UNIVERSIDADE FEDERAL DO CEARÁ - CAMPUS SOBRAL

PROGRAMA DE PÓS-GRADUAÇÃO EM ENGENHARIA ELÉTRICA E COMPUTAÇÃO

DISCIPLINA: ESTUDOS ESPECIAIS

ALUNO: IAGO MAGALHÃES DE MESQUITA

ALGORITMOS DE BUSCA

Sobral, 2023

ALGORITMOS DE BUSCA

Relatório apresentado como requisito para aprovação na disciplina de Estudos Especiais do programa de pós-graduação em engenharia elétrica e computação da Universidade Federal do Ceará.

Sobral, 2023

SUMÁRIO

1. INTRODUÇÃO

Algoritmos de busca visam solucionar problemas em que dada uma chave de busca e uma coleção de elementos, onde cada elemento possui um identificador único, desejamos encontrar o elemento da coleção que possui o identificador igual ao da chave de busca ou verificar que não existe nenhum elemento na coleção com a chave fornecida [1].

Tais algoritmos podem ser divididos em diferentes complexidades, tais como os 0(n). Um algoritmo é dito que usa tempo linear, ou tempo O(n), se sua complexidade de tempo é O(n). Informalmente, isto significa que para entradas grandes o suficiente o tempo de execução delas aumenta linearmente com o tamanho da entrada. Por exemplo, um procedimento que adiciona todos os elementos em uma lista requere tempo proporcional ao tamanho da lista. Esta descrição é levemente imprecisa, visto que o tempo de execução pode desviar significantemente de uma proporção precisa, especialmente para valores pequenos de n [2].

Em ciência da computação, a complexidade de tempo de um algoritmo quantifica a porção de tempo tomada por um algoritmo para rodar em função do tamanho da entrada do problema. A complexidade de tempo de um algoritmo é comumente expressada usando a notação big O, que suprime constantes multiplicativas e outros termos de menor ordem. Quando expressada dessa forma, a complexidade de tempo é dito ser descrita assintoticamente, i.e., como o tamanho da entrada vai para o infinito. Por exemplo, se o tempo requisitado por um algoritmo em todas as entradas de tamanho n é no máximo 5n3 + 3n, a assíntota da complexidade de tempo é O(n3) [2].

Neste trabalho iremos analisar algoritmos de busca em diferentes graus de complexidade tais como O(n), O(log n), $O(n^2)$ e $O(n^3)$, para diferentes instâncias na qual será analisado o tempo de processamento e uso de memória.

2. OBJETIVOS

2.1 Objetivo Geral

- Realizar experimentos com algoritmos de busca visando analisar tempo de execução e uso de memória para diferentes instâncias.

2.2 Objetivos Específicos

- Realizar leitura de instâncias ordenadas e não ordenadas;
- Realizar implementação dos algoritmos de busca com a linguagem Python;
- Obter tempo de processamento de cada algoritmo;
- Obter uso memória de cada algoritmo;
- Plotar gráficos de resultados de tempo e consumo de memória.

3. ESTADO DA ARTE

Complexidade de tempo é comumente estimada pela contagem do número de operações elementares realizadas pelo algoritmo, onde a operação elementar toma a quantia fixa de tempo para realizar. A quantidade de tempo tomada e o número de operações elementares realizadas pelo algoritmo diferem no máximo de um fator constante.

Um algoritmo é dito ser em tempo constante (também escrito como executado em tempo O(1)) se o valor de T(n) é limitado por uma valor que não dependa do tamanho da entrada. Por exemplo, acessando um único elemento de um array usa tempo constante, visto que uma única operação foi executada para localizá-la. Encontrar o menor valor de um vetor não ordenado, contudo, não é uma operação de tempo constante, visto que é necessário olhar sobre todos os elementos do vetor para determinar qual é o menor valor de todos. Isto é, consequentemente, uma operação em tempo linear, em relação ao tamanho da entrada, utilizando tempo O(n). Se o número de elementos é conhecido posteriormente e não se altera, contudo, tal algoritmo pode ser dito que execute em tempo constante.

