COMPLEXIDADE DE ALGORITMOS

Segmento de Soma Máxima

Segunda Prova - Parte 2 - Relatório dos Algoritmos

Discentes: Amanda Fiel, Beatriz Cavalcante e Manoel Malon Moura

Sumário

Segmento de Soma Máxima

O problema de segmento de soma maxima é dado por um certo vetor de número inteiro, esse vetor irá encontrar um segmento cuja a soma dos seus elementos seja >= a soma dos elementos de qualquer outro segmento v.

Apresentado essa definição, teremos no decorrer desse trabalho a representação de 4 exemplos de algoritmos de Segmento de Soma Máxima através da linguagem de programação pynthon.

Os quatro algoritmos e demais funcionalidades foram executados em uma máquina com o processador Intel(R) Core(TM) i5 8265U CPU @ 1.60 GHz a 3.90 GHz, com 4 núcleos e 8 threads, memória RAM de 8,00 GB e sistema operacional Windows 10 Home Single Language.

Algoritmos

Algotimos

Main.py

A função open( ) abre o arquivo denominado "resultado.txt", se não tiver no diretório, cria o arquivo com este mesmo nome. Não é colocado o nome das colunas afim de melhorar o tratamento dos dados, porém imprime na seguinte ordem em relação ao tempo de execução: tamanho ; altural ; alturaII ; alturaIII ; alturaIV.

```
1   import time
2   import random
3   import numpy as np
4   import datetime
5   from Algoritmos import Algoritmos
6
7   while(True):
8       arquivo = open('resultado.txt', 'a')
9
10      tamanho = int(input("Digite o tamanho da entrada: "))
11      limiteInferior = -100
12      limiteSuperior = 100
13      vetor = np.random.randint(limiteInferior, limiteSuperior, tamanho)
14      print("Vetor Criado:")
15      print(vetor)
16      arquivo.write(str(tamanho) + "\t\t")
```

Figura 01 - Bibliotecas e início do programa principal

Nas primeiras 4 linhas são implementadas as bibliotecas essenciais para a melhor execução do projeto, sendo seguida pela implementação da classe que possui os métodos utilizados nesse código. A partir da linha 7 o programa é iniciado e recebe um valor inteiro do usuário, determina os limites inferiores e superiores para os números aleatórios, cria um vetor com esses números e mostra esse vetor para o usuário. Logo após escreve as informações em um arquivo txt.

Algotimos

Main.py

```
48      print("\n-------------Execução do Quatro Algoritmo O(n)-------------")
49      start = time.perf_counter()
50      somaMaxima = Algoritmos.alturaIV(vetor)
51      end = time.perf_counter()
52      duracao = end - start
53      duracao = datetime.timedelta(seconds = duracao)
54      print("Tempo de duração: " + str(duracao))
55      print("O valor da altura é: " + str(somaMaxima))
56      arquivo.write(str(duracao) + "\n")
57
58      arquivo.close
59      print("-------------------------------------------------")
60      print("1 - Entrar com outro nº de entradas")
61      print("2 - Sair do programa")
62
63      opcao = int(input("> "))
64      if opcao == 1:
65          continue
66      elif (opcao == 2):
67          break
68      else:
69          print("Opcao invalida!")
```

Figura 02 - Quarto algoritmo e finalização do programa

Entre as linhas 17 à 56 são executados os 4 algoritmos. Todos seguem a mesma lógica do algoritmo 4 apresentado na figura 02, sendo essa: início do timer, chamada do método do seu respectivo algoritmo, encerramento do timer, imprime o tempo gasto em segundos, formata esse mesmo tempo, mostra o tempo e o segmento de soma máxima, escreve o tempo de execução no arquivo txt e faz tabulação.

A partir da linha 57 o arquivo é fechado e o sistema dá a opção de entrar com outro número ou sair do programa.

