Análise de Complexidade de Tempo do Método Bubble Sort

Eduardo Costa de Paiva
eduardocspv@gmail.com
Frederico Franco Calhau
fredericoffc@gmail.com
Gabriel Augusto Marson
gabrielmarson@live.com

Faculdade de Computação
Universidade Federal de Uberlândia

5 de dezembro de 2015

Lista de Figuras

Lista de Tabelas

Lista de Listagens

Sumário

Capítulo 1

Introdução

Este documento foi feito com o intuito de exibir uma análise do algoritmo Bubble Sort com relação a tempo. Além disso, será feita uma comparação da curva de tempo do que se espera do algoritmo, ou seja, $O(n^2)$ com o caso prático.

1.1 Diretório

Dada a seguinte organização das pastas, utilizamos o arquivo testdriver.py, executando, uma função conveniente por vez. Para mais informações vá até ao apêndice.

OBS.: É necessário instalar o programa tree pelo terminal. Isso pode ser feito da seguinte maneira.

```
> sudo apt-get install tree
```

A seguir é mostrada a organização das pastas sendo que os diretórios significativas para o projeto são Codigos e Relatorio além do raíz:

```
tree --charset=ASCII
.
|-- Codigos
|   |-- Bubble
|   |   |-- BubbleSort.py
|   |   `-- __pycache__
|   |       |-- BubbleSort.cpython-34.pyc
|   |       `-- testeBubble.cpython-34.pyc
|   `-- README.md
|-- memoria.py
|-- monitor.py
|-- Other
|   |-- expfit0.py
|   |-- expfit.py
|   |-- leialprof.py
|   |-- leitura1.py
|   |-- leitura2.py
|   |-- leitura.py
|   `-- logfit.py
|-- Plot
```

```
|   |-- plot1.py
|   |-- plot2.py
|   |-- plot3.py
|   `-- plot_tempo.py
|-- __pycache__
|   |-- monitor.cpython-34.pyc
|   `-- tempo.cpython-34.pyc
|-- relatorio
|   |-- imagens
|   |   |-- Bubble
|   |   |   |-- bubble_plot_1.png
|   |   |   `-- bubble_plot_3.png
|   |   `-- Merge
|   |-- Relatorio_Bubble
|   |   |-- RelatorioBubble.aux
|   |   |-- RelatorioBubble.idx
|   |   |-- RelatorioBubble.lof
|   |   |-- RelatorioBubble.log
|   |   |-- RelatorioBubble.lol
|   |   |-- RelatorioBubble.lot
|   |   |-- RelatorioBubble.out
|   |   |-- RelatorioBubble.tex
|   |   `-- RelatorioBubble.toc
|   |-- Relatorio_Merge
|   `-- Resultados
|       |-- Bubble
|       |   `-- tBolha.dat
|       `-- Merge
|-- tempo.py
|-- testdriver.py
|-- testeGeneric.py
`-- testeGeneric.py.lprof

15 directories, 35 files
```

1.2 Códigos de programas

Seguem os códigos utilizados na análise de tempo do algoritmo Bubble Sort.

1. BubbleSort.py: Disponível na Listagem 1.1.

Listagem 1.1: BubbleSort.py

```
1  import numpy as np
2
3  @profile
4  def bubble_sort(a):
5      """ Implementação do método da bolha """
6      for i in range(len(a)):
7          for j in range(len(a)-1-i):
8              if a[j] > a[j+1]:
9                  t = a[j]
10                 a[j] = a[j+1]
11                 a[j+1] = t
12
```

```
#     print(a)  O PRINT BUGA O TESTDRIVER
```

2. testeGeneric.py Disponível na Listagem 1.2

Listagem 1.2: testeGeneric.py

```
##adicionei - Serve para importar arquivos em outro diretório

import sys
sys.path.append('/home/gmarson/Git/AnaliseDeAlgoritmos/Trabalho_Final
    /Codigos/Bubble')

from monitor import *


from BubbleSort import *
import argparse

parser = argparse.ArgumentParser()
parser.add_argument("n", type=int, help="número de elementos no vetor
     de teste")
args = parser.parse_args()

v = criavet(args.n)
bubble_sort(v)



## A EXECUÇÃO DESSE ARQUIVO EH ASSIM
## NA LINHA DE COMANDO VC MANDA O NOME DO ARQUIVO E O TAMANHO DO
    ELEMNTO DO vetor
##EXEMPLO testeBubble.py 10
##ele gera um vetor aleatório (criavet) e manda pro bubble_sort
```

