Comparação da Eficiência dos Algoritmos de Busca Sequencial Padrão, Busca por Saltos e Busca Binária

Marcos Alexandre dos Anjos[1]

[1]Programa de Pós Gradução em Ciência da Computação - PPGCOMP
Universidade Estadual do Oeste do Paraná (UNIOESTE)
Caixa Postal 85.819-110 – Cascavel– PR – Brasil

dev.marcosanjos@gmail.com

Abstract. *This article empirically analyzes the efficiency of three search algorithms on static arrays with integer values: standard sequential search, jump search, and binary search. The database was generated with arrays of values ranging from 100,000 to 1 million, with intervals of 100,000 positions with random numbers. The algorithms were subjected to the same search scenarios, including worst-case, random search case, with three runs of each algorithm. The results indicated that binary search is the most efficient for large data sets, while jump search is the best option for smaller sets. This research can be useful in helping developers choose the most suitable search algorithm for their specific needs.*

Resumo. *Este artigo analisa empiricamente a eficiência de três algoritmos de busca em arranjos estáticos com valores inteiros: busca sequencial padrão, busca por saltos (Jump search) e busca binária. A base de dados foi gerada com arranjos de valores variando de 100 mil até 1 milhão, com intervalos de 100 mil posições com números aleatórios. Os algoritmos foram submetidos aos mesmos cenários de busca, incluindo pior caso, caso busca aleatória, com três execuções de cada algoritmo. Os resultados indicaram que a busca binária é a mais eficiente para grandes conjuntos de dados, enquanto a busca por saltos é a melhor opção para conjuntos menores. Esta pesquisa pode ser útil para ajudar desenvolvedores a escolher o algoritmo de busca mais adequado para suas necessidades específicas.*

1. Instrodução

O desenvolvimento e análise de algoritmos compreende nas dimensões de correntude e na complexidade. A análise de corretude pretende determinar se o algoritmo chega à solução desejada, enquanto a análise de complexidade determina qual será o custo desse algoritmo na busca pelo problema. Esses passos são importantes no desenvolvimento dos algoritmos, e por isso são estudados nas disciplinas de Estrutura de Dados e Análise de Algoritmos em cursos de computação.

Neste sentido, o presente trabalho propõe um estudo sobre os algoritmos de busca e uma análise de custo apresentado qual a eficiência, podendo de forma compará-los variando o tamanho do arranjo da amostra.

2. Referencial Teórico

O referencial teórico deste estudo abrange os conceitos e princípios fundamentais das técnicas de busca sequencial padrão, busca por saltos (jump search) e busca binária, bem como suas respectivas complexidades computacionais e limitações.

2.1. Algoritmo de Busca Sequencial padrão

Trata-se do método de busca mais simples. Dado um conjunto de números, letras, endereços ou independentemente do tipo de variável existente num programa, basta percorrer e comparar um a um partir do primeiro elemento da lista. Assim esse elemento é encontrado, seu índice é retornado e a busca é finalizada. Caso não exista o elemento no conjunto, o programa retorna algum valor que não possa ser conjunto com índice indicado que não foi encontrado (Ziviani, 2004).

Analisando o algoritmo de busca sequencial, tempos que em sue melhor caso (a) elemento na primeira posição e o algoritmo possui complexidade O(1); (b) no caso médio o elemento na posição central tem complexidade O(n/2); e (c) o pior caso é na última posição apresenta complexidade igual a O(n) (Ziviani, 2004).

2.2. Algoritmo de Busca por saltos (*Jump search*)

O algoritmo de busca Jump Search, também conhecido como Busca por Salto, é um algoritmo de busca em uma lista ordenada que utiliza saltos para percorrer a lista de forma mais eficiente que uma busca sequencial. O algoritmo é baseado em uma ideia simples de saltar a uma posição específica na lista em vez de percorrê-la item por item.

De acordo com o artigo "Jump Search Algorithm: A Review" de Ajay Kumar Pandey e Mukesh Kumar Vishwakarma, publicado na International Journal of Advanced Research in Computer Science and Software Engineering em 2013, o algoritmo de busca Jump Search possui uma complexidade de tempo de O(sqrt(n)), onde n é o tamanho da lista. Isso o torna mais eficiente do que a busca sequencial em grandes listas, enquanto ainda é fácil de implementar e requer uma quantidade razoável de memória (Pandey and Vishwakarma, 2013). O artigo também discute algumas variações e otimizações do algoritmo de busca Jump Search, como o uso de interpolação para escolher o tamanho do salto e a verificação da frequência de saltos para melhorar ainda mais o desempenho.

2.3. Algoritmo de Busca Binária

O algoritmo de busca binária é um método eficiente de encontrar um elemento em um conjunto ordenado de dados. O algoritmo divide repetidamente o conjunto ao meio e verifica se o elemento buscado está na metade esquerda ou direita. Esse processo é repetido até que o elemento seja encontrado ou a metade atual seja reduzida a zero.

