INSTITUTO DE CIÊNCIAS MATEMÁTICAS E DE COMPUTAÇÃO

UNIVERSIDADE DE SÃO PAULO

PROJETO DE ALGORITMOS E ESTRUTURA DE DADOS I

Manipulação de Dados em Memória Primária

Gabriel Van Loon Bodê da Costa Dourado Fuentes Rojas

Giovani Decico Lucafó

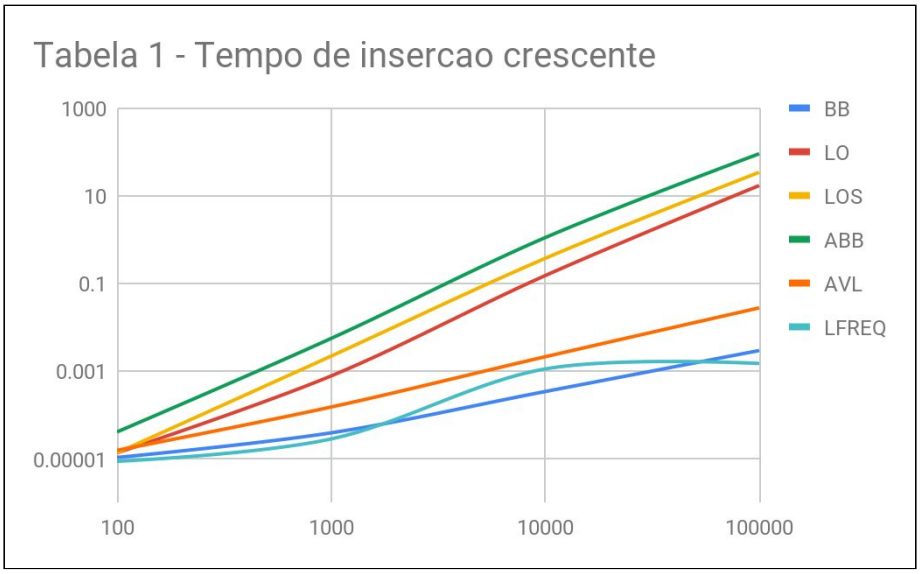
São Carlos - SP

2018

1. ANÁLISE DO TEMPO DE INSERÇÃO DE ELEMENTOS

1.1. INSERÇÃO EM ORDEM CRESCENTE

TAD x N Elementos	100	1,000	10,000	100,000
BB	0.000011	0.00004	0.000349	0.002998
LO	0.000014	0.00079	0.152589	17.186855
LOS	0.000014	0.002236	0.376629	34.415695
ABB	0.000042	0.005652	1.117814	91.007932
AVL	0.000016	0.000155	0.002163	0.028084
LFREQ	0.000009	0.000029	0.001142	0.001522



No teste de inserção crescente, as estruturas que se mostraram mais eficientes foram o Vetor de Busca Binária e a Lista Circular de Frequência, enquanto as piores, foram a Árvore Binária e a Lista Ordenada.

BB: Possui vantagem na inserção ordenada pois o custo de ajustar as posições dos elementos é muito baixa.

LO: Possui desvantagem na inserção ordenada pois precisa percorrer todos os outros elementos da lista para inserir o elemento desejado.

LOS: Assim como a LO, possui desvantagem nesse cenário por precisar percorrer todos os elementos da lista para inserir.

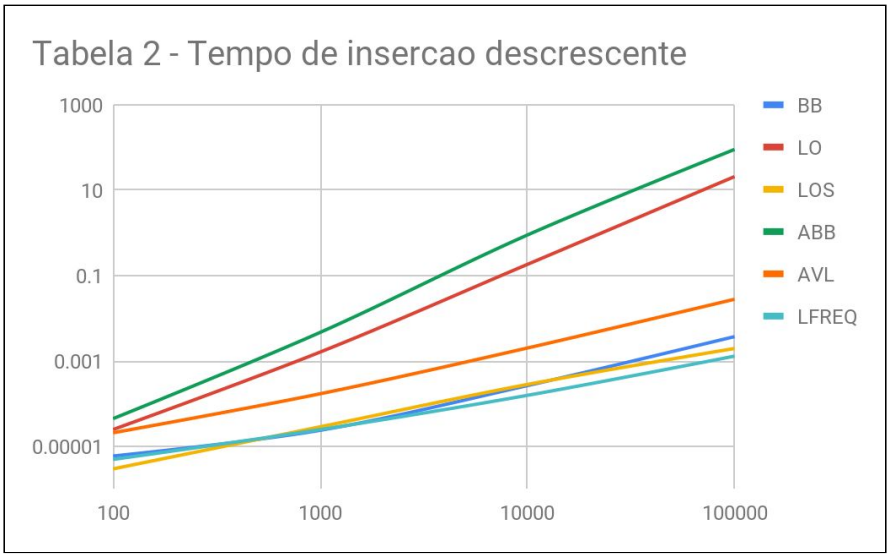
ABB: Em uma inserção crescente, a ABB se degrada em uma lista comum e, portanto, tem o pior desempenho pois precisa percorrer todos os elementos, efetuando mais operações que as listas lineares.

