Comparação de eficiência em algoritmos de busca sequencial e binária em estruturas de dados lineares e não lineares

Gabriel Ranulfo Branquinho Cirillo

[1]Mestrado em Ciência da Computação
Universidade Estadual do Oeste do Paraná (Unioeste) – Cascavel, PR – Brasil

gabriel.cirillo@unioeste.br

Abstract. *The objective of this paper is to compare and identify advantages and disadvantages of linear and non-linear data structures and their efficiency in sequential and binary search methods. The tests consider the worst cases for each scenario, as well as 100 random search cases, with the aim of diversifying the tests so that they can be reproduced in other experiments. Finally, we conclude with the identified results, that the algorithms indeed operate with the Big O complexity with the number of comparisons approaching the theoretical value for each case, and the time and memory changing according to the size of the array.*

Resumo. *O objetivo deste artigo é comparar e identificar vantagens e desvantagens de estruturas de dados lineares e não-lineares e sua eficiência em métodos de busca sequencial e binária. Os testes consideram os piores casos para cada cenário, além de 100 casos aleatórios de busca, com o interesse de diversificar os testes para que possam ser reproduzidos em outros experimentos. Por fim, concluímos com os resultados identificados, que os algoritmos trabalham de fato com a complexidade de Big O com o número de comparações se aproximando ao valor teórico para cada caso, e o tempo e memória alterando conforme tamanho do arranjo.*

1. Introdução

A eficiência na manipulação e processamento de dados é fundamental em diversas aplicações computacionais. A escolha entre arranjos estáticos e estruturas de dados lineares e não-lineares pode influenciar significativamente o desempenho de algoritmos, especialmente em operações como busca. Este estudo visa investigar e comparar a eficiência de diferentes métodos de busca em estruturas de dados como os arranjos estáticos, listas ligadas e árvores binárias de busca.

A busca de informações na memória de um computador é fundamental para recuperar dados específicos associados a uma identificação fornecida. Por exemplo, em aplicações numéricas, podemos buscar o valor de $f(x)$ em um conjunto, dada a entrada x. Este processo envolve localizar o registro que possui uma chave específica. Após a busca, pode-se determinar se ela foi bem-sucedida, encontrando o registro com a chave K, ou malsucedida, indicando que a chave K não está presente. Essa operação é essencial para muitos algoritmos, onde a eficiência na busca influencia diretamente o desempenho geral do sistema [Knuth 1998].

Métodos de busca podem ser classificados de várias maneiras. Podemos dividi-los em métodos de busca estática versus dinâmica, onde ”estática”significa que o conteúdo do conjunto de dados é essencialmente inalterado (de modo que é importante minimizar o tempo de busca sem considerar o tempo necessário para configurar o conjunto), e ”dinâmica”que significa que o conjunto está sujeito a inserções frequentes e talvez também exclusões [Knuth 1998].

Analisar um algoritmo significa prever os recursos de que o algoritmo necessitará. Ocasionalmente, recursos como memória, largura de banda de comunicação ou hardware de computador são a principal preocupação, mas com frequência é o tempo de computação que desejamos medir. Em geral, pela análise de vários algoritmos candidatos para um problema, pode-se identificar facilmente um algoritmo mais eficiente [Cormen et al. 2012].

Ao analisar os arranjos estáticos, dois métodos de busca são estudados: busca sequencial padrão e busca binária. Por outro lado, nas estruturas lineares e não-lineares, concentramo-nos na busca sequencial em lista ligada e busca em árvores binárias de busca. A comparação desses métodos permitirá uma compreensão mais aprofundada de suas vantagens e desvantagens em diferentes contextos de problema.

Para avaliar a eficiência dos algoritmos de busca, serão gerados casos de teste variando o tamanho do arranjo de 100 mil a 1 milhão de elementos únicos, estes elementos variando entre os valores de 1 a 120 milhões, por sua vez, separados em arquivos de intervalos de 100 mil. Todos os algoritmos serão testados nos mesmos cenários para garantir uma comparação justa. As métricas de desempenho consideradas incluem a média e o desvio padrão para o número de comparações, tempo de execução e consumo de memória.

Diversos cenários serão explorados, incluindo o pior caso, com 3 execuções de cada algoritmo, e casos aleatórios com 100 buscas para cada cenário. É importante ressaltar que o custo de criação das estruturas será descartado, focando apenas no desempenho das operações de busca, a maioria dos nossos testes irá considerar listas não ordenadas, apenas em um caso, o caso de busca binária em arranjo estático, será considerada a ordenação.

