TEXTO COMPLEMENTAR

Disciplina:	**Paradigmas de Linguagens**
Professor:	**Luiz Roberto Forçan**

Análise Comparativa de Linguagens de Programação a partir de Problemas Clássicos da Computação

Rodrigo Duarte Seabra[1], Isabela Neves Drummond[2], Fernando Coelho Gomes[3]

Universidade Federal de Itajubá, Minas Gerais – Brasil[1]

rodrigo@unifei.edu.br, isadrummond@unifei.edu.br, fcgomes.92@gmail.com

Resumo. *Nos estudos computacionais envolvendo as linguagens de programação, muitose discute sobre qual é a melhor opção para se utilizar em um projeto específico ou qual linguagem possui o melhor desempenho em certa aplicação. O trabalho proposto busca contribuir nesse sentido, apresentando uma análise qualitativa e quantitativa das linguagens Python, Java e C utilizando os parâmetros de tempo de execução, a quantidade de linhas de código, o tamanho do arquivo, o tempo de produção, a quantidade de acessos à documentação e o uso de funções nativas. Foram implementados seis problemas considerados clássicos na área de computação e, a partir dos resultados alcançados, foram realizadas as análises comparativas que possibilitaram a obtenção de algumas conclusões sobre o uso de cada linguagem selecionada para o estudo. O principal resultado obtido mostrou que a linguagem C possui o melhor desempenho entre as três linguagens analisadas, enquanto Python apresentou o pior resultado de tempo de execução, mas se destacou, positivamente,nos demais aspectos analisados. Finalmente, a linguagem Java não apresentou grandes resultados positivos ou negativos. Também foi possível observar que o parâmetro de análise de tempo de produção, em que se utilizou da técnica Pomodoro, não apresentou resultados concretos para a análise.*

1. Introdução

Considerando a diversidade de linguagens de programação, atualmente, disponíveis para o uso, um profissional recém-ingresso no mercado de trabalho ou, até mesmo, que esteja há mais tempo e deseja se atualizar, pode se deparar com grande impasse: qual linguagem de programação deve ser utilizada ou aprendida, agora? Mesmo conhecendo o objetivo e o paradigma de cada linguagem considerada como candidata ao uso, existe grande dúvida em como se especializar e em qual delas se aprofundar. Na internet, podem ser encontrados artigos que apresentam as linguagens que estão sendo mais utilizadas, ou qual delas escolher quando se é novo neste cenário; mas são raros os artigos que fazem uma análise mais detalhada dos problemas que uma linguagem pode apresentar durante o seu uso e como esta pode se comportar perante algumas situações que lhes são apresentadas. Além disso, aqueles que desenvolvem as suas aplicações em certa linguagem por muito tempo, normalmente, se identificam tanto com ela que se estabelece uma “zona de conforto”, acreditando-se, fielmente, que esta linguagem será a “bala de prata” (SOMMERVILLE, 2007) na computação.

Para facilitar a seleção de qual linguagem de programação deve-se começar a aprender ou se aprofundar, é necessário ter conhecimento se ela é compatível com os projetos e as situações nas quais será empregada, e se as soluções para os problemas que o programador

possui interesse em resolver podem ser alcançadas por tal linguagem. Uma análise que envolva variadas áreas de problemas da computação e utilize os parâmetros tais como a facilidade de codificação do problema, o tempo de execução da linguagem, a velocidade de solução do problema e o tamanho de código gerado são informações relevantes para um profissional ao selecionar determinada linguagem de programação. Ademais, a divisão por paradigmas, seguindo a classificação apresentada por Sebesta (2012), contribui nas decisões de modelagem da solução.

Com base no exposto, essa pesquisa tem como objetivo analisar, pelo método AHP (*Analytic Hierarchy Process*) e pelas métricas propostas no estudo, códigos implementados nas linguagens de programação C, Java e Python. Os algoritmos produzidos nessas linguagens buscam solucionar os problemas clássicos computacionais. A escolha de diferentes paradigmas de programação para a implementação dos algoritmos clássicos é o diferencial desta pesquisa, pois os variados problemas são comparados em função das métricas estabelecidas. Mas, ainda, as situações selecionadas para os fins de implementação se constituem em problemas, geralmente, utilizados como exemplos em livros didáticos explorados em cursos ou aulas de programação. Muitos dos problemas apresentados versam sobre as abordagens matemáticas devido aos fundamentos da computação. Pode-se observar, por exemplo, que o estudo de algoritmos, inicialmente, era realizado por matemáticos; assim, a Ciência da Computação.