AVL: A capacidade de se rebalancear faz com que a AVL tenha a vantagem de ter sempre uma inserção rápida.

LFREQ: A LFREQ supera todas as outras por ser possível, com ela, inserir o elemento diretamente no final da lista.

1.2. INSERÇÃO EM ORDEM DECRESCENTE

TAD x N Elementos	100	1,000	10,000	100,000
BB	0.000006	0.000024	0.000267	0.003744
LO	0.000025	0.00163	0.183616	20.86575
LOS	0.000003	0.000029	0.000284	0.001975
ABB	0.000045	0.004707	0.895505	90.26974
AVL	0.000021	0.000172	0.002026	0.028077
LFREQ	0.000005	0.000025	0.000158	0.001307



No teste de inserção decrescente, as estruturas que se mostraram mais eficientes foram: A Lista Circular de Frequência e a Lista Ordenada com Sentinela; enquanto as piores foram: A Árvore Binária e a Lista Ordenada.

BB: Possui um bom desempenho, mesmo no cenário desordenado, pois utiliza um RadixSort (que utiliza 4 Counting Sorts) após o fim de todas as inserções.

LO: Possui desvantagem na inserção desordenada porque o custo para ordenar os elementos em uma estrutura encadeada é muito alto.

LOS: Ao contrário da LO, insere os elementos já na posição ordenada e, portanto, no caso da decrescente, sempre irá inserir o elemento no início, sendo esse um cenário ótimo.

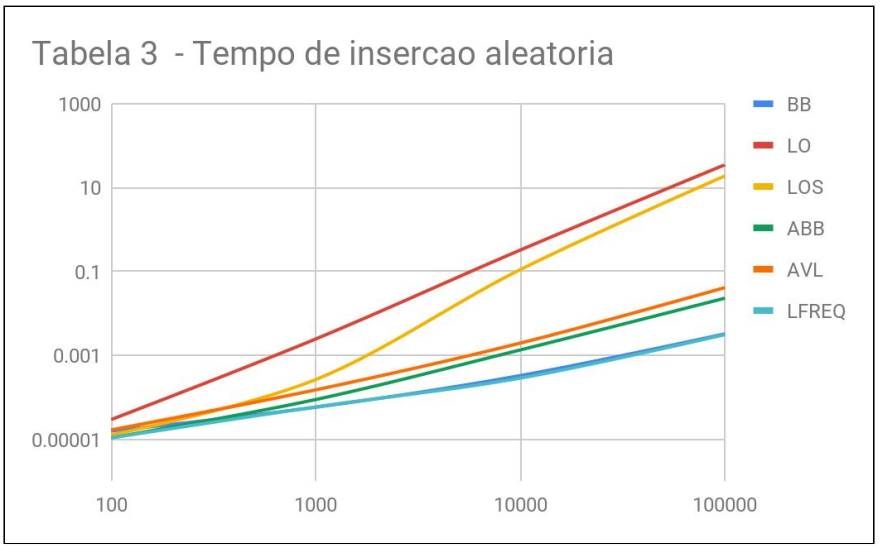
ABB: Em uma inserção decrescente, a ABB se degrada em uma lista comum e, portanto, tem o pior desempenho porque precisa percorrer todos os elementos, efetuando mais operações que as listas lineares comuns.

AVL: A capacidade de se rebalancear faz com que a AVL tenha a vantagem de ter sempre uma inserção rápida; porém, não a melhor de todas.

