UNIVERSIDADE FEDERAL DO RIO GRANDE DO NORTE - UFRN
CENTRO DE ENSINO SUPERIOR DO SERIDÓ - CERES
BACHARELADO EM SISTEMAS DE INFORMAÇÃO - BSI

JHONATAS ISRAEL DA COSTA LAURENTINO

Algoritmos de Árvores Binárias

Um Relatório Sobre Os Alguns dos Principais Algoritmos de Árvores Binárias Vistos em Estrutura de Dados

CAICÓ/RN
2019

JHONATAS ISRAEL DA COSTA LAURENTINO

Algoritmos de Busca em Listas Encadeadas e Árvores Binárias

Um Relatório Sobre Os Alguns dos Principais Algoritmos de Árvores Binárias Vistos em Estrutura de Dados

Relatório entregue à matéria Estrutura de Dados, no curso de Bacharelado em Sistemas de Informação, matéria está ministrada pelo Prof. Dr. João Paulo de Souza Medeiros, na Universidade Federal do Rio Grande do Norte (UFRN).

CAICÓ/RN
2019

Resumo

O presente relatório tem como objetivo apresentar estudos coletados durante a segunda unidade da disciplina de estruturas de dados ministrada pelo professor Dr. João Paulo de Souza Medeiros na Universidade Federal do Rio Grande no Norte. Serão apresentados gráficos e informações acerca de alguns algoritmos de busca em Árvore Binária, Árvore Balanceada e Tabela de Dispersão, além de também como conteúdo extra, uma implementação em Lista Encadeada. Para implementação dos códigos foi utilizado o sistema operacional Linux Ubuntu, a linguagem de programação C, o compilador GCC. A ferramenta GNUPLOT foi utilizada para criação dos gráficos.

Abstract

1 Introdução

Um Algoritmo de Busca em termos gerais é aquele que toma um problema como entrada e retorna uma solução, geralmente após resolver um número possível de soluções.

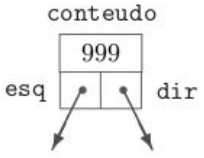
Uma solução, no aspecto de função intermediária, é um método o qual um algoritmo externo, ou mais abrangente, utilizará para solucionar um determinado problema. Esta solução é representada por elementos de um espaço de busca, definido por uma fórmula matemática ou um procedimento, tal como as raízes de uma equação com números inteiros variáveis, ou uma combinação dos dois, como os circuitos hamiltonianos de um grafo.

Já pelo aspecto de uma estrutura de dados, sendo o modelo de explanação inicial do assunto, a busca é um algoritmo projetado para encontrar um item com propriedades especificadas em uma coleção de itens. Os itens podem ser armazenadas individualmente, como registros em um banco de dados.

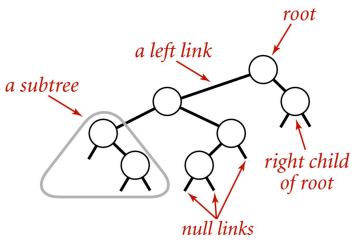
A maioria dos algoritmos estudados por cientistas da computação que resolvem problemas são algoritmos de busca. Por causa disso, o presente trabalho tratará da discussão de como se comporta um Algoritmo de Busca em determinadas estruturas de dados, mais especificamente a lista encadeada, Árvore Binária, árvore balanceada e tabela de dispersão.

1.1 O que é uma Árvore Binária?

Uma Árvore Binária (= binary tree) **BT** é um conjunto de registros que satisfaz certas condições. Os registros serão chamados nós ou células. Cada nó tem um endereço. Suporemos por enquanto que cada nó tem apenas três campos: um número inteiro e dois ponteiros para nós. Os nós podem, então, ser definidos assim:

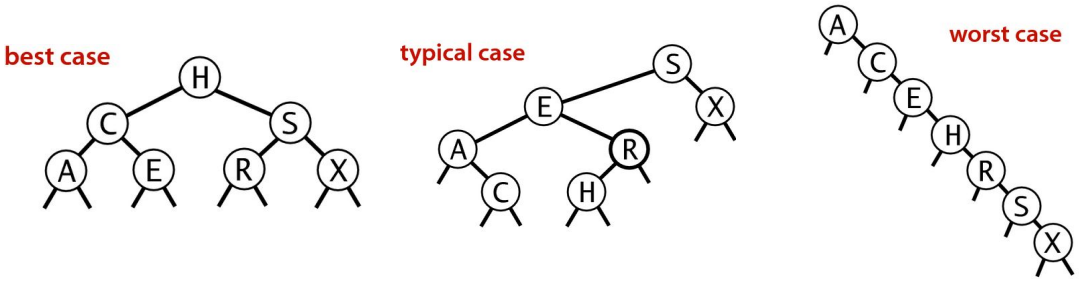


Cada nó tem no máximo dois filhos: um esquerdo e um direito.



Anatomy of a binary tree

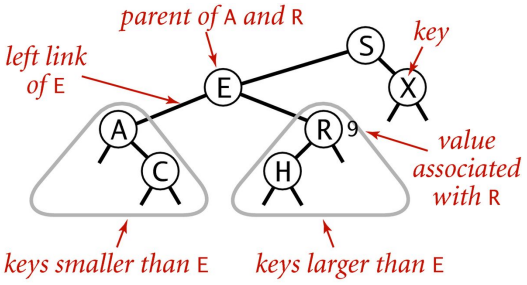
A raiz (root) é o único nó que não é filho de outro. A árvore é vazia se **root == null.** As BTs são estruturas recursivas, onde cada nó é raiz de uma sub-BT. A profundidade (depth) de um nó de uma BT é o número de links no caminho que vai da raiz até o nó. A altura (height) de uma BT é o máximo das profundidades dos nós, ou seja, a profundidade do nó mais profundo. Uma BT com N nós, tem altura no máximo N−1 e no mínimo $\lfloor \lg N \rfloor$. Se a altura estiver perto de lg N, a BT é balanceada.



O comprimento interno (internal path length) de uma BT é a soma das profundidades dos seus nós, ou seja, a soma dos comprimentos de todos os caminhos que levam da raiz até um nó. (Esse conceito é usado para estimar o desempenho esperado de TSs implementadas com BSTs).

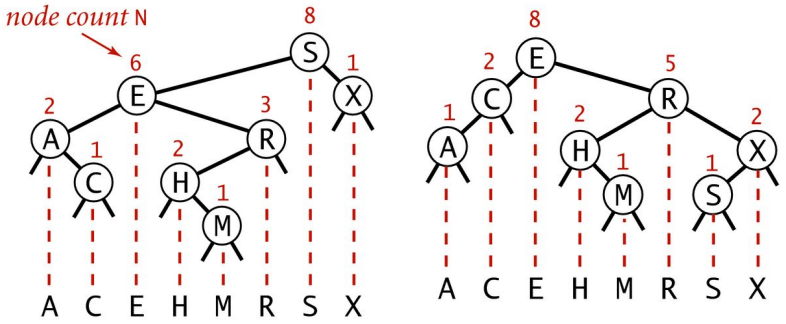
1.2 O que é uma Árvore Binária de busca?

Uma Árvore Binária de busca (BST) é um tipo especial de BT: para cada nó x, todos os nós na subárvore esquerda de x têm chave menor que x.key e todos os nós na subárvore direita de x têm chave maior que x.key. As chaves de uma BST precisam ser comparáveis.