Pode ser identificada como uma parte dos estudos matemáticos. Mas também, é passível de argumentação que o estudo de algoritmos faz parte da computação, e estes são estudados e utilizados por matemáticos. A partir dessas duas afirmações e considerando o Teorema de Cantor-Bernstein-Schroeder ambas as áreas poderiam ser consideradas equipolentes, segundo Knuth (1974). Não é possível afirmar que, ao iniciar o estudo de uma nova linguagem de programação, todo profissional direcionará a sua atenção às situações que envolvam as implementações matemáticas. No entanto, dado o histórico da computação, é possível encontrar os exemplos matemáticos sendo utilizados em várias explicações computacionais.

Após a obtenção das soluções, a análise foi realizada por meio de uma comparação entre as três linguagens selecionadas para o estudo, utilizando os seguintes parâmetros: (I) facilidade de codificação do problema; (II) tamanho de código gerado; (III) tempo de execução do algoritmo; e (IV) tamanho do arquivo que armazena o programa. Os dados que compõem o insumo para a análise foram obtidos durante a execução dos códigos de cada linguagem e serão apresentados e discutidos no decorrer do trabalho.

Apesar de a seleção da linguagem a ser utilizada em um projeto depender de variados fatores, este trabalho se propõe a realizar uma análise, não definitiva, de alguns aspectos subjetivos das linguagens tratadas no estudo com base em uma proposta de metodologia de análise. Assim, procurou-se apresentar uma metodologia diferenciada para a análise de linguagens por meio de algoritmos comuns a engenheiros e cientistas. O foco do estudo não se restringe aos *benchmarks*, tradicionalmente, disponíveis, já que estes se baseiam em consumo de recursos computacionais. Este trabalho tenta abordar o consumo de recursos humanos ao utilizar as métricas mais subjetivas, tais como: tempo de produção, uso de funções nativas, dentre outras. Desse modo, a comparação com outros *benchmarks* de tempo e consumo de recursos não se adequa a todas as métricas abordadas na análise proposta. Ademais, para que esta comparação possa ser realizada seria necessário produzir os códigos para analisar algumas métricas; todavia, essas informações não foram encontradas em outros *benchmarks*. O uso do AHP e de métricas fora do escopo puramente computacional foram as propostas principais empregadas no estudo, ao tentar diminuir a subjetividade para uma análise quantitativa.

1.1 Seleção dos problemas

Visando uma seleção imparcial dos problemas a serem analisados, foi realizada uma investigação com diversos professores da área de computação, e uma busca em variados aspectos da literatura de computação. Foi decidido que a opinião de outros estudiosos daárea seria de grande valia, assim como a busca de algoritmos capazes de serem aplicados na solução de problemas práticos, sendo selecionados os seguintes problemas: Problema 1 – Cálculo de número fatorial de forma iterativa e recursiva; Problema 2 – Cálculo de números primos; Problema 3 – Multiplicação de matrizes; Problema 4 – Fatoração de um número em números primos; Problema 5 – O caixeiro-viajante; Problema 6 – A mochila.

- Problema 1: foram implementadas duas formas de solução do problema para a comparação do desempenho de cada linguagem quanto ao tratamento de cada tipo de codificação;

- Problema 2: os códigos produzidos têm como foco descobrir os primeiros 1000 números primos. Após a obtenção dos números, é realizada uma validação do resultado, utilizando uma listagem já conhecida de números primos;

- Problema 3: o código envolvido no Problema 3 recebe as matrizes a serem multiplicadas por um arquivo de texto, previamente, gerado, e após a leitura e a inicialização de cada parâmetro a multiplicação é realizada. Foram escolhidas matrizes quadráticas de ordem 30 e, para evitar que os números fossem escolhidos pelos autores da pesquisa, cada elemento da matriz foi determinado por um programa auxiliar que tem como objetivo gerar números aleatóriosdentro da faixa -1000 a 1000;