LFREQ: Assim como na inserção crescente, o fato de inserir elementos em custo constante é diferencial para sua boa performance.

1.3. INSERÇÃO EM ORDEM ALEATÓRIA

TAD x N Elementos	100	1,000	10,000	100,000
BB	0.000016	0.000059	0.000334	0.003298
LO	0.00003	0.002498	0.328815	35.386956
LOS	0.000013	0.000269	0.112563	19.180297
ABB	0.000011	0.00009	0.001368	0.023374
AVL	0.000017	0.000153	0.001992	0.041798
LFREQ	0.000011	0.00006	0.000292	0.003173



No teste de inserção aleatória, as estruturas que se mostraram mais eficientes foram: A Lista Circular de Frequência e o Vetor de Busca Binária; enquanto as piores foram: A Árvore Binária e a Lista Ordenada.

BB: Possui um bom desempenho porque utiliza um RadixSort (que utiliza 4 Counting Sorts) após o fim de todas as inserções. Seus resultados ficam muito próximos do LFREQ.

LO: Possui desvantagem na inserção porque o custo para ordenar os elementos em uma estrutura encadeada é muito alto.

LOS: Como insere os elementos na posição correta, seu custo é maior que a das árvores e do que a LFREQ porque na média, visita metade de seus elementos por inserção.

ABB: Possui um bom desempenho pois, como a inserção é aleatória, a árvore fica bem balanceada.

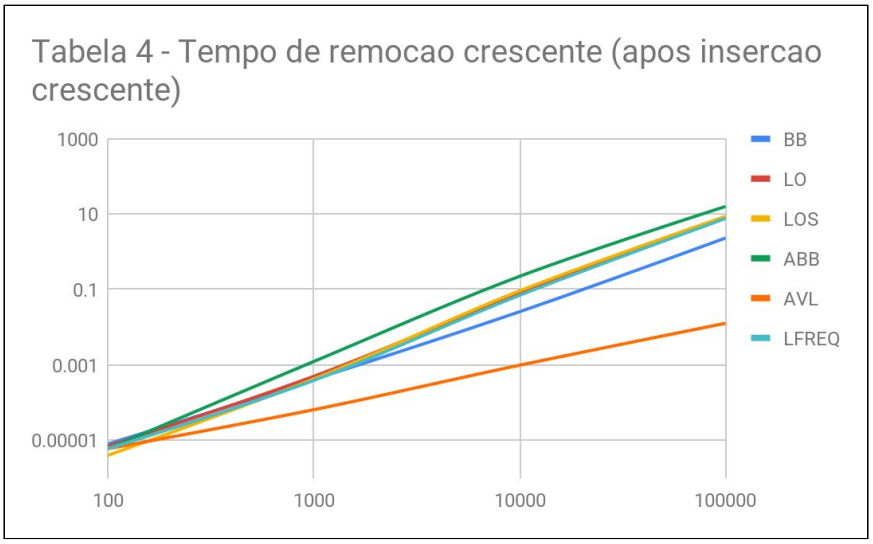
AVL: Similar à ABB, porém, seu desempenho é um pouco menor devido ao custo das rotações realizadas.

LFREQ: Como nas anteriores, vantagem por realizar inserção em tempo constante.

2. ANÁLISE DO TEMPO DE REMOÇÃO DE ELEMENTOS

2.1. REMOÇÃO EM ORDEM CRESCENTE

TAD x N Elementos	100	1,000	10,000	100,000
BB	0.000008	0.000415	0.025846	2.32506
LO	0.000007	0.000499	0.079814	8.537066
LOS	0.000004	0.000426	0.092253	8.667998
ABB	0.000006	0.001236	0.224425	15.980565
AVL	0.000006	0.000065	0.000984	0.012709
LFREQ	0.000006	0.000389	0.070358	7.667056



No teste de remoção crescente, as estruturas que se mostraram mais eficientes foram: A AVL e o LFREQ. Todas as outras empatam ou se aproximam como pior performance.

BB: Possui um bom desempenho para buscar pelo elemento desejado, porém, possui o custo de mover os elementos restantes da lista, sendo a remoção ordenada seu pior cenário.

LO: A LO, assim como as outras 3 listas lineares, tem um bom desempenho nesse cenário pois o elemento que será removido se encontra no início da lista.

