Cláudia L. Poiet Sampedro
Igor N. Faustino
João Victor Nascimento
Letícia Mazzo Portela

Avaliação empírica de algoritmos

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

Desde agosto de 2017 foram expostos diversos conceitos na disciplina de Análise de Algoritmos, tendo esta importância elevada, já que é caracterizada como fundamental na área de Computação. Em vista disso, o presente trabalho contempla aplicações práticas dos conceitos vistos em sala de aula, especificamente, na análise empírica de algoritmos. Para tal feito, foram realizadas comparações, em quatro linguagens de programação diferentes, de quatro versões do algoritmo Subvetor Máximo. Dessa forma, pretende-se mostrar como o comportamento de um mesmo algoritmo pode variar, levando em conta a linguagem em que foi elaborado, o hardware em que foi executado, o software utilizado e, também, certas configurações específicas.

Palavras-chave: Análise empírica. Algoritmos. Linguagens de programação.

Sumário

1 Introdução

Os algoritmos têm por finalidade resolver problemas por meio de uma sequência bem definida de ações, sendo possível existir diversos algoritmos, elaborados de formas diferentes, para resolver o mesmo problema. Em vista disso, existe a Análise de Algoritmos, cujo propósito é o de fazer uma previsão dos recursos que determinado algoritmo fará uso (FOLEISS, 20–a). Assim, ao contrapor dois algoritmos que resolvem o mesmo problema, é possível determinar, através da Análise de Algoritmos, qual deles é o mais eficiente, levando em conta fatores como tempo de execução, quantidade de entradas, dentre outros.

Ao longo do segundo semestre de 2017, foram abordados diversos conceitos e demonstrados exemplos na disciplina de "Análise de Algoritmos", ministrada pelo professor Rodrigo Campiolo, no quarto período do curso de Bacharelado em Ciência da Computação. Sendo assim, com base no que fora apresentado, solicitou-se a elaboração do presente relatório, no qual se encontram considerações a respeito da avaliação empírica de algoritmos.

Assim, para a elaboração do respectivo relatório, foi tomado como base o Problema do Subvetor Máximo. Para resolver tal problema, foram apresentadas quatro maneiras: Enumeração, Enumeração Melhorada, Divisão e Conquista, e Programação Dinâmica (BORRADAILE, 20–).

Para tanto, solicitou-se que para cada maneira de se resolver o problema, fossem desenvolvidas quatro versões em linguagens de programação diferentes, sendo exigido que pelo menos uma dessas linguagens fosse compilada e pelo menos uma interpretada. Dessa forma, as linguagens escolhidas foram: C (compilada), Pascal (compilada), Java (interpretada) e Python (interpretada). Em seguida, realizou-se a análise empírica dos algoritmos desenvolvidos.

2 Objetivos

Como mencionado na seção anterior, o presente relatório tem como intuito aplicar o conhecimento que fora adquirido em sala de aula. Além disso, por meio da análise empírica, pretende-se mostrar como o comportamento de um mesmo algoritmo pode variar, levando em conta a linguagem em que foi elaborado, o hardware em que foi executado, o software utilizado e, também, certas configurações específicas.

3 Fundamentação

3.1 Análise Empírica

A análise empírica ou avaliação empírica de algoritmos, se caracteriza como o estudo do tempo despendido para a execução de certos algoritmos. A medição deste tempo pode ser realizada através da contagem do número de vezes que cada instrução presente no código em questão é executada ou pela temporização de cada trecho do algoritmo (FABRO, 2009). Assim, por meio dos procedimentos citados, é possível efetuar a otimização do algoritmo, pois se consegue localizar os pontos em que leva mais tempo para executar.

Importante salientar que o hardware e software utilizados na execução do algoritmo, além de determinadas configurações, são influenciadores para o tempo de execução despendido pelo mesmo e, também, para a forma com que será utilizada a memória.

Sendo assim, segundo Fabro (2009), os principais objetivos da análise empírica são: "avaliar a corretude dos algoritmos, comparar a eficiência de diferentes algoritmos, comparar implementações alternativas do mesmo algoritmo, identificar a complexidade do algoritmo".

3.2 Problema do Subvetor Máximo

Seja um vetor *V[1..n]* com n números inteiros positivos e negativos, o subvetor *V[e..d]*, em que $e >= 1$, $d <= n$ e $\sum_{i=e}^{d} V[i]$ é máxima, é denominado subvetor máximo (FOLEISS, 20–b).

Por exemplo, dado o vetor *V[31, -41,* ***59, 26, -53, 58, 97****, -93, -23, 84]*, a soma dos elementos em destaque é igual a 187, sendo o subvetor máximo, portanto, identificado por estes elementos.