- Problema 4: utiliza uma lista de números primos, previamente, definidos e, após aleitura de todos os valores, o programa realiza as multiplicações entre os elementos dalista para verificar se algum deles pode ser validado como a fatoração de um número em números primos. O valor a ser fatorado é, previamente, definido e proveniente da multiplicação de elementos primos aleatórios;

- Problema 5: o código envolvido lê os nomes das cidades e uma matriz de conexões entre elas. A partir das distâncias entre cada cidade, representadas em uma matriz, são realizados os cálculos necessários para que um caminho que passe por todos os nós seja apresentado;

- Problema 6: a implementação definida, nesta pesquisa, foi a mochila binária, em virtude da complexidade e da variedade dos algoritmos da mochila fracionária. O algoritmo produzido realiza uma verificação binária dos elementos que podem estar na mochila e define as combinações que podem ser realizadas, já que cada elemento a ser inserido na mochila possui uma característica própria. Para definir a entrada na mochila são analisados o peso e o valor de cada possível objeto.

2. Método

A fim de manter a máxima precisão possível nas medições, principalmente, na medição de produtividade de codificação de cada um dos algoritmos, foi criado um roteiro que dividiu o processo de codificação em três etapas: **estudo dirigido**, **execução de validações** e **implementação**. As três atividades foram realizadas durante o período de uma semana para

cada um dos problemas propostos. O tempo dedicado a cada conjunto de três algoritmos foi determinado com o propósito de uma análise técnica mais aprofundada dos autores da pesquisa sobre cada um dos problemas, para que, assim, durante o período de implementação, fosse possível a apresentação de uma solução com maior embasamento e consolidação.

Durante a tarefa de **estudo dirigido**, foram realizadas diversas pesquisas sobre o problema, focando, sempre, em formas de implementação e métodos de aplicação do problema em questão. Os resultados obtidos foram utilizados para realizar o embasamento do código a ser validado, e da forma de solução a ser apresentada nas outras linguagens. O contato com as implementações realizadas nas linguagens propostas por esta pesquisa foi evitado para prevenir as possíveis influências. Foram analisadas, principalmente, as soluções teóricas, ou seja, as propostas de estruturas de solução para os problemas.

A **validação** foi utilizada como o método de auxiliar o estudo dirigido. Como foram encontradas diversas soluções, foi necessária a escolha de uma forma de implementação. Utilizando a linguagem de programação PHP, a qual os autores têm conhecimento básico, foram feitos testes de estruturas de implementação a serem utilizadas. Esta validação foi realizada em uma linguagem de programação diferente das propostas no estudo para evitar, também, uma possível influência no processo de implementação.

Para o processo de **implementação** do problema foram seguidos alguns passos sempre que se dava início à implementação em uma das linguagens. Os passos seguidos visavam a maior eficiência e o menor número de interrupções possíveis durante esta tarefa. Nesse contexto, os passos seguidos foram: (I) iniciar todos os programas necessários para começar o desenvolvimento. No caso das três linguagens, foram utilizados o SublimeText3 e o GNOME terminal; (II) preparação da documentação da linguagem que seria utilizada naquele momento; (III) preparação do ambiente de implementação. Em todos os casos, o ambiente consistiu em sua mesa de trabalho; (IV) preparação do contador Pomodoro[3] utilizando um aplicativo simples desenvolvido pelos autores (https://goo.gl/Alkatn) para o controle do tempo decorrido na implementação; (V) preparação de um gravador de áudio utilizando o aplicativo Voice Recorder. Foi realizada uma gravação do período de implementação para que quaisquer comentários relevantes pudessem ser transcritos no processo de análise; (VI) início da implementação.

A implementação de cada problema seguiu, sempre, a mesma ordem de linguagens, começando a tarefa com C, seguindo em Java e finalizando com Python. Além disso, foram efetuadas no mesmo dia, com um intervalo de uma hora ao final de cada implementação. É importante salientar que, durante o processo de implementação, outras métricas, além do **tempo de produção**, foram coletadas: (I) a métrica de **uso de funções nativas**, obtida por meio das notas de áudio do programador; (II) o **acesso à documentação**: essa aquisição de dados se tornou possível em função das gravações realizadas durante a seção de codificação do algoritmo, já que, a cada acesso da documentação, foi realizada uma notação em áudio para marcar este evento. Ainda no ambiente de desenvolvimento, foram coletados os seguintes dados: (I) **quantidade de linhas**: após a finalização de todos os problemas foram abertos os arquivos fonte e anotadas as quantidades de linhas demarcadas pelo SublimeText3; (II) **tamanho do arquivo**: utilizando o gerenciador de arquivos Nemo, tomou-se nota da quantidade de espaço ocupado por cada arquivo fonte.

