Universidade de Brasília
Instituto de Ciências Exatas
Departamento de Matemática
Programa de Mestrado Profissional
em Matemática em Rede Nacional

Avaliação de Algoritmos Ensinados no Ensino Fundamental e Médio Sob a Perspectiva da Análise de Complexidade

Bruno Macedo Alves

Brasília

2018

Bruno Macedo Alves

Avaliação de Algoritmos Ensinados no Ensino Fundamental e Médio Sob a Perspectiva da Análise de Complexidade

Dissertação apresentada ao Departamento de Matemática da Universidade de Brasília, como parte dos requisitos do "Programa" de Mestrado Profissional em Matemática em Rede Nacional - PROFMAT, para obtenção do grau de Mestre.

Universidade de Brasília - UnB

Departamento de Matemática - MAT

PROFMAT - SBM

Orientador: Prof. Dr. Edson Alves da Costa Júnior

Brasília

2018

Posição vertical

Bruno Macedo Alves
Avaliação de Algoritmos Ensinados no Ensino Fundamental e Médio Sob a Perspectiva da Análise de Complexidade/ Bruno Macedo Alves. – Brasília, 2018-
79 p. : il. (algumas color.) ; 30 cm.

Orientador: Prof. Dr. Edson Alves da Costa Júnior

Dissertação de Mestrado – Universidade de Brasília - UnB
Departamento de Matemática - MAT
PROFMAT - SBM, 2018.

1. Algoritmos. 2. Complexidade. I. Edson Alves da Costa Júnior. II. Universidade de Brasília. III. PROFMAT - SBM. IV. Avaliação de Algoritmos Ensinados nos Ensinos Fundamental e Médio Sob a Perspectiva da Análise de Complexidade

CDU XYZ 02:141:005.7

Universidade de Brasília
Instituto de Ciências Exatas
Departamento de Matemática

Avaliação de Algoritmos Ensinados no Ensino Fundamental e Médio Sob a Perspectiva da Análise de Complexidade

por

Bruno Macedo Alves

Dissertação apresentada ao Departamento de Matemática da Universidade de Brasília, como parte dos requisitos do "Programa" de Mestrado Profissional em Matemática em Rede Nacional - PROFMAT, para obtenção do grau de

MESTRE

Brasília, 11 de junho de 2018

Comissão Examinadora:

Prof. Dr. Edson Alves da Costa Júnior- FGA/UnB (Orientador)

Prof. Dr. Vinícius de Carvalho Ríspoli - FGA/UnB (Membro)

Prof. Dr. Fábio Macedo Mendes - FGA/UnB (Membro)

Dedico este trabalho a todos os professores de Matemática que, incansavelmente, esforçam-se para fazer com que seus alunos aprendam este incrível conhecimento, apesar de todas as dificuldades.

Agradecimentos

A Deus. Aos meus pais, por terem me ajudado a ser quem sou e por todo o amor dispendido comigo. À minha esposa Raíza, pela compreensão, suporte e carinho nas horas em que mais precisei. À minha irmã Carolina, por ter sido sempre uma referência para mim. Ao meu irmão Marcelo, por ser meu melhor amigo e por ouvir sempre as ideias que quis colocar neste trabalho. À minha família e amigos, por me darem apoio e ficarem sempre ao meu lado.

Agradecimentos especiais ao meu orientador, professor Edson, por todo o suporte dado durante o trabalho com dicas e com soluções geniais para problemas que, sozinho, eu não conseguiria resolver. Sem sua ajuda não teria terminado este trabalho. Agradeço também a meus amigos do MAT 2/2004 pelo encorajamento nestes dois árduos anos, especialmente aos amigos Henrique Zannata, Thiago Wiliams e Matheus Bernardini pelo tempo gasto me ajudando a resolver problemas relacionados a essa dissertação.

Não posso deixar de mencionar também meus colegas e amigos da turma do PROFMAT 2016, que me inspiraram com experiências, modos de agir e pensar, os quais fizeram-me crescer muito como professor e como estudante de matemática. Agradeço também a meus professores, os quais abriram minha mente para novas ideias e novos modos de pensar matemática.