LOS: Possui vantagem semelhante à da LO.

ABB: Está degradada em formato de lista e, portanto, possui um desempenho semelhante à elas, porém, pior devido ao custo adicional.

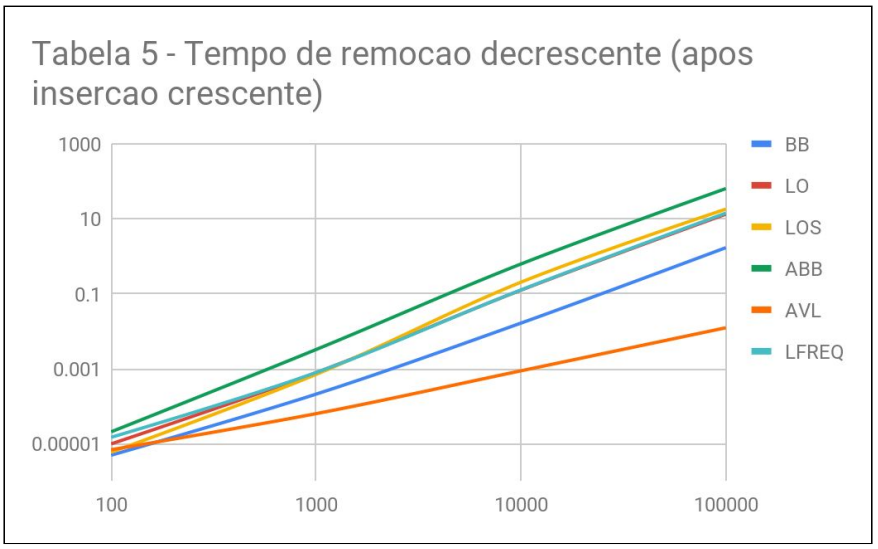
AVL: Possui um bom desempenho porque devido ao bom balanceamento, e, como a ordenação é crescente, maior parte das remoções serão de folhas.

LFREQ: Possui o desempenho bem próximo ao das outras listas, uma vez que os elementos

removidos foram mantidos no início devido às buscas.

2.2. REMOÇÃO EM ORDEM DECRESCENTE

TAD x N Elementos	100	1,000	10,000	100,000
BB	0.000005	0.000212	0.016611	1.704765
LO	0.00001	0.000792	0.125558	13.319824
LOS	0.000006	0.000711	0.204729	18.147194
ABB	0.000021	0.003279	0.625446	64.097014
AVL	0.000007	0.000064	0.000888	0.012486
LFREQ	0.000015	0.000798	0.12729	14.201342



No teste de remoção decrescente, as estruturas que se mostraram mais eficientes foram: A Árvore AVL e o Vetor de Busca Binária; enquanto as piores foram: A Árvore Binária e a Lista Ordenada com Sentinela.

BB: Possui um bom desempenho porque a remoção decrescente não afeta a ordenação dos elementos.

LO: Possui desvantagem nesse cenário porque precisa passar por todos os elementos em cada uma das remoções.

LOS: Assim como a LO, precisa passar por todos os elementos para realizar cada remoção.

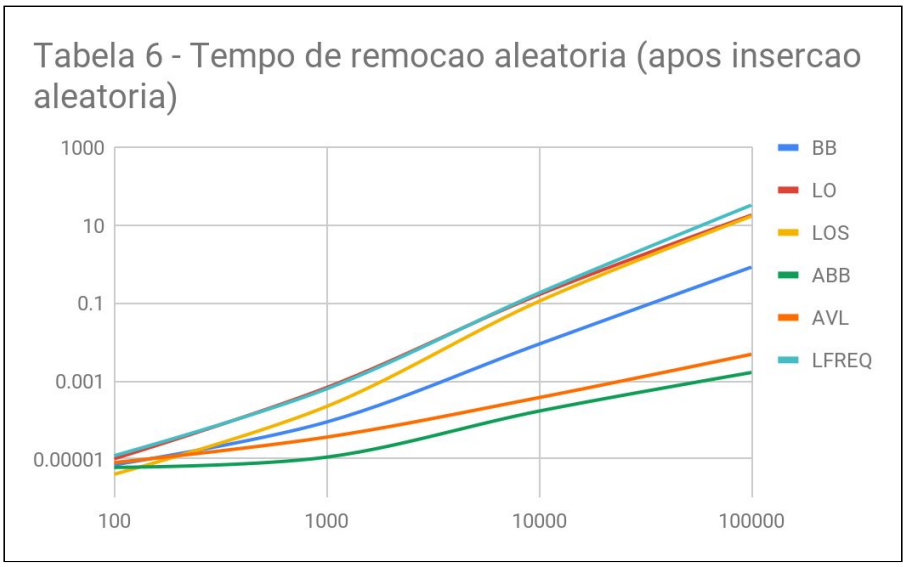
ABB: Como foi feita após uma inserção decrescente, a ABB se mostra similar à uma lista e possui as mesmas desvantagens que a LO e a LOS, porém, com mais custo de memória.