Para realizar a coleta dos dados de tempo de execução necessários para a análise, foi gerada uma máquina virtual, utilizando o *software* VirtualBox, e preparado um ambiente de execução utilizando o Sistema Operacional (SO) Arch Linux. Esta distribuição Linux foi selecionada pela familiaridade dos autores e por se tratar de uma versão mais limpa, quando comparada ao quesito de *software* pré-instalados. A versão do Arch Linux utilizada foi a 2015.08.01. Após a configuração básica do sistema, foram executados todos os algoritmos, previamente, gerados, e coletados os dados. Para a aquisição do tempo de execução, utilizou-se a função *time* do sistema operacional. A escolha de um sistema diferente do utilizado

diariamente pelo autor foi realizada como uma tentativa de ampliar a estabilidade entre as execuções. Almejando atingir um resultado mais significativo no período de execução dos algoritmos, estes foram executados repetidamente 10, 100 e 1000 vezes, sendo contabilizado o tempo médio total da execução, utilizando um *script* desenvolvido pelos autores, em cada conjunto de repetições. A diferença de tempo se torna mais clara quando abordada desta forma. Apesar de demonstrado o resultado de múltiplas execuções, este resultado não foi contabilizado, juntamente do *framework* de análise, sendo apresentado, somente, para fins de melhoria da percepção dos resultados.

3. Análise

O problema de **cálculo de números fatoriais** foi dividido em iterativo e recursivo para as execuções; todavia, no período de produção do algoritmo, foi considerado como um único problema. A separação para a execução teve como objetivo a verificação da diferença de tempo entre a versão recursiva e iterativa de um algoritmo muito comum naárea da computação. Durante a produção do código, não houve acesso à documentação de nenhuma das linguagens. O número escolhido como teste de execução das implementações foi 20. Tal valor foi escolhido por limitações da simplicidade do algoritmo. O maior valor armazenado por uma variável em C, um *unsigned long long*, é extrapolado a partir do valor do fatorial de 21. Assim como em Java, este valor limite é atingido após o fatorial de 70. Essas características não são consideradas como falhas dalinguagem, mas, sim, limitações da sua estrutura básica. No comparativo de linguagens, comoPython está no nível mais alto, a estrutura de *overflow* de uma variável é tratada automaticamente, enquanto em C ou Java, deve-se preparar o código para estas eventualidades. Com isso, o limite máximo de uma estrutura simples em C foi escolhidopara manter o código similar e para demonstrar como cada uma das linguagens trataseus tipos básicos. Para o **cálculo de números primos**, durante a produção do código, não houve acesso à documentação de nenhuma das linguagens. Como parâmetro de execução e teste deste problema, os primeiros 1000 números primos foram calculados. No caso do problema de **multiplicação de matrizes**, durante a produção do código, foi realizado um acesso à documentação da linguagem C. Para os testes executados, foi gerada uma matriz quadrada de ordem 30 com valores aleatórios de -1000 a 1000. A matriz escolhida foi transcrita em um arquivo texto para que os mesmos dados fossem inseridos de forma mais simples para todos os algoritmos.

Para o problema de **fatoração em números primos**, durante a produção do código, não foi realizado qualquer acesso à documentação das linguagens. A lista contendo os 1000 primeiros números primos e o número 33770797 foram utilizados como dados de entrada para o algoritmo de fatoração em números primos. Este valor foiescolhido por representar a fatoração de dois números primos contidos na listagem. O método de escolha de quais valores da lista seriam utilizados foi baseado em um métodoaleatório, no qual foram gerados dois valores entre 1 e 1000, e, assim, selecionado o valor da lista referente ao resultado da metodologia de escolha. Durante a produção do código do problema do **caixeiro-viajante**, não foi realizado qualquer acesso à documentação das linguagens. Para a aplicação dos testes, foi gerado um arquivo contendo 15 cidades, nomeadas de A a U, e uma matriz de distâncias entre as cidades. Aforma de entrada de dados matricial foi escolhida devido à solução desenvolvida e proposta. Para o último problema, **mochila booleana**, durante a produção do código,não foi realizado qualquer acesso à documentação das linguagens. Diferentemente dos algoritmos, previamente, apresentados, o problema da mochila tem os seus valores de teste nativamente dentro do código. Sendo assim, foi selecionado um peso total para amochila e gerados dois vetores, um referente ao peso e o outro relativo ao valor de cada item. Foi escolhida uma capacidade total de 400 para a mochila, e 22 itens com pesos e valores distintos.