Ao IMPA e à SBM pela iniciativa do PROFMAT. Este curso tem contribuído de forma significativa para a melhoria do ensino da matemática no Brasil.

Por fim, agradeço ao meu amigo Bruno Henrique por ter disponibilizado seu tempo e sua casa para me ajudar com os algoritmos contidos neste trabalho, ao Vinícius Rangel, pela disponibilidade para me ajudar com os bugs do sistema, ao professor Julwaity Quaresma, pela ajuda com as traduções para o inglês e ao Dhiego Loyola pela ajuda com o Latex.

"Não podemos decidir quando nascemos. Tudo o que podemos decidir é o que fazer com o tempo que nos é dado. "

J. R. R. Tolkien

Resumo

Este trabalho faz uma comparação entre algoritmos ensinados na escola para a potenciação, o máximo divisor comum e os determinantes, do ponto de vista da análise de complexidade. O trabalho busca determinar qual o algoritmo mais eficiente dentre aqueles comumente apresentados. A análise é feita primeiramente escrevendo os algoritmos em pseudocódigo, para em seguida descrever o comportamento da função que dá o tempo de execução do algoritmo em função do tamanho da entrada.

Palavras-chaves: Algoritmos. Complexidade.

Abstract

This work compares the algorithms teached at school to the exponentiation, the greatest comon divisor and the determinants, from the point of view of the complexity analysis. The work intends to determine which is the most efficient algorithm among those commonly presented. The analysis is done by firstly writing the algorithms in a pseudocode, then describing the behavior of the function that gives the time of execution of the algorithm based on the input size.

Key-words: Algorithms. Complexity.

Lista de ilustrações

Lista de tabelas

Sumário

Introdução

O aprendizado de matemática passa pela execução de algoritmos. Desde crianças, aprendemos algoritmos para somar e subtrair, por exemplo. Contudo, durante muito tempo, as pessoas tem sido ensinadas, de geração em geração, a executarem os mesmos algoritmos. Mas será que esses algoritmos são os mais eficientes, dentre aqueles que poderiam ser ensinados aos estudantes?

Estudando os algoritmos para efetuar uma potenciação com base e expoente naturais, para determinar o máximo divisor comum entre dois números naturais e para calcular determinantes de matrizes, busca-se avaliar, neste trabalho, a eficiência destes algoritmos. Será que os professores de matemática tem ensinado os melhores algoritmos que existem, do ponto de vista da análise de complexidade assintótica?

Por isso, acreditamos que há uma necessidade imediata de rever tais algoritmos e de propor sugestões de mudanças. Para que isto ocorra, entretanto, temos que fazer uma análise da complexidade assintótica desses algoritmos. Esta análise descreve o comportamento do tempo de execução do algoritmo em função da quantidade de operações realizadas à medida que o tamanho da entrada aumenta. A entrada varia conforme o problema: o expoente, no caso da potenciação, o tamanho dos números, no caso do máximo divisor comum e a dimensão da matriz, no caso do determinante.

Analisando a literatura disponível, encontramos o trabalho de Esquinca (2014). Neste trabalho, Esquinca analisa a complexidade de diversos algoritmos e os escreve em pseudocódigo. Contudo, seus objetivos são um pouco diferentes, pois são mais voltados para a apresentação dos algoritmos e o ensino de programação na escola básica. Nosso objetivo neste trabalho será o de comparar os algoritmos e verificar, por meios indiretos, quais os que são comumente adotados nas escolas.

O trabalho busca responder à seguinte pergunta de pesquisa: *Os algoritmos ensinados no ensino médio e fundamental para o cálculo de potenciações, máximo divisor comum e determinantes são computacionalmente eficientes?*

Objetivos

Este trabalho tem o objetivo de avaliar os algoritmos usualmente ensinados na escola para o cálculo de potenciações, máximo divisor comum e determinantes, sugerindo, ao final, que sejam adotados os métodos que forem computacionalmente mais eficiente.

Para isto, iremos fazer um estudo da análise de complexidade assintótica, consultando a literatura básica sobre o assunto. Depois, iremos aplicar os conceitos da análise de

complexidade assintótica no caso dos algoritmos de potenciações, máximo divisor comum e determinantes. Em seguida, faremos uma revisão dos livros didáticos comumente adotados no 6ª ano do ensino fundamental e no 2º ano do ensino médio para verificar quais algoritmos são apresentados nestes livros. Por fim, escreveremos os algoritmos em Python e, por meio de amostras aleatórias, confirmaremos os resultados obtidos na análise.