AVL: Possui vantagem devido à sua busca eficiente e ao fato da remoção não interferir nas subárvores não percorridas.

LFREQ: Como a remoção foi feita após as buscas, que desordenou os primeiros elementos, a LFREQ possui um custo pior do que as outras listas.

2.3. REMOÇÃO EM ORDEM ALEATÓRIA

TAD x N Elementos	100	1,000	10,000	100,000
BB	0.000007	0.000088	0.008868	0.843631
LO	0.00001	0.000677	0.165392	18.30705
LOS	0.000004	0.000221	0.111788	17.141212
ABB	0.000006	0.000011	0.000168	0.001649
AVL	0.000008	0.000036	0.000374	0.004854
LFREQ	0.000012	0.000624	0.182981	33.155617



No teste de remoção aleatória, as estruturas que se mostraram mais eficientes foram: A Árvore AVL e a Árvore de Busca Binária; enquanto as piores foram: A Lista Ordenada e a Lista Circular de Frequência.

BB: Possui um desempenho melhor do que as listas porque o custo de mover pequenos elementos, mesmo em grandes quantidades, é menor se comparado ao custo de percorrer as listas.

LO: Possui desvantagem nesse cenário porque precisa passar por diversos elementos em cada uma das remoções para encontrar o valor desejado e removê-lo.

LOS: Assim como a LO, precisa passar por muitos elementos em cada remoção.

ABB: Possui um bom desempenho pois, como a inserção é aleatória, o balanceamento é o melhor possível para esse estrutura.

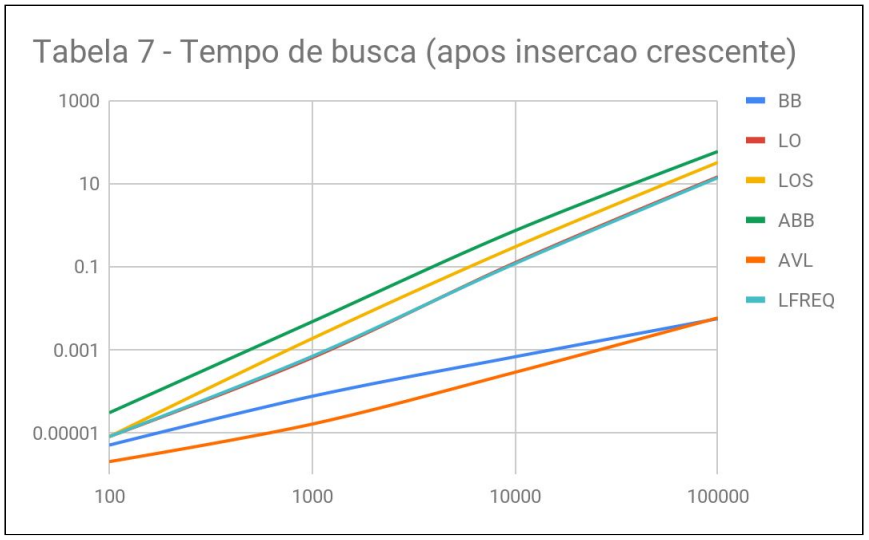
AVL: Possui a mesma vantagem da ABB, porém, com um pequeno custo extra decorrente das rotações de ajuste.

LFREQ: Assim como a LO, precisa passar por muitos elementos em cada remoção.