De acordo com o artigo "A Binary Search Implementation for Embedded Systems" (2019), a busca binária tem uma complexidade de tempo de O(log n), onde n é o número de elementos no conjunto de dados (Ali, 2019). Isso significa que, para grandes conjuntos de dados, a busca binária é significativamente mais rápida do que a busca sequencial. Além disso, o artigo "Analysis of Binary Search Algorithm" (2017) apresenta uma análise matemática mais detalhada da eficiência do algoritmo de busca binária (Rahman and Islam, 2017).

3. Metodologia

Neste trabalho, objetivo é comparar empiricamente a eficiência de três algoritmos de busca em arranjos estátisticos com valores inteiros. Os algoritmos escolhidos são a busca sequencial padrão, busca por saltos (junp search) e busca binária.

Para isso, foi implementado os algoritmos em código em Python contendo vários scripts necessários para análise da proposta. Inicialmente, foi criado um script para gerar a base de dados, que consiste em arranjos de valores variando de 100 mil até 1 milhão, com intervalos de 100 mil posições com numeros aleatórios.

Segunda etapa do processo, foi realizado uma análise dos algoritmos de busca, e realizado a implementação somado aos scripts para controle de repetição. Onde cada algotimo foi realizado o mesmo teste três vezes, para obter um tempo médio. Para que seja possivel realizar os calculos de média, desvio padrão. Os três algoritmos foram submetidos aos mesmos cenarios, que são eles: (a) pior caso, com três execuções de cada algoritmo; (b) caso aleatório com cem buscas para cada cenário.

É importante destacar que o custo de criação do arrnjo e calculo desvio padrão, criação de arquivos, tempo de ordenação foi descartado, foco principal está analise do tempo de busca.

4. Resultados

Os resultados exprementais foram obtidos a partir dos testes realizados apartir de uma maquina com as configurações detalhadas apresentada na Tabela 1.

Table 1. Detalhes sobre Ambiente de Testes

Configuração	Detalhes
Marca do notebook	Avell Liv
Processador	i7 - 1050H
Memoria RAM	16 gigas - DDR4
Sistema Operacional	Windows 11 PRO
Linguagem de programação	Python 3.11.3

4.1. Algoritimo de Busca Sequencial

Os resultados obtidos pelo algoritmo de busca sequencial estão apresentados na Tabela 2 (Anexos). Podemos comparar os cenários A e B e observar que o custo de comparações desse algoritmo é extremamente alto. No caso A, o custo cresce proporcionalmente à quantidade de valores presentes na lista. Já no caso B, que envolve a busca por um valor aleatório, os resultados são dispersos, podendo apresentar resultados menos custosos em alguns casos.

Entretanto, ao se observar a média, o custo da busca é alto, já que muitas comparações precisam ser feitas. Na Figura 1, podemos ter uma visualização da comparação dos dois cenários, em azul representa cenário A e vermelho representa cenário B.

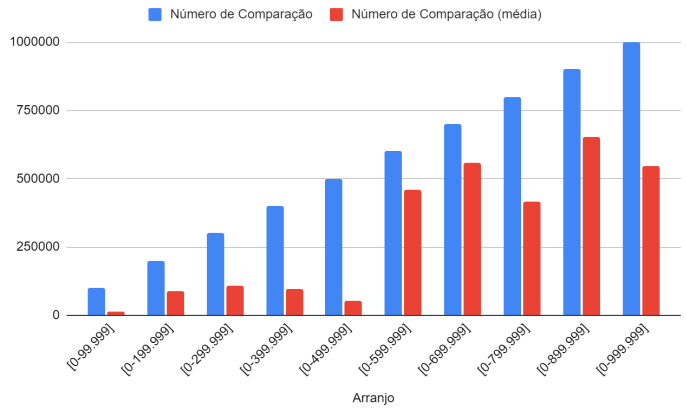

Figura 1. Grafico relação número de comparação case A vs número comparação case B.

4.2. Algoritimo de Busca por Saltos (jump search)

O algoritmo de busca por saltos, também conhecido como jump search, apresentou um desempenho superior em relação ao algoritmo de busca sequencial. Analisando os dados da Tabela 3 (Anexos), os custo deste algoritmo, observando o número de comparações necessárias para realizar as buscas nos cenários A e B, os resultados foram surpreendentes. No caso A, foram necessárias entre 17 a 20 comparações para realizar a busca. Já no caso B, os resultados foram semelhantes, ficando entre 15 a 19 comparações. Isso nos permite entender que este algoritmo apresenta alta performance de busca.