Com essas considerações em mente, este projeto busca fornecer *insights* valiosos sobre a escolha e aplicação adequadas de estruturas de dados e algoritmos de busca em diferentes contextos computacionais.

2. Fundamentação Teórica

2.1. Estruturas de Dados Lineares e Não Lineares

As estruturas de dados lineares são aquelas em que os elementos são organizados de forma sequencial, ou seja, cada elemento tem um sucessor e, opcionalmente, um antecessor. Nessas estruturas, a ordem dos elementos é importante e as operações de inserção e remoção geralmente ocorrem em uma extremidade da estrutura. As estruturas de dados não lineares são aquelas em que os elementos não são organizados de forma sequencial. Em vez disso, os elementos são organizados de maneira hierárquica, formando relações complexas entre si [Cormen et al. 2012].

2.2. A notação Big O

A notação Big O informa quão eficiente é um algoritmo em termos de escalabilidade. Por exemplo, se temos uma lista de tamanho n, e o tempo de execução do algoritmo é $O(n)$, isso significa que o tempo de execução do algoritmo cresce linearmente com o tamanho da lista. No entanto, a notação Big O não fornece uma medida direta do tempo em segundos. Em vez disso, ela permite comparar o número de operações realizadas pelo algoritmo para diferentes tamanhos de entrada. Essa notação nos ajuda a entender como o desempenho do algoritmo se comporta à medida que o tamanho da entrada aumenta [Cormen et al. 2012].

2.3. Arranjos Estáticos

Arranjos estáticos, também conhecidos como *arrays* estáticos, são estruturas de dados que representam uma coleção de elementos de tamanho fixo e contíguos na memória. Eles são usados para armazenar dados do mesmo tipo em uma sequência ordenada, onde cada elemento pode ser acessado por meio de um índice específico [Cormen et al. 2012].

2.4. Listas Ligadas

Uma lista ligada é uma estrutura de dados na qual os objetos estão organizados em ordem linear. Entretanto, diferentemente de um arranjo, no qual a ordem linear é determinada pelos índices do arranjo, a ordem em uma lista ligada é determinada por um ponteiro em cada objeto[Cormen et al. 2012].

2.5. Busca Sequencial Padrão

A busca sequencial padrão é um método simples para encontrar um elemento em uma lista. Ele percorre cada elemento da lista, um por um, até encontrar o elemento desejado ou até o final da lista ser alcançado. A complexidade desse método é linear, o que significa que o tempo de execução cresce proporcionalmente ao tamanho da lista. A complexidade deste algoritmo é $O(n)$, onde n é o número de elementos na lista. Isso ocorre porque, no pior caso, pode ser necessário percorrer todos os n elementos da lista para encontrar o elemento desejado [Bhargava 2016].

2.6. Busca Binária

A busca binária é um algoritmo eficiente para encontrar um elemento específico em um arranjo ordenado. Ele opera dividindo repetidamente o arranjo ao meio e comparando o elemento alvo com o elemento central da sub lista atual. Dependendo do resultado da comparação, o algoritmo decide em qual metade do arranjo continuar a busca. Esse processo é repetido até que o elemento desejado seja encontrado ou a sub lista se torne vazia. A complexidade da busca binária é $O(\log n)$, onde n é o número de elementos no arranjo [Dasgupta et al. 2006].

2.7. Busca Sequencial em Lista Ligada

A busca sequencial em uma lista ligada é uma estrutura de dados na qual cada elemento (nó) contém um valor e uma referência (ou ponteiro) para o próximo elemento na lista na busca sequencial, percorremos a lista ligada começando pelo primeiro nó e avançando para o próximo nó até encontrar o elemento desejado ou chegar ao final da lista (quando

o próximo nó é nulo). Durante o processo de busca, cada nó é examinado para verificar se seu valor corresponde ao valor que está sendo buscado.

A complexidade da busca sequencial em uma lista ligada é linear, ou seja, $O(n)$, onde n é o número de elementos na lista. Isso ocorre porque, em caso de pior cenário, precisamos percorrer todos os elementos da lista até encontrar o elemento desejado ou chegar ao final da lista [Cormen et al. 2012].