Algoritmos

Altural, II, III, IV.py

```python
import matplotlib.pyplot as plt
import time
import random
import numpy as np
import datetime
from Algoritmos import Algoritmos

arquivo = open('resultadoAlturaI.txt', 'a')

for i in range(0, 100):
    tamanho = i+1
    limiteInferior = -100
    limiteSuperior = 100
    vetor = np.random.randint(limiteInferior, limiteSuperior, tamanho)
    print("tamanho: " + str(tamanho))
    print("Vetor Criado:")
    print(vetor)
    start = time.perf_counter()
    somaMaxima = Algoritmos.alturaI(vetor)
    end = time.perf_counter()
    duracao = end - start
    print("Tempo de duração: " + str(duracao))
    print("O valor da altura é: " + str(somaMaxima)+"\n")
    arquivo.write(str(tamanho) + "\t\t")
    arquivo.write(str(duracao) + "\n")
    plt.scatter(tamanho, duracao)

plt.xlabel('Entradas')
plt.ylabel('Tempo(s)')
plt.show()
arquivo.close
```

Figura 03 - Algoritmo da altura I

Assim como no programa anterior, as primeiras linhas são para importar as bibliotecas e classes utilizados durante este código. O sistema é iniciado, define os limites inferiores e superiores dos números aleatórios a serem gerados, cria e imprime esse vetor, inicia o timer, chama o método de seu respectivo algoritmo, encerra o timer e mostra os resultados de seu tempo e segmento de soma máxima. Ao final, o sistema prepara um gráfico para mostrar o seu resultado.

Esse mesmo algoritmo se repete para as outras três alturas, mudando somente a palavra destacada na Figura 03 (mudança de método), ao colocar o número da altura no final.

Algoritmos

ALGORITMO 1

```python
class Algoritmos:

    @staticmethod
    def alturaI(vetor):
        x = vetor[0]
        for i in range(0, len(vetor)):
            for k in range(i, len(vetor)):
                s = 0
                for j in range(i, k+1):
                    s = s + vetor[j]
                if s > x:
                    x = s
        return x
```

Esta classe contém os métodos para calcular cada um dos algoritmos de segmento de soma máxima. Todos são estáticos para que não seja preciso instanciar um objeto que chame um método. A Figura 04 mostra o método do primeiro algoritmo.

Figura 04 - Primeiro Algoritmo
teta(n^3) = n*n*n

Algoritmos

ALGORITMO 2

```
15      @staticmethod
16      def alturaII(vetor):
17          x = vetor[0]
18          for q in range(1, len(vetor)): #
19              s = 0
20              for j in range(q, 1, -1):
21                  s = s + vetor[j]
22                  if s > x:
23                      x = s
24          return x
```

Figura 05 - Segundo Algoritmo
teta(n^2) = n*n

Algoritmos

ALGORITMO 3

```
26      @classmethod
27      def max(cls, num1, num2, num3):
28          if num3 == None:
29              num3 = 0
30          if num2 == None:
31              num3 = 0
32          if num1 == None:
33              num3 = 0
34
35          if num1 > num2 and num1 > num3:
36              return cls.num1
37          if num2 > num1 and num2 > num3:
38              return cls.num2
39          if num3 > num1 and num3 > num2:
40              return cls.num3
```

Figura 06 - Método responsável por determinar o maior número entre 3. Esse é necessário na execução de alturaIII.

```
42      @staticmethod
43      def alturaIII(vetor, p, r):
44          if p == r:
45              return vetor[p]
46          else:
47              q = (p+r)//2
48              x1 = Algoritmos.alturaIII(vetor, p, q)
49              x2 = Algoritmos.alturaIII(vetor, q+1, r)
50              y1 = s = vetor[q]
51              for i in range(q-1, p, -1):
52                  s = vetor[i] + s
53                  if s > y1:
54                      y1 = s
55              y2 = s = vetor[q+1]
56              for j in range(q+2, r):
57                  s = s + vetor[j]
58                  if s > y2:
59                      y2 = s
60              x = max(x1, y1 + y2, x2)
61              return x
```

Figura 07 - Terceiro Algoritmo teta(nlogn)

Algoritmos

ALGORITMO 4

```
63      @staticmethod
64      def alturaIV(vetor):
65          n = len(vetor)
66          semi = vetor.copy()
67          semi[0] = vetor[0]
68          for q in range(1, n):
69              if semi[q-1] >= 0:
70                  semi[q] = semi[q-1]+vetor[q]
71              else:
72                  semi[q] = vetor[q]
73          x = semi[0]
74          for q in range(2, n):
75              if semi[q] > x:
76                  x = semi[q]
77          return x
```