3.3 Linguagens de Programação

3.3.1 C

A linguagem de programação C foi criada em 1970 por Dennis Ritchie, no ATT Bell Labs, nos Estados Unidos. Sua criação foi baseada na linguagem B, criada por Ken Thompson (FONSECA IGUATEMI EDUARDO DA; ALEXANDRE, 2014).

C se caracteriza como uma linguagem estruturada, assim sendo, é necessário que os programas desenvolvidos nessa linguagem sigam uma disciplina, facilitando sua correção. Além disso, C é *case sensitive*, ou seja, diferencia letras maiúsculas de minúsculas.

3.3.2 Pascal

Pascal é uma linguagem de programação criada em 1968 por Niklaus Wirth, do Instituto de Informática ETH (Eidgenossische Technische Hochschule), em Zurique, Suiça. Seu nome é uma homenagem ao matemático Blaise Pascal (1623-1662) e, além disso, foi criada tendo como base as linguagens ALGOL e PL/I (CASTILHO, 2009).

Assim como C, Pascal também é considerada uma linguagem de paradigma estruturado. Todavia, devido ao fato de ter surgido antes de C, é reconhecida como uma linguagem que revolucionou o que se conhecia por linguagens de programação até então, por conta dos novos conceitos que trouxe, bem como criação de novos tipos, uso de funções e procedimentos recursivos, etc. (CASTILHO, 2009).

3.3.3 Java

A linguagem Java teve sua primeira versão lançada em 1996, no entanto, começou a ser elaborada em 1991 em alguns projetos de um grupo de funcionários da Sun Microsystems (hoje pertencente a Oracle Corporation), nos Estados Unidos. Seu nome é uma homenagem às xícaras de café, já que a equipe, durante o seu desenvolvimento, ingeria muito café (CLARO DANIELA BARREIRO; SOBRAL, 2008).

Java foi utilizada, inicialmente, no ambiente web, onde obteve grande sucesso. Com o tempo, passou por várias evoluções e nos dias atuais é utilizada em diversas aplicações e dispositvos presentes no cotidiano (CLARO DANIELA BARREIRO; SOBRAL, 2008). Vale ressaltar que o paradigma utilizado por essa linguagem é o de orientação a objetos, onde os principais conceitos utilizados são os de Classes, Objetos, Associação e Encapsulamento.

3.3.4 Python

Python é uma linguagem de programação que foi criada em 1982 por Guido Van Rossun, no CWI (Centrum Wiskund & Informatica), em Amsterdâ, Holanda. Essa linguagem foi desenvolvida com o intuito de sempre possuir liberdade, ser gratuita e de código aberto, além de ter disponibilidade para vários sistemas e possuir clareza em sua sintaxe (LABAKI, 20–).

Da mesma forma que Java, Python também se enquadra no paradigma de orientação a objetos e, além disso, possui alta modularidade. Um fato interessante a respeito dessa linguagem, é que a mesma já foi utilizada pela empresa Industrial Lights and Magic para realizar o controle de certos efeitos de Star Wars, dentre outras aplicações (LABAKI, 20–).

4 Materiais

Para a execução e avaliação dos algoritmos, foi utilizado um notebook com processador Intel Core i7 com velocidade de 2.2GHz, 8GB de memória RAM e 1TB de disco rígido. Além disso, o sistema operacional era o Linux Debian versão 9.

5 Procedimentos

Como previamente citado e referenciado, foram implementadas quatro formas do mesmo algoritmo em quatro linguagens, as quais possuem complexidade e tempo de execução distintos (BORRADAILE, 20–). Todos os códigos foram testados com 5 entradas díspares, entretanto, estas possuem a mesma semente de inicialização. Além disso, foi usada uma função para gerar números aleatórios, e estes foram salvos em arquivos. Todos os códigos e linguagens foram providos dos mesmos arquivos gerados, os quais possuíam entradas de 500, 1000, 2500, 5000 e 7500 dados.

A primeira versão do algoritmo, chamada de Enumeração, percorria com um laço de repetição a cada par do vetor, executando a soma e mantendo esta salva. A cada iteração dos laços, a soma anterior era comparada a atual, sendo armazenado o maior valor entre elas. Este algoritmo possui complexidade $T(n) = O(n)$, sendo bastante custoso e delongado quando executado com entradas robustas.

A segunda implementação mantinha a mesma lógica da anterior, entretanto, eliminava o laço que percorria cada par e, por consequência, a soma não era computada tantas vezes. Este algoritmo possui complexidade $T(n) = O(n)$, caracterizando-se como mais viável do que o anterior.

Na terceira implementação foi tirado proveito do conceito de Divisão e Conquista, o qual consiste em dividir o vetor em partes menores, para facilitar o processamento, e juntar as mesmas depois de processadas, chegando ao resultado desejado. No caso deste algoritmo, ao dividir o vetor existe a possibilidade da soma máxima estar na primeira ou segunda metade, ou então, entre estas. Esta versão possui complexidade $T(n) = O(nlgn)$.