2.8. Árvore Binária de Busca

Uma árvore binária de busca *(BST - Binary Search Tree)* é uma estrutura de dados em árvore na qual cada nó tem no máximo dois filhos, com a seguinte propriedade: para cada nó, todos os elementos na subárvore à esquerda desse nó são menores que o valor armazenado no nó e todos os elementos na subárvore à direita são maiores.

Essa propriedade permite a rápida busca, inserção e exclusão de elementos na árvore, tornando as árvores binárias de busca uma estrutura de dados eficiente para muitas operações. Por exemplo, ao buscar um elemento em uma árvore binária de busca, é possível eliminar metade dos elementos restantes a cada passo, resultando em uma busca eficiente em tempo médio de $O(\log n)$, onde n é o número de elementos na árvore. No entanto, em casos extremos, a árvore pode degenerar em uma lista vinculada, resultando em uma complexidade de busca de $O(n)$ [Cormen et al. 2012].

3. Materiais e Métodos

Neste capítulo, serão detalhados os materiais utilizados e os métodos empregados na implementação e avaliação dos diferentes métodos de busca em arranjos estáticos e estruturas de dados lineares e não-lineares.

3.1. Materiais Utilizados

Para a realização deste projeto, foram utilizados os seguintes materiais:

- *Notebook Dell Inspiron 5406 2-in-1,* executando *Microsoft Windows 11 Home Single Language (x64),* Versão *23H2, Build 22631.3527,* equipado com um processador *Intel(R) Core(TM) i3-1115G4* de *11ª* geração *@ 3.00GHz e 16 GBytes* de memória RAM DDR4.
- Ambiente de desenvolvimento integrado (IDE): *Visual Studio Code* (versão 1.89.12).
- Linguagem de programação: *Python* 3.10.9.
- Conjunto de dados gerados para os testes, variando o tamanho do arranjo de 100 mil a 1 milhão de elementos únicos.
- Bibliotecas utilizadas: *csv, random, os, psutil* e *time.*
- Estruturas de dados específicas e métodos de busca implementados em estruturas lineares e não-lineares, como listas ligadas e árvores binárias de busca e arranjos estáticos.

O código fonte e os materiais utilizados neste projeto podem ser encontrados no seguinte repositório do GitHub `https://github.com/gabrielranulfo/Avaliacao-EDAA`.

3.2. Geração dos Casos de Teste

Para avaliar a eficiência dos algoritmos de busca, os casos gerados de teste variam o tamanho do arranjo de 100 mil a 1 milhão de elementos únicos, este elementos estão entres os valores de 1 a 120 milhões, por sua vez, separados em arquivos de intervalos de 100 mil. Todos os algoritmos foram testados nos mesmos cenários para garantir uma comparação justa.

3.3. Coleta de Dados

Para cada método de busca, ordenação e cenários de teste, foram coletados os seguintes dados:

- Número de comparações realizadas durante a busca.
- Tempo de execução do algoritmo em microssegundos (µs).
- Consumo de memória durante a execução em *bytes*.

Os dados foram registrados para análise posterior, incluindo o cálculo da média e do desvio padrão para as métricas coletadas.

3.4. Considerações Adicionais

Além dos métodos de busca propriamente ditos, foram descartados custo de criação das estruturas, custo de criação dos arquivos, já o custo de ordenação será discutido separadamente.

Nos casos em que os valores foram arredondados para números inteiros, optamos por um arredondamento para baixo. Não há motivo específico, apenas uma escolha oferecida pela biblioteca do código gerado.

Este capítulo detalhou os materiais utilizados e os métodos empregados na implementação e avaliação dos métodos de busca estudados neste projeto. Os próximos capítulos abordarão os resultados obtidos e sua análise.

4. Resultados

Os resultados dos testes de pior cenário para busca sequencial em listas ligadas são apresentados na Tabela 1. Optamos por escolher três números que não existiam em um arranjo não ordenado, de modo que a busca representasse o pior caso e percorresse todo o arranjo sem identificar e encontrar o número procurado.

Como resultado, vemos que a média de memória oscila entre 39.751.680 *bytes* e 249.253.888 *bytes*, e o desvio padrão não altera no mesmo arranjo. A razão mais provável é que a memória foi alocada uma vez e, em seguida, realocada para outros tamanhos de arranjo. Isso pode ser atribuído a diversas razões, como otimizações do compilador, gerenciamento de memória do sistema operacional ou mesmo da linguagem de programação utilizada. No entanto, o que se percebe é a necessidade de uma maior alocação de memória com crescimento linear conforme o tamanho do arranjo.