3. ANÁLISE DO TEMPO DE BUSCA DE ELEMENTOS

3.1. BUSCA EM ORDEM CRESCENTE

TAD x N Elementos	100	1,000	10,000	100,000
BB	0.000005	0.000075	0.000669	0.00553
LO	0.000008	0.000635	0.122927	14.389923
LOS	0.000008	0.001834	0.295747	31.864758
ABB	0.00003	0.004662	0.726129	58.489723
AVL	0.000002	0.000016	0.000283	0.00579
LFREQ	0.000008	0.000684	0.116672	13.698359



No teste de busca crescente, as estruturas que se mostraram mais eficientes foram: A Árvore AVL e o Vetor de Busca Binária; enquanto as piores foram: As listas e a Árvore BB.

BB: Possui um bom desempenho devido ao método de busca binária ser rápido.

LO: Possui desvantagem pois precisa passar por diversos elementos até encontrá-lo, percorrendo toda a lista no pior caso.

LOS: Assim como a LO, precisa passar por muitos elementos em cada busca.

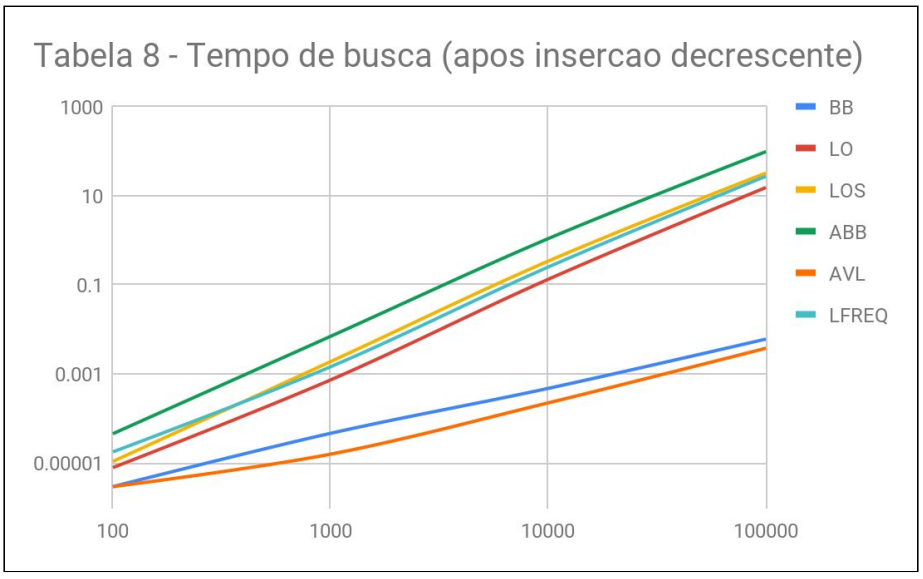
ABB: Como a árvore está degradada em uma lista, possui o pior desempenho nesse cenário.

AVL: Possui a melhor performance, pois utiliza de lógica análoga à da busca binária utilizada pelo Vetor de Busca Binária.

LFREQ: Assim como as outras listas, precisa passar por muitos elementos em cada busca e por toda a lista no pior cenário

3.2. BUSCA EM ORDEM DECRESCENTE

TAD x N Elementos	100	1,000	10,000	100,100
BB	0.000003	0.000047	0.000478	0.006088
LO	0.000008	0.000738	0.133304	15.232182
LOS	0.000011	0.001899	0.340379	32.146891
ABB	0.000046	0.007006	1.087367	96.927441
AVL	0.000003	0.000016	0.000226	0.003791
LFREQ	0.000018	0.001455	0.25071	27.223575



No teste de busca decrescente, as estruturas que se mostraram mais eficientes foram: A Árvore AVL e o Vetor de Busca Binária; enquanto as piores foram: As listas e a Árvore BB.

BB: Possui um bom desempenho devido ao método de busca binária ser rápido.

LO: Possui desvantagem porque precisa passar por quase todos os elementos em cada busca, sendo esse o pior cenário das listas ordenadas.

LOS: Assim como a LO, precisa passar por quase todos os elementos em cada busca, assim, possuindo baixo desempenho.