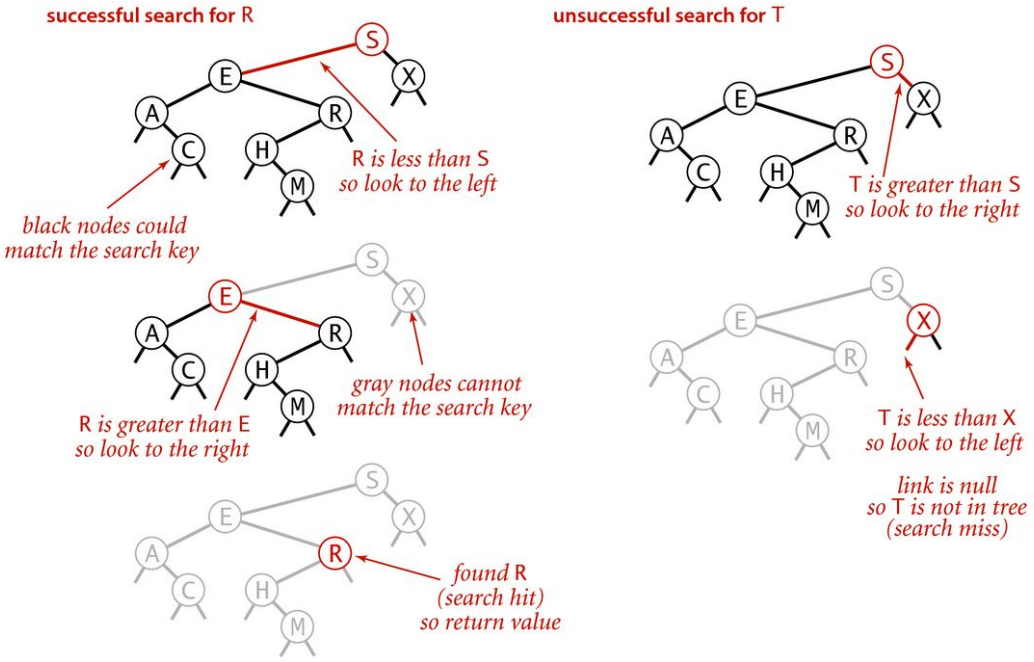
Anatomy of a binary search tree

Exemplo: Duas BSTs que representam o mesmo conjunto de chaves:



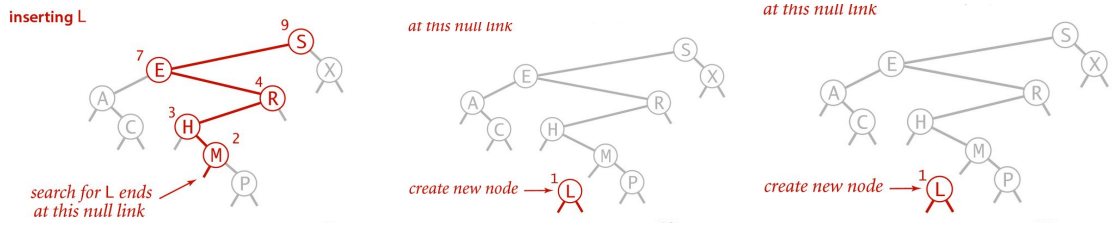
Two BSTs that represent the same set of keys

A busca (GET) em uma BST: o processo é muito parecido com a busca binária em um vetor ordenado:



Search hit (left) and search miss (right) in a BST

A inserção (put) em uma BST: o processo é muito mais barato que a inserção em um vetor ordenado, pois não envolve movimentação de dados.



2. Busca em Estruturas de Dados

2.1 Busca em Árvore Binária

Como dito no capítulo 1 uma Árvore Binária de Busca é uma Estrutura de Dados baseada em nós sendo que todos os valores numéricos à esquerda do nó são inferiores a ele e todos os valores numéricos à direita são maiores do que ele, essencialmente. O objetivo dessa estrutura é permitir que uma busca seja feita em ordem O ($log_2 n$).

Uma busca realizada em uma Árvore Binária tem tempo de execução O (n) em seu pior caso, O $(log_2 n)$ no médico caso, e tem tempo de execução O (1) em seu melhor caso.

A cada passo que é executado, é possível garantir que nenhuma outra parte da árvore contém a chave sendo buscada, o procedimento encerra quando o nó com x é encontrado, senão, chega-se a NULL, mostrando assim que o valor não está na árvore, formando assim o pior caso.

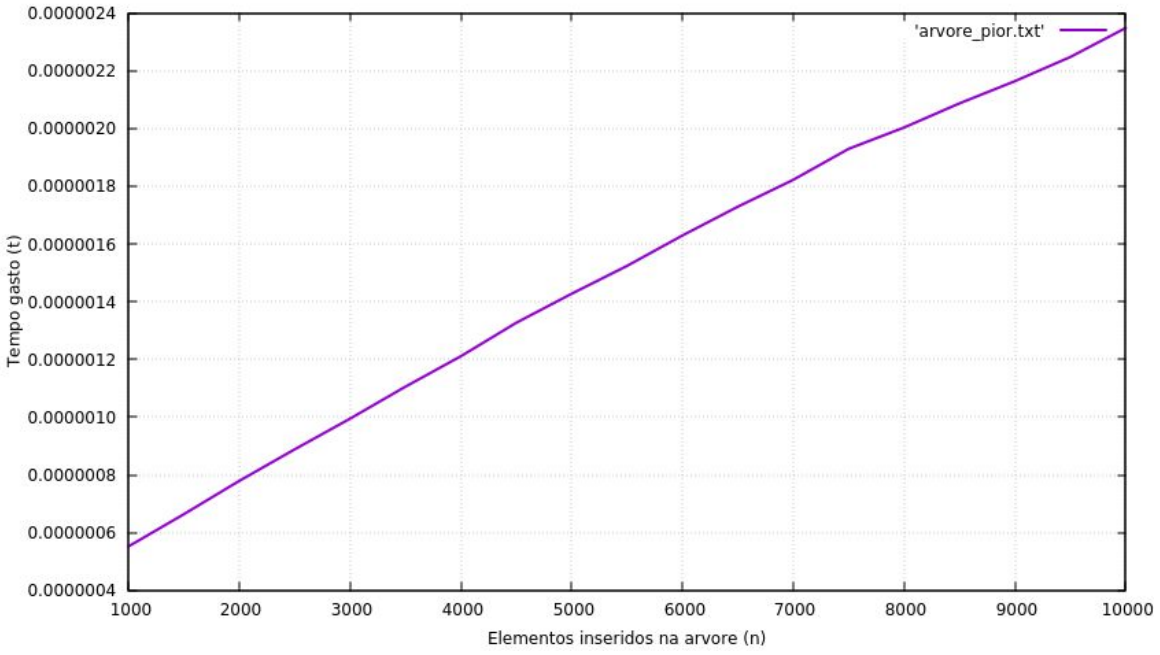


Figure 1: Tempo de execução do pior caso da árvore de busca binária

O pior caso ocorre quando árvore é preenchida em ordem crescente ou decrescente tornando-a semelhante à uma lista encadeada, já que todos elementos estarão apenas de um lado. Neste caso, seu tempo de execução é linear O (n). Por causa disso, na geração do gráfico a lista foi preenchida em ordem crescente e o elemento procurado não foi encontrado.

O melhor caso da busca em uma Árvore Binária, assim como quase todas outras estruturas, ocorre quando o elemento buscado é encontrado logo na primeira verificação, portanto sendo de tempo de execução constante O (1). Seu melhor caso pode ser equacionado como mostrado

$$T_b(n) = C_1 + C_2 + C_3$$

É possível visualizar o gráfico de tempo de execução na figura 6

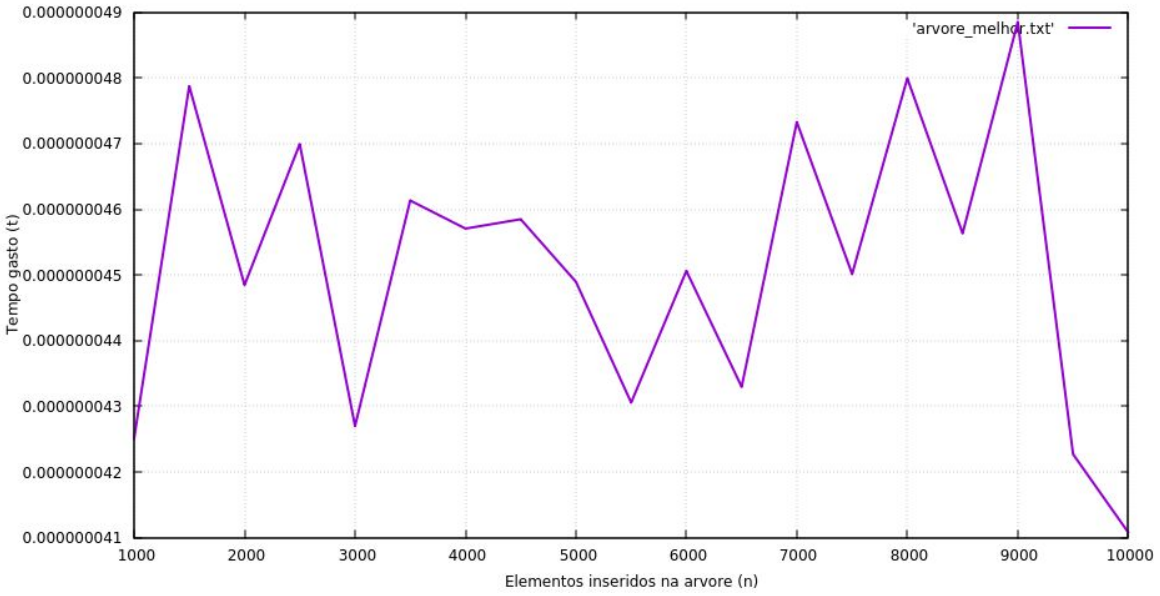


Figure 2: Tempo de execução do melhor caso da árvore de busca binária

Para geração do gráfico a árvore foi preenchida de forma aleatória e o elemento procurado sempre foi encontrado na primeira verificação.

