Relatório

Projeto Prático nº1

Algoritmos e Estruturas de Dados

Licenciatura em Engenharia Informática

2016/2017

João Moreira - 2015230374

Ricardo Tavares – 2014230130

Introdução

Este primeiro projeto prático consiste em idealizar e construir três abordagens diferentes usando estruturas de dados que, lendo um ficheiro com vários dados sobre um país, possam armazená-los e nos permitam manipular essa mesma informação, de forma a fazermos um estudo comparativo do seu desempenho face a diversas necessidades.

Nas páginas que se seguem encontra-se a explicação de cada uma das estruturas e uma descrição geral do trabalho realizado, bem como os resultados obtidos e respetiva análise.

Interface:

A interface é igual para as três estruturas, sendo esta um menu em terminal que nos permite escolher de entre as seguintes opções: pesquisa, adicionar elementos, editar, eliminar, imprimir os elementos da estrutura e sair.

A pesquisa funciona através do nome ou abreviatura do país e faz print das suas informações, ou, caso não exista, print de uma mensagem a informar isso mesmo.

O comando adicionar entrada pede o nome, abreviatura, ano e valor (para adicionar mais valores recorremos ao comando edição), depois cria a lista de anos e insere o valor nesse mesmo ano. No final adiciona um nó com esta informação à estrutura.

A edição funciona através de uma pesquisa do país que pretendemos editar, faz print do mesmo, após isso, escolhemos que tipo de edição queremos fazer (nome, abreviatura ou dados).

O remover procura o nome do país que pretendemos remover e apaga-o da estrutura.

A opção de imprimir a lista chama uma função que está definida na estrutura para listar os seus elementos.

A opção de saída é apenas um "break" do ciclo "while true".

Abordagem nº 1

Lista Ligada

Breve descrição:

Composta por nós que têm um ponteiro para o seguinte. As listas estão ordenadas por ordem de inserção.

Node.py

Este ficheiro contem a definição do nó. Para armazenar os dados recorremos a uma variável para o nome, outra para a abreviatura e uma lista para guardar os valores de cada ano.

LinkedList.py

Aqui está definida a lista e as funções que permitem inserir dados, alterar o seu tamanho, remover dados, imprimir a lista e apagar dados que estejam em duplicado.

LinkedListRun.py

A partir deste ficheiro podemos correr o programa, lendo os dados de um ficheiro .csv. Após o ficheiro ser carregado para a lista, temos a interface onde podemos escolher uma das opções definidas.

Abordagem nº 2

Árvore AVL

Breve descrição:

A AVL é uma árvore binária de pesquisa balanceada. O balanceamento é feito através de operações que são rotações (Left rotation, Right rotation, Left-Right rotation e Right-Left rotation).

Node.py

Definido da mesma forma que na Lista Ligada.

AVL.py

Aqui está definida a árvore (altura e atualização da altura, balanceamento e sua atualização, e todas as rotações) e as funções que permitem inserir dados, remover dados eimprimir a árvore.

AVLRun.py

A partir deste ficheiro podemos correr o programa, lendo os dados de um ficheiro .csv. Após o ficheiro ser carregado para a árvore, temos a interface onde podemos escolher uma das opções definidas.

Abordagem nº3

Splay Tree

Breve descrição:

Esta é uma árvore mais simples que a AVL visto que não força equilíbrio e não mantém informação da altura. Qualquer alteração não perturba o equilíbrio, simplesmente leva a árvore a ajustar-se mediante o nó que foi acedido (SPlaying)

Node.py

Definido da mesma forma das outras abordagens.

Splay.py

Aqui está definida a arvore (raiz e headers) e as funções que permitem inserir dados, remover dados e imprimir a árvore, assim como a função de SPlaying.

