Alexandre Yuji Kajihara
Clodoaldo Basaglia da Fonseca

# Avaliação empírica

Relatório técnico de atividade prática solicitado pelo professor Rodrigo Campiolo na disciplina de Análise de Algoritmos do Bacharelado em Ciência da Computação da Universidade Tecnológica Federal do Paraná.

Universidade Tecnológica Federal do Paraná – UTFPR

Departamento Acadêmico de Computação – DACOM

Bacharelado em Ciência da Computação – BCC

Campo Mourão

Novembro / 2017

# Resumo

O objetivo dessa avaliação prática supervisionada foi realizar a avaliação empírica dos diferentes algoritmos que tentam resolver o problema do máximo valor do subarranjo, implementados em diferentes linguagens de programação, utilizando sempre os mesmos arranjos e comparar o desempenho entre as linguagens e os algoritmos. De inicio foi implementado um algoritmo que gerasse números aleatórios para os diferentes tamanhos que incluíssem mais valores positivos do que negativos. Após isso, implementamos o algoritmo de máximo valor do subarranjo em diferentes linguagens de programação. O resultado é que conseguimos encontrar os mesmos valores de máximos valores de subarranjo para todos os programas, mas com valores *wall-clock time* diferentes. Concluímos que existe diferença notória entre os diferentes algoritmos que tentam resolver o problema proposto, e que a linguagem também afeta muito, já que tivemos um caso em uma das linguagens apresentou um desempenho não satisfatório.

**Palavras-chave**: Golang. C. Java. Python.

# Sumário

# 1 Introdução

Para realizar avaliação empírica é necessário entender o problema: o problema do máximo valor do subarranjo tenta encontrar o maior de valor de um subarranjo dado um arranjo de entrada. Além disso, é indispensável ter conhecimento de programação, para escrever os algoritmos que tentam resolver o problema do máximo valor do subarranjo, de onde deve ser invocada as funções que irão mensurar o tempo *wall-clock time* do nosso algoritmo, de como compilar e executar os diferentes de programas de diferentes linguagens de programação.

# 2 Objetivos

O objetivo dessa avaliação prática supervisionada é comparar *wall-clock time* da implementação do algoritmo em diferentes linguagens e com diferentes entradas.

# 3 Fundamentação

Os fundamentos necessários para desenvolver a avaliação empírica foram compreender as particularidades de cada linguagem de como declarar uma função ou método, variáveis, laços de repetição, incrementar variáveis, imprimir os resultados que desejavamos na tela e onde invocar as funções que irão medir o tempo dos algoritmos.

Tanto para a Java quanto para Go, os comandos para instalação foram parecidos, como pode ser observado a baixo:

Java

```
$ sudo add-apt-repository ppa:webupd8team/java
$ sudo apt-get update
$ sudo apt-get install oracle-java8-installer
```

Go

```
$ sudo add-apt-repository ppa:ubuntu-lxc/lxd-stable
$ sudo apt-get update
$ sudo apt-get install golang
```

Após a instalação precisamos configurar alguns parâmetros, como o *GOPATH* e o *GOBIN*, que pode ser feito da seguinte maneira:

```
$ export GOPATH=caminho_para_seu_workspace/go/
$ export PATH=$PATH:$GOPATH/bin
$ export GOBIN=caminho_para_seu_workspace/go/bin
```

Nos comandos acima o "caminho_para_seu_workspace"deve ser substituído pela localização dá pasta onde se deseja criar os seus arquivos no formato ".go". Para evitar de executar esses três comandos toda vez que ligar a sua máquina, você pode adicionar os três comandos acima em um arquivo chamado ".profile". As demais linguagens escolhidas não foram necessárias instalação, pois no nosso sistema operacional em que realizamos a avalição empírica já vieram instaladas que foram o Python e o C.