Para o caso médio esperado, a árvore foi preenchida de forma aleatória e foi procurado um número aleatório.

Na figura 3 é possível visualizar seu tempo de execução.

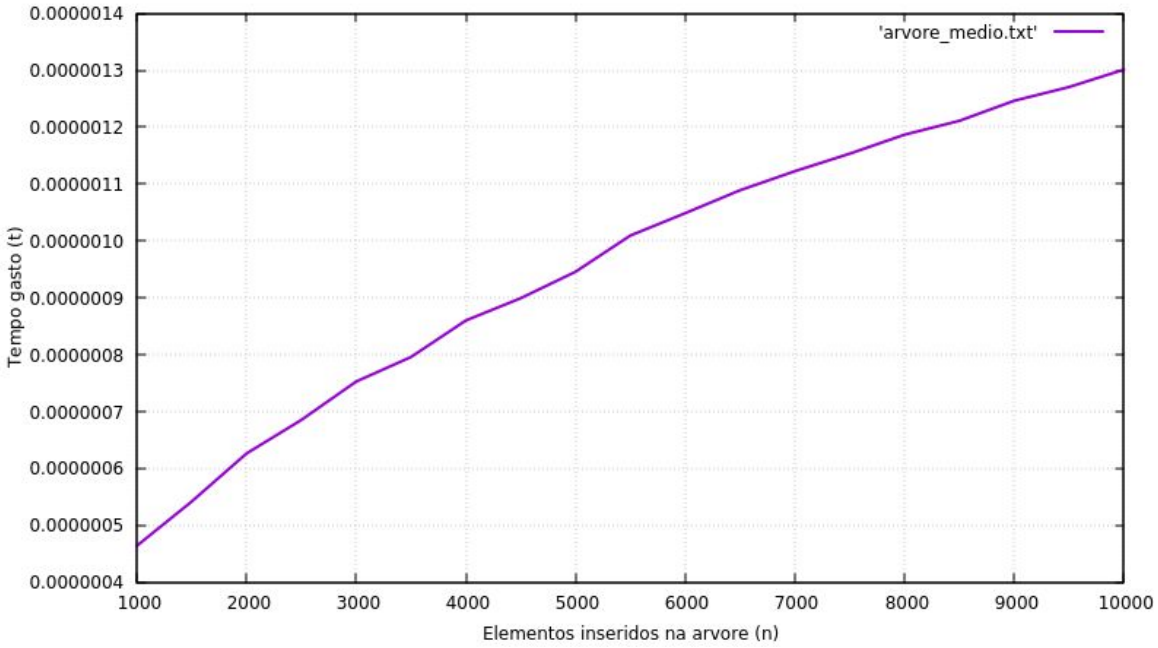


Figura 3: Tempo de execução do caso médio da busca em Árvore Binária

Na figura 4 é possível visualizar a comparação entre o tempo de execução dos três casos.

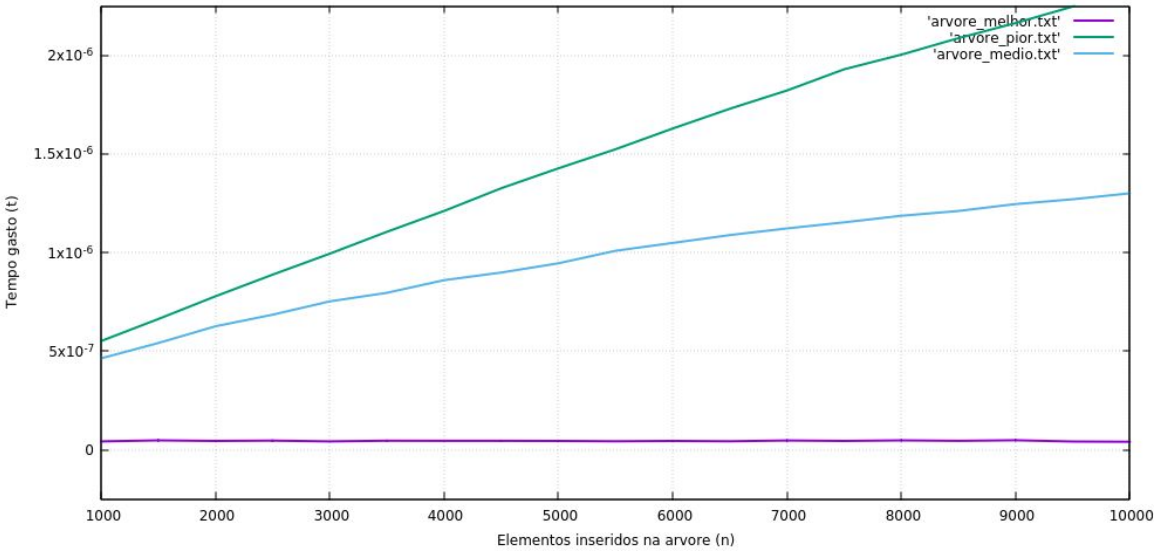


Figure 4: Tempo de execução dos três casos da busca em Árvore Binária

2.2 Busca em árvore balanceada

Árvore balanceada ou árvore AVL funciona de maneira semelhante à Árvore Binária, entretanto, mantendo um balanceamento entre o lado direito e esquerdo de cada nó, garantindo assim que seja efetuado um menor número de verificações. Para manter esse balanceamento geralmente se verifica a cada nova inserção se existe algum lado desbalanceado, se houver, é necessário fazer algumas rotações para torná-la balanceada novamente. O pior e médio caso da busca em árvore balanceada tem tempo de execução O ($\log_2 n$) e seu melhor caso O (1).

O melhor caso da busca em uma árvore balanceada ocorre quando o elemento buscado é encontrado logo na primeira verificação. O melhor caso pode ser equacionado como

$$T_b(n) = C_1 + C_2 + C_3$$

O pior caso da busca ocorre quando o elemento buscado não é encontrado, pode ser equacionado como

$$T_w(n) = C_1 + C_2 + C_{4,6} + C_{3,5} + T_w\left(\frac{n}{2}\right)$$

observa-se que se obtém uma relação de recorrência que tem como caso base

$$T(1) = C_1$$

então, calculando a próxima relação

$$T_w\left(\frac{n}{2}\right) = C_1 + C_2 + C_{4,6} + C_{3,5} + T_w\left(\frac{n}{4}\right)$$

$$T_w\left(\frac{n}{2}\right) = C_1 + C_2 + C_{4,6} + C_{3,5} + T_w\left(\frac{n}{4}\right)$$

$$T_w(n) = C_1 + C_2 + C_{4,6} + C_{5,7} + \left[C_1 + C_2 + C_{4,6} + C_{3,5} + T_w\left(\frac{n}{4}\right)\right] =$$

$$2(C_1 + C_2 + C_{4,6} + C_{5,7}) + T_w\left(\frac{n}{4}\right)$$

$$T_w(n) = x(C_1 + C_2 + C_{4,6} + C_{5,7}) + T_w\left(\frac{n}{2^x}\right)$$

$$\frac{n}{2^x} = 0$$

$$n = 2^x$$

$$log_2 n = x$$

$$T_w(n) = log_2 n(C_1 + C_2 + C_{4,6} + C_{5,7}) + T_w\left(\frac{log_2 n}{log_2 n}\right) =$$