A ultima versão implementada usava programação dinâmica como base, partindo da premissa que o subvetor máximo usa ou não o último elemento do vetor principal. Esta é a implementação menos custosa, possuindo complexidade $T(n) = O(n)$.

6 Discussão dos Resultados

Os resultados das comparações entre os algoritmos foram reunidos em gráficos, os quais relacionam o tamanho das entradas com o tempo real de execução, medido em escala

logarítmica, apenas da função que encontra o maior subvetor. Para melhor aplicação da análise empírica, foram ignoradas as seções de entrada e saída.

Cada gráfico retrata uma implementação, usando linhas de cores distintas para representar cada uma das linguagens. No eixo X estão contidos os tamanhos das entradas e no Y o tempo gasto em milissegundos.

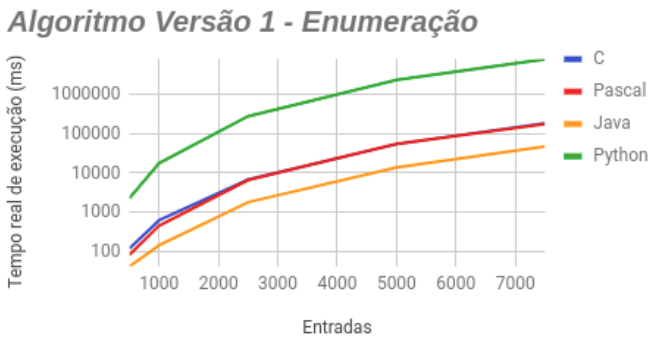

Figura 1 – Comparação das linguagens no algoritmo de Versão 1.

Na Figura 1, onde está representada a primeira versão, é visível que o algoritmo implementado em Python possui desempenho degradado em relação as outras linguagens utilizadas. Quando esta implementação é executada com entradas robustas, independente da linguagem, é perceptível um considerável prolongamento no tempo de execução.

A linguagem Java veio a se destacar pelo seu desempenho, mesmo com entradas numerosas mostrou menor tempo de execução quando comparada as outras linguagens testadas. C e Pascal apresentaram desempenho semelhante, expressando tempo médio em relação às outras ferramentas utilizadas.

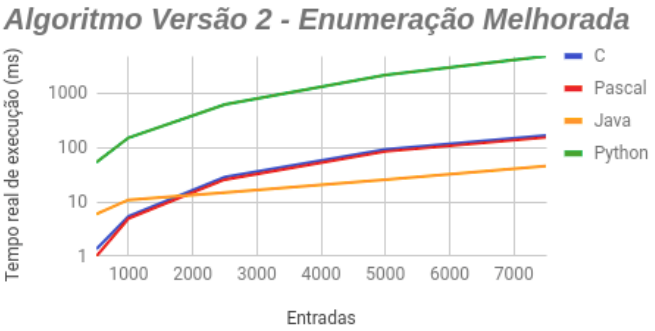

Figura 2 – Comparação das linguagens no algoritmo de Versão 2.

A Figura 2 representa o algoritmo de Enumeração Melhorada. Neste, é visível comportamento próximo ao da Figura 1, quando empregadas entradas numerosas: Python apresentando pior desempenho, Java o melhor, C e Pascal desempenho mediano. Entretanto, é perceptível que nos testes com os menores arquivos, até aproximadamente 1800 entradas, as linguagens C e Pascal se destacam e manifestam um ótimo desempenho.

Figura 3 – Comparação das linguagens no algoritmo de Versão 3.

O terceiro algoritmo, Divisão e Conquista, está com seus resultados representados pela Figura 3. Nesta, é notável o distinto comportamento das linguagens para entradas pequenas, mas, a medida que estas crescem, os resultados tendem a se parelhar de maneira geral.

Para arquivos pequenos e médios, Pascal situa-se com os melhores resultados, os quais estão relativamente distantes das outras linguagens. As outras 3 implementações não possuem distanciamento de valores tão abruptos, embora Java principie com os piores resultados.

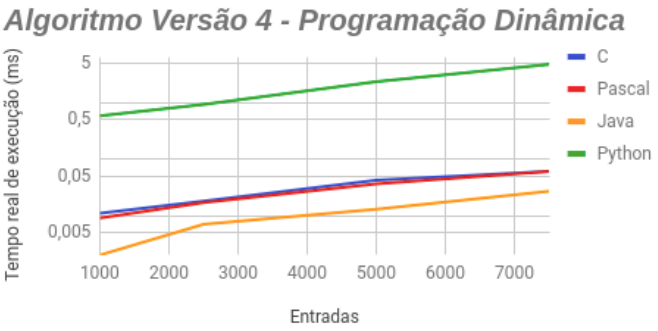

Figura 4 – Comparação das linguagens no algoritmo de Versão 4.