Estrutura do Trabalho

O trabalho está dividido em 4 capítulos. No Capítulo 1, faz-se uma exposição dos conceitos teóricos básicos necessários ao entendimento do trabalho. Serão abordados o conceito de algoritmo, de complexidade assintótica, bem como resultados básicos relacionados à potenciação, ao máximo divisor comum e aos determinantes.

O Capítulo 2 discute a metodologia utilizada para a escolha dos algoritmos e dos exemplos didáticos abordados ao longo do trabalho. Nele também são apresentados os algoritmos escolhidos. Após a sua apresentação, faz-se uma análise de complexidade baseada na contagem das operações e no tempo de execução. O tempo de execução é considerado constante para cada operação do mesmo tipo, o que simplifica a análise. Busca-se, portanto, determinar não uma função específica, mas sim uma classe de funções que represente o tempo de execução de cada algoritmo.

O Capítulo 3 cuida da aplicação dos algoritmos em exemplos de livros didáticos comumente adotados nos ensinos médio e fundamental e computacionais, com entradas aleatórias geradas em linguagem Python. Estes exemplos mostram a grande diferença que faz a escolha por um algoritmo eficiente. Segue-se uma análise das escolhas de algoritmos feitas no contexto escolar.

O Capítulo 4 apresenta a opinião do autor, respondendo a pergunta de pesquisa proposta na introdução, a qual defende a escolha pelos algoritmos mais eficientes.

1 Fundamentação Teórica

Neste capítulo iremos apresentar os conceitos relacionados a algoritmos. Faremos também uma breve exposição sobre o uso da notação assintótica no estudo do crescimento de funções e suas aplicações à análise da eficiência de algoritmos. Na Seção 1.1 estabeleceremos o conceito de algoritmo e apresentaremos a linguagem que será utilizada para a descrição dos mesmos. Na Seção 1.2 trataremos da análise de algoritmos e dos critérios fundamentais para esta análise. Um desenvolvimento sobre a notação assintótica e seu uso na análise de algoritmos é feito na Seção 1.3. O capítulo termina com uma revisão de conceitos e resultados relacionados à potenciação, ao máximo divisor comum e aos determinantes.

1.1 Algoritmos

Um algoritmo é qualquer procedimento computacional bem definido que toma algum valor ou conjunto de valores como entrada e produz algum valor ou conjunto de valores como saída. O valor ou conjunto de valores de entrada é denominado instância do problema. Por exemplo, um algoritmo que some dois inteiros transforma os valores de entrada $(25, 57)$ no valor de saída 82, que é o resultado da soma.

Dizemos que um algoritmo é correto se, para toda instância de entrada, ele produzir a saída correta. Logo, um algoritmo é incorreto se, para uma certa instância, não parar (um computador pode não concluir um algoritmo ou demorar mais tempo do que estamos dispostos a esperar) ou se produzir uma resposta incorreta (CORMEN et al., 2012).

Mesmo que um algoritmo esteja correto, é importante analisar o seu desempenho mediante determinados critérios de eficiência. O motivo para tal análise é que algoritmos utilizam espaço em memória para armazenar dados enquanto operam, e seus procedimentos levam um determinado intervalo de tempo para serem executados. Ora, a memória disponível em um computador não é infinita e, tampouco, gratuita. Além disso, não se pode esperar indefinidamente para que um algoritmo produza a resposta certa para uma instância de um problema. Daí a importância de se medir a eficiência de um algoritmo, eficiência esta, em geral, medida pelo seu tempo de execução. Dois algoritmos distintos, mesmo que sejam ambos corretos, podem levar tempos diferentes para resolver uma mesma instância de um problema.