Para uma melhor visualização a Figura 2, apresenta uma comparação dos cenários A e B com suas respectivas cores azul e vermelho. Em que podemos observar visualmente que case A teve um numero de comparações superior ao case B. Mas mesmo assim os resultados são proximos.

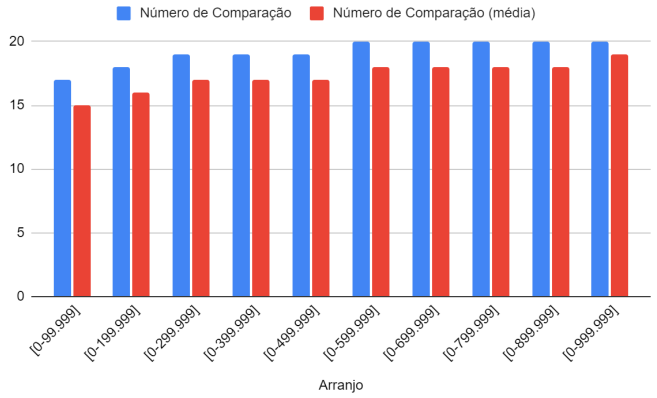

Figura 2. Grafico relação número de comparação case A vs número comparação case B.

Embora tenha uma alta performance, um ponto fraco deste algoritmo é sua complexidade para implementação. Para obter resultados de tempo mais palpáveis, foi adicionado um sleep de 0.001 segundo. No entanto, em comparação com o algoritmo de

busca sequencial, o jump search se mostrou muito mais eficiente na busca de elementos em uma lista ordenada.

4.3. Algoritimo de Busca Binária

Ao analisar os resultados da Tabela 4 (Anexos), é possível constatar que o algoritmo de busca binária apresentou uma performance surpreendente em ambos os cenários, quando comparado ao algoritmo de busca por saltos. Observando a média de comparações necessárias para realizar a busca, podemos perceber que o algoritmo de busca binária obteve resultados muito eficientes. No entanto, é importante ressaltar que no cenário A, a média de comparações foi ligeiramente superior em comparação ao cenário B. Isso ocorre porque, no cenário B, os dados são gerados aleatoriamente, o que pode influenciar diretamente na relação entre o número de comparações e o tempo médio de execução.

A implementação do algoritmo de busca binária é relativamente simples, mas seu desempenho é bastante eficiente na busca em listas ordenadas. Devido à sua alta performance, foi adicionado um sleep de 0.001 segundos para apresentar uma medida de tempo mais fácil de entender e visualizar. Com essa medida, é possível observar que o algoritmo de busca binária apresenta resultados muito bons em termos de tempo de execução e número de comparações realizadas, principalmente quando comparado aos algoritmos de busca sequencial e busca por saltos. Na Figura 3, podemos ter uma visualização sobre o tempo case A e B com suas respectivas cores azul e vermelho. Onde que temos numero de comparalhos são relativamente pequenas, quando comparamos o mesmo grafico com busca sequencial.

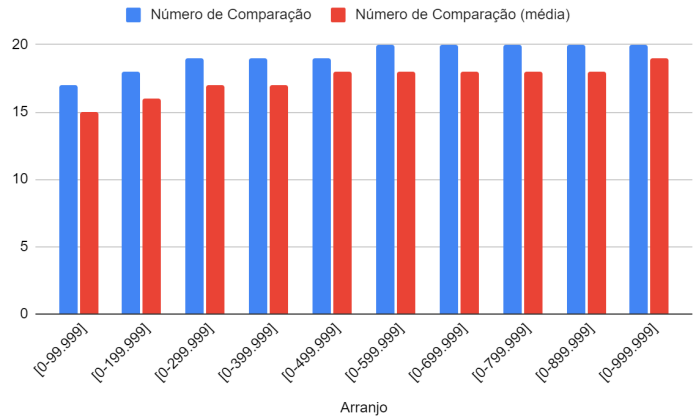

Figura 3. Grafico relação número de comparação case A vs número comparação case B.

5. Conclusão

Após a realização dos experimentos, podemos concluir que o algoritmo de busca sequencial é o menos eficiente em termos de performance. O custo de comparações é proporcional à quantidade de valores presentes na lista, o que o torna inviável para grandes conjuntos de dados.

Já o algoritmo de busca por saltos apresentou um desempenho superior em relação ao sequencial, realizando uma quantidade significativamente menor de comparações. Isso o torna uma opção mais viável para grandes conjuntos de dados. Porém, sua implementação é um pouco mais complexa do que a busca binária.

Falando na busca binária, este algoritmo apresentou uma performance surpreendente em ambos os cenários. A quantidade média de comparações realizadas foi significativamente menor do que a da busca sequencial, e em alguns casos até mesmo menor do que a da busca por saltos. Além disso, sua implementação é relativamente simples, o que o torna uma opção bastante viável em diversas situações.