Um algoritmo é dito ter tempo logaritmo se T(n) = O(log n). Devido ao uso do sistema de números binários pelos computadores, o logaritmo é frequentemente de base 2 (isto é, log2 n, muitas vezes escrito lg n). Contudo, pela troca da equação base para logaritmos, loga n and logb n diferem apenas por uma constante multiplicadora,

porque na notação big-O é descartada; logo O(log n) é a notação padrão para algoritmos de tempo logaritmo independente da base do logaritmo.

Algoritmos que executam em tempo logarítmico são comumente encontrados em operações em árvores binárias ou quando se usa busca binária.

Um algoritmo é dito rodar em tempo polilogaritmo se T(n) = O((log n)k), para alguma constante k. Por exemplo, a ordenação de uma cadeia de matrizes pode ser resolvida em tempo poli-logarítmico em uma máquina paralela de acesso aleatório.

Um algoritmo é dito que usa tempo linear, ou tempo O(n), se sua complexidade de tempo é O(n). Informalmente, isto significa que para entradas grandes o suficiente o tempo de execução delas aumenta linearmente com o tamanho da entrada. Por exemplo, um procedimento que adiciona todos os elementos em uma lista requere tempo proporcional ao tamanho da lista. Esta descrição é levemente imprecisa, visto que o tempo de execução pode desviar significantemente de uma proporção precisa, especialmente para valores pequenos de n.

Um algoritmo é dito subquadrático se T(n) = o(n2). Por exemplo, a maioria dos algoritmos baseados em comparações ingênuas são quadráticos (por exemplo insertion sort), mais algoritmos mais avançados que são encontrados são subquadráticos (por exemplo shell sort). Nenhum tipo de ordenação com intúitos gerais roda em tempo linear, mas a mudança de quadrático para subquadrático é de grande importância prática.

Um algoritmo é dito ser de tempo polinomial se o tempo de execução dele é limitado superiormente por uma expressão polinomial do tamanho da entrada para o algoritmo, T(n) = O(nk) para algumas constantes k. Problemas para algoritmos de tempo polinomial existem pertencendo a classe de complexidade P, que é central no campo da teoria da complexidade computacional. A tese de Bobham afirma que tempo polinomial é um sinônimo para "dócil", "viável", "eficiente", ou "veloz".

4. METODOLOGIA

Neste trabalho, foram analisados seis algoritmos de busca, sendo eles:

- Busca Linear V1
- Busca Linear V2
- Busca Binária
- Busca Quadrática
- Busca Ternária
- Busca Cúbica

Eles se dividem em diferentes níveis de complexidade, sendo assim, temos algoritmos de O(n), O(log n), $O(n^2)$ e $O(n^3)$. Os testes realizados foram utilizados instâncias com valores numéricos ordenados e não ordenados, em relação a máquina na qual os dados foram processados, este trabalho se utilizou de duas máquinas:

- Notebook Lenovo, AMD Ryzem 5, 8Gb de RAM
- Google Colab, I7, 16Gb de RAM e GPU Tesla T4 de 16Gb

As seguintes instâncias foram utilizadas para realizar as análises, sendo elas ordenadas e não ordenadas:

- 100
- 200
- 1000
- 2000
- 5000
- 10000
- 50000
- 100000
- 500000
- 1000000
- 5000000
- 10000000
- 100000000

Devido a questões computacionais e de tempo, as 4 últimas instâncias não foram utilizadas para análise dos algoritmos de buscas. Sendo eles:

- 1000000
- 5000000
- 10000000
- 100000000

Ao final da execução de todos os algoritmos, foram analisadas o tempo e o consumo de memória, plotado gráficos para realização de comparação entre eles e todos os scrips desenvolvidos foram postados no GitHub.