É possível notar que, neste teste de pior cenário para a busca sequencial, a quantidade de comparações é extremamente grande, pois o algoritmo percorre cada elemento da lista até o final, validando sua complexidade de $O(n)$, onde n é a quantidade de elementos de cada lista.

O tempo necessário para percorrer a lista apresenta comportamento linear em relação à quantidade de comparações e ao tamanho do arranjo variando de 23.490 µs em média para o arranjo de 100.000 elementos a 232.737 µs no arranjo de 1.000.000 elementos em média, comportamento apresentado devido à estrutura utilizada neste cenário.

Tabela 1. Busca Sequencial em Lista Ligada para Pior Cenário

Arranjo	Tempo (µs)		Comparações		Memória (*bytes*)	
	Média	Std.	Média	Std.	Média	Std.
100.000	23.490	6.765	100.000	0	39.751.680	0
200.000	39.906	453	200.000	0	62.824.448	0
300.000	79.553	9.289	300.000	0	85.970.944	0
400.000	85.638	8.006	400.000	0	109.236.224	0
500.000	106.145	5.824	500.000	0	132.599.808	0
600.000	160.916	73.882	600.000	0	156.053.504	0
700.000	137.439	658	700.000	0	179.363.840	0
800.000	153.093	5.665	800.000	0	202.619.563	19.309
900.000	177.654	6.592	900.000	0	225.935.360	0
1.000.000	232.737	47.787	1.000.000	0	249.253.888	0

Os testes de busca sequencial em listas ligadas para 100 casos aleatórios são apresentados na Tabela 2. Optamos por escolher 100 diferentes números aleatórios que pertenciam ao arranjo de modo que a busca resultasse em sucesso em todos os casos, variando os resultados para cada número devido à sua posição no arranjo não ordenado.

Observa-se que, seguindo o padrão dos testes de busca sequencial de lista ligada para pior cenário, mesmo em buscas onde o número é encontrado, a quantidade de memória e tempo tem comportamento linear ao tamanho do arranjo.

A média de memória oscila de 39.739.515 *bytes* no arranjo de 100.000 elementos a 251.043.840 *bytes* no arranjo de 1.000.000 elementos, oscilação ligeiramente menor quando em cenário de pior caso.

É possível observar que, neste teste de busca aleatória, a quantidade de comparações apresenta alta variação: 51.814 comparações para o arranjo de 100.000 elementos em média a 450.976 comparações em média no arranjo de 1.000.000 de elementos, com desvio padrão oscilando de 28.090 a 292.157 em média respectivamente, pois o algoritmo percorre cada elemento da lista até identificar e encontrar o procurado. Ainda assim, o custo é menor quando comparado com três execuções de pior cenário onde é necessário percorrer o arranjo inteiro.

O tempo tem comportamento semelhante ao teste anterior, porém, é razoavelmente mais baixo, variando de 37.974 µs em média no arranjo de 100.000 elementos a 89.128 µs no arranjo de 1.000.000, o desvio padrão também oscila muito devido à busca por diferentes números.

Tabela 2. Busca Sequencial em Lista Ligada para 100 Números Aleatórios

Arranjo	Tempo (µs)		Comparações		Memória (*bytes*)	
	Média	Std.	Média	Std.	Média	Std.
100.000	37.974	48.789	51.814	28.090	39.739.515	22.288
200.000	23.476	14.014	98.589	60.607	62.811.709	12.729
300.000	35.414	24.906	144.951	81.008	85.962.752	0
400.000	37.337	22.907	176.534	116.434	109.318.144	0
500.000	57.876	30.771	276.820	147.149	132.628.480	0
600.000	71.975	41.696	319.489	179.918	156.217.344	0
700.000	82.423	62.767	352.684	195.705	179.585.024	0
800.000	90.462	36.984	452.821	198.159	202.899.661	5.861
900.000	82.037	58.763	400.328	276.857	226.252.390	7.815
1.000.000	89.128	54.516	450.976	292.157	251.043.840	0

A Tabela 3 mostra os resultados para busca sequencial do pior cenário em arranjos estáticos, uma estrutura mais simples que a lista ligada e que apesar de seguir o mesmo padrão de comportamento observado nos testes anteriores, neste cenário, destaca-se por uma redução no uso de memória, oscilando entre 20.922.368 *bytes* em média no arranjo de 100.000 elementos a 92.684.288 *bytes* em média para o arranjo de 1.000.000 elementos.