Ao longo deste trabalho iremos utilizar um pseudocódigo semelhante a algumas linguagens de programação, como C, Python, Java ou Pascal. Um pseudocódigo é uma lista de comandos escrita em linguagem comum para descrever um algoritmo, mas que

incorpora algumas das principais funções de uma linguagem de programação padrão, como as citadas acima. Iremos utilizar as convenções usadas em (CORMEN et al., 2012):

(a) o recuo indica estrutura de bloco. Assim, as linhas de comando dentro de uma estrutura qualquer devem ter um recuo maior. Uma nova estrutura deve receber um recuo igual ao que iniciou a estrutura anterior;

(b) os laços `for`, `while`, `if-else`, `repeat-until` tem a mesma interpretação das linguagens de programação C, C++, Pascal, Java ou Python;

(c) o símbolo = tem o sentido de atribuição. Por exemplo, se escrevemos $i = i + 1$ significa que à variável i será atribuído o valor $i + 1$;

(d) elementos de um vetor serão acessados especificando-se o nome do vetor, seguido pelo índice entre colchetes. Por exemplo, $A[i]$ indica o i-ésimo elemento do vetor A;

(e) os atributos de um objeto serão indicados por pontos. Por exemplo, $A.n$ significa que o vetor A tem n elementos;

(f) se um dado valor de saída é nulo ou vazio, usaremos o valor especial NIL.

1.2 Análise de Algoritmos

Analisar um algoritmo significa prever os recursos de que o algoritmo necessita. Recursos como memória ou *hardware* podem fazer parte desta análise, mas frequentemente o tempo de computação é o principal parâmetro considerado. Neste trabalho, iremos utilizar um modelo de computação genérico de máquina com memória de acesso aleatório (*Random Access Machine* - RAM) com um único processador como tecnologia de implementação. Além disso, iremos considerar que cada instrução é executada em tempo constante.

A análise do tempo que um algoritmo leva para ser executado, em geral, depende do tamanho n da entrada. Para fazer esta análise, contamos o número de operações primitivas para uma dada entrada de tamanho n. Cada linha no código do algoritmo, por hipótese, leva um tempo constante para ser executada e o número de vezes em que a linha é executada é uma função de n. Somando-se o tempo de execução de cada passo, obtemos o tempo de execução total do algoritmo, o qual, consequentemente, também será uma função de n.

É fácil ver que o tempo de execução de um algoritmo também depende da instância, em particular, que está sendo processada. Por exemplo, um algoritmo que ordene um vetor de números pode levar um tempo menor para ser executado quando a entrada está parcialmente ordenada do que quando a entrada está completamente desordenada.

Deste modo, a análise de algoritmos contempla o pior caso, o melhor caso e o caso médio. O pior caso representa a situação na qual o tempo de execução é o maior possível, e o melhor caso a situação na qual o tempo de execução é o menor possível. Já o caso médio é definido como a média sobre o número de etapas executadas pelo algoritmo em cada entrada, ponderadas pela probabilidade de ocorrência de cada entrada (DROZDEK, 2002).

O estudo do pior caso é o mais importante por três motivos: em primeiro lugar, o tempo de execução do pior caso de um algoritmo estabelece um limite superior para o tempo de execução de qualquer instância. Conhecê-lo nos dá uma garantia de que o algoritmo nunca demorará mais do que esse tempo. Em segundo lugar, para alguns algoritmos, o pior caso ocorre com bastante frequência. Por exemplo, na pesquisa de um banco de dados em busca de determinada informação, o pior caso do algoritmo de busca ocorre quando a informação não está presente no banco de dados. Por fim, muitas vezes, o tempo de execução do caso médio tende a ser igual ao do pior caso.

1.3 Notação Assintótica

Para dar ênfase à ordem de crescimento do tempo de execução de um algoritmo em função do tamanho da entrada n, iremos estudar a eficiência assintótica dos algoritmos, ou seja, estaremos preocupados em como o tempo de execução aumenta quando tomamos o limite desta função com n tendendo a infinito.

Como vimos, o tempo de execução é uma função do tamanho da entrada n. Iremos considerar que tais funções tem como domínio o conjunto dos números naturais.

Definição 1. (Notação Θ) Dizemos que uma função f é $\Theta(g(n))$ se existirem constantes positivas c_1, c_2 e n_0 tais que $0 \leq c_1 g(n) \leq f(n) \leq c_2 g(n)$ para todo $n > n_0$.