então

$$log_2 n(C_1 + C_2 + C_{4,6} + C_{5,7}) + C_1$$

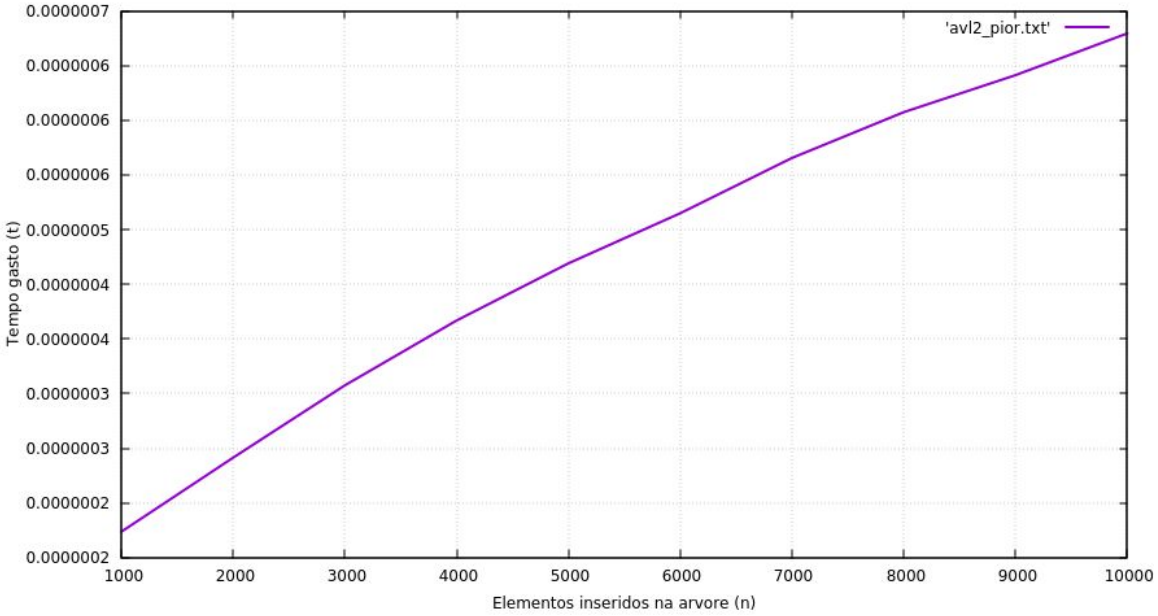


Figura 5: Tempo de execução do pior caso da árvore balanceada

Para geração do gráfico a árvore foi preenchida de forma aleatória e o elemento procurado não foi encontrado. Para o caso médio esperado, a árvore foi preenchida de forma aleatória e foi procurado um número aleatório. Na figura 6 é possível visualizar seu tempo de execução.

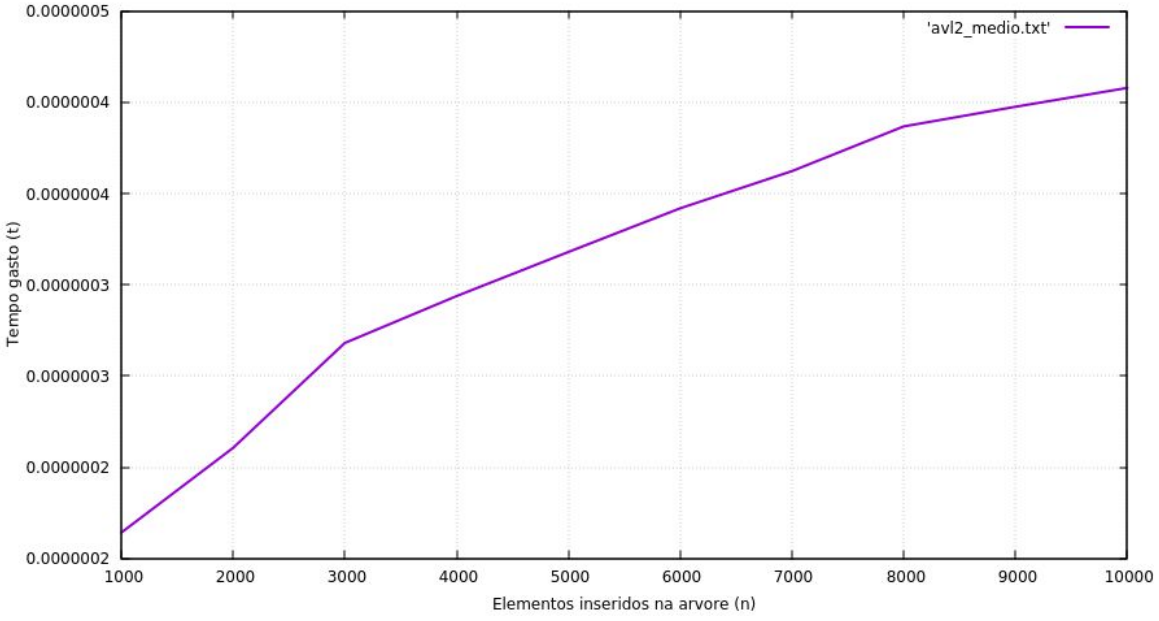


Figura 6: Tempo de execução do caso médio da árvore balanceada

Na figura 7 é possível visualizar a comparação entre o tempo de execução dos dois casos.

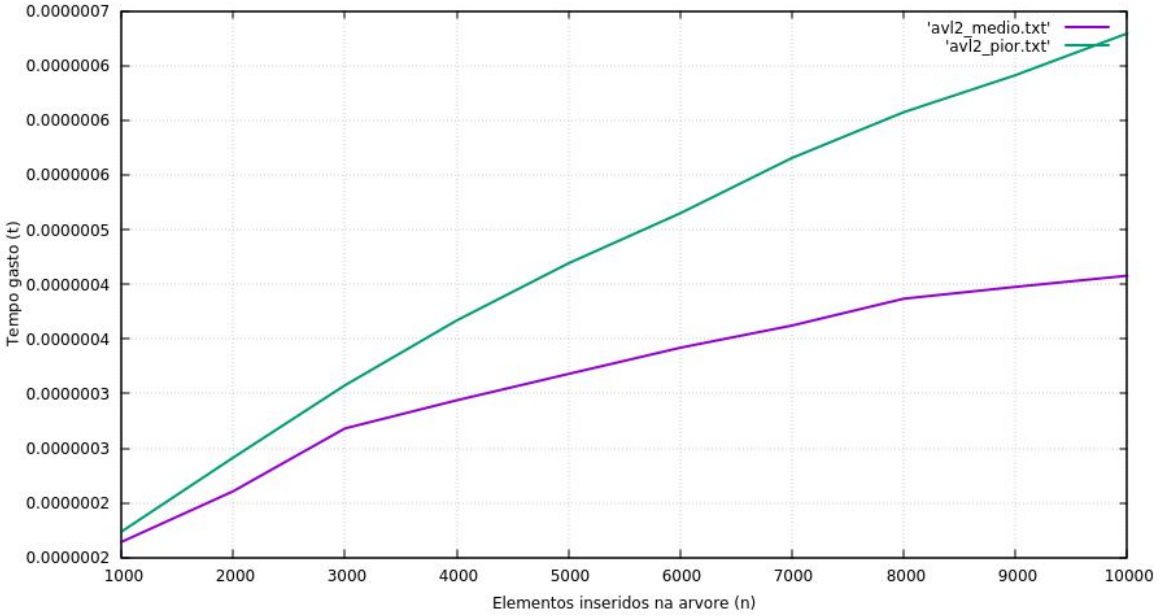


Figura 7: Tempo de execução dos casos da busca em árvore balanceada

2.3 Busca em tabela de dispersão

Tabela de dispersão, também conhecida como tabela hash ou tabela de espalhamento, é é um vetor cada uma de cujas posições armazena zero, uma, ou mais chaves que funcionam associando chaves à valores. O principal objetivo dessa estrutura de dados é fazer com que a partir de uma chave seja possível fazer uma rápida busca e encontrar o elemento desejado.