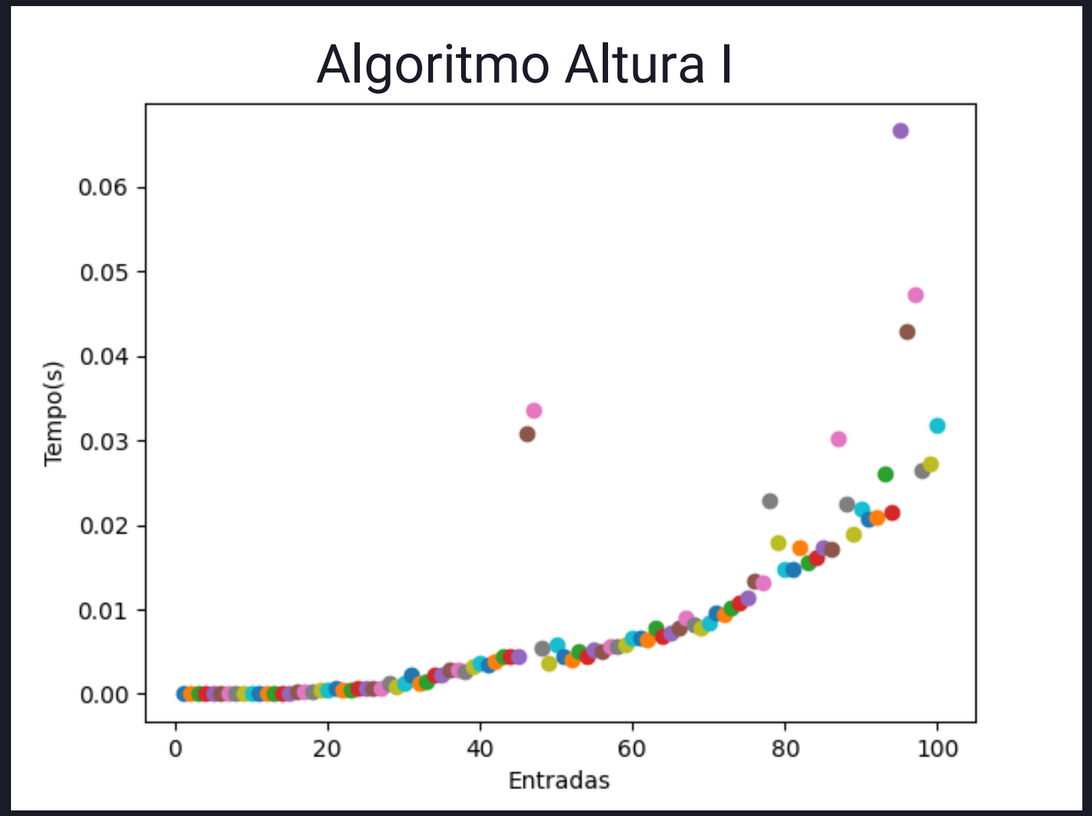
Figura 08 - Quarto Algoritmo Teta(n)