## 4 Materiais

Os materiais necessários foram um *laptop* com as seguintes configurações:

- Sistema Operacional Ubuntu 17.04 64 bits
- Intel Core i7-6500U
- 16 GB de memória RAM (*Random Access Memory*)
- Placa de vídeo NVIDIA Geforce 940M com 4GB
- GCC compilador para linguagem de programação C na versão 6.3.0
- Java compilador para linguagem de programação Java na versão 8
- Go compilador para linguagem de programação Golang na versão 1.7.4
- Python3 compilador para linguagem de programação Python na versão 3.5.3
- Editor de texto Sublime Text 3143

## 5 Procedimentos e Resultados

A primeira etapa foi definir as quatros linguagens de programação que iriam implementar as diferentes formas de solucionar o problema do máximo valor do subarranjo. Dessas quatros linguagens uma delas deveria ser interpretada e a outra compilada. Então escolhemos a linguagem de programação C que é uma linguagem compilada (BEZERRA, 2015), Python que é uma linguagem interpretada (JAISWAL, 2014), Java que também é uma linguagem interpretada e compilada (KAY, 1999), Golang que é uma linguagem que nasceu em 2007 a partir de algumas necessidades do Google(RYCHLEWSKI, 2016) e a mesma é uma linguagem compilada (SUMMERFIELD, 2012) e algumas companhias que utilizam são: Facebook, Netflix, Dropbox (NAGESH, 2017). Definida as linguagens, começamos a escrever os algoritmos (todos os algoritmos que iremos citar abaixo estão no Github, que pode ser acessado a partir de um link presente nas referências (KAJIHARA, 2017)).

O primeiro algoritmo implementado foi o de gerar arranjos com valores aleatórios. O mesmo foi necessário devido cada linguagem implementar diferentimente a geração de números aleatórios. Esse algoritmo recebe três valores de entrada, sendo esses o maior e o menor valor a ser gerado, bem como a quantidade de valores a serem gerados. A semente escolhida foi a de valor 10000, isso porque testamos vários valores diferentes para a semente, e esse foi o que apresentou mais valores positivos do que negativos.

Com esse algoritmo escrito, pegamos a saída dele e salvamos em arquivos no formato ".txt". Os tamanhos de arranjos escolhidos foram: 128, 256, 512, 1024, 2048, 4096 e 8192, e decidimos por esses valores que devido acreditamos que iria ser bem notório a diferença do *wall-clock time* entre os diferentes tamanhos de entrada, os algoritmos e as linguagens de programação. O intervalo de valores que escolhemos foi de -8192 até 8192, devido que o maior valor de tamanho de arranjo que temos é de 8192, e já que se utilizarmos outro intervalos de valores poderia afetar no *wall-clock time* das diferentes linguagens de programação e algoritmos

Após termos todas as entradas, começamos a implementar os algoritmos que tentam resolver o problema do máximo valor do subarranjo. Então escrevemos o algoritmo que utiliza três laços de repetição aninhado, o segundo que utiliza apenas dois laços de repetição, o terceiro que utiliza a técnica de divisão e conquista e o último que utiliza o conceito de programação dinâmica, para as linguagens Golang, Java, C e Python, nessa ordem respectivamente. Como foi dito nas seções anteriores optamos por mensurar utilizando o *wall-clock time*, que também pode ser chamado de *wall time* ou de *real-world time*, refere-se ao tempo decorrido, conforme um cronômetro ou um relógio de pulso ou relógio de parede (ROUSE, 2011). Um outro motivo pelo qual escolhemos o *wall-clock time* é que todas as linguagens de programação que foram escolhidas existe a possibilidade invocar funções que permitem medir o tipo de medição que foi escolhido.

Com todos os algoritmos que resolvem o problema do máximo valor do subarranjo implementados para todas as linguagens que foram escolhidas, executamos os setes arranjos de tamanhos diferentes para cada algoritmo, ou seja, cada linguagem escolhida implementamos os quatros algoritmos, e para cada um desses algoritmos executamos os setes diferentes arranjos. Por exemplo, esse último processo, foi repetido por cinco vezes, ou seja, na linguagem de programação Java, usando o algoritmo que utiliza a técnica de programação dinâmica, com o arranjo de entrada tendo o valor de 8192 cinco vezes, para que possamos ter uma média dos *wall-clock time*. Isso foi feito em todos os algoritmos, de todas as linguagens e com todos os tamanhos de arranjo proposto.