A quantidade de comparações e o tempo de execução mantêm-se consistentes, seguindo o padrão observado nos testes anteriores no entanto um pouco mais eficientes em tempo neste cenário de busca.

Tabela 3. Busca Sequencial em Arranjos Estáticos para Pior Cenário

Arranjo	Tempo (µs)		Comparações		Memória (*bytes*)	
	Média	Std.	Média	Std.	Média	Std.
100.000	18.601	4.603	100.000	0	20.922.368	0
200.000	34.956	2.982	200.000	0	28.995.584	0
300.000	55.451	4.428	300.000	0	37.007.360	0
400.000	121.319	7.939	400.000	0	45.117.440	0
500.000	115.483	35.024	500.000	0	53.186.560	0
600.000	107.015	8.461	600.000	0	61.444.096	0
700.000	109.786	8.195	700.000	0	69.443.584	0
800.000	125.643	10.499	800.000	0	77.545.472	0
900.000	222.060	31.473	900.000	0	85.647.360	0
1.000.000	248.823	91.048	1.000.000	0	92.684.288	0

A Tabela 4 apresenta os resultados dos testes de busca sequencial de 100 números aleatórios em arranjos estáticos não ordenados. Observa-se que, assim como nos testes de pior cenário a memória oscila bastante de acordo com o tamanho do arranjo, mas não tanto quanto no cenário de listas ligadas, utilizando cerca de 20.966.359 *bytes* em média no arranjo de 100.000 elementos a 69.840.896 *bytes* em média no arranjo de 1.000.000 elementos.

A quantidade de comparações exibe uma alta variação e desvio padrão, uma vez que o algoritmo percorre cada elemento do vetor até identificar o elemento buscado, variando de 54.441 comparações em média no arranjo de 100.000 elementos a 480.054 comparações no arranjo de 1.000.000 de elementos em média.

Quanto ao tempo, seu comportamento é semelhante ao teste anterior de busca em 100 números nas listas ligadas, porém, razoavelmente mais baixo, foram necessários em média 12.528 µs para identificar todos os números no arranjo de 100.000 elementos e 83.203 µs para a busca no arranjo de 1.000.000 de elementos, nota-se neste cenário que no arranjo de 800.000 elementos o tempo médio para a busca foi de 98.708 µs o que é cerca de 15.505 µs de diferença a mais que o teste no arranjo de 1.000.000 de elementos, essa oscilação referente aos outros arranjos pode ser compreendida pela posição dos números dentro de cada arranjo e resultando portanto em um tempo maior entre as buscas, mesmo o arranjo sendo menor que um outro.

Tabela 4. Busca Sequencial em Arranjos Estáticos para 100 Números Aleatórios

Arranjo	Tempo (µs)		Comparações		Memória (*bytes*)	
	Média	Std.	Média	Std.	Média	Std.
100.000	12.528	7.451	54.441	28.581	20.966.359	23.110
200.000	20.164	11.053	92.388	52.948	26.920.878	15.155
300.000	40.076	94.582	140.025	86.591	32.833.536	0
400.000	30.255	20.399	180.169	121.250	37.855.232	0
500.000	41.350	25.896	244.032	139.788	43.843.584	0
600.000	52.773	31.629	300.899	173.516	48.975.872	0
700.000	62.931	37.226	344.426	191.535	55.066.624	0
800.000	98.708	121.903	366.976	223.871	59.941.069	5.861
900.000	83.778	52.057	427.555	253.238	65.871.872	0
1.000.000	83.203	51.610	480.054	281.981	69.840.896	0

Veremos a partir de agora os resultados de testes em estruturas de dados que têm por objetivo serem mais eficientes em tempo e comparações do que aqueles cuja busca é realizada de maneira sequencial simples, ou seja, busca caso a caso um determinado número, e como consequência tem um número maior de comparações e uso dos recursos do computador.

A Tabela 5 apresenta os resultados dos testes de busca binária em arranjos estáticos para situações de pior cenário. No entanto, aqui, neste cenário, o arranjo precisou passar primeiro por uma ordenação, que será discutida separadamente na Tabela 9. O pior cenário segue os testes anteriores, com o intuito de buscar valores não presentes no arranjo.