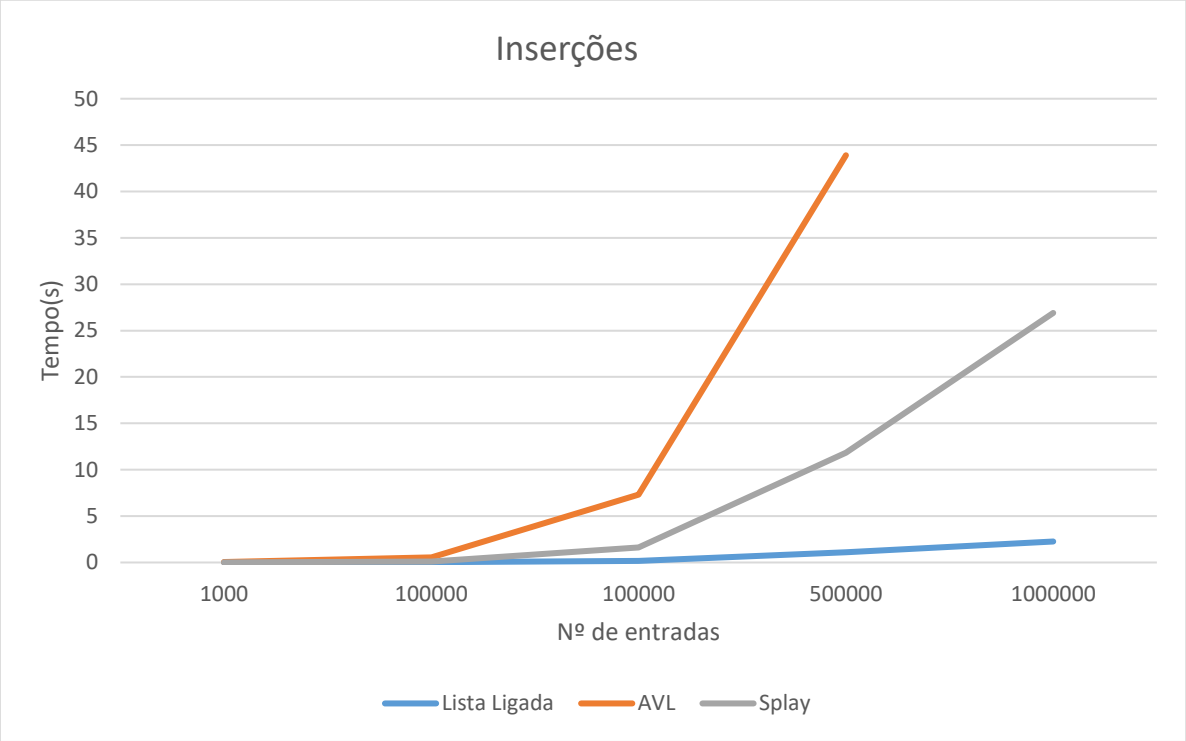
SplayRun.py

A partir deste ficheiro podemos correr o programa, lendo os dados de um ficheiro .csv. Após o ficheiro ser carregado para a árvore, temos a interface onde podemos escolher uma das opções definidas.

Análise de Complexidade

Inserção

Para testar o tempo de inserção criámos ficheiros de texto com nomes aleatórios de países, que foram depois carregados por cada estrutura.



	1000	100000	100000	500000	1000000
Lista Ligada	0,00183	0,0165	0,1837	1,1072	2,2582
AVL	0,04889	0,5588	7,3223	43,9021	
SPlay	0,0085	0,1082	1,6258	11,8477	26,9044

Visto que a Lista Ligada não tem qualquer tipo de ordenação o seu tempo de inserção é proporcional ao número de entradas, tem por isso complexidade O(1).