A Tabela 2 apresenta os resultados numéricos obtidos para os parâmetros de tamanho

de código, tamanho de arquivo e tempo de produção para cada um dos problemas e cada uma das linguagens.

Tabela 2. Resultados gerais para cada um dos problemas e linguagens

Problema	Linguagem	Tamanho de código (linhas)	Tamanho do arquivo (kB)	Tempo de produção (ciclos)
Cálculo de números fatoriais	C	22	7,2 + 0,445 kB	0,5 ciclo
	Java	20	0,467 + 0,732 kB	0,5 ciclo
	Python	20	0,72 + 0,372 kB	0,5 ciclo
Cálculo de números primos	C	31	7,3 + 0,647 kB	0,5 ciclo
	Java	30	1 + 0,708 kB	0,5 ciclo
	Python	26	0,64 + 0,483 kB	0,5 ciclo
Multiplicação de matrizes	C	54	9,4 + 1,1 kB	0,5 ciclo
	Java	58	1,5 + 2,2 kB	0,5 ciclo
	Python	52	1,1 + 1,2 kB	0,5 ciclo
Fatoração em números primos	C	30	7,7 + 0,597 kB	0,5 ciclo
	Java	37	0,866 + 1,7 kB	0,5 ciclo
	Python	30	0,549 + 0,773 kB	0,5 ciclo
Mochila booleana	C	53	1,1 + 8,4 kB	1,5 ciclo
	Java	54	1,2 + 1,6 kB	1,5 ciclo
	Python	40	0,938 + 2 kB	1,5 ciclo
Caixeiro-viajante	C	79	9,6 + 1,8 kB	1 ciclo
	Java	81	2,1 + 2,4 kB	1 ciclo
	Python	58	1,5 + 1,4 kB	1 ciclo

Para que a análise entre as linguagens seja realizada, é necessário que alguns valores referentes à matriz comparativa do AHP sejam calculados. A Tabela 3 mostra tanto os valores obtidos na matriz comparativa normalizada quanto à matriz comparativa original. Em seguida, a Tabela 4 expõe os valores referentes ao vetor de Eigen; ou seja, o valor de influência de cada um dos parâmetros de análise. Também é importante observar que o coeficiente de consistência (CR) obtido ficou abaixo de 10%; assim, segundo a teoria AHP, a tabela comparativa está apta para o uso. O valor de Eigen apresentado na Tabela 4 se relaciona, diretamente, com a Tabela 3, dado que os valores referentes a cada um dos problemas são calculados utilizando os valores normalizados. Por fim, o coeficiente de consistência, obtido por meio dos valores de Eigen, demonstra o balanceamento da Tabela 4. Este coeficiente valida os pesos escolhidos na Tabela 3.

Tabela 3. Normalização da tabela comparativa

	Tempo de execução	Tamanho de código	Tamanho de arquivo	Tempo de produção	Uso de funções nativas	Acesso à documentação
Tempo de execução	1	9	6	2	6	6
Tamanho de código	0,11	1	0,17	0,11	0,2	0,2
Tamanho de arquivo	0,17	6	1	0,2	4	4
Tempo de produção	0,5	9	5	1	6	6
Uso de funções nativas	0,17	5	0,25	0,17	1	1
Acesso à documentação	0,17	5	0,25	0,17	1	1
Total	2,11	35	12,67	3,64	18,2	18,2
Tempo de execução	0,47	0,26	0,47	0,55	0,33	0,33
Tamanho de código	0,05	0,03	0,01	0,03	0,01	0,01
Tamanho de arquivo	0,08	0,17	0,08	0,05	0,22	0,22
Tempo de produção	0,24	0,26	0,39	0,27	0,33	0,33
Uso de funções nativas	0,08	0,14	0,02	0,05	0,05	0,05
Acesso à documentação	0,08	0,14	0,02	0,05	0,05	0,05