Neste tipo de busca, observamos um comportamento notavelmente diferente. No cenário de pior caso, podemos perceber uma qualidade e velocidade na busca extremamente mais ágil e rápida do que em situações onde utilizamos busca sequencial, devido à complexidade da busca binária que é $O(\log n)$, onde n é o número de elementos no arranjo.

A utilização da memória, por sua vez, ainda é semelhante aos testes anteriores, variando de 21.671.936 *bytes* em média no arranjo de 100.000 elementos a 69.320.704 *bytes* em média no arranjo de 1.000.000 de elementos. O desvio padrão mostra que possivelmente não houve realocação de memória quando a busca era no mesmo arranjo.

A quantidade de comparações e tempo aqui são os grandes diferenciais. O número médio de comparações alterna entre 16 para o arranjo de 100.000 elementos e 19 para o arranjo de 1.000.000 de elementos. Uma observação quanto aos valores: aqui era esperado

um valor aproximado de 16,60 para o arranjo de 100.000 elementos e 19,93 comparações em média para o cenário de 1.000.000 de elementos, que é o resultado de $O(\log n)$ onde n é a quantidade de elementos do arranjo. No entanto, nosso código arredondou estes números para baixo, 16 e 19 respectivamente, e não 17 e 20, e o mesmo para os outros arranjos, o que gera uma sutil mudança no valor, mas não interfere diretamente na análise e proposta deste artigo.

A média de tempo, por sua vez, foi de 143 µs para o arranjo de 100.000 elementos e 201 µs para o arranjo de 1.000.000 de elementos, com desvio padrão de 54 a 132, respectivamente. O que é uma unidade de tempo extremamente veloz ao comparar com a busca sequencial observada nos testes anteriores. Não devemos esquecer que é importante considerar o custo de ordenação, o que discutiremos mais adiante.

Tabela 5. Busca Binária em Arranjos Estáticos para Pior Cenário

Arranjo	Tempo (µs)		Comparações		Memória (*bytes*)	
	Média	Std.	Média	Std.	Média	Std.
100.000	143	54	16	0	21.671.936	0
200.000	173	92	17	0	27.004.928	0
300.000	107	36	18	0	32.841.728	0
400.000	138	62	18	0	37.765.120	0
500.000	148	64	18	0	43.704.320	0
600.000	112	18	19	0	48.664.576	0
700.000	171	80	19	0	54.681.600	0
800.000	134	39	19	0	59.514.880	0
900.000	121	30	19	0	65.421.312	0
1.000.000	201	132	19	0	69.320.704	0

A Tabela 6 apresenta os resultados dos testes de busca binária em arranjos estáticos ordenados para 100 números aleatórios presentes nos arranjos. A média e o desvio padrão de memória variam entre 21.460.910 *bytes* no arranjo de 100.000 elementos e 69.111.808 *bytes* no arranjo de 1.000.000 de elementos.

A quantidade de comparações segue próxima do valor de $O(\log n)$, onde n é a quantidade de elementos do arranjo, sendo 16 no arranjo de 100.000 elementos e 19 para o arranjo de 1.000.000 elementos, com desvio padrão oscilando entre 1 e 2, devido ao fato do número ser encontrado com menos comparações do que em outras buscas.

Comparadas às pesquisas simples, as comparações são significativamente reduzidas devido à eliminação de parte da lista nesse cenário. Isso resulta em um tempo médio de execução e identificação do número também menor, assim como no teste anterior, variando entre 231 µs no arranjo de 100.000 elementos e 191 µs no arranjo de 1.000.000 de elementos, o que neste caso equiparou o tempo e até foi mais veloz em um cenário com muito mais elementos, essa característica também pode ter causa atribuída a posição de cada número dentro do arranjo durante a busca.

Tabela 6. Busca Binária em Arranjos Estáticos para 100 Números Aleatórios

Arranjo	Tempo (µs)		Comparações		Memória (*bytes*)	
	Média	Std.	Média	Std.	Média	Std.
100.000	231	283	16	1	21.460.910	22.992
200.000	207	123	17	1	26.985.513	13.654
300.000	153	57	17	1	32.792.576	0
400.000	181	70	17	2	37.726.290	2.046
500.000	165	63	18	2	43.541.709	70.757
600.000	165	53	18	2	48.459.776	0
700.000	154	65	18	1	56.311.808	0
800.000	194	242	19	2	59.322.368	0
900.000	196	189	19	2	65.228.800	0
1.000.000	191	214	19	1	69.111.808	0

A Tabela 7 apresenta os resultados de piores cenários nos testes de buscas em árvores binárias de busca desbalanceadas para arranjos não ordenados. Assim como em testes anteriores, escolhemos três números que não existiam no cenário de modo a identificar o pior caso, escolhendo o primeiro elemento do arranjo como sua raiz.