5. RESULTADOS

Todos os scripts desenvolvidos podem ser visualizados no GitHub no seguinte endereço: https://github.com/IagoMagalhaes23/BBP1008---ESTUDOS-ESPECIAIS/tree/main/Trabalho%20esquenta

A seguir será analisado cada um dos algoritmos de forma individual com o tempo de processamento e memória utilizada, além disso, as instâncias de 1, 5, 10 e 100 milhões não foram testadas devido a problemas com memória e tempo de execução.

ALGORITMO 1 – Busca Linear V1

A busca linear é o algoritmo de busca mais simples para vetores e qualquer outro tipo de estrutura de dados linear. A ideia básica do algoritmo é comparar o elemento procurado com cada elemento do vetor até encontrá-lo partindo, em geral, da primeira posição do vetor [3].

Tabela 01. Busca linear v1 para instâncias ordenadas.

Instância	Tempo	Memória
100	0.0007576942443847656s	215937024
200	0.0005022287368774414s	201719808
1000	0.0005040884017944336s	215937024
2000	0.000639653205871582s	201719808
5000	0.0009804248809814453s	201719808

10000	0.0016920328140258788s	216203264
50000	0.006335234642028809s	201719808
100000	0.012808132171630859s	201719808
500000	0.035229849815368655s	218005504
1000000	X	X
5000000	X	X
10000000	X	X
100000000	X	X

Fonte: Autor.

Como apresentado na tabela 01 o algoritmo de busca linear v1 atingiu resultados muito promissores, as médias de tempo não atingiram nem meio segundo, mostrando a eficiência do algoritmo. Cada instância foi testada 10 vezes e obtida a média de tempo e memória. O número buscado é 1000. E em algumas instâncias ele não existe, em outras ele está no início, meio ou fim da instância.

Tabela 02. Busca linear v1 para instâncias não ordenadas.

Instância	**Tempo**	**Memória**
100	0.5956899166107178s	574349312
200	0.6102594852447509s	621883392
1000	0.7205517768859864s	597209088
2000	0.5974762678146363s	624881664
5000	0.7288527965545655s	637485056
10000	0.7271636962890625s	602103808
50000	0.7261827230453491s	651628544
100000	0.7182903528213501s	608268288
500000	0.6100410699844361s	650600448
1000000	X	X
5000000	X	X
10000000	X	X
100000000	X	X

Fonte: Autor.

Como apresentado na tabela 02 o algoritmo de busca linear v1 também atingiu resultados muito promissores para valores desordenados, as médias de tempo

atingiram cerca de pouco mais de meio segundo, mostrando a eficiência do algoritmo. Cada instância foi testada 10 vezes e obtida a média de tempo e memória. O número buscado é 1000. E em algumas instâncias ele não existe, em outras ele está no início, meio ou fim da instância.

ALGORITMO 2 – Busca Linear V2

A busca linear v2 é basicamente uma versão melhorada através da engenharia de software, na qual otimiza o mesmo algoritmo da busca linear v1 e o otimiza via código.

Tabela 03. Busca linear v2 para instâncias ordenadas.

Instância	**Tempo**	**Memória**
100	0.0006129264831542969s	218005504
200	0.00010700225830078125s	206901248
1000	0.0003278017044067383s	218005504
2000	0.00010645389556884766s	206901248
5000	0.0003641366958618164s	206901248
10000	0.0006176471710205078s	218271744
50000	0.00011277198791503906s	206901248
100000	0.00012264251708984374s	206901248
500000	0.03119683265686035s	223236096
1000000	X	X
5000000	X	X
10000000	X	X
100000000	X	X

Fonte: Autor.

Como apresentado na tabela 03 o algoritmo de busca linear v2 melhora consideravelmente o v1, através de uma melhor organização de código. O número buscado é 1000. E em algumas instâncias ele não existe, em outras ele está no início, meio ou fim da instância. O uso de memória se mostra muito baixo também.

Tabela 04. Busca linear v2 para instâncias não ordenadas.