3. monitor.py Disponível na Listagem 1.3

Listagem 1.3: monitor.py

```
# Para instalar o Python 3 no Ubuntu 14 ou 15
#
# sudo apt-get install python3 python3-numpy python3-matplotlib
    ipython3 python3-psutil
#

from math import *
import gc
import random
import numpy as np

from tempo import *

# Vetores de teste
def troca(m,v,n): ## seleciona o nível de embaralhamento do vetor
    m = trunc(m)
    mi = (n-m)//2
    mf = (n+m)//2
    for num in range(mi,mf):
        i = np.random.randint(mi,mf)
        j = np.random.randint(mi,mf)
```

```
21          print("i= ", i, " j= ", j)
22          t = v[i]
23          v[i] = v[j]
24          v[j] = t
25      return v
26
27
28  def criavet(n, grau=0, inf=-1000, sup=1000):
29      passo = (sup - inf)/n
30      if grau < 0.0:
31          v = np.arange(sup, inf, -passo)
32          if grau <= -1.0:
33              return v
34          else:
35              return troca(-grau*n, v, n)
36      elif grau > 0.0:
37          v = np.arange(inf, sup, passo)
38          if grau >= 1.0:
39              return v
40          else:
41              return troca(grau*n, v, n)
42      else:
43          return np.random.randint(inf, sup, size=n)
44
45
46  def executa(fn, v):
47      gc.disable()
48      with Tempo(True) as tempo:
49          fn(v)
50      gc.enable()
```

4. testdriver.py Referenciado no apêndice A.

Capítulo 2

Imagens

Se desejar incluir imagens você poderá usar o comando a seguir:

```
\begin{figure}[!ht]
\centering
\includegraphics[scale=0.5]{imagens/bolha1.png}
\caption{Complexidade de tempo do método da bolha \label{fig:1}}
\end{figure}
```

O comando anterior gerará a imagem a seguir, dado que o arquivo `fig1.png` esteja no subdiretório `imagens`.

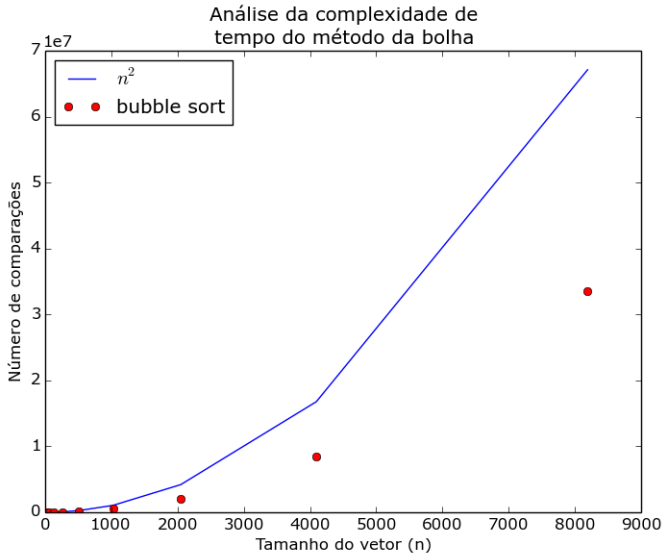

Figura 2.1: *Complexidade de tempo do método da bolha*

Note que a Figura 2.1 pode ser referenciada em qualquer parte do documento. Você também pode incluir diretamente outros formatos de imagens tais como o `jpg`.

Capítulo 3

Tabelas

Aqui vão alguns exemplos de criação de tabelas.