A hash possui parâmetros importantes:

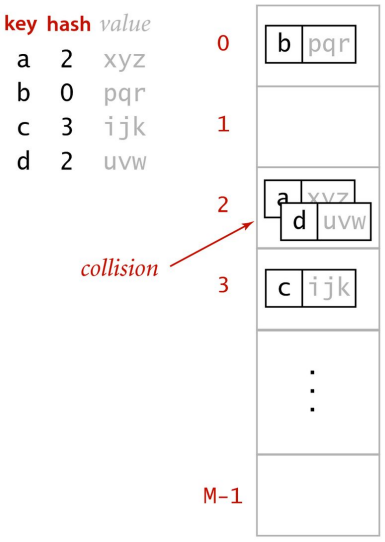
M : número de posições na tabela de hash

N : número de chaves da tabela de símbolos

α = N/M : fator de carga (load factor)

A função de hash transforma cada chave em um índice da tabela de hash, é nela que a função de hashing espalha as chaves pela tabela de hash.

A função de hashing associa um valor hash (hash value), entre 0 e M−1, a cada chave. No exemplo dos CPFs temos $\alpha < 1$ e a função de hashing é a identidade. No exemplo dos nomes de pessoas temos $\alpha > 1$ e a função de hashing é nome.charAt(0). A função de hashing produz uma colisão quando duas chaves diferentes têm o mesmo valor hash e portanto são levadas na mesma posição da tabela de hash:



Hashing: the crux of the problem

A tabela de dispersão tem tempo de execução O (1) em seu melhor e médio caso e, pode chegar a ter tempo de execução até O (n) em seu pior caso.

É possível equacionar o tempo de execução esperado de uma tabela de dispersão como

$$T_a(n) = \frac{1}{n} + (m-1)\left(\frac{1}{m}\right) =$$

$$\frac{1}{n}n + \left(\frac{n-1}{m}\right) =$$

$$1 + 1 - \frac{1}{n}$$

O gráfico de tempo de execução médio da tabela de dispersão pode ser visto na figura 7.

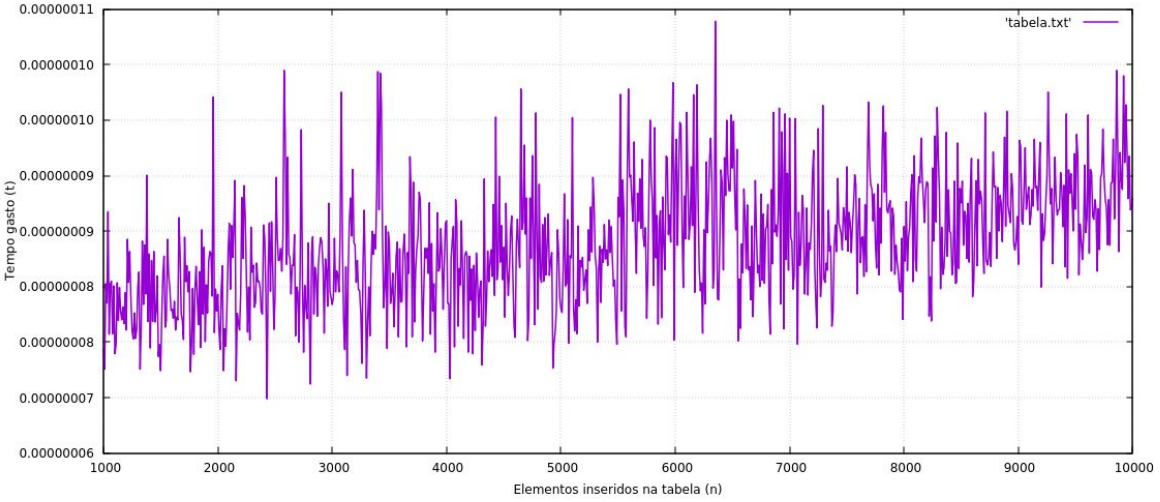


Figure 8: Tempo de execução médio de busca em uma tabela hash

2.4 Busca em lista encadeada

A fim de tornar trabalho mais complexo, foi desenvolvido como conteúdo extra, um breve relato sobre lista de busca encadeada.

Uma lista encadeada é uma maneira de representar um conjunto de dados que são adicionados em células, ou seja, o primeiro elemento é adicionado na primeira célula, o segundo elemento na segunda célula e assim por diante.

Uma busca realizada em uma lista encadeada tem tempo de execução O (1) em seu melhor caso e O (n) em seu médio e pior caso. O melhor caso da busca em uma lista encadeada ocorre quando o elemento buscado é encontrado logo na primeira verificação, sendo assim tendo tempo de execução constante O (1). Seu melhor caso pode ser equacionado como

$$T_b(n) = C_1$$

onde C_1 é a primeira linha executada da função de busca. É possível visualizar o gráfico de tempo de execução na figura 9.

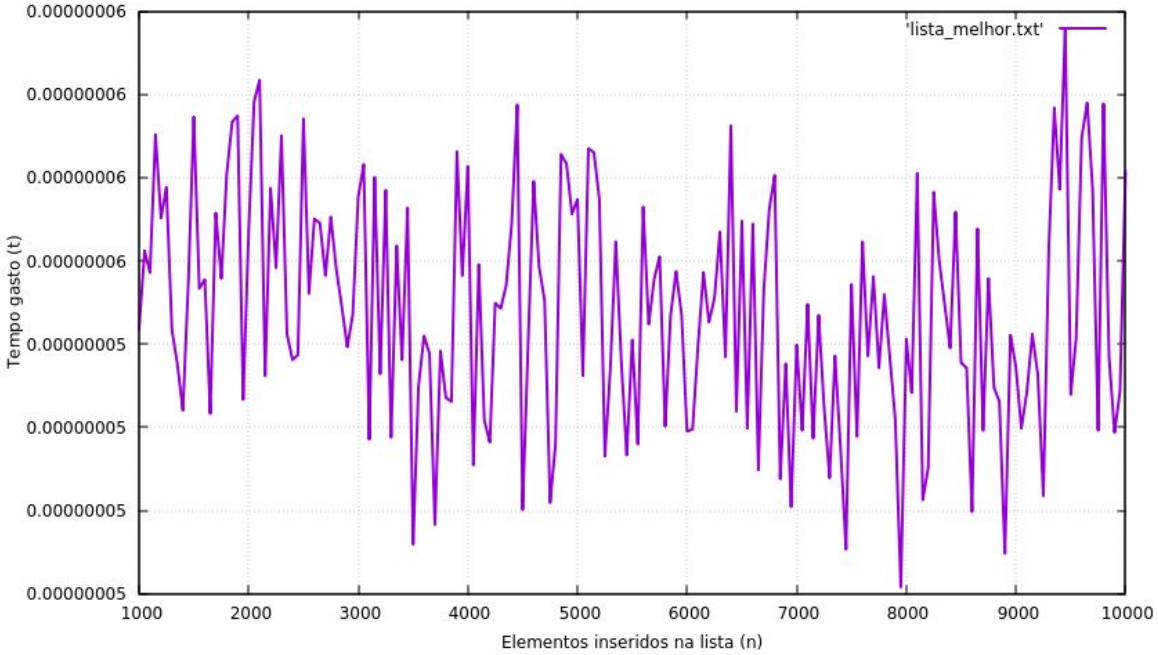


Figura 9: Tempo de execução do melhor caso na busca em lista encadeada

Nota-se que embora haja variações o tempo de execução é constante em um intervalo. Para geração do gráfico a lista foi preenchida de forma aleatória e o elemento procurado sempre foi encontrado na primeira verificação. O pior caso ocorre quando o elemento buscado não está presente na lista, assim tendo tempo de execução linear O (n). O pior caso pode ser equacionado como

$$T_w(n) = C_1 + C_2 + C_3 + T_w(n-1)$$

observa-se que se obtém uma relação de recorrência que tem como caso base

$$T_w(0) = C_1$$

então, calculando a próxima relação

$$T_w(n-1) = C_1 + C_2 + C_3 + T_w(n-2)$$

$$T_w(n) = C_1 + C_2 + C_3 + [C_1 + C_2 + C_3 + T_w(n-2)] =$$

$$2(C_1 + C_2 + C_3) + T_w(n-2) =$$

substituindo pelo padrão de troca

$$x(C_1 + C_2 + C_3) + T_w(n-x)$$

$$n - x = 0$$

$$x = n$$

então,

$$n(C_1 + C_2 + C_3) + C_1$$

É possível visualizar o gráfico de tempo de execução na figura 10.