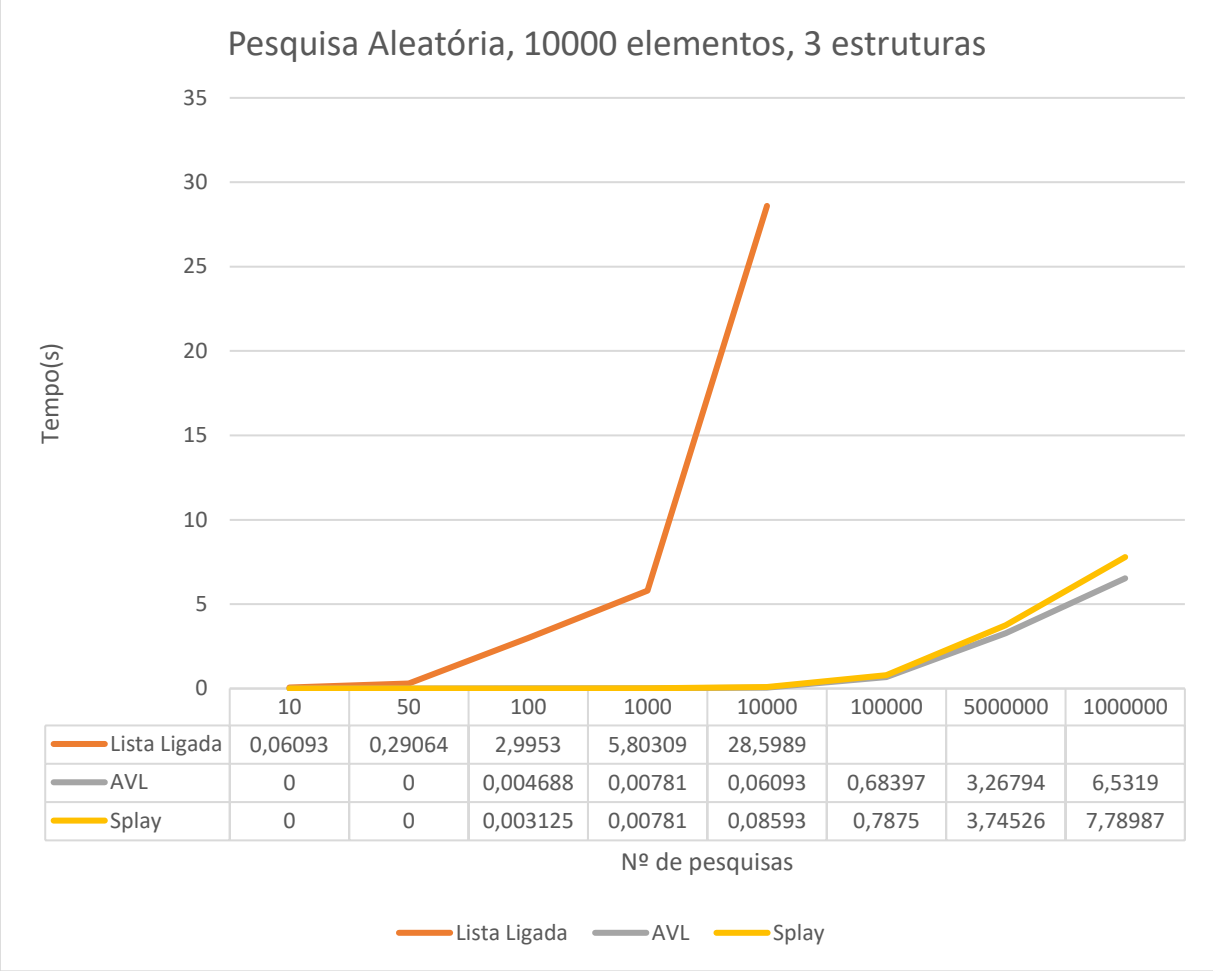
A árvore AVL vai fazendo rotações a cada inserção de forma a manter-se balanceada o que pesa nos tempos de inserção mas a torna eficiente noutras operações. Tem complexidade O(log(n)).

As árvores SPlay são semelhantes às AVL mas acabam por ser ligeiramente mais eficientes pois não fazem as rotações das AVL e o último elemento é sempre posto na raiz. Têm também complexidade O(log(n)).

