**Avaliação de algoritmos de busca – Parte 1**

Marcos Agnes

1 Unioeste, Cascavel – Paraná, BR

Estrutura de Dados e Análise e Algoritmos - EDAA

**Resumo.** A utilização de algoritmos otimizados pode acarretar ganhos consideráveis, principalmente quando se está trabalhando com uma grande quantidade de registros. Este trabalho apresenta a comparação de três algoritmos lineares, que utilizando estratégias diferentes buscam um valor em uma lista.

**Keywords:** Busca Sequencial, Busca Binaria, Busca de Saltos.

1. Introdução

No desenvolvimento de sistemas e aplicações, a eficiência dos algoritmos de busca desempenha um papel fundamental. A escolha adequada da estrutura de dados é essencial para otimizar o desempenho desses algoritmos. Neste contexto, este estudo tem como objetivo implementar e comparar quatro estruturas de dados amplamente utilizadas: Sequential Search, Binary Search, Linked List Search e Binary Search Tree.

O objetivo principal deste estudo é avaliar o desempenho e a eficiência das estruturas de dados mencionadas por meio de uma análise comparativa. Pretendemos identificar qual estrutura é mais adequada para diferentes cenários e tipos de dados. Além disso, buscamos responder ao seguinte problema de pesquisa: "Qual estrutura de dados oferece o melhor desempenho em termos de tempo de execução e eficiência na busca de elementos?".

1. Revisão de Literatura

Sequential Search, também conhecida como busca sequencial, é um método simples e direto de busca de elementos em uma lista. Ela examina cada item da lista sequencialmente até encontrar uma correspondência ou atingir o final da lista. A complexidade de tempo da Sequential Search é O(n), onde 'n' é o tamanho da lista, o que pode torná-la ineficiente para grandes conjuntos de dados [1].

Busca binária (Binary Search) é amplamente utilizado. A busca binária é aplicável apenas em listas ordenadas e utiliza uma abordagem de divisão e conquista. Ao comparar o elemento buscado com o elemento do meio da lista, o algoritmo descarta metade dos elementos em cada iteração, reduzindo significativamente o número de comparações necessárias. A complexidade de tempo da busca binária é O(log n), onde 'n' é o tamanho da lista. Essa característica torna a busca binária extremamente rápida, especialmente em grandes conjuntos de dados ordenados [2].

Linked List Search é uma estrutura de dados encadeada, em que cada elemento contém um ponteiro para o próximo elemento. Embora seja fácil de manipular e inserir novos elementos, a busca em uma lista encadeada requer percorrer a lista item por item, o que resulta em uma complexidade de tempo de O(n) no pior caso [3].

Binary Search Tree (Árvore de Busca Binária) é uma estrutura de dados hierárquica que permite buscas eficientes e outras operações, como inserção e exclusão de elementos. A árvore é organizada de forma a garantir que todos os elementos na sub-árvore esquerda de um nó sejam menores que o próprio nó, e todos os elementos na sub-árvore direita sejam maiores. A complexidade de tempo para buscar um elemento em uma Binary Search Tree é O(log n) no caso médio, onde 'n' é o número de elementos na árvore [4].

1. Materiais e Métodos

Neste capítulo será abordado os materiais utilizados para execução das comparações e os softwares utilizados.

A máquina que foi utilizada para a execução das comparações foi um macbook com um chip M1 Pro de 10 núcleos e 16 GB RAM. Os softwares utilizados foram: VSCode (Como IDE de desenvolvimento) com uma extensão Jupiter e Python 3.9 (Para utilização da linguagem de programação Python.

A linguagem de desenvolvimento selecionada para a implementação dos algoritmos e a lógica de comparação foi o Python. Utilizando o Python juntamente com o Jupiter Notebook é possível executar o código em “blocos”, assim permitindo reaproveitar processamentos previamente executados, essa abordagem traz um ganho de tempo considerável, pois não necessita a execução de todos o código de uma única vez.

Foi utilizado uma abordagem quantitativa, onde cada algoritmo foi executado em dez cenários diferentes (Arrays com tamanhos diferentes) e executado 100 vezes para cada um dos cenários.

Cada execução gera uma saída com o nome do algoritmo, tempo de execução em nanosegundo, tamanho do array, número de comparações e altura (este só é alimentado na arvore binária de busca, os demais recebem zero).

Para cada cenário é gerado um valor aleatório para ser utilizado como chave de busca. Para isso foi utilizado a função *randrange* do pacote *random* do Python.

As implementações de cada estrutura foram retiradas do Github [5], [6], [7.] e [8], já para criar o array e realizar a operação de inicialização com valores aleatórios foi utilizado a biblioteca Numpy.