Os resultados demonstram que se comparam menos elementos que nas estruturas sequenciais, variando de 10 comparações em média no arranjo de 100.000 elementos a 14 comparações no arranjo de 1.000.000 de elementos. Neste cenário, mesmo arranjos maiores podem oferecer menos comparações que outros devido à natureza da estrutura, uma vez que árvores desbalanceadas podem ter diferentes tamanhos em cada um de seus lados e, como consequência, a busca pode percorrer um destes lados e finalizar antes, sendo este maior ou não de acordo com a divisão do arranjo e também baseado na escolha do valor da sua raiz.

A utilização da memória foi superior aos cenários onde utilizamos arranjos e listas ligadas devido à estrutura da árvore necessária, variando em média de 37.462.016 *bytes* para o arranjo de 100.000 elementos a 219.545.600 *bytes* para o arranjo de 1.000.000 de elementos.

Tabela 7. Busca em Árvore Binária de Busca para Pior Cenário

Arranjo	Tempo (µs)		Comparações		Memória (*bytes*)	
	Média	Std.	Média	Std.	Média	Std.
100.000	14	5	10	0	37.462.016	0
200.000	19	7	14	0	56.393.728	0
300.000	27	6	21	0	76.460.032	0
400.000	72	75	19	0	96.624.640	0
500.000	18	9	14	0	116.736.000	0
600.000	17	7	10	0	137.076.736	0
700.000	35	30	7	0	157.237.248	0
800.000	30	17	15	0	179.437.568	0
900.000	27	12	11	0	197.558.272	0
1.000.000	24	5	14	0	219.545.600	0

A Tabela 8 apresenta os resultados dos testes de buscas na árvore binária de busca desbalanceadas para 100 buscas aleatórias. Os resultados demonstram que compara-se

menos quando comparadas a estruturas sequencias, no entanto a um custo alto de memória para todas as etapas, em média 36.912.087 *bytes* para o arranjo de 100.000 elementos e 218.613.146 *bytes* para o arranjo de 1.000.000 elementos.

A quantidade média de comparações varia entre 22 para o arranjo de 100.000 elementos e 26 para o arranjo de 1.000.000 de elementos, com desvio padrão médio oscilando entre 5 a 6 ligeiramente maiores do que os valores em arranjos usados para buscas binárias, mas ainda muito menores que os valores observados nas buscas sequenciais.

Outra questão que podemos notar é que os cenários podem variar muito devido ao não balanceamento da árvore, e com alguns casos podendo degenerar a árvore tendendo ela para uma lado extremamente maior que o outro.

Tabela 8. Busca em Árvore Binária de Busca para 100 Números Aleatórios

Arranjo	Tempo (µs)		Comparações		Memória (*bytes*)	
	Média	Std.	Média	Std.	Média	Std.
100.000	227	89	22	5	36.912.087	21.391
200.000	221	74	22	5	57.492.111	14.548
300.000	236	124	22	6	77.578.240	0
400.000	223	68	23	4	97.894.400	0
500.000	268	125	27	5	118.018.048	0
600.000	240	99	25	5	138.186.752	0
700.000	224	95	25	5	158.290.903	3.891
800.000	267	180	26	5	178.409.472	0
900.000	245	96	25	4	198.533.120	0
1.000.000	236	119	26	5	218.613.146	1.954

Discutiremos a seguir, na Tabela 9, os custos para ordenação dos arranjos. Utilizamos o algoritmo padrão da linguagem *Python* na versão 3.10.9, que utiliza o algoritmo *TimSort* [Peters 2002], criado e implementado por *Tim Peters* a partir da versão 2.3 e substituído pelo algoritmo *PowerSort* [Munro and Wild 2018] na versão 3.11.0 [Python Software Foundation 2024].

Segundo [Peters 2002], o *TimSort* é um algoritmo de ordenação híbrido e estável, derivado do *merge sort* [Cormen et al. 2012] e do *insertion sort* [Knuth 1998], projetado para ter um bom desempenho em muitos tipos de dados do mundo real. De acordo com o mesmo, o algoritmo possui complexidade $O(n \log n)$ no pior caso, $O(n)$ no melhor caso e $O(n \log n)$ em casos médios.