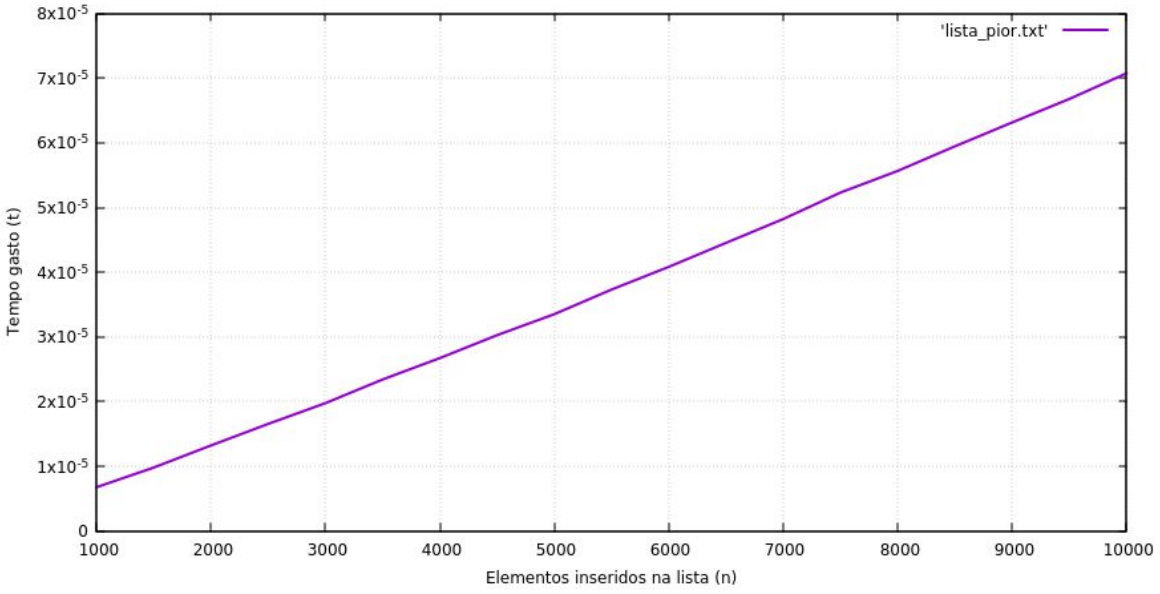


Figura 10: Tempo de execução do pior caso da busca em lista encadeada

Para geração do gráfico a lista foi preenchida de forma aleatória e o elemento procurado não foi encontrado. Para o caso médio esperado, a lista foi preenchida de forma aleatória e foi procurado um número aleatório. Na figura 11 é possível visualizar seu tempo de execução.

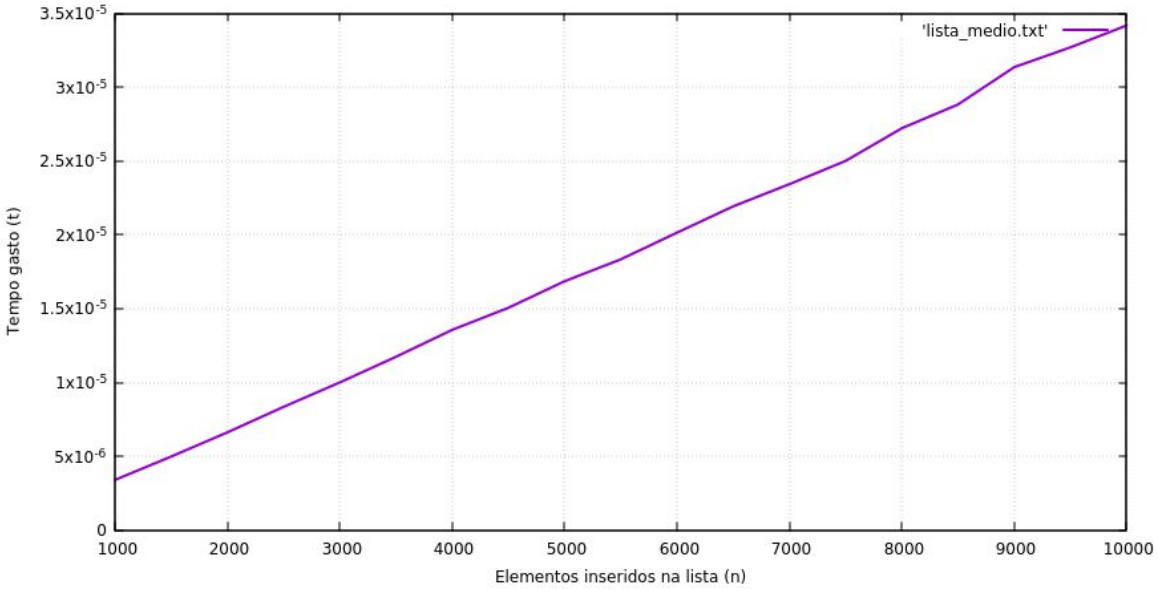


Figure 11: Tempo de execução do caso médio a busca em lista encadeada

Na figura 12 é possível visualizar a comparação entre o tempo de execução dos três casos.

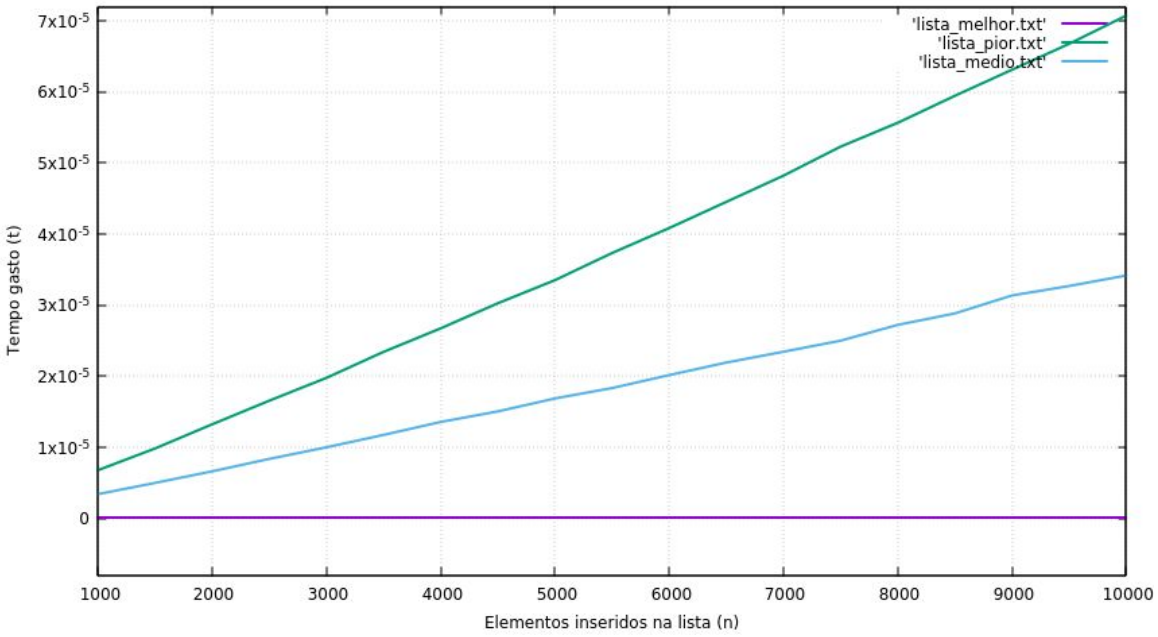


Figura 12: Tempo de execução do caso médio da busca em lista encadeada

4. Comparação entre as buscas em estruturas de dados

Todas as estruturas utilizadas para a análise foram testadas com uma quantidade de elementos inseridos variando entre 1000 e 10000. Essa sessão tratará de comparar as estruturas entre elas am de verificar a diferença entre os tempos de execução, para isso, as comparações serão separadas em algumas ordens, que serão:

- **ordem 1**, onde serão comparados todas as estruturas em seu pior caso (se tiver);
- **ordem 2**, onde serão comparados todos os casos médios;
- **ordem 3**, onde serão comparados os casos da Árvore Binária e árvore balanceada especi-camente;
- **ordem 4**, onde serão comparados todos os casos de todas as estruturas;
- **ordem 5**, onde serão mostrados todos os casos de todas estruturas exceto a lista encadeada.

4.1 Ordem 1

No seguinte tópico sera comparadas todas estruturas em seu pior caso. É possível visualizar os tempos de execução na figura 13.