Pesquisa

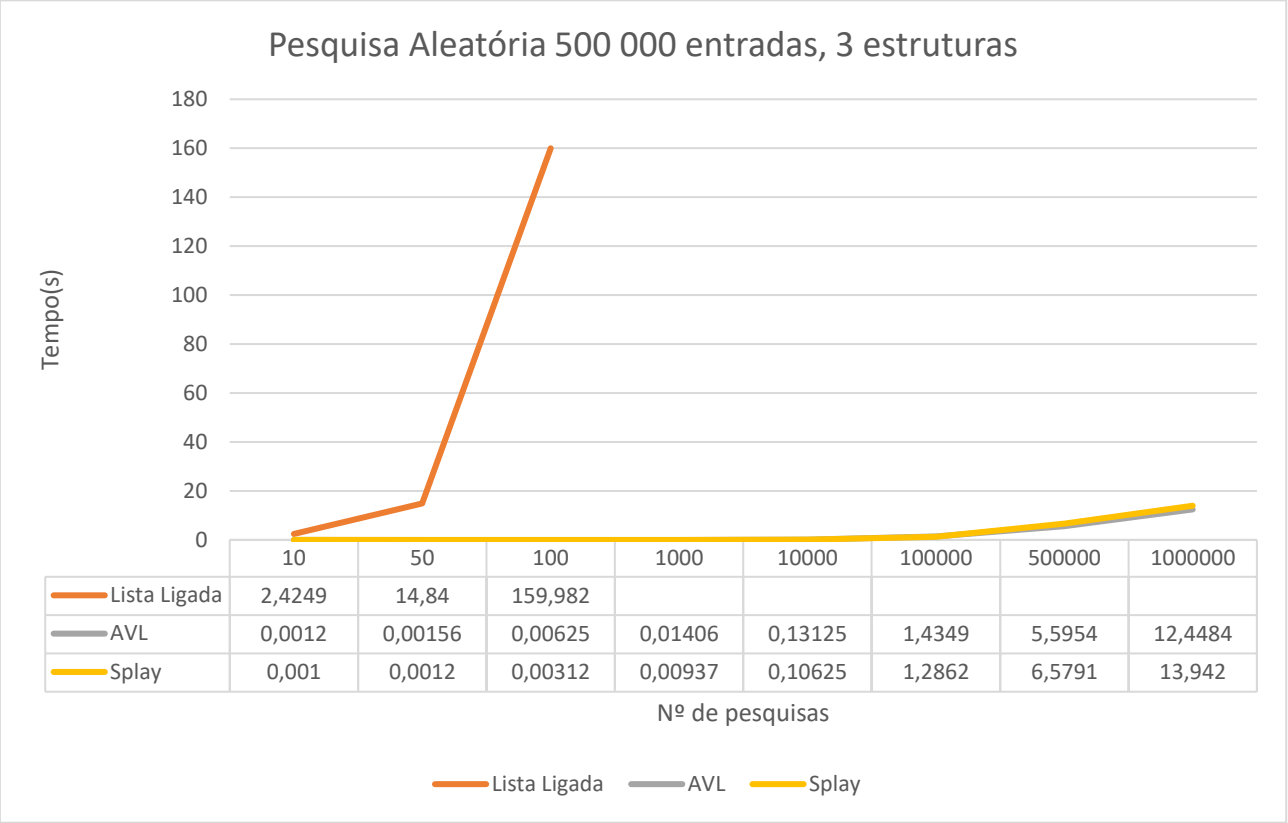
Para a pesquisa fizemos um estudo de 3 formas:

- Pesquisa Aleatória em estruturas de 10000 elementos:

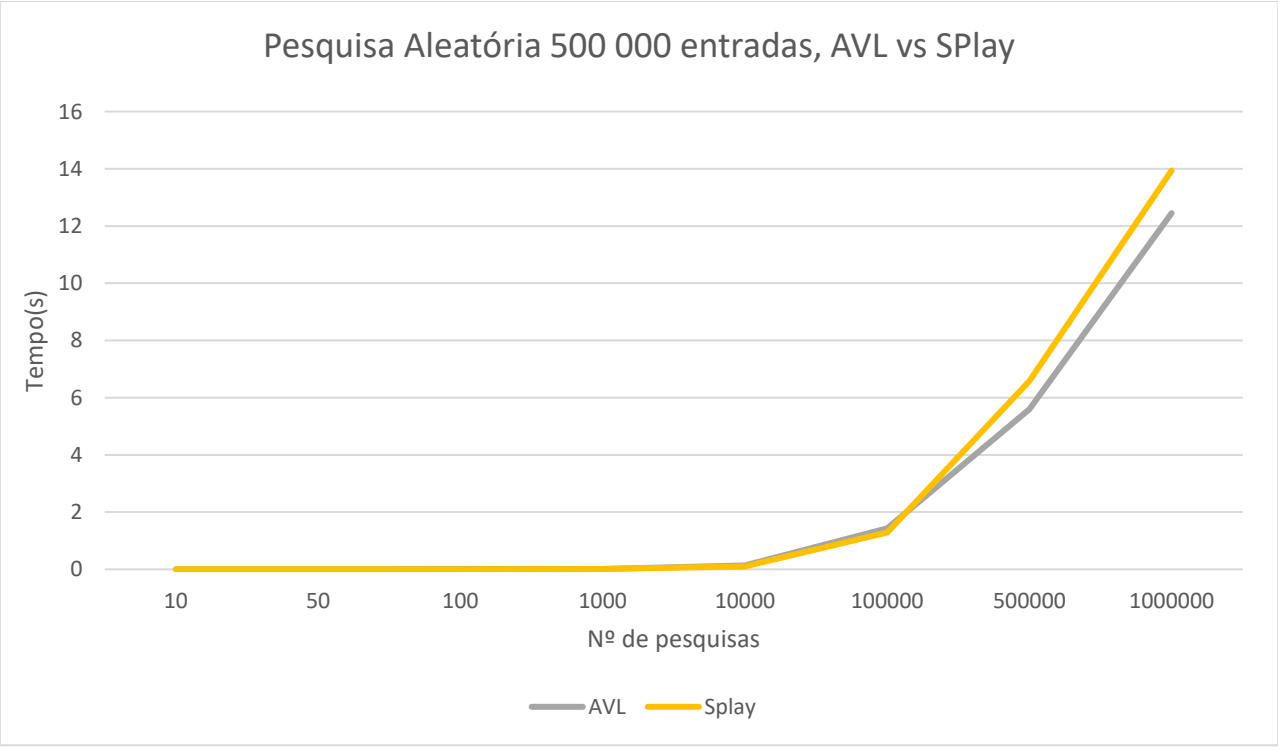


	10	50	100	1000	10000	100000	5000000	1000000
Lista Ligada	0,06093	0,29064	2,9953	5,80309	28,5989			
AVL	0	0	0,004688	0,00781	0,06093	0,68397	3,26794	6,5319
Splay	0	0	0,003125	0,00781	0,08593	0,7875	3,74526	7,78987

- Pesquisa Aleatória em estruturas de 500000 elementos:



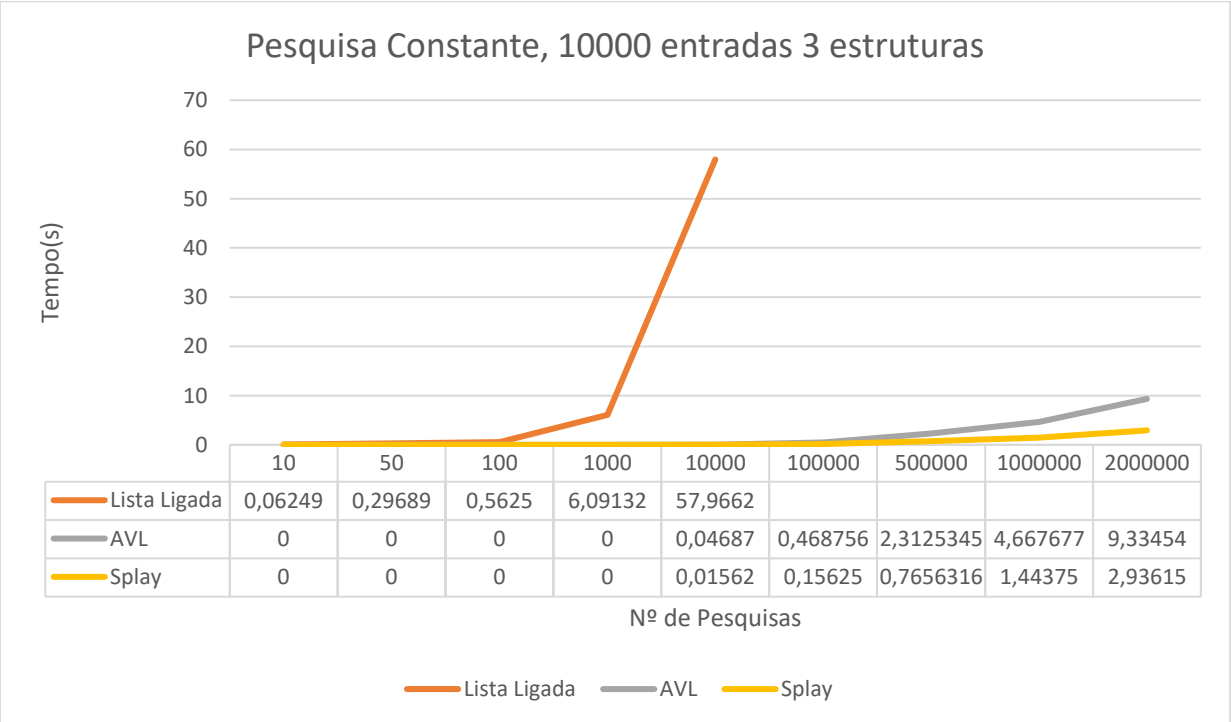
	10	50	100	1000	10000	100000	500000	1000000
Lista Ligada	2,4249	14,84	159,982					
AVL	0,0012	0,00156	0,00625	0,01406	0,13125	1,4349	5,5954	12,4484
Splay	0,001	0,0012	0,00312	0,00937	0,10625	1,2862	6,5791	13,942