Instância	Tempo	Memória
100	0.6174428462982178s	701321216
200	0.6696840524673462s	712585216
1000	0.6259521007537842s	675803136
2000	0.6122011423110962s	710987776
5000	0.6060641050338745s	716689408
10000	0.7260002136230469s	707444736
50000	0.7227292776107788s	749314048
100000	0.7195951461791992s	702537728
500000	0.7215059995651245s	743751680
1000000	X	X
5000000	X	X
10000000	X	X
100000000	X	X

Fonte: Autor.

Como apresentado na tabela 04 o algoritmo de busca linear v2 também se comporta de forma bastante eficiente com valores de entrada não ordenados, porém, já com elevação de tempo e uso de memória. E em algumas instâncias ele não existe, em outras ele está no início, meio ou fim da instância. O uso de memória se mostra muito baixo também.

ALGORITMO 3 – Busca Binária

A busca binária é um eficiente algoritmo para encontrar um item em uma lista ordenada de itens. Ela funciona dividindo repetidamente pela metade a porção da lista que deve conter o item, até reduzir as localizações possíveis a apenas uma [4].

Tabela 05. Busca binária para instâncias ordenadas.

Instância	Tempo	Memória
100	0.00025112628936767577s	55083008
200	0.0007008075714111328s	61120512
1000	0.0006930828094482422s	55300096
2000	0.0s	61120512

5000	0.0005475044250488281s	61124608
10000	0.0003372907638549805s	56930304
50000	0.0004980564117431641s	62111744
100000	0.0004746198654174805s	63217664
500000	010676383972167969s	83861504
1000000	X	X
5000000	X	X
10000000	X	X
100000000	X	X

Fonte: Autor.

Como apresentado na tabela 05 o algoritmo de busca binária é muito eficiente e utiliza pouca memória e obtêm respostas muito rápido. E em algumas instâncias ele não existe, em outras ele está no início, meio ou fim da instância.

ALGORITMO 4 – Busca Quadrática

Tabela 06. Busca quadrática para instâncias ordenadas.

Instância	**Tempo**	**Memória**
100	-	-
200	-	-
1000	-	-
2000	-	-
5000	-	-
10000	-	-
50000	-	-
100000	-	-
500000	-	-
1000000	X	X
5000000	X	X
10000000	X	X
100000000	X	X

Fonte: Autor.

A busca quadrática se mostrou muito ineficiente em relação a tempo e uso de memória, demorando um tempo muito grande para pequenas instâncias e muito mais tempo e memória para instâncias maiores. Após 12 horas de teste, percebi que o algoritmo iria necessitar de bastante tempo para processar instâncias ordenadas com poucos valores, sendo inviável concluir todo o processo de teste.

Tabela 07. Busca quadrática para instâncias não ordenadas.

Instância	**Tempo**	**Memória**
100	-	-
200	-	-
1000	-	-
2000	-	-
5000	-	-
10000	-	-
50000	-	-
100000	-	-
500000	-	-
1000000	X	X
5000000	X	X
10000000	X	X
100000000	X	X

Fonte: Autor.

Infelizmente para o algoritmo de busca quadrática para instâncias não ordenadas, não foi possível realizar os testes devido ao prazo de entrega do trabalho e o tempo que o mesmo exigiria para execução seria bastante grande.

ALGORITMO 5 – Busca Ternária

Na Pesquisa Ternária, dividimos nosso array em três partes (*tomando dois meados*) e descarte dois terços do nosso espaço de busca em cada iteração. À primeira vista, parece que a pesquisa ternária pode ser mais rápida que a pesquisa binária, pois sua complexidade de tempo em uma entrada contendo n os itens devem ser O(log3n), que é menor que a complexidade de tempo da busca binária O(log2n) [5].

Tabela 08. Busca ternária para instâncias ordenadas.

