# Relatório Projeto 3.3 AED


Filipe David Amado Mendes
**Nº Estudante:** 2020218797
***Login* no *Mooshak:*** 2020218797
**Turma:** PL3
**Docente Responsável:** Prof. Doutor Ivo Gonçalves

## Cenário 1: 10% de inserções

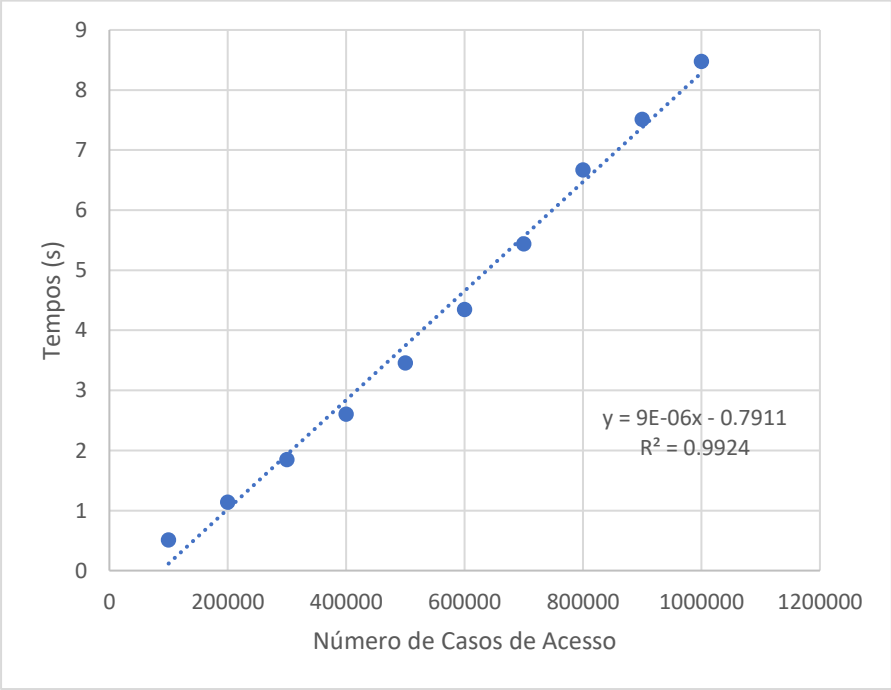
### Tabela

| Num Casos Acesso | Tempos(s) |
|---|---|
| 100000 | 0.512260437 |
| 200000 | 1.140298843 |
| 300000 | 1.851530075 |
| 400000 | 2.605567932 |
| 500000 | 3.456614494 |
| 600000 | 4.350059986 |
| 700000 | 5.442472219 |
| 800000 | 6.671000719 |
| 900000 | 7.513597488 |
| 1000000 | 8.476368189 |