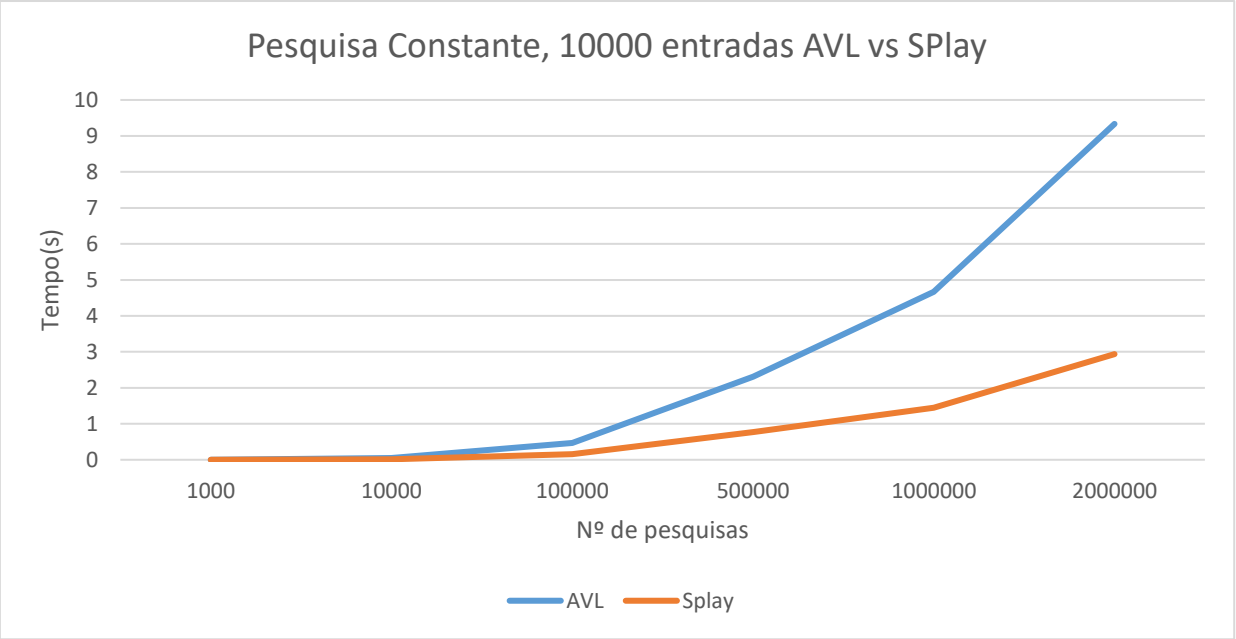
Como podemos observar, o tamanho da estrutura não altera a complexidade da pesquisa, neste caso a Lista Ligada perde muito em comparação às outras estruturas pois a cada nova pesquisa tem de percorrer a lista toda desde o ínicio até encontrar o elemento que procura, se este estiver para o fim da lista torna-se um processo muito pesado (O(n)).

As duas árvores têm tempos muito semelhantes, com um ligeiro avanço na SPlay. Ambas têm mecanismos de ordenação e balanceamento, sendo que as SPlay não foram feitas para pesquisas completamente aleatórias, ambas têm também complexidade O(log(n)).