Tendo a Figura 4 como reprodução dos resultados obtidos através da execução do 4º algoritmo, pautado por programação dinâmica, foram obtidos resutados bastante satisfatórios. Assim, devido a complexidade do algoritmo ser T(n) = O(n), mesmo em entradas maiores o tempo se manteve baixo.

Python foi a linguagem com pior desempenho, gastanto 5 milissegundos para a maior entrada, enquanto Java, com o melhor desempenho, levou em média 0,03ms. C e Pascal mantiveram resultados parecidos e com pouca diferença do melhor obtido.

7 Conclusões

Após a analise dos resultados, pode-se concluir que a 1ª versão do algoritmo de SubVetor Máximo é muito custosa e quase inviável para entradas maiores. O pior desempenho nesta versão foi encontrado na implementação em Python, que para uma entrada de 7500 dados levou algo em torno de 2 horas para a execução. O melhor desemenho vem a cargo do Java, que demorou aproximadamente 45 segundos para o processamento da mesma entrada.

A 2ª versão, apesar de melhor que a primeira, também apresenta tempos de execução altos. O padrão das linguagens da primeira versão se mantém, mas é notável que para arquivos de até aproximadamente 1800 entradas, C e Pascal demonstraram melhor comportamento.

A implementação com base em Divisão e Conquista foi a que mais aproximou os resultado das 4 linguagens. Para entradas grandes, todas apresentaram tempo de execução parecido, em torno de 50 milissegundos. Considerando arquivos de até algo próximo a 5000 entradas, Pascal se destaca com os menores tempos.

A ultima versão é a com melhor complexidade (O(n)), rendendo ótimos resultados mesmo para entradas robustas, em todas as linguagens. Entretanto, Python obteve o pior resultado, chegando a apresentar índice de piora de quase 100 vezes para a maior entrada, quando comparado aos resultados medianos, obtidos por C e Pascal.

Se tratando dos algoritmos, conforme foram mudando as versões, sequencialmente, o tempo real de execução foi decrescendo, mas é discrepante a melhora no tempo quando comparada a 1ª com a última versão, independente da linguagem avaliada.

A respeito das linguagens, de maneira geral, Python se portou com os piores resultados, enquanto Java com os melhores. C e Pascal apresentaram valores bastantes semelhantes e, por várias vezes, medianos em relação as outras duas linguagens escolhidas.

8 Referências

BORRADAILE, G. *Test your knowledge: Solve the max subarray problem 4 ways.* Corvallis, EUA: Oregon State University, 20–. Disponível em: <https://web.engr-.oregonstate.edu/~glencora/wiki/uploads/max-subarray-project.pdf>. Acesso em: 23.11.2017. Citado 2 vezes nas páginas 4 e 7.

CASTILHO, M. e. a. *Guia rápido de referência da linguagem Pascal: Versão Free Pascal.* Universidade Federal do Paraná, 2009. Disponível em: <http://www-.inf.ufpr.br/cursos/ci055/pascal.pdf>. Acesso em: 25.11.2017. Citado na página 6.

CLARO DANIELA BARREIRO; SOBRAL, J. B. M. *Programação em JAVA*. Florianópolis: Copyleft Pearson Education, 2008. Disponível em: <http://www-.carlosmatos.com.br/images/noticias/1157/arquivo.pdf>. Acesso em: 25.11.2017. Citado na página 6.

FABRO, J. A. *Análise Empírica de Algoritmos*. Curitiba: Universidade Tecnológica Federal do Paraná, 2009. Disponível em: <http://www.dainf.ct.utfpr.edu.br/~fabro-/IF64C/Analise_Empirica_de_AlgoritmosOK.pdf>. Acesso em: 23.11.2017. Citado na página 5.

FOLEISS, J. H. *Análise de Algoritmos – Aula 1*. Campo Mourão: Universidade Tecnológica Federal do Paraná, 20–. Citado na página 4.

FOLEISS, J. H. *Análise de Algoritmos – Aula 10*. Campo Mourão: Universidade Tecnológica Federal do Paraná, 20–. Citado na página 5.

FONSECA IGUATEMI EDUARDO DA; ALEXANDRE, E. d. S. M. *Linguagem de Programação I: Programação Estruturada usando C*. João Pessoa: Editora da Universidade Federal da Paraíba, 2014. Disponível em: <http://producao.virtual.ufpb-.br/books/edusantana/linguagem-de-programacao-i-livro/livro/livro.pdf>. Acesso em: 25.11.2017. Citado na página 5.

LABAKI, J. *Introdução a Python: Módulo A*. Ilha Solteira: Grupo Python Universidade Estadual Paulista, 20–. Disponível em: <http://www.dcc.ufrj.br/~fabiom/python-/pythonbasico.pdf>. Acesso em: 25.11.2017. Citado na página 6.