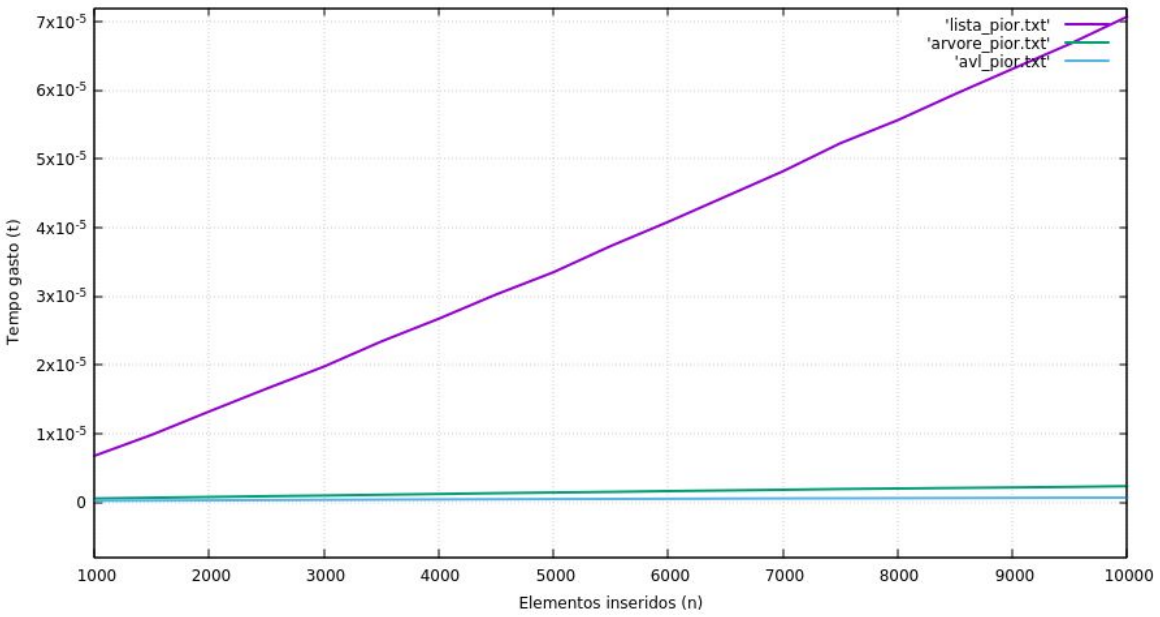


Figura 13: Tempo de execução das estruturas em seu pior caso

A partir do gráfico é possível concluir que o tempo de execução de uma lista encadeada em seu pior caso (linear) é muito superior às demais estruturas no mesmo caso.

4.2 Ordem 2

Aqui serão comparadas todas estruturas em seu caso médio. É possível visualizar os tempos de execução na figura 14.

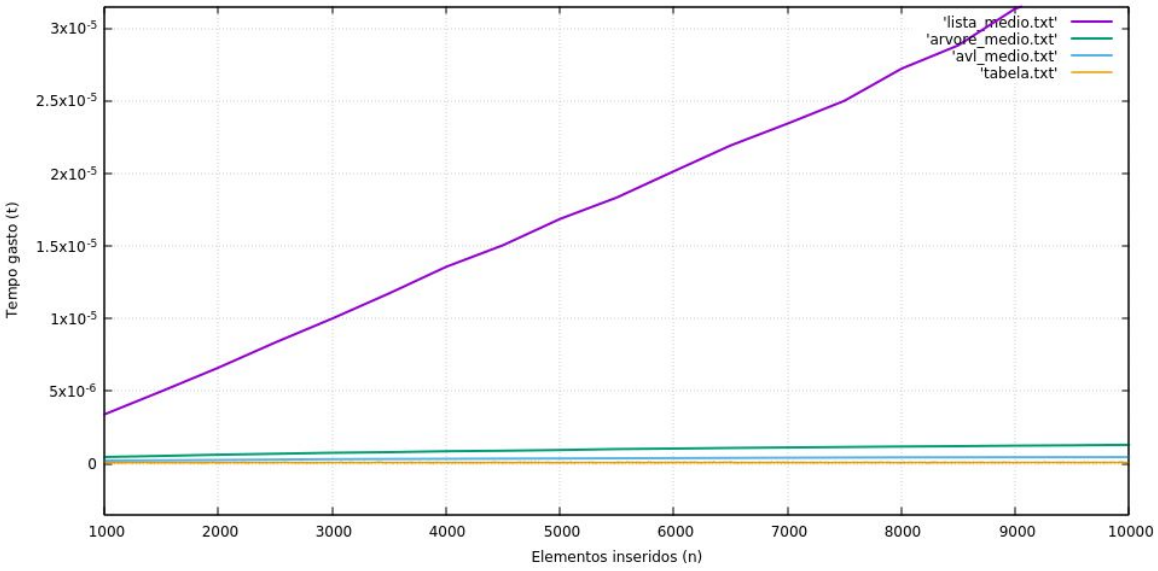


Figura 14: Tempo de execução das estruturas em seu caso médio

Assim como foi visto na ordem 1, o tempo de execução da lista encadeada em seu caso médio é muito superior às demais estruturas no mesmo caso.

4.3 Ordem 3

Aqui serão comparadas os casos da árvore balanceada e binária, mas especialmente. Na figura 14 é possível visualizar o tempo de execução da árvore balanceada e binária em seus piores casos.

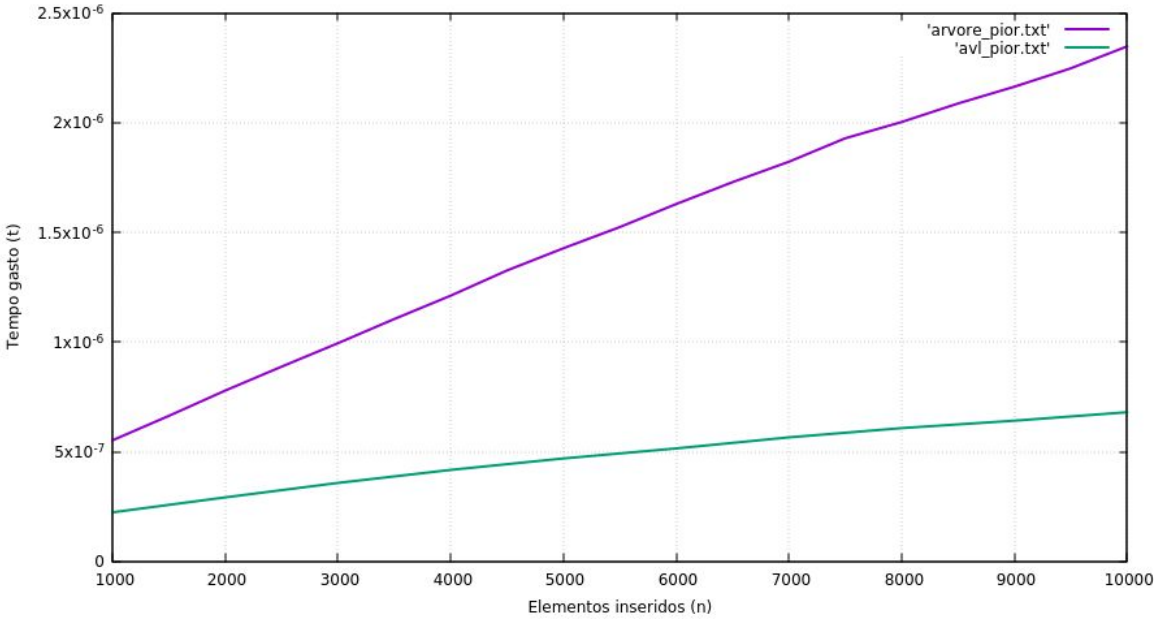


Figure 14: Tempo de execução das árvore balanceada e binária em seu pior caso

A partir da figura 14 é possível concluir que o tempo de execução da árvore balanceada em seu pior caso é inferior à árvore binária no mesmo caso.

Na figura 15 é mostrado o tempo de execução das árvores balanceada e binária em seus casos médios.