- Pesquisa Constante em estruturas de 10000 elementos:



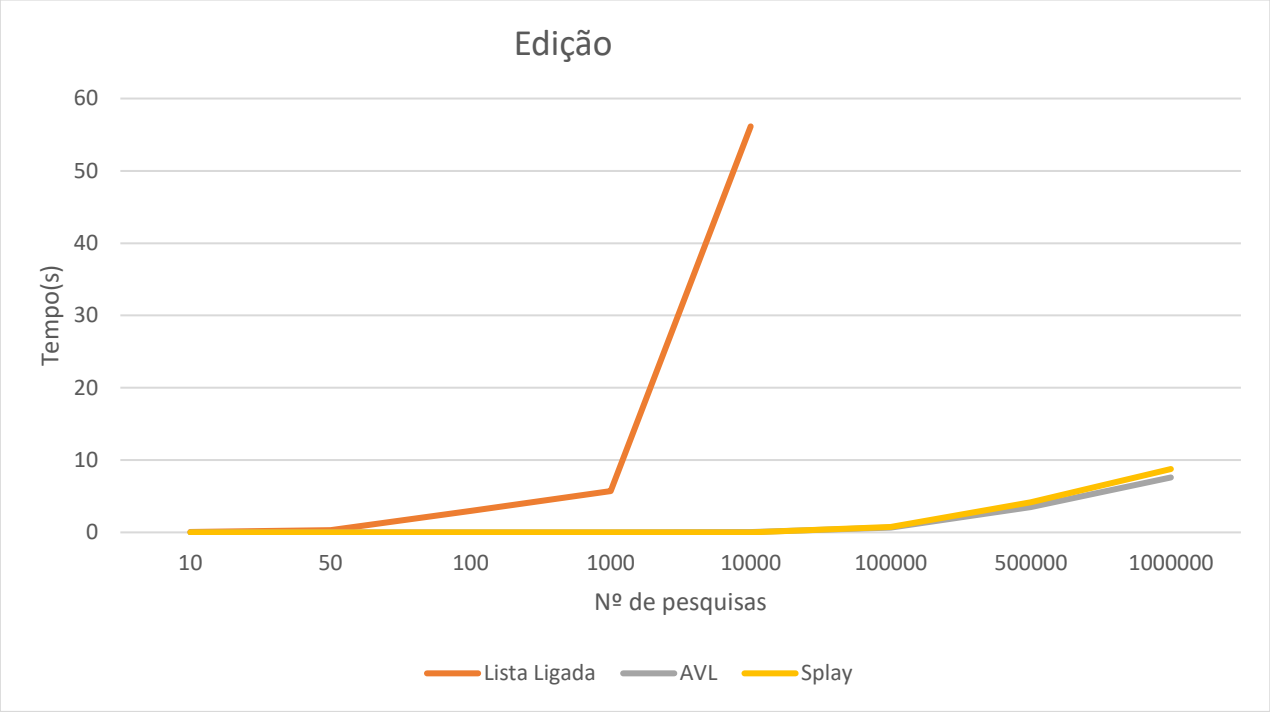
	10	50	100	1000	10000	100000	500000	1000000	2000000
Lista Ligada	0,06249	0,29689	0,5625	6,09132	57,9662				
AVL	0	0	0	0	0,04687	0,468756	2,3125345	4,667677	9,33454
Splay	0	0	0	0	0,01562	0,15625	0,7656316	1,44375	2,93615



Com uma pesquisa constante torna-se claro que as SPlay têm melhor desempenho, o facto a cada vez que um nó é acedido é "empurrado" para cima torna múltiplas pesquisas do mesmo elemento muito vantajosas em árvores SPlay, fazendo com que os últimos elementos a serem procurados estejam no topo da árvore.