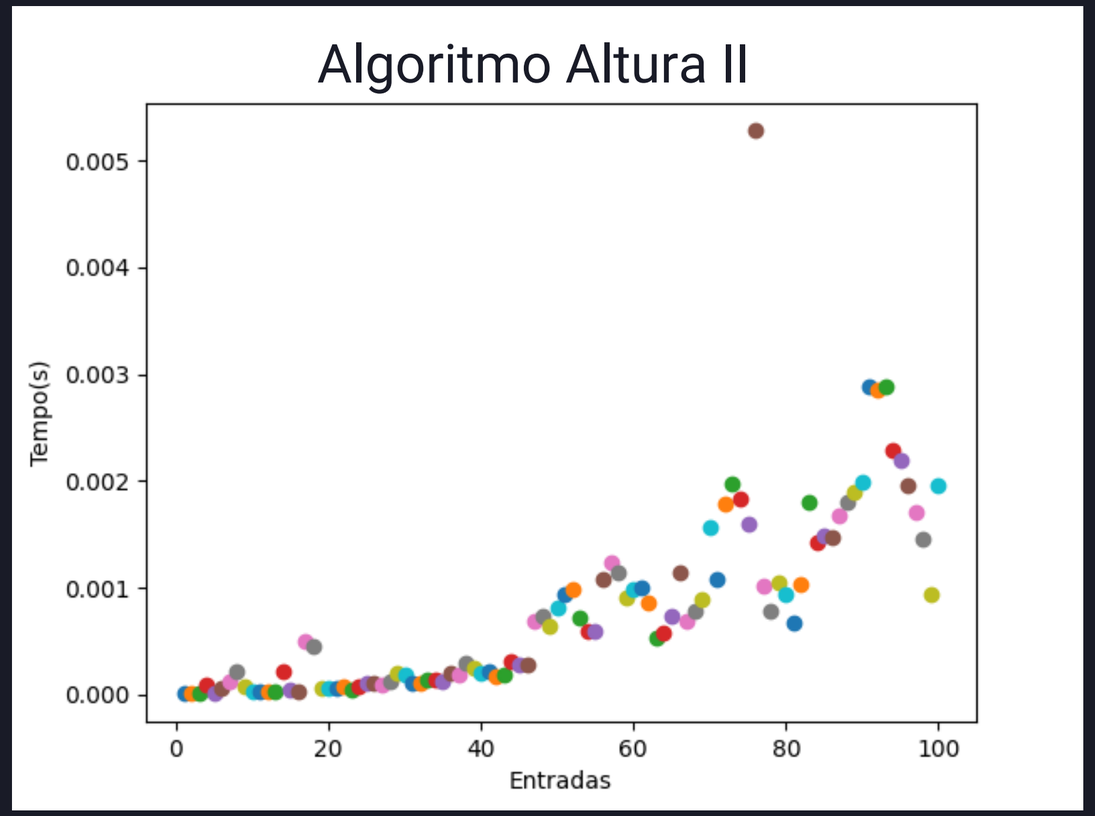
Tabela de Comparação dos Algoritmos

Tabela 1

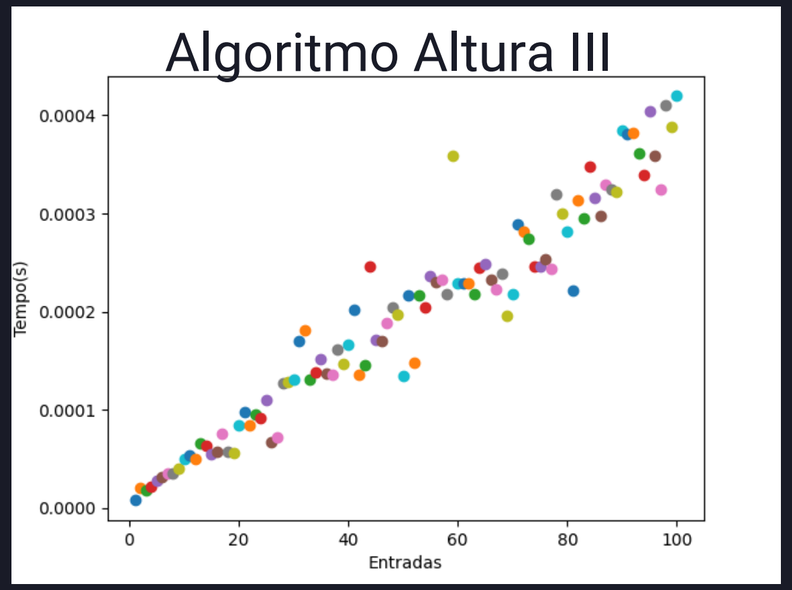
N	O(N^3)	O(N^2)	O(n log n)	O(n)
1	0:00:00.000461	0:00:00.000007	0:00:00.000006	0:00:00.000013
10	0:00:00.000212	0:00:00.000057	0:00:00.000374	0:00:00.000067
50	0:00:00.006734	0:00:00.000458	0:00:00.000170	0:00:00.000285
100	0:00:00.035041	0:00:00.001050	0:00:00.000340	0:00:00.000132
500	0:00:02.577834	0:00:00.019725	0:00:00.001249	0:00:00.000374
1000	0:00:20.733868	0:00:00.076549	0:00:00.003507	0:00:00.007392
2000	0:03:11.075214	0:00:00.291757	0:00:00.004841	0:00:00.001555