Os resultados obtidos foram interessantes, tanto no valor de máximo subarranjo que foi encontrado entre os diferentes tamanhos de arranjo de entrada, quanto no *wall-clock time* entre as diferentes linguagens de programação.

# 6 Discussão dos Resultados

O primeiro resultado que iremos discutir, é o dos valores que foram encontrados quando resolvemos o problema do máximo valor do subarranjo, para os diferentes tamanho de arranjo de entrada, em que a primeira linha refere-se aos diferentes tamanhos de arranjo utilizado na entrada dos algoritmos. Os valores da tabela 1 foram os mesmos para todos os algoritmos, para todas as linguagens e sempre utilizando as mesmas entradas.

Tabela 1: Valores máximo do subarranjo para diferentes tamanhos de entrada

| | 128 | 256 | 512 | 1024 | 2048 | 4096 | 8192 |
|---|---|---|---|---|---|---|---|
| Algoritmo | 104294 | 104294 | 104294 | 131699 | 245447 | 245447 | 314688 |

A tabela 1, apresenta alguns valores iguais com diferentes tamanhos de arranjos de entrada. Acreditamos que essa coincidência aconteceu devido a utilização da mesma semente para os diferentes tamanhos de arranjo de entrada, e que no caso da repetição do 128, 256 e 512.Se notarmos as entradas, os primeiros 128 valores estão presentes nas 128 primeiras posições no arranjo de tamanho 256 são o mesmo, e o mesmo vale para o arranjo de tamanho 256 em que seus valores estão contidos no arranjo de tamanho 512, e o mesmo vale para os arranjos com tamanho de 2048 e 4096. Os resultados do *wall-clock time* que obtivemos podem ser visualizados nas tabelas 2, 3, 4, 5, 6, 7, 8 e 9.

Tabela 2: Média de *Wall-clock time* obtido da linguagem de programação C em segundos para os arranjos 128, 256, 512 e 1024.

| | Linguagem de programação C | | | |
|---|---|---|---|---|
| | 128 | 256 | 512 | 1024 |
| Algoritmo 1 | 0,0040546 | 0,0375932 | 0,1013786 | 0,5431192 |
| Algoritmo 2 | 0,000218 | 0,0004436 | 0,001181 | 0,0041086 |
| Algoritmo 3 | 0,000186 | 0,0001752 | 0,00022 | 0,0002796 |
| Algoritmo 4 | 0,0001238 | 0,0001092 | 0,0001538 | 0,0002518 |

Tabela 3: Média de *Wall-clock time* obtido da linguagem de programação C em segundos para os arranjos 2048,4096 e 8192.

| | Linguagem de programação C | | |
|---|---|---|---|
| | 2048 | 4096 | 8192 |
| Algoritmo 1 | 3,9162902 | 30,9227674 | 265,8441478 |
| Algoritmo 2 | 0,0131044 | 0,0515182 | 0,1149056 |
| Algoritmo 3 | 0,0004658 | 0,0009078 | 0,0027186 |
| Algoritmo 4 | 0,0002266 | 0,0002726 | 0,0004236 |

Tabela 4: Média de *Wall-clock time* obtido da linguagem de programação Java em segundos para os arranjos 128, 256, 512 e 1024.

| | Linguagem de programação Java | | | |
|---|---|---|---|---|
| | 128 | 256 | 512 | 1024 |
| Algoritmo 1 | 0,0031202 | 0,00596 | 0,017463 | 0,1023826 |
| Algoritmo 2 | 0,000331 | 0,000829 | 0,0017044 | 0,0034878 |
| Algoritmo 3 | 0,0002986 | 0,0004414 | 0,0005936 | 0,0008596 |
| Algoritmo 4 | 0,0002316 | 0,0002616 | 0,0003384 | 0,0004144 |

Tabela 5: Média de *Wall-clock time* obtido da linguagem de programação Java em segundos para os arranjos 2048,4096 e 8192.