Por exemplo, a função definida por $f(n) = 3n^2 - 2$ é $\Theta(n^2)$ pois, dados $c_1 = 4$ e $c_2 = 2$, tem-se que $2n^2 \leq f(n) \leq 4n^2$ para todo $n \geq n_0 = 2$. Neste caso, cometendo um abuso de notação, escrevemos $f(n) = \Theta(n^2)$ e lemos f é $\Theta(n^2)$.

Dizer que uma função f é $\Theta(g(n))$ significa dizer que, para valores de n suficientemente grandes, os valores de f estarão limitados pelos valores das funções $c_1 g$ e $c_2 g$. Embora estes valores de g dependam das constantes c_1 e c_2 escolhidas, em geral não nos preocupamos com os valores destas constantes, mas sim com a existência delas.

Definição 2. (Notação O) Dizemos que f é $O(g(n))$ se existem constantes positivas c e n_0 tais que $0 \leq f(n) \leq cg(n)$ para todo $n > n_0$.

Enquanto a notação $\Theta(g(n))$ dá limites assintóticos acima e abaixo para a função

$f(n)$, a notação $O(n)$ dá um limite assintótico superior apenas. Se $f(n)$ representar o tempo de execução de um algoritmo para uma instância de tamanho n, dizer que $f(n)$ é $O(n^2)$, por exemplo, significa dizer que o tempo de execução do pior caso cresce como uma função quadrática, isto é, proporcional a n^2.

De modo semelhante, a notação Ω nos dá um limite assintótico inferior para uma dada função $f(n)$.

Definição 3. (Notação Ω) Uma função f é dita $\Omega(g(n))$ se existirem constantes positivas c e n_0 tais que $0 \leq cg(n) \leq f(n)$ para todo $n > n_0$.

As notações Θ, O e Ω se relacionam por meio do importante teorema a seguir.

Teorema 1. Para quaisquer duas funções $f(n)$ e $g(n)$, temos $f(n) = \Theta(g(n))$ se, e somente se, $f(n) = O(g(n))$ e $f(n) = \Omega(g(n))$.

Demonstração: Suponha que $f(n) = \Theta(g(n))$. Isto implica que existem constantes positivas c_1 e c_2 tais que $0 \leq c_1 g(n) \leq f(n) \leq c_2 g(n)$. Logo, $0 \leq c_1 g(n) \leq f(n)$ e $0 \leq f(n) \leq c_2 g(n)$, ou seja, $f(n) = \Omega(g(n))$ e $f(n) = O(g(n))$.

Por outro lado, suponha que $f(n) = O(g(n))$ e $f(n) = \Omega(g(n))$. Por definição, existem constantes positivas c_1 e c_2 tais que $0 \leq f(n) \leq c_2 g(n)$ e $0 \leq c_1 g(n) \leq f(n)$, respectivamente. Logo, $0 \leq c_1 g(n) \leq f(n) \leq c_2 g(n)$, o que mostra que $f(n) = \Theta(g(n))$.

□

1.4 Conceitos Elementares

Potenciação

A potenciação é a operação que escreve uma multiplicação de fatores iguais de forma resumida.

Definição 4. (Potenciação) Dados dois números naturais a e n, a expressão a^n representa um produto de n fatores iguais a a, ou seja,

$$a^n = \underbrace{a \cdot a \cdot \ldots \cdot a}_{n \text{ fatores}}.$$

Neste caso, dizemos que a é a base e n é o expoente da potenciação. Além disso, definimos os casos especiais $a^1 = a$ e $a^0 = 1$.

A Definição 4 nos diz que 7^4, por exemplo, é igual a $7 \times 7 \times 7 \times 7$. Para computar este número, devemos multiplicar os dois primeiros fatores, obtendo 49. Este resultado

deve ser multiplicado pelo próximo fator, donde se tem 343. Por fim, multiplicamos 343 pelo quarto e último fator, obtendo o resultado 2401.

Algumas propriedades da potenciação são bem conhecidas e estão enumeradas a seguir.