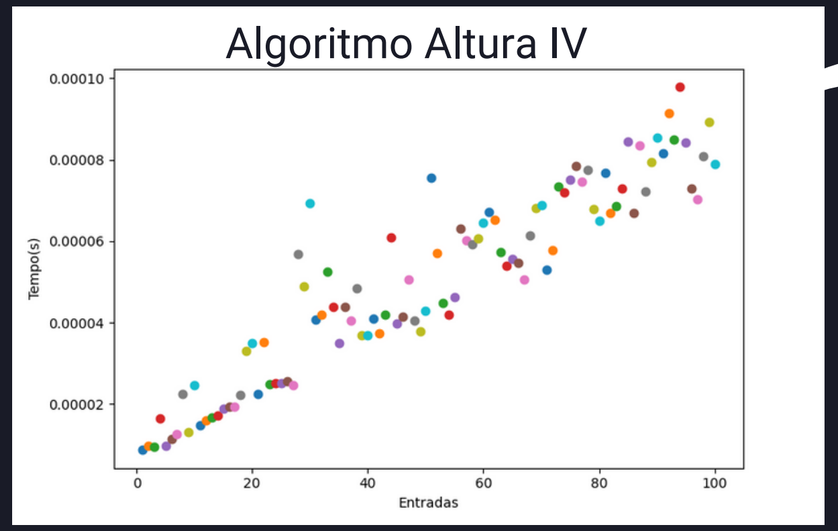
Gráficos Plotados dos Algoritmos

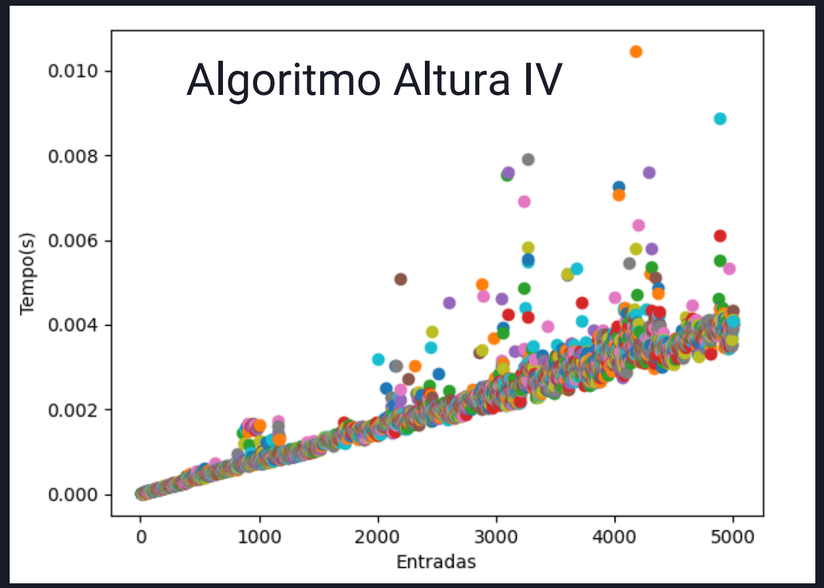




Gráficos Plotados dos Algoritmos







Implementação

Conclusão

Na contemporaneidade, onde a sociedade está cada vez mais cercada por tecnologias, percebesse a necessidade de buscar se inteirar das diversidades que a ciência do ramo de complexidade de algoritmos oferece.

Utilizando-se de diretrizes estabelecidas e elaborando métodos robustos, e consistentes da linguagem de programação pynthon, constatou-se a importância da elaboração do presente trabalho e tornou-se evidente que a implementação dos quatro algoritmos de segmento soma máxima auxiliou de forma ampla na compreensão dos códigos na pratica.

Obrigado!