

Sobre Métodos de Ordenação

Mário Leite

...

Vocês já notaram que os números da Mega Sena, ou de qualquer outra loteria, sempre são apresentados em ordem crescente!? Quando se trata de mostrar uma sequência de números, ou mesmo os nomes de pessoas, os elementos da lista sempre são exibidos ordenadamente. E quando se deseja ordenar uma lista de elementos, a primeira ideia que vem à cabeça de um programador (principalmente os iniciantes) é o clássico **“Método da Bolha”**, ou como gostam os amantes do dialeto inglês: *“Bubble Sort”*. Esse é o método usado por nove entre dez programadores, mesmo negando perante os colegas para não ser taxado de “pobre”! Assim explica a Wikipédia sobre este método: *“a ideia é percorrer o **vector** diversas vezes, e a cada passagem fazer flutuar para o topo o maior elemento da sequência. Essa movimentação lembra a forma como as bolhas em um **tanque** de água procuram seu próprio nível, e disso vem o nome do algoritmo”*.

Eu prefiro defini-lo de outro modo, mais perto da lógica de programação; desta maneira: ***“processo para classificar um vetor de n elementos ($n > 1$) que envolve dois loops aninhados: um mais externo e outro mais interno; enquanto no laço mais externo a variável i varia de 1 até $(n-1)$, no laço mais interno a variável j vai de $(i+1)$ até n . Dentro do laço mais interno as comparações dos elementos de índice i e j são feitas; e se houver necessidade haverá trocas de posições entre esses dois elementos. Assim, ao final desses dois loops o vetor estará com seus elementos ordenados...”***

Embora o “Método da Bolha” seja muito popular e muito fácil de entender, sua aplicação em vetores com muitos elementos (na ordem de milhares) é muito ineficiente; o processo fica muito lento, e o tempo de processamento aumenta assustadoramente. Existem outros métodos, embora mais complexos, porém, muito mais eficientes; e entre eles podemos destacar dois: **“Método da Inserção Direta”** e **“Método Shell”**.

O “Método da Inserção Direta” é baseado na divisão de um vetor de n ($n > 1$) elementos em dois segmentos, como mostra o esquema do **quadro 4**:

- O primeiro segmento (mais à esquerda) contém os elementos já ordenados no momento, e inicialmente contém o primeiro elemento do vetor original.
- O segundo segmento (mais à direita) contém os elementos ainda não ordenados e inicialmente contém $n-1$ elementos.

De modo sequencial é inserido no segmento ordenado um elemento do segmento não ordenado para que fique na posição correta no primeiro segmento.

O “Método Shell” é um aperfeiçoamento do método da inserção, utilizando um algoritmo que passa várias vezes pelos elementos do vetor, dividindo o grupo maior em grupos menores, fazendo inserções nesses grupos menores; o que diminui, drasticamente, o tempo do processo de ordenação. Os programas **“InsertxBolha”** e **“ShellxBolha”**, implementados em Visualg, mostram os resultados da aplicação desses métodos, comparando com o “Método da Bolha”. Esses resultados são bem expressivos; mostrando que o “Método da Bolha” é realmente muito lento quando comparado com outros métodos de classificação, como pode ser observado nos **quadros 1** e **2**. É claro

que esses tempos de processamento podem ser bem menores quando o programa é implementado em linguagens reais de programação, como C, VB, Python, PHP, etc, e com *hardware* mais potente; mas, o importante é observar a diferença de tempo de processamento quando comparamos outros métodos de classificação com o “Método da Bolha”, mesmo rodando em máquinas mais modestas.