Proposição 1. Dados os números naturais a, b, m e n, temos que

1. $a^m \cdot a^n = a^{m+n}$
2. $\dfrac{a^m}{a^n} = a^{m-n}$, se $m \geq n$
3. $(a^m)^n = a^{m \cdot n}$
4. $a^m \cdot b^m = (a \cdot b)^m$

Demonstração

1. $a^m \cdot a^n = \left(\underbrace{a \cdot a \cdot \ldots \cdot a}_{m \text{ fatores}}\right) \cdot \left(\underbrace{a \cdot a \cdot \ldots \cdot a}_{n \text{ fatores}}\right) = \underbrace{a \cdot a \cdot \ldots \cdot a}_{m+n \text{ fatores}} = a^{m+n}$.
2. Se $m \geq n$, então $\frac{a^m}{a^n} = \dfrac{\underbrace{a \cdot a \cdot \ldots \cdot a}_{m \text{ fatores}}}{\underbrace{a \cdot a \cdot \ldots \cdot a}_{n \text{ fatores}}}$. Os fatores a do denominador cancelarão os fatores a do numerador. Restarão, no máximo, $m - n$ fatores. Logo,
$$\frac{a^m}{a^n} = a^{m-n}.$$
3. $(a^m)^n = \underbrace{a^m \cdot a^m \cdot \ldots \cdot \ldots \cdot a^m}_{n \text{ fatores}} = a^{\underbrace{m+m+\ldots+m}_{n \text{ parcelas}}} = a^{mn}$;
4. $a^m \cdot b^m = \underbrace{a \cdot a \cdot \ldots \cdot a}_{m \textbf{ fatores}} \cdot \underbrace{b \cdot b \cdot \ldots \cdot b}_{m \textbf{ fatores}} = \underbrace{(ab) \cdot (ab) \cdot \ldots \cdot (ab)}_{m \textbf{ fatores}} = (ab)^m$.

□

Máximo Divisor Comum

Quando dividimos um número a por b, obtemos um quociente q e um resto r, com $0 \leq r < q$. Sabe-se que, para a e b quaisquer, com $b \neq 0$, q e r existem e são únicos e

podemos escrever $a = bq + r$, o que é garantido pelo Teorema 3. Antes de demonstrá-lo, porém, precisamos do princípio da boa ordenação, uma propriedade fundamental dos números naturais.

Teorema 2. (Princípio da Boa Ordenação) Todo subconjunto dos números naturais possui um menor elemento.

A demonstração foge do escopo deste trabalho e será omitida. Para mais detalhes, consulte (LIMA, 2007).

Teorema 3. (Divisão Euclidiana) Dados os números naturais a e b, $b \neq 0$, existem únicos q e r, também naturais, tais que $a = bq + r$, com $0 \leq r < b$.

Demonstração:

(Existência) Se $a < b$, basta tomar $q = 0$ e $r = b$. Por outro lado, se $a = b$, basta tomar $q = 1$ e $r = 0$. Finalmente, se $a > b$, considere o conjunto

$$S = \{x = a - bq \text{ tais que } a - bq \geq 0\},$$

o qual está contido no conjunto dos números naturais. Note que $b - a > 0$ implica que $S \neq \emptyset$. Logo, pelo princípio da boa ordenação, S possui um menor elemento, digamos r.

Por definição, $r = a - bq \geq 0$ para algum $q \in \mathbb{N}$. Afirmamos que $r < b$ pois, caso contrário, teríamos $r = a - bq \geq b$, donde $a - q(b + 1) \geq 0$, o que implica que $y = a - q(b + 1) \in S$. Mas $y = a - q(b + 1) < a - bq$, o que contradiz o fato de que r é o menor elemento de S. Portanto, $0 \leq r < b$.

(Unicidade) Se $a = bq_1 + r_1$ e $a = bq_2 + r_2$, com $0 \leq r_1, r_2 < b$, então

$$bq_1 + r_1 = bq_2 + r_2$$
$$b(q_1 - q_2) = r_2 - r_1$$

Sendo assim, $r_2 - r_1$ é múltiplo de b. Mas, como $|r_2 - r_1| < b$, o único valor que $r_2 - r_1$ pode assumir é zero. Logo, $r_2 = r_1$, donde segue que $q_2 = q_1$

□

Um dos conceitos centrais da divisão é o de máximo divisor comum de dois números ou, abreviadamente, MDC. O máximo divisor comum nos é apresentado no 6º ano do ensino fundamental. Na escola, este conceito é utilizado na simplificação de frações ou de equações.