Tabela 4. Valores de Eigen por comparativo e valor CR

Comparativos	Vetor de Eigen
Tempo de execução	40,21%
Tamanho de código	2,45%
Tamanho de arquivo	13,73%
Tempo de produção	30,37%
Uso de funções nativas	6,62%
Acesso à documentação	6,62%
Total	100%
CR (< 10%)	9,5%

Na comparação entre os dados de tempo de execução, é, facilmente, observado que a linguagem C obteve grande vantagem e resultados muito mais eficientes quando comparada a Python e Java. Não, somente, o tempo total de execução, mas como cadaum dos tempos individuais são bem menores quando executados em C. Ao comparar Java e Python, individualmente, tem-se que a primeira linguagem somente se sobressai nos problemas que envolvem os números primos e, apesar disso, o tempo final das execuções em Java é menor do que em Python. Analisando os resultados, juntamente, do método de implementação, a diferença entre os tempos pode ser ocasionada tanto pelo nível da linguagem quanto pela forma escolhida para implementar os problemas. O paradigma para qual a linguagem foi desenvolvida pode, muitas vezes, influenciar na forma em que o código é executado e compilado. Os tempos de execução de cada problema em cada uma das linguagens são apresentados na Tabela 5.

Tabela 5 – Resultado do tempo de execução de cada linguagem

TEMPO DE EXECUÇÃO								
	PROBLEMA							TOTAL
LINGUAGEM	P1 ITER	P1 REC	P2	P3	P4	P5	P6	
C	0,001	0,001	0,04	0,002	0,007	0,001	0,001	0,053
JAVA	0,074	0,06	0,132	0,1	0,081	0,068	0,067	0,582
PYTHON	0,049	0,05	0,457	0,049	0,153	0,032	0,026	0,816

Ao analisar a quantidade de linhas que cada implementação obteve, observou-se que, em ambos os casos, na comparação total e parcial, Python se sobressaiu. Assim como os resultados do tempo de execução, o paradigma sobre o qual a linguagem foi desenvolvida apresenta uma influência na quantidade de código necessária para executar uma ação. Um exemplo é Python, que não necessita de chaves para as funções, os métodos, os blocos condicionais ou os blocos de repetição, fazendo com que algumas linhas sejameconomizadas. As quantidades de linhas de cada problema, em cada uma das linguagens, são apresentadas na Tabela 6.

Tabela 6. Resultado do tamanho de código de cada linguagem

TAMANHO DE CÓDIGO (LINHAS)							
	PROBLEMA						TOTAL
LINGUAGEM	P1	P2	P3	P4	P5	P6	
C	22	31	54	30	53	79	269
JAVA	20	30	58	37	54	81	280
PYTHON	20	26	52	30	40	58	226

O tamanho de arquivo resultante, a soma entre o código fonte e a versão compilada, apresentou resultados nos quais os algoritmos em Python, com a exceção do problema da mochila booleana,

apresentaram resultado muito melhor. No geral, a diferença entre Python e Java não foi elevada, ao contrário da linguagem C, que se destoou das outras.O espaço ocupado pela versão compilada foi o maior diferencial entre as linguagens, dado que este fator depende do compilador. Os arquivos fonte não apresentaram muita diferença, visto que podem ser considerados como, somente, arquivos de texto normais. Os tamanhos de arquivo de cada problema, em cada uma das linguagens, são apresentados naTabela 7. Para o tempo de produção, obtiveram-se algumas conclusões diferenciadas. Todos os resultados foram, aproximadamente, iguais quando analisados no quesito temporeal. O método utilizado foi o Pomodoro, que consiste em um conjunto de regras e técnicas para que seja ampliada a produtividade e a qualidade do trabalho, por meio domelhor gerenciamento de foco e tempo. As regras são simples e objetivas, de fácil aplicação, além de apresentar resultados palpáveis de melhoria em projetos de *software*, como discutido no estudo de Patrício, Macedo e França (2011). Para a aplicação do método, é necessário manter o foco, sem qualquer distração, em um trabalho a ser executado durante um ciclo de 25 minutos. Após um ciclo, deve-se interromper o trabalho por cinco minutos e, somente, descansar. Ao concluir um conjunto de quatro ciclos de trabalho é recomendado um tempo de 30 minutos de *long break* para o descanso. Segundo os apoiadores do projeto, ao seguir este procedimento de execução de tarefas, há uma melhoria no foco, na agilidade de pensamento e, até mesmo, na saúde física.