| | Linguagem de programação Java | | |
|---|---|---|---|
| | 2048 | 4096 | 8192 |
| Algoritmo 1 | 0,759792 | 5,8092908 | 48,806878 |
| Algoritmo 2 | 0,007644 | 0,0132856 | 0,0358966 |
| Algoritmo 3 | 0,0011792 | 0,0015106 | 0,0019914 |
| Algoritmo 4 | 0,0004568 | 0,0006352 | 0,000938 |

Tabela 6: Média de *Wall-clock time* obtido da linguagem de programação Golang em segundos para os arranjos 128, 256, 512 e 1024.

| | Linguagem de programação Golang | | | |
|---|---|---|---|---|
| | 128 | 256 | 512 | 1024 |
| Algoritmo 1 | 0,0016004 | 0,0124554 | 0,0406806 | 0,1299182 |
| Algoritmo 2 | 0,0001762 | 0,000475 | 0,001582 | 0,0061084 |
| Algoritmo 3 | 0,0004524 | 0,0002146 | 0,0002374 | 0,000494 |
| Algoritmo 4 | 0,0000752 | 0,0001188 | 0,0001252 | 0,0001292 |

Tabela 7: Média de *Wall-clock time* obtido da linguagem de programação Golang em segundos para os arranjos 2048,4096 e 8192.

| | Linguagem de programação Golang | | |
|---|---|---|---|
| | 2048 | 4096 | 8192 |
| Algoritmo 1 | 0,7830532 | 5,9709588 | 48,5996486 |
| Algoritmo 2 | 0,022969 | 0,0408362 | 0,0915008 |
| Algoritmo 3 | 0,0024038 | 0,0015654 | 0,0026466 |
| Algoritmo 4 | 0,000176 | 0,0002242 | 0,0003572 |

Tabela 8: Média de *Wall-clock time* obtido da linguagem de programação Python para os arranjos 128, 256, 512 e 1024.

| | Linguagem de programação Python | | | |
|---|---|---|---|---|
| | 128 | 256 | 512 | 1024 |
| Algoritmo 1 | 0,033163 | 0,2353152 | 1,9783996 | 18,256785 |
| Algoritmo 2 | 0,0020034 | 0,0094096 | 0,046559 | 0,115786 |
| Algoritmo 3 | 0,0007772 | 0,0016868 | 0,0025162 | 0,0048354 |
| Algoritmo 4 | 0,0001988 | 0,0003986 | 0,0007238 | 0,0043714 |

Tabela 9: Média de *Wall-clock time* obtido da linguagem de programação Python em segundos para os arranjos 2048,4096 e 8192.

| | Linguagem de programação Python | | |
|---|---|---|---|
| | 2048 | 4096 | 8192 |
| Algoritmo 1 | 287,107713 | 2181,80139 | 12302,71828 |
| Algoritmo 2 | 0,5349128 | 2,1712456 | 8,7654736 |
| Algoritmo 3 | 0,0148798 | 0,0217884 | 0,051592 |
| Algoritmo 4 | 0,0094546 | 0,0230022 | 0,0551476 |

Os resultados que obtivemos já eram esperado em relação aos algoritmos, em que o algoritmo quatro é superior aos demais e que o algoritmo um é o que têm um desempenho inferior, devido que o quarto algoritmo usa a técnica de programação dinâmica, que é uma tradução iterativa inteligente da recursão e pode ser definido, vagamente, como "recursão com apoio de uma tabela"(FEOFILOFF, 2015). Já o algoritmo 1 sabíamos que teria o pior desempenho, devido que durante a sua implementação percebemos que o mesmo possui três laços de repetição aninhados o que acarretaria em um *wall-clock time* elevado. Em relação aos demais já esperavámos que eles iriam ser melhores que o primeiro algoritmo, mas com desempenho inferior ao quarto. Com as entradas utilizadas, já tinhamos a expectativa que as mesmas iam piorando o seu *wall-clock time* a medida que o tamanhos dos arranjos de entradas aumentam, já que aumenta a quantidade de elementos para encontrar o valor máximo do subarranjo. Agora o que realmente nós surpreendeu foi a linguagem de programação, vimos que as compiladas tiveram um desempenho superior as interpretadas, porém não achávamos que iria ser de horas a diferença entre as linguagens, mas sim de minutos ou talvez de segundos.

