****

**UNIVERSIDADE FEDERAL DO CEARÁ - CAMPUS SOBRAL**

**PROGRAMA DE PÓS-GRADUAÇÃO EM ENGENHARIA ELÉTRICA E COMPUTAÇÃO**

**DISCIPLINA: ESTUDOS ESPECIAIS**

**ALUNO: IAGO MAGALHÃES DE MESQUITA**

**ALGORITMOS PARA ENCONTRAR O MAIOR VALOR**

Sobral, 2023

**ALGORITMOS PARA ENCONTRAR O MAIOR VALOR**

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........................................................................................................... 3**

**2 OBJETIVOS................................................................................................................ 4**

**3 METODOLOGIA....................................................................................................... 4**

**4 RESULTADOS........................................................................................................... 5**

**5 CONCLUSÃO............................................................................................................ 11**

**6 REFERÊNCIAS......................................................................................................... 12**

1. **INTRODUÇÃO**

Algoritmos para encontrar o maior valor funcionam de forma simples, dado uma entrada se busca pelo maior presente nessa instância. Porém, existem diferentes formas de se encontrar esse valor, sendo assim, pode-se construir algoritmos com melhores desempenhos, reduzindo o tempo de processamento e uso de memória.

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 [1].

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) [1].

Neste trabalho iremos analisar algoritmos para encontrar o maior valor em uma instância de entrada. Observaremos o tempo necessário para realizar essa busca e o consumo de memória em dois algoritmos.

1. **OBJETIVOS**
   1. Objetivo Geral

* Realizar experimentos com algoritmos para encontrar o maior valor, visando analisar tempo de execução e uso de memória para diferentes instâncias.
  1. Objetivos Específicos
* Realizar leitura de instância não ordenadas;
* Realizar implementação dos algoritmos para encontrar o maior valor 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.

1. **METODOLOGIA**

Neste trabalho, foram analisados dois algoritmos para encontrar o maior valor, sendo eles:

* maxVal1;
* maxVal2.

Os testes realizados foram utilizados instâncias com valores numéricos não ordenados, em relação a máquina na qual os dados foram processados, este trabalho se utilizou da seguinte máquina:

* Notebook Lenovo, AMD Ryzem 5, 8Gb de RAM.

As seguintes instâncias foram utilizadas para realizar as análises, sendo elas 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.

1. **RESULTADOS**

Todos os scripts desenvolvidos podem ser visualizados no GitHub no seguinte endereço: <https://github.com/IagoMagalhaes23/BBP1008---ESTUDOS-ESPECIAIS/tree/main/Trabalho_esquenta_02>

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 – maxVal1

O algoritmo maxVal1 funciona de forma bem simples, dado uma instância de entrada, uma variável assume que a primeira posição do vetor é o maior valor. Logo depois, esse valor é comparado com os demais valores do vetor, percorrendo todas as posições e realizando as comparações. Caso algum outro valor seja maior que o valor presente na variável de máximo valor, a mesma será atualizada e comparação seguirá agora com o valor atualizado. E caso encontre novos valores maiores, tal valor será atualizado até que todos os itens do vetor sejam analisados.

Na tabela 01, são apresentados os resultados de tempo e consumo de memória para cada instância em que o algoritmo foi executado.

Tabela 01. Algoritmo para encontrar o maior valor – maxVal1.

|  |  |  |
| --- | --- | --- |
| **Instância** | **Tempo** | **Memória** |
| 100 | 0.0s | 54640640 |
| 200 | 0.0s | 58413056 |
| 1000 | 0.0002499818801879883s | 54775808 |
| 2000 | 0.00015957355499267577s | 57376768 |
| 5000 | 0.000507354736328125s | 57389056 |
| 10000 | 0.004066729545593261s | 56139776 |
| 50000 | 0.003972244262695312s | 59904000 |
| 100000 | 0.006930685043334961s | 63889408 |
| 500000 | 0.043287229537963864 | 79704064 |
| 1000000 | X | X |
| 5000000 | X | X |
| 10000000 | X | X |
| 100000000 | X | X |

Fonte: Autor.

Na figura 01, é apresentado um gráfico de linha mostrando a evolução do tempo de processamento da menor instância para a maior.

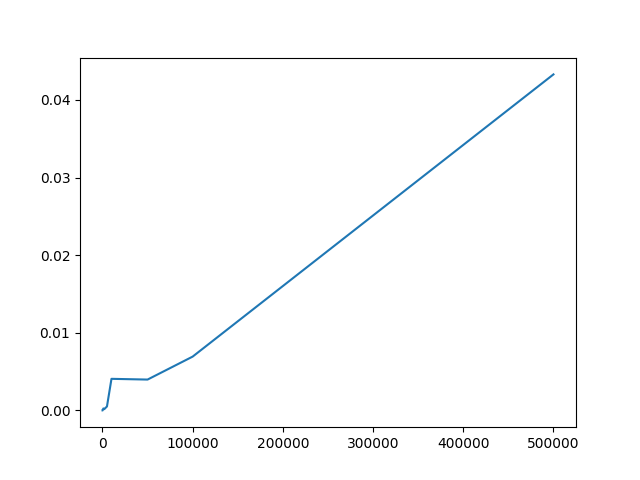


Figura 01. Evolução de tempo de processamento para o algoritmo maxVal1.

Na figura 02, é mostrado um gráfico mostrando o consumo de memória para cada instância analisada.