Dessa forma, podemos concluir que a escolha do algoritmo de busca mais adequado dependerá das características específicas de cada problema. Para conjuntos de dados pequenos, a busca sequencial pode ser suficiente. Para conjuntos de dados maiores, a busca por saltos pode ser uma opção melhor do que a sequencial. E para grandes conjuntos de dados, a busca binária pode ser a opção mais eficiente em termos de performance e simplicidade de implementação.

References

Boulic, R. and Renault, O. (1991) “3D Hierarchies for Animation”, In: New Trends in Animation and Visualization, Edited by Nadia Magnenat-Thalmann and Daniel Thalmann, John Wiley & Sons ltd., England.

Dyer, S., Martin, J. and Zulauf, J. (1995) “Motion Capture White Paper”, http://reality.sgi.com/employees/jam_sb/mocap/MoCapWP_v2.0.html, December.

Holton, M. and Alexander, S. (1995) “Soft Cellular Modeling: A Technique for the Simulation of Non-rigid Materials”, Computer Graphics: Developments in Virtual Environments, R. A. Earnshaw and J. A. Vince, England, Academic Press Ltd., p. 449-460.

Knuth, D. E. (1984), The TeXbook, Addison Wesley, 15th edition.

Smith, A. and Jones, B. (1999). On the complexity of computing. In *Advances in Computer Science*, pages 555–566. Publishing Press.

Anexos

Tabela 2. Resultados da busca por Sequêncial Aplicado nos Cenários A e B

	Case A			Case B		
Arranjo	**Tempo de Execução Médio**	**Desvio Padrão**	**Número de Comparação**	**Tempo de Execução Médio**	**Desvio Padrão**	**Número de Comparação (média)**
[0-99.999]	0,00469	0,000581	100000	0,00278	0,00008	13122
[0-199.999]	0,01065	0,001161	200000	0,00561	0,00021	89166
[0-299.999]	0,01541	0,001655	300000	0,00771	0,00062	107146
[0-399.999]	0,02119	0,001330	400000	0,01140	0,00092	94739
[0-499.999]	0,02819	0,001405	500000	0,01392	0,00142	53554
[0-599.999]	0,03066	0,001522	600000	0,01636	0,00067	460920
[0-699.999]	0,03633	0,002517	700000	0,02034	0,00159	557652
[0-799.999]	0,04236	0,002469	800000	0,02297	0,00160	416054
[0-899.999]	0,04816	0,003019	900000	0,02374	0,00127	652826
[0-999.999]	0,04901	0,001505	1000000	0,02760	0,00092	546230

Tabela 3. Resultados da Busca por Saltos Aplicado nos Cenários A e B

	Case A			Case B		
Arranjo	**Tempo de Execução Médio**	**Desvio Padrão**	**Número de Comparação**	**Tempo de Execução Médio**	**Desvio Padrão**	**Número de Comparação (média)**
[0-99.999]	0,36978	0,01242	17	0,50088	0,01584	15
[0-199.999]	0,67416	0,15786	18	0,70114	0,06839	16
[0-299.999]	0,78894	0,02179	19	0,77301	0,10126	17
[0-399.999]	0,90170	0,00720	19	1,03045	0,03086	17
[0-499.999]	1,00172	0,03159	19	1,11747	0,07748	17
[0-599.999]	1,12439	0,02475	20	1,11025	0,03356	18
[0-699.999]	1,21971	0,02933	20	1,24855	0,04266	18

[0-799.999]	1,23653	0,00446	20	1,28364	0,05018	18
[0-899.999]	1,16488	0,08673	20	1,35538	0,02959	18
[0-999.999]	1,33264	0,03699	20	1,39686	0,02817	19

Tabela 4. Resultados da Busca por Binária Aplicado nos Cenários A e B

	Case A			**Case B**		
Arranjo	**Tempo de Execução Médio**	**Desvio Padrão**	**Número de Comparação**	**Tempo de Execução Médio**	**Desvio Padrão**	**Número de Comparação (média)**
[0-99.999]	0,00192	0,00046	17	0,02147	0,00041	15
[0-199.999]	0,00174	0,00034	18	0,02420	0,00101	16
[0-299.999]	0,00150	0,00012	19	0,02425	0,00138	17
[0-399.999]	0,00236	0,00011	19	0,02441	0,00022	17
[0-499.999]	0,00157	0,00047	19	0,02422	0,00077	18
[0-599.999]	0,00210	0,00015	20	0,02423	0,00046	18
[0-699.999]	0,00197	0,00040	20	0,02573	0,00037	18
[0-799.999]	0,00191	0,00021	20	0,02493	0,00023	18
[0-899.999]	0,00166	0,00060	20	0,02708	0,00130	18
[0-999.999]	0,00133	0,00026	20	0,02590	0,00018	19