Tabela 7. Resultado do tamanho de arquivo de cada linguagem

TAMANHO DE ARQUIVO (kB)							
LINGUAGEM	PROBLEMA						TOTAL
	P1	P2	P3	P4	P5	P6	
C	7,65	7,95	10,5	8,3	9,5	11,4	55,29
JAVA	1,2	1,71	3,7	2,57	2,8	4,5	16,47
PYTHON	1,09	1,12	2,3	1,32	2,94	2,9	11,68

Ao utilizar a métrica e a técnica Pomodoro, foi possível perceber que os tempos se igualaram. Isso pode ser justificado por alguns fatores: (I) o estudo prévio de cada programa com uma definição de estrutura pode ter ocasionado uma normalização do tempo de produção; (II) os problemas selecionados para o desenvolvimento podem não ser,totalmente, adequados para a métrica Pomodoro; (III) a métrica utilizada, a técnicaPomodoro, por trabalhar em ciclos de 30 minutos, pode não ser a técnica mais precisa e adequada para aplicar nesta situação. Os tempos de produção de cada problema, em cada uma das linguagens, são apresentados na Tabela 8.

Tabela 8. Resultado do tempo de produção de cada linguagem

TEMPO DE PRODUÇÃO (CICLOS)							
LINGUAGEM	PROBLEMA						TOTAL
	P1	P2	P3	P4	P5	P6	
C	0,5	0,5	0,5	0,5	1,5	1	4,5
JAVA	0,5	0,5	0,5	0,5	1,5	1	4,5
PYTHON	0,5	0,5	0,5	0,5	1,5	1	4,5

O uso de funções nativas também foi influenciado pelo paradigma e pelo nívelda linguagem implementada. Pode-se observar que Python fez uso do maior número de funções prontas. A linguagem C ficou em segundo lugar, neste quesito, pois a leitura de arquivos em Java se utiliza de conceitos do paradigma orientado aos objetos. Mas é interessante observar, também, o momento de uso das funções nativas. Um exemplo é a linguagem Python, que utiliza o método *range* para gerar um vetor de valores sequenciais e, assim, iterar entre esses valores

para gerar um *looping*. Os números de chamadas às funções nativas de cada problema, em cada uma das linguagens, são apresentados na Tabela 9. A última métrica, o acesso à documentação, foi a mais difícil de mensurar, já que esta depende da memória do programador e de sua aptidão com a linguagem. O resultado apresentado, somente, um acesso à documentação da linguagem C durante a produção do problema de multiplicação de matrizes, ocorreu, pois, o método de leitura de caracteres de um arquivo não é tão trivial como em outras linguagens. Mesmo não realizando tantos acessos, alguns comentários sobre cada forma de acesso à documentação são válidos: (I) documentação C, utilizando um sistema Linux os acessosàs funções e às bibliotecas se tornam mais fáceis por meio da linha de comando; (II) documentação Java, é necessário conhecer o local de instalação da documentação *off-line*. Apesar disso, o acesso utilizando o *browser* se torna bem intuitivo; (III) documentação Python, a utilização do *shell* interativo torna o acesso à documentação simplificado como em C e, ainda, permite que os testes sejam realizados mais facilmente.

Tabela 9. Resultado final de chamadas de funções nativas de cada linguagem

	FUNÇÕES NATIVAS						
	PROBLEMA						TOTAL
LINGUAGEM	P1	P2	P3	P4	P5	P6	
C	1	1	5	5	2	5	19
JAVA	1	1	3	3	1	3	12
PYTHON	2	3	5	5	2	5	22

Uma conclusão geral dos resultados é apresentada na Tabela 10. É possível observar que C obteve melhor desempenho, somente, no tempo de execução, mas o valore o peso desta métrica fazem com que, dentre as análises realizadas, C tenha o melhor desempenho geral. Os resultados gerais de Python e Java obtiveram grande variedade. Enquanto Python se destacou nos demais parâmetros de análise, a linguagem obteve os problemas de tempo, fazendo com que a sua preferência fosse afetada. Apesar de Java, somente, apresentar pior desempenho na métrica relativa ao tamanho do código, os resultados da linguagem com relação ao tempo se aproximam mais dos resultadosobtidos em Python.