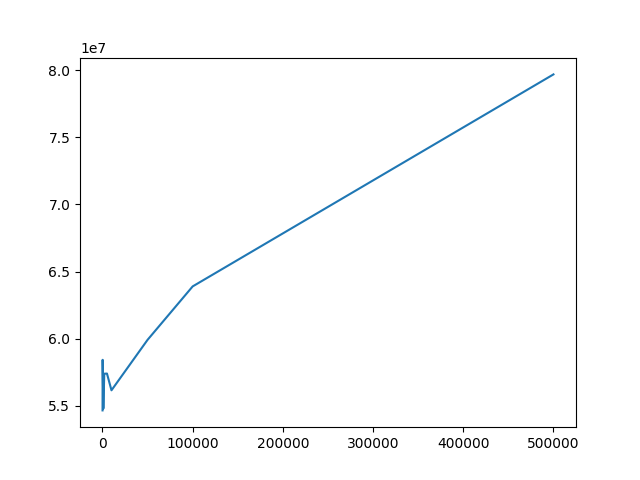


Figura 02. Consumo de memória para o algoritmo maxVal1.

ALGORITMO 2 – maxVal2

O algoritmo maxVal2 funciona utilizando o conceito de recursividade. Recebendo como parâmetros um vetor, o valor inicial e o final. Através de uma estrutura de condição é inicialmente verificado se o valor inicial menos o final é menor ou igual a 1, caso positivo, a função retornará o maior valor entre esses dois através da função max. Caso negativo, uma variável m é criada recebendo o valor da divisão por 2 da soma do valor inicial e do final, logo após uma variável v1 recebe o retorno da própria função maxVal2, sendo que agora os parâmetros são o vetor, o valor inicial e m. O mesmo processo na linha seguinte, agora com v2 recebendo o retorno da função com os parâmetros sendo o vetor, m mais 1 e o valor final do vetor. Após tais valores de v1 e v2 serem encontrados a função maxVal2 retorna o máximo valor entre v1 e v2 através da função max.

Na tabela 02, são apresentados os resultados de tempo e consumo de memória para cada instância em que o algoritmo foi executado.

Tabela 02. Algoritmo para encontrar o maior valor – maxVal2.

|  |  |  |
| --- | --- | --- |
| **Instância** | **Tempo** | **Memória** |
| 100 | 0.0s | 54820864 |
| 200 | 0.0s | 58245120 |
| 1000 | 0.0s | 54919168 |
| 2000 | 0.0005017757415771484s | 58245120 |
| 5000 | 0.0010704994201660156s | 57196544 |
| 10000 | 0.0s | 56242176 |
| 50000 | 0.0009789228439331054s | 60260352 |
| 100000 | 0.0011159181594848633s | 63250432 |
| 500000 | 0.0012822628021240234s | 79765504 |
| 1000000 | X | X |
| 5000000 | X | X |
| 10000000 | X | X |
| 100000000 | X | X |

Fonte: Autor.

Na figura 03, é apresentado um gráfico de linha mostrando a evolução do tempo de processamento da menor instância para a maior.

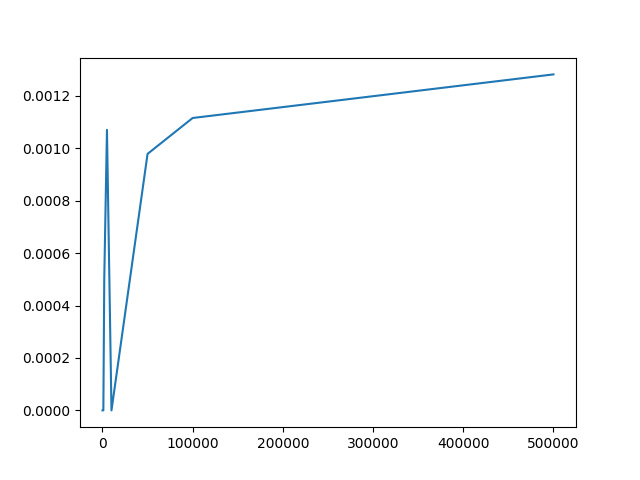


Figura 03. Evolução de tempo de processamento para o algoritmo maxVal2.

Na figura 04, é mostrado um gráfico mostrando o consumo de memória para cada instância analisada.

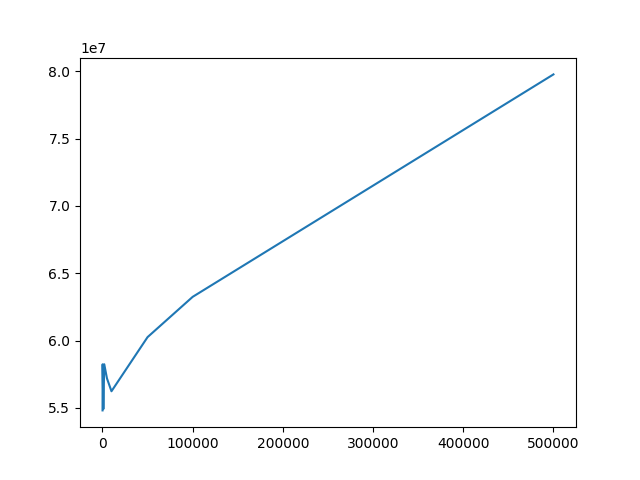


Figura 04. Consumo de memória para o algoritmo maxVal2.

1. **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 que tanto memória e tempo de processamento crescem exponencialmente em ambos os algoritmos, porém, no algoritmo maxVal2 o consumo de tempo e memória é bem maior que o maxVal1.

1. **REFERÊNCIAS**

[1] 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.