### Gráfico

Equação: **y** = 9E-06x - 0.7911

$R^2 = 0.9924$



## Cenário 2: 90% de inserções

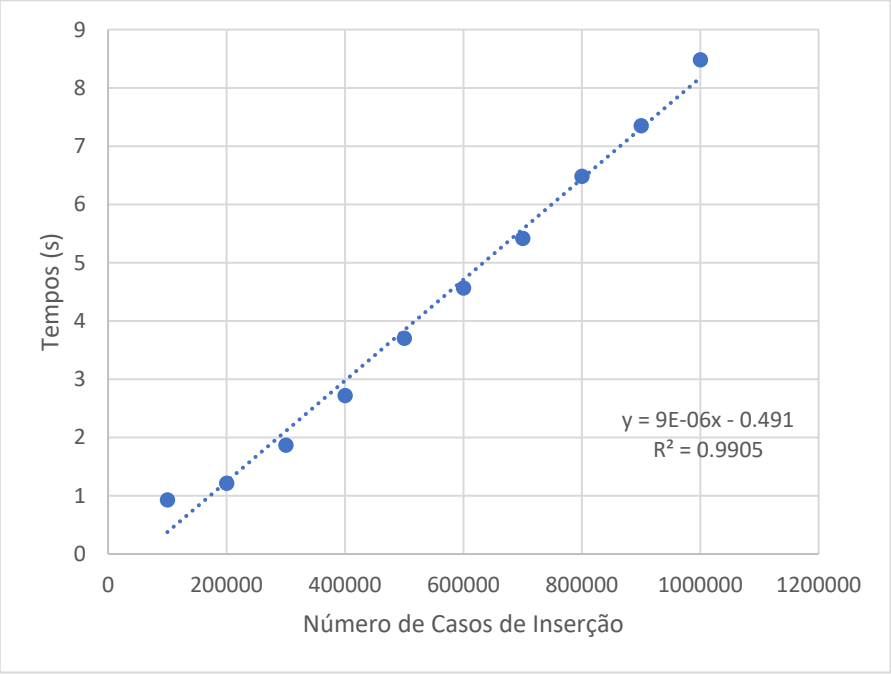
## Tabela

| Num Casos Inserção | Tempos(s) |
|---|---|
| 100000 | 0.930285454 |
| 200000 | 1.213748455 |
| 300000 | 1.870087385 |
| 400000 | 2.721723557 |
| 500000 | 3.701350927 |
| 600000 | 4.568072319 |
| 700000 | 5.415672302 |
| 800000 | 6.484638929 |
| 900000 | 7.352390766 |
| 1000000 | 8.480639458 |

## Gráfico

Equação: **y** = 9E-06x - 0.491

$R^2$ = 0.9905



## Introdução

Árvores binárias têm como função organizar dados de forma ordenada para que sejam acessíveis de forma rápida e eficiente. Para isso existem vários tipos de árvores binarias diferentes que se baseiam no mesmo princípio: os nós filhos à esquerda do nó pai são sempre menores que o nó pai e os nós à direita maiores que o mesmo. Para este projeto foi usado uma AVL Tree, uma árvore de pesquisa bastante eficiente para um grande número de consultas.

Uma árvore AVL recorre a um algoritmo de rotação dos nós para manter a árvore sempre equilibrada, isto é, todos os ramos têm de ter o mesmo tamanho, sendo apenas aceite um ramo ter mais um nível que os outros. Isto torna a pesquisa muito eficiente, porém a inserção de novos nós na árvore mais lenta, visto que tem que se verificar e possivelmente rodar as subárvores a cada inserção.

## Análise de Resultados

Para testar o algoritmo utilizado foram realizados dez testes iguais para dois cenários diferentes. No primeiro cenário de testes 10% dos comandos introduzidos foram inserções e os outros 90% consultas à árvore. Por outro lado, no segundo cenário foi feito o inverso, sendo 90% de inserções e apenas 10% de consultas.

Tal como previsto, no primeiro cenário o algoritmo utilizado foi bastante eficiente. O algoritmo tem complexidade linear, f(N) = N, como podemos através do gráfico e da equação da regressão linear, com um $R^2 = 0.9924$. Podemos também ver através da tabela que mesmo para o teste com 1000000 de comandos o tempo de execução do programa foi bastante baixo, o que mostra a eficácia deste tipo de árvore para pesquisas extensas.

Por outro lado, os resultados obtidos para o segundo cenário foram bastante surpreendentes. Visto que o algoritmo desenvolvido tem como objetivo a rapidez de pesquisa e não de inserção, os tempos obtidos, praticamente iguais aos do cenário 1, são chocantes. Apesar do número de inserções, o algoritmo mostra ainda assim uma complexidade linear, f(N) = N, com um erro muito baixo, $R^2 = 0.9905$. Podemos conferir pela tabela de tempos e pelo gráfico, que até mesmo no teste com 1000000 de casos o algoritmo implementado conseguiu um tempo praticamente igual no primeiro e no segundo cenário, mostrando assim uma eficiência muito grande.

## Conclusão

Com estes resultados, podemos concluir que o algoritmo desenvolvido é bastante eficiente tanto para os casos para os quais foi desenvolvido tanto como para outros. Isso mostra que, mesmo podendo não ser o melhor algoritmo de pesquisa, será bastante eficaz para situações em que seja necessário tanto um grande número de inserções como de consultas na árvore.