# 7 Conclusões

Após realizar essa atividade prática supervisionada, em que realizamos avaliação empírica dos algoritmos que tentam encontrar o máximo valor do subarranjo, vimos que existem muitos fatores que influenciam no *wall-clock time*, como a técnica ou a maneira que pensamos para tentar resolver um problema, a entrada e a linguagem de programação.

Um fato muito importante que essa atividade prática nos fez refletir é que os fatores que influenciam no *wall-clock time* que citamos anteriormente devem ser muito bem escolhidos. Isso porque, se temos um problema que irá utilizar entradas muito grandes ou irá receber entradas de vários usuários com diferentes entradas ou entradas muitos grandes, o algoritmo ele tem que ser otimizado para entregar a saída em um tempo satisfatório a quem solicitou, então saber que tipo de entradas serão permitidas irá saber definir que técnica deverá ser empregada ao algoritmo irá tentar resolver o problema, e também saber as pecularidades de cada tipo de linguagem irá auxiliar a escolher uma linguagem mais adequada para ser empregada. Por fim, um outro tópico que essa avaliação empírica nos fez pensar é que muitas vezes quando tentamos resolver um problema, tentamos resolver de maneira rápida, só para que resolva o problema que queremos que seja resolvido no momento, portanto acreditamos que devemos parar de pensar dessa maneira, e tentar resolver o problema de maneira mais geral, não envolvendo apenas um caso, mas sim todos os casos possíveis que podem ocorrer, para que tenhamos o *wall-clock time* com o menor valor possível tanto para entradas grandes quanto pequenas.

# 8 Referências

BEZERRA, R. F. S. *Compiladores x Intepretadores*. 2015. SlideShare. Disponível em: <https://pt.slideshare.net/100002330670325/linguagem-compiladas-e-interpretadas>. Citado na página 5.

FEOFILOFF, P. *Programação dinâmica*. 2015. IME (Instituto Militar de Engenharia) - USP (Universidade de São Paulo). Disponível em: <https://www.ime.usp.br-/~pf/analise_de_algoritmos/aulas/dynamic-programming.html>. Citado na página 9.

JAISWAL, S. *Python: Interpreted or Compiled*. 2014. Swati Jaiswal. Disponível em: <http://swatij.me/python/python-interpreted-or-compiled/>. Citado na página 5.

KAJIHARA, A. Y. *Análise empírica - Análise de algoritmos - (2017-2)*. 2017. Github. Disponível em: <https://github.com/xaaaandao/analise-empirica-20172>. Citado na página 5.

KAY, J. *Compiled vs Interpreted*. 1999. Purdue University - Departament of Computer Science. Disponível em: <https://www.cs.purdue.edu/homes/cs290w/javaLecs/wk1-/minilec2.html>. Citado na página 5.

NAGESH, K. *Golang & Docker—Microservices (RESTful API) for Enterprise Model*. 2017. Hackernoon. Disponível em: <https://hackernoon.com/golang-docker-microservices-for-enterprise-model-5c79addfa811>. Citado na página 5.

ROUSE, M. *wall time (real-world time or wall-clock time)*. 2011. WhatIs.com. Disponível em: <http://whatis.techtarget.com/definition/wall-time-real-world-time-or-wall-clock-time>. Citado na página 6.

RYCHLEWSKI, F. *Trabalhando com Go (GoLang), a linguagem do Google*. 2016. IMasters. Disponível em: <https://imasters.com.br/linguagens/trabalhando-com-go-golang-a-linguagem-do-google/?trace=1519021197>. Citado na página 5.

SUMMERFIELD, M. *Getting Going with Go*. 2012. Dr.Dobb's - THE WORLD OF SOFTWARE DEVELOPMENT. Disponível em: <http://www.drdobbs.com/open-source/getting-going-with-go/240004971>. Citado na página 5.