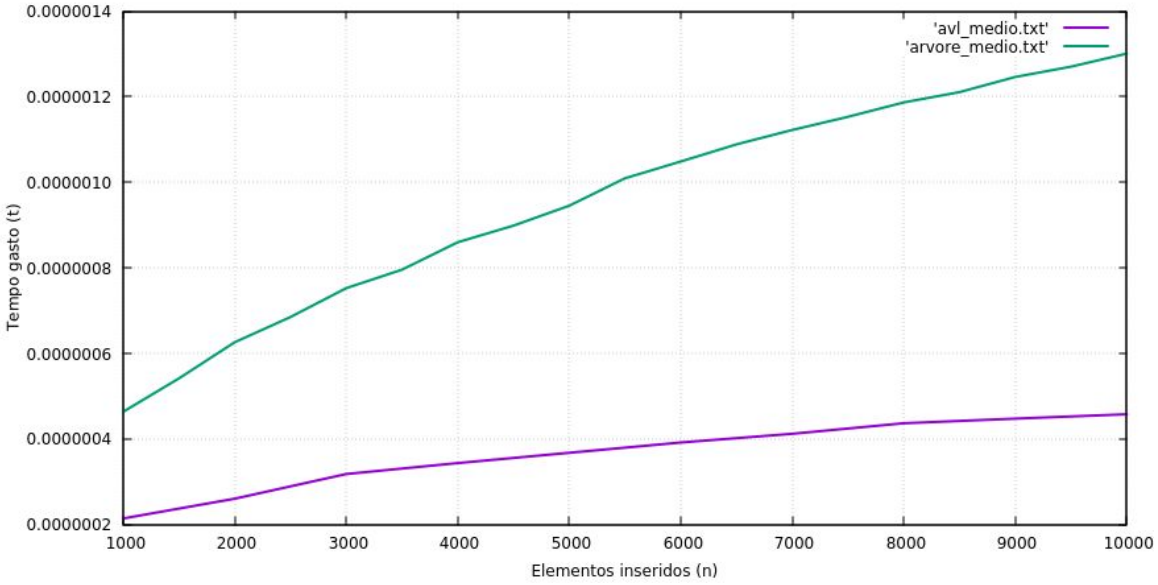


Figure 15: Tempo de execução das árvore balanceada e binária em seu caso médio

Nota-se que, assim como no pior caso de ambas, no caso médio o tempo de execução da árvore balanceada segue sendo inferior à árvore binária.

4.4 Ordem 4

Nesta ordem serão apresentadas todos os casos de todas as estruturas. Na figura 16 é possível visualizar os tempos de execução.

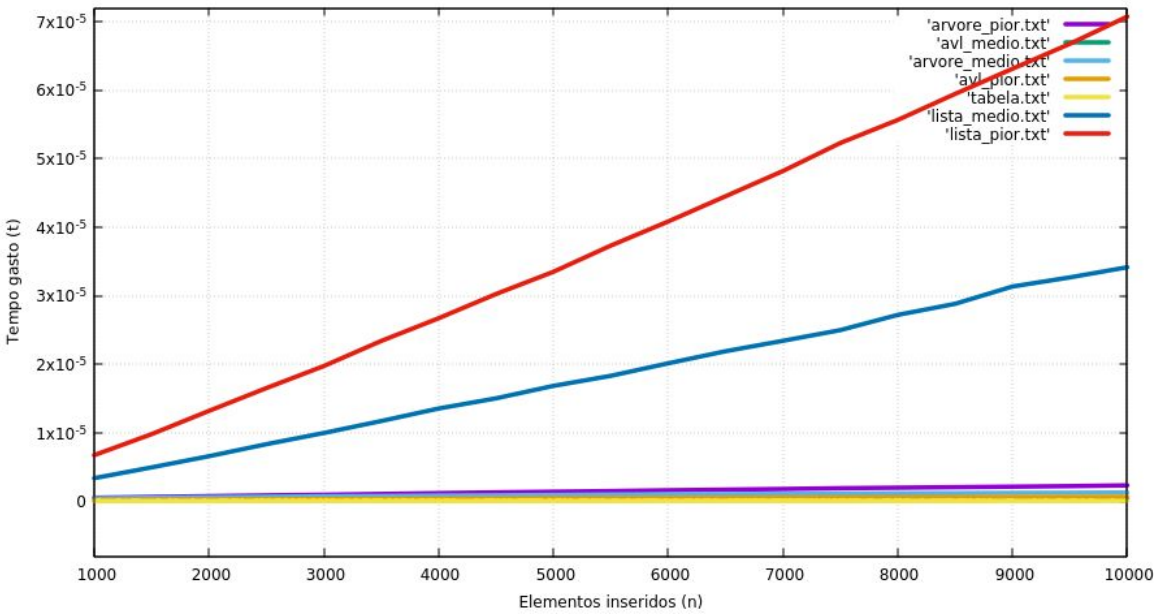


Figure 16: Tempo de execução de todos os casos de todas estruturas

Como já mostrado anteriormente neste trabalho e reafirmado na figura 16, o tempo de execução da busca em uma lista encadeada tanto no pior quanto no médio caso é superior às demais estruturas, causando até mesmo uma dificuldade para verificar a diferença no tempo de execução das demais. Por conta disso, na ordem 5 será excluída a lista encadeada para melhor visualização.

4.5 Ordem 5

Nesta ordem será comparado o tempo de execução entre os casos da árvore binária, balanceada e tabela hash. O gráfico pode ser visto na figura 17.

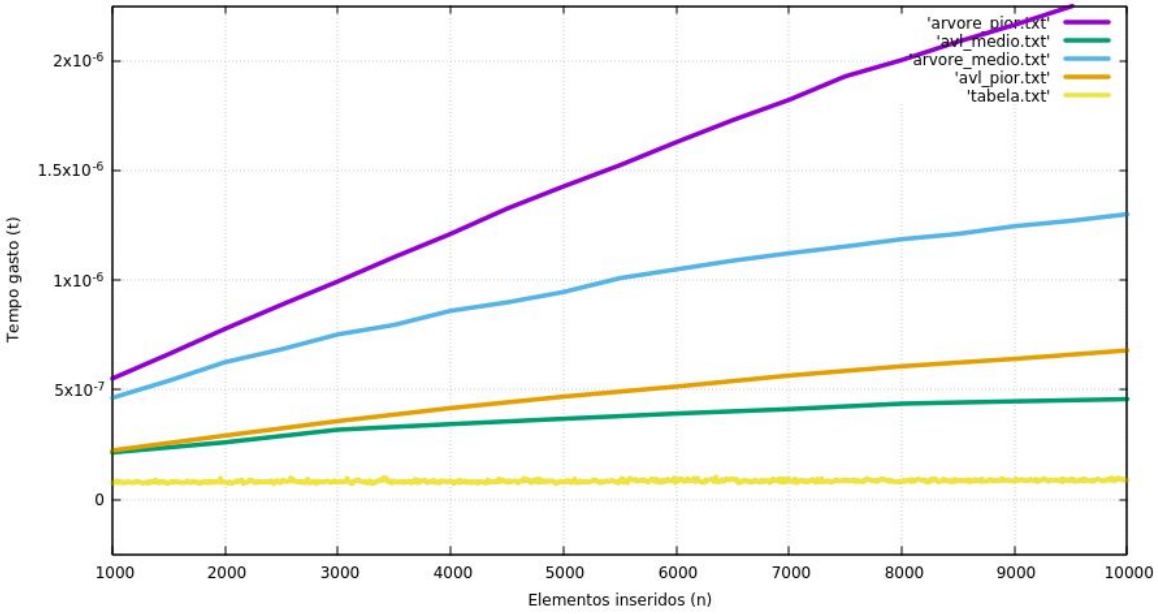


Figure 17: Tempo de execução de todos os casos de todas estruturas

Analisando o gráfico da figura 17 verifica-se que a busca em uma tabela hash contém o menor tempo de execução dentre as demais estruturas. A busca em uma árvore balanceada, em ambos os casos, tem tempo de execução superior à tabela hash mas inferior à busca em uma árvore binária, que por sua vez contém o maior tempo de execução dentre as demais, em ambos os casos.

5. Referências

O projeto está disponibilizado no seguinte link:
https://github.com/jhonatasisraelcl/DataStructurePrograms
https://www.ime.usp.br/~pf/algoritmos/aulas/binst.html
https://www.ime.usp.br/~pf/algoritmos/aulas/bint.html
https://www.ime.usp.br/~pf/estruturas-de-dados/aulas/index.html
https://pt.wikipedia.org/wiki/Algoritmo_de_busca
https://www.ime.usp.br/~pf/estruturas-de-dados/aulas/st-hash.html