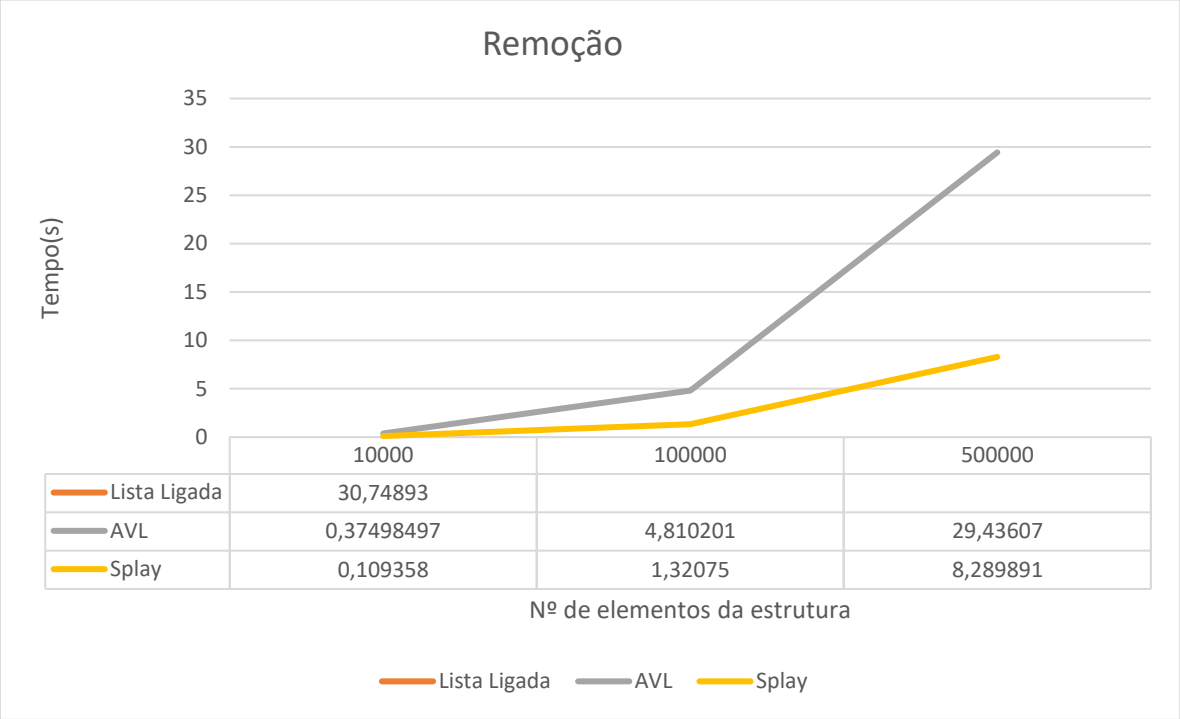
A vantagem das árvores AVL é que garantem uma pesquisa rápida (O(log(n))), enquanto que as SPlay garantem que no pior dos casos a operação demore O(n log(n)), podendo ser mais rápidas que as AVL, mas também bastante mais lentas.

Edição



A edição é idêntica à pesquisa, com a diferença de dar a opção de mudar um atributo do nó. Isto só vai ter impacto nas AVL e se o nome ou abreviatura mudarem de forma significativa , o pode levar a várias rotações para voltar a balancear a árvore.

Remoção



	10000	100000	500000
Lista Ligada	30,74893		
AVL	0,37498497	4,810201	29,43607
Splay	0,109358	1,32075	8,289891

As árvores são muito mais eficientes, mas as AVL deveriam ter vantagem sobre as SPlay pois garantem uma complexidade de remoção constante, enquanto que as SPlay podem ter operações individuais muito pesadas (mesma questão da pesquisa)

Numa Lista Ligada (simples) para remover um nó é preciso navegar a lista e fazer com que o ponteiro do nó anterior aponte para o seguinte, não havendo melhor forma de saber onde na lista é que está o nó que se quer remover. Tem por isso um mau desempenho com complexidade O(n).

Conclusões finais

Feito o estudo a cada estrutura, temos uma conclusão que contradiz os nossos resultados, a árvore AVL é das 3 a estrutura mais equilibrada e que melhor se adapta e diversas necessidades. O facto de se manter sempre balanceada permite tempos constantes e não lentos, o que torna numa estrutura robusta. O facto de não ser evidente nos nossos resultados pode ser devido aos inputs aleatórios que não sejam muito diferentes entre si, beneficiando assim as SPlay.

A simplicidade da Lista Ligada é evidente, pode servir como uma estrura auxiliar de fácil implementação, para guardar poucos elementos, mas operações mais exigentes não podem depender dela.

A árvore SPlay tem vantagens evidentes no acesso aos nós mas na prática não existem muitas situações de inputs repetidos ou múltiplas pesquisas sobre o mesmo assunto, o que a torna pouco fiável.

Percentagem de trabalho:

João Moreira – 30%

Ricardo Tavares – 70%