Tabela 10. Resultados por parâmetro de análise

	Prioridade	Melhor desempenho	Pior desempenho
Tempo de execução	40,21%	C	Python
Tamanho de código	2,45%	Python	Java
Tamanho de arquivo	13,73%	Python	C
Tempo de produção	30,37%	-	-
Uso de funções nativas	6,62%	Python	C
Acesso à documentação	6,62%	Python/Java	C

4. Considerações Finais

A busca por melhor desempenho em todos os aspectos de produção sempre foi um atributo muito presente no desenvolvimento da tecnologia. Os desenvolvedores e os apoiadores da tecnologia, sempre almejam os *hardware* e *software* que consigam atingir o máximo potencial disponível. Um aspecto claro, nesse ponto, é a comparação. Em todo ambiente em que se buscam melhorias, principalmente, relativas ao desempenho, faz-se necessário que os dados, as informações e os conhecimentos sejam analisados de forma concorrente. Na área de desenvolvimento de *software*, uma das comparações que mais tende a ser realizada é a

relacionada às linguagens de programação. Uma boa parte se deve ao custo de operação, ao desenvolvimento e à manutenção que certa linguagem pode exigir. Nesta pesquisa, os resultados obtidos para cada parâmetro analisado por linguagem demonstraram que apesar de não se destacar em nenhum dos outros parâmetros, somente, no tempo de execução, a linguagem C seria a melhor opção a se escolher. Apesar de esta linguagem apresentar o pior resultado em tamanho de arquivo,e por se tratar de uma linguagem mais complexa, o que exigiria mais acessos à documentação e pouco uso de métodos nativos, o seu desempenho seria de grande valia para um projeto. A linguagem Python, apesar de apresentar pior desempenho quando comparada às demais, seria uma boa escolha para as implementações nas quais se necessitade um resultado quase que imediato. A escolha do Java se daria mais por conta de experiência e aptidão do desenvolvedor, já que os seus resultados gerais não se demonstraram sobressair em nenhum dos aspectos. Além disso, é importante observar que as linguagens de mais alto nível, como Python, tendem a apresentar as estruturas básicas em um formato mais complexo. Um exemplo apontado neste trabalho foi o *overflow* de tipos, ou, até mesmo, as listas encadeadas que, em Python, representam os vetores. Também pode-se observar uma correlação entre o nível de abstração de cada linguagem, a eficiência computacional e o parâmetro de facilidade de codificação. Apesar de a análiseser um pouco imprecisa, por utilizar as medidas subjetivas, ainda, sim, é possível identificarque as linguagens de mais alto nível tendem a perder em desempenho por conta da abstração. Ou seja, quando se é realizada a decisão de abstrair um processo em uma linguagem de programação, pode-se resultar na queda de *performance* computacional, juntamente, do ganho no tempo de produção. É importante observar, também, que os aspectos de afinidade, capacidade e experiência do programador podem influenciar na análise. Esta influência se deve ao caráter da análise, alguns parâmetros subjetivos e, até mesmo, as diferenças entre os paradigmas utilizados. Apesar disso, foi realizada uma tentativa de quantificar alguns parâmetros qualitativos de forma que, utilizando o AHP, fosse possível a realização de uma decisão concreta quando se trata da seleção de linguagens de programação. Outro aspecto importante de ser observado é o fato de o consumo de memória volátil não ser considerado como parâmetro. O principal fator que levou a esta decisão, de não incluir o consumo de memória como métrica, se relaciona ao fato de que a complexidade dos problemas e as entradas escolhidas não proporcionaram os resultados conclusivos.

Como conclusão final desta pesquisa, considerando os resultados e as avaliações realizadas, o objetivo de definição de linguagens de programação para os projetos, apesar de, ainda, nebuloso, pode ser tratado de forma quantitativa e bem definido pelo AHP. Apesar de não se poder afirmar uma solução final para as decisões envolvendo as linguagens de programação, este trabalho pode contribuir para os futuros métodos dedecisão ou, até mesmo, como uma forma de aquisição de dados sobre as linguagens abordadas.