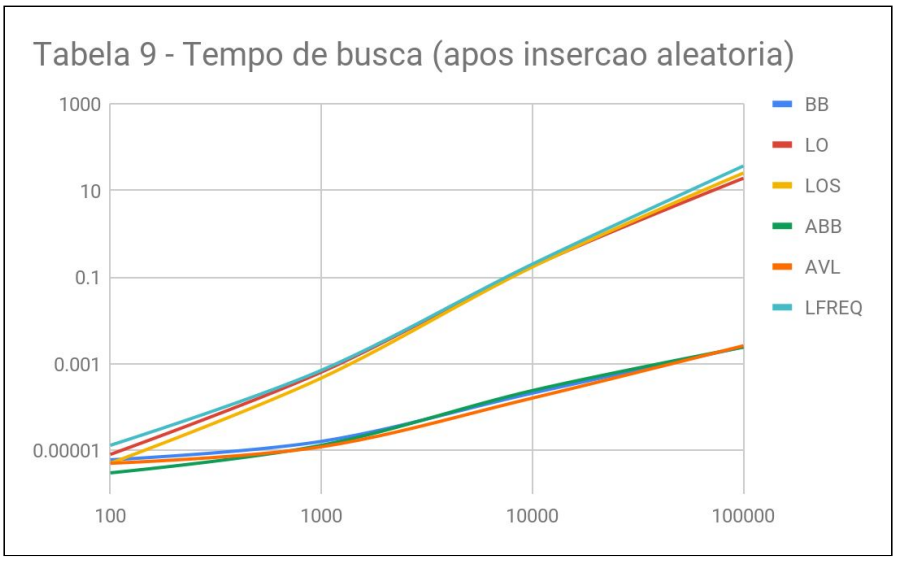
ABB: Como a árvore está degradada em uma lista, possui o pior desempenho nesse cenário.

AVL: Possui a melhor performance, pois utiliza de lógica análoga à da busca binária utilizada pelo Vetor de Busca Binária.

LFREQ: As buscas afetam a ordem de seus elementos, tendo portanto comportamento próximo ao das outras listas e as mesmas desvantagens.

3.3. BUSCA EM ORDEM ALEATÓRIA

TAD x N Elementos	100	1,000	10,000	100,100
BB	0.000006	0.000016	0.000208	0.002434
LO	0.000008	0.000637	0.177252	19.21446
LOS	0.000005	0.000465	0.171168	25.35419
ABB	0.000003	0.000013	0.000239	0.002405
AVL	0.000005	0.000012	0.00016	0.002592
LFREQ	0.000013	0.000702	0.19934	36.921734



No teste de busca aleatória, as estruturas que se mostraram mais eficientes foram: A Árvore AVL o Vetor de Busca Binária e a Árvore de Busca binário; enquanto as piores foram: As listas no geral.

BB: Possui um bom desempenho devido ao método de busca binária ser rápido.

LO: Possui desvantagem porque precisa passar por diversos elementos em cada busca, sendo o caso médio navegar por metade dos elementos em cada uma delas.

LOS: Assim como a LO, precisa passar por muitos elementos em cada busca.

ABB: Possui um bom desempenho pois, como a inserção é aleatória, o balanceamento é o melhor possível para esse estrutura.

AVL: Possui performance próxima à ABB e ao vetor BB. Sendo essas três estruturas ideias para esse cenário de buscas aleatórias.

LFREQ: Como as buscas são aleatórias, o fato de a lista ordenar por frequência pouco afeta a sua performance, tendo portanto as mesmas desvantagens das outras listas.

4. ANÁLISE DO CASO GERAL

Para o caso geral, a melhor estrutura de dados que pode ser utilizada é a da Árvore AVL. Isso se deve ao fato de sua capacidade de se manter balanceada permitir que os elementos sejam sempre acessados no melhor tempo possível.

As rotações que ela executa são simples e não despendem muita memória, dessa forma mesmo em seus piores cenários de inserção, como no caso da crescente e decrescente, o custo de memória não é tão afetado como ocorre com as outras estruturas.

Por fim, pela análise dos gráficos, vemos que o custo de tempo de suas operações sempre se encontra como melhor caso ou próximo dele.

5. CONCLUSÃO

O projeto foi importante para demonstrar como diferentes estruturas podem atuar de maneira tão discrepante dependendo do cenário em que elas estão contidas.

Vimos também que estruturas lineares tendem a se comportar de maneira pior do que estruturas em árvore quando estamos em cenários aleatórios e com grande quantidade de elementos. Sendo, portanto, essencial ao usuário saber quais ferramentas utilizar em sua aplicação.