1. Resultados

Os algoritmos Busca Binaria e Pesquisa de Saltos apresentam um resultado tão melhor que a busca sequencial, que teve de ser adicionado uma instrução “sleep” (dormir), que visa fazer o código pausar e aguardar 10 milissegundos.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Nome** | **AVG Time (ns)** | **AVG Comp.** | **STD Time (ns)** | **STD Comp.** |
| **Sequential Search 3000** | 178620.0 | 76632.3 | 89033.22750524098 | 43760.10731876694 |
| **Binary Search 3000** | 4420.0 | 537.42 | 1069.392350823588 | 308.3061848228154 |
| **Linked List Search 3000** | 317640.0 | 81826.11 | 163707.1482862004 | 46595.40367995431 |
| **Binary Search Tree 3000** | 308120.0 | 50.5 | 196798.59145837402 | 28.86607004772212 |
| **Sequential Search 6000** | 353380.0 | 146365.27 | 184404.38064210946 | 87476.56799427547 |
| **Binary Search 6000** | 4840.0 | 593.64 | 1488.086019019062 | 339.9665430597546 |
| **Linked List Search 6000** | 599310.0 | 147731.32 | 343388.7503981457 | 88572.98767941387 |
| **Binary Search Tree 6000** | 624790.0 | 50.5 | 403040.5263742097 | 28.86607004772212 |
| **Sequential Search 9000** | 485380.0 | 201986.02 | 286440.8064504776 | 123434.07598503584 |
| **Binary Search 9000** | 4790.0 | 622.59 | 908.7904048789247 | 352.53553281903373 |
| **Linked List Search 9000** | 912230.0 | 237540.11 | 522695.20477999415 | 131093.5346757341 |
| **Binary Search Tree 9000** | 941190.0 | 50.5 | 595315.7598283453 | 28.86607004772212 |
| **Sequential Search 12000** | 610470.0 | 266892.3 | 326807.4801775504 | 153201.9909379444 |
| **Binary Search 12000** | 4790.0 | 633.01 | 1210.7435731813734 | 359.20129440189936 |
| **Linked List Search 12000** | 1293350.0 | 341347.17 | 682244.7563008454 | 187311.54302183594 |
| **Binary Search Tree 12000** | 1305210.0 | 50.5 | 690887.9546641409 | 28.86607004772212 |
| **Sequential Search 15000** | 895540.0 | 395848.78 | 479457.4521268806 | 235918.74374176294 |
| **Binary Search 15000** | 5240.0 | 647.23 | 2506.4716236175504 | 368.8995487934351 |
| **Linked List Search 15000** | 1426580.0 | 378409.68 | 890060.1685279484 | 202585.1580400144 |
| **Binary Search Tree 15000** | 1580250.0 | 50.5 | 933566.6593768224 | 28.86607004772212 |
| **Sequential Search 18000** | 1108250.0 | 484862.08 | 592390.9752013446 | 272675.06243471114 |
| **Binary Search 18000** | 5230.0 | 665.73 | 1355.3966209195005 | 378.4905508728058 |
| **Linked List Search 18000** | 1673520.0 | 419182.45 | 1078135.9235272703 | 240433.75276189385 |
| **Binary Search Tree 18000** | 2203350.0 | 50.5 | 1141516.0303298417 | 28.86607004772212 |
| **Sequential Search 21000** | 1238680.0 | 542437.1 | 605813.3851277966 | 312840.98639831383 |
| **Binary Search 21000** | 5210.0 | 680.43 | 1185.7065404222076 | 390.2109751147448 |
| **Linked List Search 21000** | 1942000.0 | 490134.4 | 1263024.7107638076 | 276893.3211836284 |
| **Binary Search Tree 21000** | 2483860.0 | 50.5 | 1375362.635961876 | 28.86607004772212 |
| **Sequential Search 24000** | 1375590.0 | 581035.08 | 740236.5850861466 | 353001.5304533871 |
| **Binary Search 24000** | 5340.0 | 693.32 | 907.9647570252934 | 392.50403513849386 |
| **Linked List Search 24000** | 2387350.0 | 596696.19 | 1329830.1423490145 | 343744.0158024484 |
| **Binary Search Tree 24000** | 2620710.0 | 50.5 | 1508561.1840094521 | 28.86607004772212 |
| **Sequential Search 27000** | 1461760.0 | 635057.57 | 763136.8569267245 | 349043.96580325684 |
| **Binary Search 27000** | 5210.0 | 700.3 | 951.7877914745492 | 403.2070311886934 |
| **Linked List Search 27000** | 2790670.0 | 734886.72 | 1499995.8536942694 | 416184.123543993 |
| **Binary Search Tree 27000** | 3118650.0 | 50.5 | 1761629.1458476724 | 28.86607004772212 |
| **Sequential Search 30000** | 1839510.0 | 803105.0 | 884717.8137123723 | 465663.85206092603 |
| **Binary Search 30000** | 5180.0 | 701.86 | 942.1252570651102 | 400.8364509372869 |
| **Linked List Search 30000** | 2945960.0 | 750010.56 | 1664365.1217205918 | 448953.5064985754 |
| **Binary Search Tree 30000** | 3388880.0 | 50.5 | 1832967.4971477264 | 28.86607004772212 |