Para os dedicados *pythonistas*, uma boa notícia: usando o conceito de **Lista[]** poderão gerar os números com o método **randint()** da classe random; incluir cada um deles na lista com o método **append()**, ordenar essa lista com o método **sorted()** e, finalmente, exibí-los um a um, ordenados, como elementos de um vetor. Para os que codificam em Python é fácil, né!? Mas, e para outras dezenas de linguagens, será que possuem a mesma facilidade?! Portanto, PENSAR LOGICAMENTE é o caminho mais seguro para criar o programa que resolva o problema; independente da linguagem de programação! Os **quadros 1** e **2** mostram as saídas dos programas “**InsertxBolha**” e “**ShellxBolha**”, ambos, implementados em Visualg, comparando os tempos de processamentos dos métodos de “Inserção Direta” e “Shell” com o “Método da Bolha”, respectivamente, para ordenação de um vetor de 1000 elementos inteiros. O **quadro 3** mostra a saída do programa “**MedeTempo**”, codificado em Python, na medição do tempo do mesmo vetor de 1000 elementos inteiros. Observe que a diferença de tempo no processamento em Python, comparando com os programas escritos em Visualg é infinitamente menor. É claro que isto tinha que acontecer, pois enquanto o Visualg é apenas um testador de algoritmos (ferramenta de auxílio ao aprendizado), Python é uma linguagem real de programação. MAS, eu só codifiquei em Python, DEPOIS de testar os algoritmos dos programas no Visualg.

Portanto: “*Programar é solucionar o problema com Lógica, e não procurar facilidades na linguagem de programação!*”

Algoritmo "InsertxBolha"

//Gera vetor de 1000 elementos e o ordena com o "Método da inserção Direta"
//e com o "Método da Bolha".

//Autor: Mário Leite

//-----

Const TamVet=1000

Var VetNum, VetOrig: **vetor**[0..TamVet] **de inteiro**

i, j, n, Aux, incr, Elem, NumRand: **inteiro**

Inicio

Escreval("Processando... Aguarde!")

Para j De 1 Ate TamVet Faca

NumRand <- Randi(1000) //elemento de 1 a 999

VetNum[j] <- NumRand

VetOrig[j] <- VetNum[j] //preserva o vetor original

FimPara

{Ordenação como Método da Inserção}

Cronometro on

Para i De 1 Ate TamVet Faca

Elem <- VetNum[i]

j <- i - 1

Enquanto ((j>=1) e (Elem<VetNum[j])) Faca

VetNum[j+1] <- VetNum[j]

j <- j - 1

FimEnquanto

VetNum[j+1] <- Elem //inserção do elemento

FimPara

Escreval("Tempo gasto com o Método da Inserção:")

Cronometro off

Escreval("")

{Ordenação como Método da Bolha}

Cronometro on

Para i De 1 Ate (Tamvet-1) Faca

Para j De (i+1) Ate Tamvet Faca

Se(VetOrig[i]>VetOrig[j]) Entao

Aux <- VetOrig[i]

VetOrig[i] <- VetOrig[j]

VetOrig[j] <- Aux

FimSe

FimPara

FimPara

Escreval("Tempo gasto com o Método da Bolha:")

