.6 mb	100	12.6 mb	0.5
500	1.9	12.6 mb	500	12.6 mb	1.1
1000	1.3	12.6 mb	1000	12.6 mb	1.6
10000	1.6	13.6 mb	10000	12.6 mb	1.1
20000	0.8	13.6 mb	20000	12.6 mb	1.0
100000	1.1	13.6	100000	13.6 mb	0.6
1000000	9.8	20.9 mb	1000000	18 mb	1.9
10000000	13.8	400 mb	10000000	59 mb	7.2

Operação BUSCA			
Árvore de busca		Árvore AVL	
número de elementos	CPU usage	Número de elementos	CPU usage
20000	1.0	20000	0.5
100000	1.3	100000	1.1
1000	1.1	1000	0.5
500	0.9	500	0.6
100	1.5	100	1.0
1000	0.5	1000	1.5

Operação REMOVER			
Árvore de busca		Árvore AVL	
número de elementos	CPU usage	Número de elementos	CPU usage
100000	2.1	100000	1.3
20000	2.0	20000	1.3
10000	2.1	10000	1.1
1000	0.7	1000	1.1
500	0.9	500	1.2
100	0.2	100	0.9