..

|  |  |  |
| --- | --- | --- |
| **Nome** | **Worse Time (ns)** | **Worse Comp.** |
| **Sequential Search 3000** | 336000 | 148222 |
| **Binary Search 3000** | 11000 | 1062 |
| **Linked List Search 3000** | 617000 | 160642 |
| **Binary Search Tree 3000** | 671000 | 100 |
| **Sequential Search 6000** | 796000 | 299821 |
| **Binary Search 6000** | 17000 | 1174 |
| **Linked List Search 6000** | 1173000 | 302271 |
| **Binary Search Tree 6000** | 1276000 | 100 |
| **Sequential Search 9000** | 1030000 | 420117 |
| **Binary Search 9000** | 10000 | 1228 |
| **Linked List Search 9000** | 1712000 | 469367 |
| **Binary Search Tree 9000** | 1970000 | 100 |
| **Sequential Search 12000** | 1299000 | 532865 |
| **Binary Search 12000** | 14000 | 1253 |
| **Linked List Search 12000** | 2390000 | 653517 |
| **Binary Search Tree 12000** | 2614000 | 100 |
| **Sequential Search 15000** | 1653000 | 786485 |
| **Binary Search 15000** | 26000 | 1291 |
| **Linked List Search 15000** | 2850000 | 736085 |
| **Binary Search Tree 15000** | 3331000 | 100 |
| **Sequential Search 18000** | 2592000 | 949601 |
| **Binary Search 18000** | 15000 | 1307 |
| **Linked List Search 18000** | 3602000 | 841723 |
| **Binary Search Tree 18000** | 4217000 | 100 |
| **Sequential Search 21000** | 2299000 | 1082803 |
| **Binary Search 21000** | 14000 | 1346 |
| **Linked List Search 21000** | 4254000 | 952131 |
| **Binary Search Tree 21000** | 5198000 | 100 |
| **Sequential Search 24000** | 2685000 | 1170673 |
| **Binary Search 24000** | 11000 | 1369 |
| **Linked List Search 24000** | 4769000 | 1207206 |
| **Binary Search Tree 24000** | 5364000 | 100 |
| **Sequential Search 27000** | 2818000 | 1284816 |
| **Binary Search 27000** | 12000 | 1389 |
| **Linked List Search 27000** | 5231000 | 1457590 |
| **Binary Search Tree 27000** | 6143000 | 100 |
| **Sequential Search 30000** | 3246000 | 1630198 |
| **Binary Search 30000** | 12000 | 1387 |
| **Linked List Search 30000** | 6074000 | 1531549 |
| **Binary Search Tree 30000** | 6599000 | 100 |

..

|  |  |
| --- | --- |
| **Nome** | **Height AVG** |
| **Binary Search Tree 3000** | 2998.0 |
| **Binary Search Tree 6000** | 5998.0 |
| **Binary Search Tree 9000** | 8998.0 |
| **Binary Search Tree 12000** | 11998.0 |
| **Binary Search Tree 15000** | 14998.0 |
| **Binary Search Tree 18000** | 17998.0 |
| **Binary Search Tree 21000** | 20998.0 |
| **Binary Search Tree 24000** | 23998.0 |
| **Binary Search Tree 27000** | 26998.0 |
| **Binary Search Tree 30000** | 29998.0 |

1. Referências

[1] BRESHEARS, Clay. The art of concurrency: A thread monkey's guide to writing parallel applications. " O'Reilly Media, Inc.", 2009.

[2] KNUTH, Donald Ervin. The art of computer programming: sorting and searching. In: The art of computer programming: sorting and searching. 1975. p. 723-723.

[3] LAFORE, Robert. Data structures and algorithms in Java. Sams publishing, 2017.

[4] WEISS, Mark Allen. Data structures and algorithm analysis in C++. 2014.

[5] Algoritmos e Estruturas de Dados. Github.com, 2023. Disponível em: https://github.com/kelvins/Algoritmos-e-Estruturas-de-Dados/blob/main/src/python/busca\_binaria.py. Acesso em: 16/05/2023.

[6] BinarySearchTree\_naive\_version.py. Github.com, 2023. Disponível em: https://gist.github.com/zjplab/b052af68196bec5ce48dcc39b0acef2f#file-binarysearchtree\_naive\_version-py. Acesso em: 16/05/2023.

[7] Linked-List-in-Python. Github.com, 2023. Disponível em: https://github.com/M2skills/Linked-List-in-Python/blob/cc1065ee6409e94813137900e58d17afac73ec16/LinkedList.py#L213. Acesso em: 16/05/2023.

[8] Algoritmos-e-Estruturas-de-Dados. Github.com, 2023. Disponível em: https://github.com/kelvins/Algoritmos-e-Estruturas-de-Dados/blob/main/src/python/busca\_sequencial.py. Acesso em: 16/05/2023.