Cronometro off

FimAlgoritmo

Algoritmo "ShellxBolha"

```
//Gera vetor de 1000 elementos e o ordena com "Método Shell" e com  
//o Método da Bolha".  
//Autor: Mário Leite
```

```
-----  
Const TamVet=1000  
Var VetNum, VetOrig: vetor[0..TamVet] de inteiro  
    i, j, n, Aux, incr, Elem, NumRand: inteiro  
Inicio  
  Escreval("Processando... Aguarde!")  
  Para j De 1 Ate TamVet Faca  
    NumRand <- Randi(1000) //elemento de 1 a 999  
    VetNum[j] <- NumRand  
    VetOrig[j] <- VetNum[j] //preserva o vetor original  
  FimPara  
  {Ordenação como Método Shell}  
  Cronometro on  
  incr <- 1  
  Repita  
    incr <- 3*incr+1  
  Ate(incr>=TamVet)  
  Repita  
    incr <- Int(incr/3)  
    Para i De incr Ate (TamVet-1) Faca  
      Elem <- VetNum[i]  
      j <- i - incr  
      Enquanto ((j>=0) e (Elem<=VetNum[j])) Faca  
        VetNum[j + incr] <- VetNum[j]  
        j <- j - incr  
        Se(j <= -1) Entao  
          Interrompa //abandona o loop incondicionalmente  
      FimSe  
      FimEnquanto  
      VetNum[j + incr] <- Elem  
    FimPara  
  Ate(incr<=1) //fim do algoritmo Shell  
  Escreval("Tempo gasto com o Método Shell:")  
  Cronometro off  
  Escreval("")  
  {Ordenação como Método da Bolha}  
  Cronometro on  
  Para i De 1 Ate (Tamvet-1) Faca  
    Para j De (i+1) Ate Tamvet Faca  
      Se(VetOrig[i]>VetOrig[j]) Entao  
        Aux <- VetOrig[i]  
        VetOrig[i] <- VetOrig[j]  
        VetOrig[j] <- Aux  
      FimSe  
    FimPara  
  FimPara  
  Escreval("Tempo gasto com o Método da Bolha:")  
  Cronometro off  
FimAlgoritmo
```

```

#MedeTempo.py - medindo tempo de processamento
#Autor: Mário Leite
#-----
import math
import random
import time
EndWhile = "EndWhile"
EndIf = "EndIf"
ListNum = []
j = 1
n = 1000
#loop para gerar e armazenar os 1000 números randômicos
while(j<=n):
    NumRand = random.randint(1,999)
    if(NumRand<10):
        StrRand = "0" + str(NumRand)
    else:
        StrRand = str(NumRand)
    EndIf
    ListNum.append(StrRand)
    j = j + 1
EndWhile
#Inicia algoritmo Shell
inicio = time.time()
incr = 1
while(incr<n):
    incr = 3*incr + 1
    #print(incr)
EndWhile
while(incr>1):
    incr = math.trunc(incr/3)
    i = incr
    while(i<=(n-1)):
        Elem = ListNum[i]
        j = i - incr
        while(j>=0 and (Elem <= ListNum[j])):
            ListNum[j+incr] = ListNum[j]
            j = j - incr
            if(j <= -1):
                break
        EndIf
        EndWhile
        ListNum[j + incr] = Elem
        i = i + 1
    EndWhile
EndWhile #fim do algoritmo Shell
fim = time.time()
Tempo = fim - inicio
Tempo = math.trunc(Tempo*(10**2)+0.50)/10**2 #arredonda com duas decimais
print("Tempo gasto com o Método Shell:{} segundos".format(Tempo))
#-----

```

```
Início da execução
Processando... Aguarde!

Cronômetro iniciado.
Tempo gasto com o Método da Inserção:

Cronômetro terminado. Tempo decorrido: 20 segundo(s) e 156 ms.

Cronômetro iniciado.
Tempo gasto com o Método da Bolha:

Cronômetro terminado. Tempo decorrido: 109 segundo(s) e 594 ms.

Fim da execução.
```

Quadro 1 - Método da Inserção versus Método da Bolha

```
Cronômetro iniciado.
Tempo gasto com o Método Shell:

Cronômetro terminado. Tempo decorrido: 2 segundo(s) e 62 ms.

Cronômetro iniciado.
Tempo gasto com o Método da Bolha:

Cronômetro terminado. Tempo decorrido: 108 segundo(s) e 375 ms.

Fim da execução.
```

Quadro 2 - Método Shell versus Método da Bolha

```
"D:\Curso Phyton\Listas\venv\Scripts\python.exe" "D:/Curso Phyton/Listas/MedeTempo.py"
Tempo gasto com o Método Shell: 0.02 segundos

Process finished with exit code 0
```

Quadro 3 - Tempo de processamento do Método Shell em Python

1ª Iteração	17 15 94 30 99 34 103 50 62 70
2ª Iteração	15 17 94 30 99 34 103 50 62 70
3ª Iteração	15 17 94 30 99 34 103 50 62 70
4ª Iteração	15 17 30 94 99 34 103 50 62 70
5ª Iteração	15 17 30 94 99 34 103 50 62 70
6ª Iteração	15 17 30 34 94 99 103 50 62 70
7ª Iteração	15 17 30 34 94 99 103 50 62 70
8ª Iteração	15 17 30 34 50 94 99 103 62 70
9ª Iteração	15 17 30 34 50 62 94 99 103 70
Vetor Ordenado	15 17 30 34 50 62 70 94 99 103

Quadro 4 - Esquema de ordenação pelo “Método de Inserção Direta”
 Fonte: Leite (2006, p. 117)