Utilizamos a ordenação para realizar os testes em busca binária nos arranjos estáticos, uma vez que a ordenação se faz necessária para que o algoritmo possa trabalhar corretamente a partir da sua divisão ao meio e comparando primeiro o elemento do meio com o valor procurado, de modo que descarte a busca na outra metade do arranjo após validar se este é menor ou maior ao número comparado [Dasgupta et al. 2006].

O tempo total de ordenação oscilou de 14.316 µs para o arranjo de 100.000 elementos a 359.885 µs no arranjo de 1.000.000 de elementos, com o uso de memória variando de um total de 19.824.640 *bytes* no arranjo de 100.000 elementos para 93.679.616 *bytes* no arranjo de 1.000.000 de elementos.

É importante observar e também acrescentar este tempo ao tempo da busca binária

em arranjos estáticos, uma vez que essa ordenação é necessária no cenário para que, por fim, possa ser verificado o tempo e custo necessário para as buscas.

Tabela 9. Custo de Ordenação

Arranjo	Tempo (µs)	Memória (*bytes*)
100.000	14.316	19.824.640
200.000	33.483	27.893.760
300.000	139.639	35.991.552
400.000	73.259	44.134.400
500.000	116.211	53.690.368
600.000	125.418	60.469.248
700.000	150.001	69.611.520
800.000	174.107	76.521.472
900.000	222.831	86.028.288
1.000.000	359.885	93.679.616

5. Conclusão

Com base nos resultados dos testes realizados para diferentes métodos de busca em arranjos estáticos e estruturas de dados lineares e não-lineares, podemos inferir como os algoritmos operam. No caso da busca sequencial, os testes corroboram a notação de Big O, demonstrando que a quantidade de comparações aumenta linearmente com o tamanho da lista, o que confirma a complexidade $O(n)$.

Os testes também confirmam que a mesma relação de Big O se aplica a listas de complexidade $O(\log n)$, onde a quantidade de comparações varia em média de acordo com o logaritmo do número total de elementos. Isso evidencia que tais algoritmos são adequados para cenários onde se busca minimizar tanto o tempo quanto a quantidade de comparações necessárias.

A seleção do método ideal para uma aplicação específica depende da implementação. Em conjuntos pequenos ou quando a ordenação prévia é impraticável, a busca sequencial pode ser suficiente. No entanto, para conjuntos de dados grandes, ordenados ou não, a busca binária é, indubitavelmente, a melhor escolha.

Conclui-se, portanto, que a escolha da estrutura de dados influencia diretamente no método de busca. As listas ligadas são destacadas em situações que demandam flexibilidade, enquanto as árvores são mais adequadas para cenários de busca.

É importante ressaltar que as análises e conclusões apresentadas neste relatório são derivadas dos resultados obtidos nos testes conduzidos. Contudo, é fundamental reconhecer que tais conclusões podem não ser generalizadas para todas as situações, uma vez que o desempenho e a eficácia da estrutura de dados podem variar conforme as características específicas do problema ou do ambiente de implementação.

Referências

Bhargava, A. Y. (2016). *Entendendo Algoritmos, um guia ilustrado para programadores e outros curiosos*. Novatec Editora, São Paulo, Brasil.

Cormen, T. H., Leiserson, C. E., Rivest, R. L., and Stein, C. (2012). *Algoritmos - Teoria e Prática*. Elsevier, Rio de Janeiro, 3 edition.

Dasgupta, S., Papadimitriou, C. H., and Vazirani, U. V. (2006). *Algorithms*. McGraw-Hill, New York, NY.

Knuth, D. E. (1998). *The Art of Computer Programming, Volume 3: (2nd Ed.) Sorting and Searching*. Addison Wesley Longman Publishing Co., Inc., Redwood City, CA, USA.

Munro, J. I. and Wild, S. (2018). Nearly-Optimal Mergesorts: Fast, Practical Sorting Methods That Optimally Adapt to Existing Runs. In Azar, Y., Bast, H., and Herman, G., editors, *26th Annual European Symposium on Algorithms (ESA 2018)*, volume 112 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 63:1–63:16, Dagstuhl, Germany. Schloss Dagstuhl – Leibniz-Zentrum für Informatik.

Peters, T. (2002). Sorting in python: The history of timsort. Access 2024-05-18.

Python Software Foundation (2024). Python 3.11.0 changelog. Access 2024-05-18.