Vamos criar uma tabela simples com o código a seguir. O resultado é a Tabela 3.1

```
\begin{table}[h]
  \centering
  \caption{Alguns opcodes do Dalvik \label{tab:opcode1}}
  \begin{tabular}{lll} \hline
  {\bf Opcode (hex)} & {\bf Nome do opcode} & {\bf Explicação} \\ \hline
  00 & nop & Somente gasta alguns ciclos do processador \\ \hline
  01 & move vx, vy & Move o conteúdo de vy em vx. Ambos os registradores
   devem estar no intervalo de 0 a 255 \\\hline
  \end{tabular}
\end{table}
```

Tabela 3.1: *Alguns opcodes do Dalvik*

Opcode (hex)	Nome do opcode	Explicação
00	nop	Somente gasta alguns ciclos do processador
01	move vx, vy	Move o conteúdo de vy em vx. Ambos os registradores devem es

Note que o texto sob a coluna `Explicação` é muito longo e a tabela 3.1 ficou mal formatada. Podemos resolver esse problema usando colunas de largura fixa, conforme mostrado no código a seguir e cujo resultado é a tabela 3.2.

```
\begin{table}[h]
  \centering
  \caption{Alguns opcodes do Dalvik \label{tab:opcode2}}
  \begin{tabular}{p{1.5cm}lp{9cm}} \hline
  {\bf Opcode (hex)} & {\bf Nome do opcode} & {\bf Explicação} \\ \hline
  00 & nop & Somente gasta alguns ciclos do processador \\ \hline
  01 & move vx, vy & Move o conteúdo de vy em vx. Ambos os registradores
   devem estar no intervalo de 0 a 255 \\\hline
  \end{tabular}
\end{table}
```

Tabela 3.2: *Alguns opcodes do Dalvik*

Opcode (hex)	**Nome do opcode**	**Explicação**
00	nop	Somente gasta alguns ciclos do processador
01	move vx, vy	Move o conteúdo de vy em vx. Ambos os registradores devem estar no intervalo de 0 a 255

Capítulo 4

Citações e referências bibliográficas

Todas as fontes usadas para a confecção de seu relatório devem ser citadas. Isso inclui livros e documentos da internet, tais como tutoriais e páginas.