Instância	Tempo	Memória
100	0.0020212650299072264s	55836672
200	0.0015661001205444336s	61251584
1000	0.0019561052322387695s	56066048
2000	0.001390552520751953s	61259776
5000	0.0017381668090820312s	61267968
10000	0.002491450309753418s	57753600
50000	0.002777242660522461s	62734336
100000	0.002236294746398926s	64503808
500000	0.0014180898666381835s	84283392
1000000	X	X
5000000	X	X
10000000	X	X
100000000	X	X

Fonte: Autor.

Como apresentado na tabela 08 o algoritmo de busca ternária é muito eficiente e utiliza pouca memória e obtem respostas muito rápido. E em algumas instâncias ele não existe, em outras ele está no início, meio ou fim da instância. Ele é uma versão melhorada do algoritmo de busca binária.

ALGORITMO 6 – Busca Cúbica

Tabela 09. Busca cúbica para instâncias ordenadas.

Instância	Tempo	Memória
100	0.021051526069641113s	1559371776
200	12.380260109901428s	1559638016
1000	12.611485481262207s	1559638016
2000	102.88274283409119s	1559638016
5000	X	X
10000	X	X
50000	X	X
100000	X	X

500000	X	X
1000000	X	X
5000000	X	X
10000000	X	X
100000000	X	X

Fonte: Autor.

Considerei esse algoritmo um crime, extremamente demorado para instâncias ordenadas e tanto minha máquina quanto o Google Colab não conseguiram executar instâncias maiores que 5000 em tempo considerável.

Tabela 10. Busca cúbica para instâncias não ordenadas.

Instância	**Tempo**	**Memória**
100	-	-
200	-	-
1000	-	-
2000	-	-
5000	X	X
10000	X	X
50000	X	X
100000	X	X
500000	X	X
1000000	X	X
5000000	X	X
10000000	X	X
100000000	X	X

Fonte: Autor.

Com dados ordenados o tempo já era demorado, com não ordenados o tempo de processamento cresceu exponencialmente e tanto minha máquina quanto o Google Colab não conseguiram executar instâncias maiores que 5000 em tempo considerável. Com valores desordenados, ao chegar a instância com 100 valores o tempo foi de 6 horas e não houve conclusão da análise e acabei abortando este teste.

6. CONCLUSÃO

O trabalho foi realizado com êxito, sendo possível implementar todos os algoritmos utilizando a linguagem Python, realizando a leitura de todas as instâncias no formato ‘txt’. As métricas solicitadas, tempo e memória, foram obtidas e analisadas neste trabalho.

Com este trabalho foi possível ver que dado um problema, existem diferentes maneiras de soluciona-los e que utilizando uma melhor organização de código e recursos de hardware é possível melhorar os desempenhos.

7. REFERÊNCIAS

[1] ALGORITMOS de Busca - Algoritmos e Programação de Computadores. [S. l.], 1 jan. 2023. Disponível em: https://ic.unicamp.br/~mc102/aulas/aula11.pdf. Acesso em: 5 set. 2023.

[2] COMPLEXIDADE de tempo. [S. l.], 8 jan. 2023. Disponível em: https://pt.wikipedia.org/wiki/Complexidade_de_tempo#Tabela_de_complexidade_de_tempo_comum. Acesso em: 5 set. 2023.

[3] Busca Linear. [S. l.], 14 set. 2017. Disponível em: https://www.blogcyberini.com/2017/09/busca-linear.html. Acesso em: 4 set. 2023.

[4] Busca Binária. Disponível em: https://pt.khanacademy.org/computing/computer-science/algorithms/binary-search/a/binary-search#:~:text=A%20busca%20bin%C3%A1ria%20%C3%A9%20um,localiza%C3%A7%C3%B5es%20poss%C3%ADveis%20a%20apenas%20uma.. Acesso em: 4 set. 2023.

[5] Pesquisa ternária vs Pesquisa binária. Disponível em: https://www.techiedelight.com/pt/ternary-search-vs-binary-search/. Acesso em: 4 set. 2023.