Apêndice A

Códigos extensos

A.1 testdriver.py

Listagem A.1: testdriver.py

```
import subprocess
import numpy as np
import matplotlib.pyplot as plt
import sys , shutil


##PRA CADA NOVO METODO TEM QUE MUDAR a chamada das funções, o shutil.move(), e o savefig

sys.path.append('/home/gmarson/Git/AnaliseDeAlgoritmos/Trabalho_Final/Codigos/Bubble')  ## adicionei o código de ordenação
sys.path.append('/home/gmarson/Git/AnaliseDeAlgoritmos/Trabalho_Final/relatorio/Resultados/Bubble') ## adicionei o resultado do executa_teste


def executa_teste(arqteste, arqsaida, nlin, intervalo):
    """Executa uma sequência de testes contidos em arqteste, com:
       arqsaida: nome do arquivo de saída, ex: tBolha.dat
       nlin: número da linha no arquivo gerado pelo line_profiler contendo
             os dados de interesse. Ex: 14
       intervalo: tamanhos dos vetores: Ex: 2 ** np.arange(5,10)
    """
    f = open(arqsaida,mode='w', encoding='utf-8')
    f.write('#      n   comparações      tempo(s)\n')

    for n in intervalo:
        cmd = ' '.join(["kernprof -l -v", "testeGeneric.py", str(n)])
        str_saida = subprocess.check_output(cmd, shell=True).decode('utf-8')
        linhas = str_saida.split('\n')
        unidade_tempo = float(linhas[1].split()[2])
        #print("CMD:", cmd, "\nSTR_SAIDA: ",str_saida,"\nLINHAS: ",linhas,"\nUNIDADE_TEMPO: ",unidade_tempo)
        #print("Linhas4:",linhas[4]," ----->  Linhas 4 float: ",linhas[4].split()[2])
        tempo_total = float(linhas[3].split()[2])
```

```
31          lcomp = linhas[nlin].split()
32          num_comps = int(lcomp[1])
33          str_res = '{:>8} {:>13} {:13.6f}'.format(n, num_comps, tempo_total
                )
34          print(str_res)
35          f.write(str_res + '\n')
36      f.close()
37      shutil.move("tBolha.dat", "/home/gmarson/Git/AnaliseDeAlgoritmos/
            Trabalho_Final/relatorio/Resultados/Bubble/tBolha.dat")
38
39  #executa_teste("testeGeneric.py", "tBolha.dat", 14, 2 ** np.arange(5,14))
40
41  def plota_teste1(arqsaida):
42      n, c, t = np.loadtxt(arqsaida, unpack=True)
43      #print("n: ",n,"\nc: ",c,"\nt: ",t)
44
45      plt.plot(n, n ** 2, label='$n^2$')  ## custo esperado bubble Sort
46      plt.plot(n, c, 'ro', label='bubble sort')
47
48      # Posiciona a legenda
49      plt.legend(loc='upper left')
50
51      # Posiciona o título
52      plt.title('Análise da complexidade de \ntempo do método da bolha')
53
54      # Rotula os eixos
55      plt.xlabel('Tamanho do vetor (n)')
56      plt.ylabel('Número de comparações')
57
58      plt.savefig('relatorio/imagens/Bubble/bubble_plot_1.png')
59      plt.show()
60
61  #plota_teste1("/home/gmarson/Git/AnaliseDeAlgoritmos/Trabalho_Final/
        relatorio/Resultados/Bubble/tBolha.dat")
62
63  def plota_teste2(arqsaida):
64      n, c, t = np.loadtxt(arqsaida, unpack=True)
65      plt.plot(n, n ** 2, label='$n^2$')
66      plt.plot(n, t, 'ro', label='bubble sort')
67
68      # Posiciona a legenda
69      plt.legend(loc='upper left')
70
71      # Posiciona o título
72      plt.title('Análise da complexidade de \ntempo do método da bolha')
73
74      # Rotula os eixos
75      plt.xlabel('Tamanho do vetor (n)')
76      plt.ylabel('Tempo(s)')
77
78      plt.savefig('bubble2.png')
79      plt.show()
80
81  def plota_teste3(arqsaida):
82      n, c, t = np.loadtxt(arqsaida, unpack=True)
83
84      # Calcula os coeficientes de um ajuste a um polinômio de grau 2 usando
85      # o método dos mínimos quadrados
86      coefs = np.polyfit(n, t, 2)
```

```
87      p = np.poly1d(coefs)
88
89      plt.plot(n, p(n), label='$n^2$')
90      plt.plot(n, t, 'ro', label='bubble sort')
91
92      # Posiciona a legenda
93      plt.legend(loc='upper left')
94
95      # Posiciona o título
96      plt.title('Análise da complexidade de \ntempo do método da bolha')
97
98      # Rotula os eixos
99      plt.xlabel('Tamanho do vetor (n)')
100     plt.ylabel('Tempo(s)')
101
102     plt.savefig('relatorio/imagens/Bubble/bubble_plot_3.png')
103     plt.show()
104
105 #plota_teste3("/home/gmarson/Git/AnaliseDeAlgoritmos/Trabalho_Final/
        relatorio/Resultados/Bubble/tBolha.dat")
106
107
108 def plota_teste4(arqsaida):
109     n, c, t = np.loadtxt(arqsaida, unpack=True)
110
111     # Calcula os coeficientes de um ajuste a um polinômio de grau 2 usando
112     # o método dos mínimos quadrados
113     coefs = np.polyfit(n, c, 2)
114     p = np.poly1d(coefs)
115
116     plt.plot(n, p(n), label='$n^2$')
117     plt.plot(n, c, 'ro', label='bubble sort')
118
119     # Posiciona a legenda
120     plt.legend(loc='upper left')
121
122     # Posiciona o título
123     plt.title('Análise da complexidade de \ntempo do método da bolha')
124
125     # Rotula os eixos
126     plt.xlabel('Tamanho do vetor (n)')
127     plt.ylabel('Número de comparações')
128
129     plt.savefig('bubble4.png')
130     plt.show()
131
132 def plota_teste5(arqsaida):
133     n, c, t = np.loadtxt(arqsaida, unpack=True)
134
135     # Calcula os coeficientes de um ajuste a um polinômio de grau 2 usando
136     # o método dos mínimos quadrados
137     coefs = np.polyfit(n, c, 2)
138     p = np.poly1d(coefs)
139
140     # set_yscale('log')
141     # set_yscale('log')
142     plt.semilogy(n, p(n), label='$n^2$')
143     plt.semilogy(n, c, 'ro', label='bubble sort')
144
```

```
145     # Posiciona a legenda
146     plt.legend(loc='upper left')
147 
148     # Posiciona o título
149     plt.title('Análise da complexidade de \ntempo do método da bolha')
150 
151     # Rotula os eixos
152     plt.xlabel('Tamanho do vetor (n)')
153     plt.ylabel('Número de comparações')
154 
155     plt.savefig('bubble5.png')
156     plt.show()
```