Definição 5. (Máximo Divisor Comum) Dados dois números naturais a e b, dizemos que o número natural d é o máximo divisor comum (MDC) entre a e b se:

1. $d|a$ e $d|b$

2. se $m \in \mathbb{N}$ é tal que $m|a$ e $m|b$, então $m|d$.

A primeira propriedade da Definição 5 diz que, para um número d ser o MDC entre a e b, d tem que dividir a e b, isto é, ser um divisor comum de a e b. A segunda propriedade diz que qualquer divisor comum de a e b divide o MDC, ou seja, o MDC é o maior dentre os divisores comuns.

Por exemplo, o MDC entre 12 e 8 é 4, pois 4 é divisor comum entre 12 e 8 e qualquer outro divisor comum positivo entre 12 e 8 (no caso, 1 e 2) divide 4.

Apresentamos, no Teorema 4, um importante resultado acerca da divisão entre dois números naturais.

Teorema 4. Sejam a e b dois números naturais com $a \geq b > 0$. Se $a = bq + r$, com q, $r \in \mathbb{N}$ e $0 \leq r < b$, então $r < \frac{a}{2}$.

Demonstração:

Consideremos dois casos: $a < 2b$ e $a \geq 2b$. No primeiro caso, se $a \geq 2b$, temos que $b \leq \frac{a}{2}$. Como, por hipótese, $r < b$, então $r < \frac{a}{2}$.

No segundo caso, se $a < 2b$ então, por hipótese, $a = bq+r$, onde $q \in \mathbb{N}$ e $0 \leq r < b$, $r \in \mathbb{N}$. Como $a < 2b$, segue que $q = 1$. Logo, $r = a - b$. Observe que $a < 2b \Rightarrow b > \frac{a}{2}$. Portanto, $r = a - b < a - \frac{a}{2} = \frac{a}{2}$

□

Muitas vezes, nos preocupamos apenas com o resto da divisão de um número natural por outro. A Definição 6 apresenta o conceito de congruência módulo n.

Definição 6. Dados os números naturais a, b e n, dizemos que a é congruente a b módulo n se a divisão de a por n deixa resto b. Neste caso, escrevemos $a \equiv b \mod n$.

Outro importante conceito da Teoria dos Números é o de número primo. Os números primos formam a base multiplicativa dos números naturais.

Definição 7. (Número primo) Um número natural é dito primo se possui exatamente dois divisores positivos distintos.

Os números primos se distribuem irregularmente no conjunto dos números naturais. Até o presente momento, não foi descoberto um padrão para essa distribuição, mas existem ferramentas que nos ajudam a entender melhor seu comportamento. Embora haja

infinitos números primos (GONçALVES, 2006), podemos usar uma função para contar a quantidade de primos menores que ou iguais a um número natural dado.

Definição 8. Denotamos por $\pi(n)$ a quantidade de números primos menores ou iguais ao número natural n.

Determinante

Conceito central da álgebra linear, os determinantes desempenham papel fundamental quando queremos analisar a existência de soluções de um sistema linear, bem como a determinação de tais soluções. A todo sistema está associada uma matriz.

Definição 9. (Matriz) Uma matriz é uma tabela de elementos dispostos em linhas e colunas.

Um elemento qualquer de uma matriz A ocupa uma posição em uma linha e em uma coluna. Denotaremos o elemento da linha i e coluna j por a_{ij}. Quando uma matriz A possui o mesmo número n de linhas e de colunas, dizemos que A é uma matriz quadrada de ordem n. Uma matriz que possua apenas uma linha é chamada vetor-linha. Analogamente, uma matriz que possua apenas uma coluna é chamada vetor-coluna.

Um tipo especial de matriz é a matriz identidade, representada pela letra I. Esta consiste de uma matriz quadrada na qual todos os elementos da diagonal principal são iguais a 1 e todos os elementos restantes são iguais a zero. Uma matriz identidade que possua n linhas e n colunas é chamada matriz identidade de ordem n e representada por I_n. A toda matriz quadrada está associado um número, chamado determinante da matriz. A Definição 10 foi extraída de (STRANG, 2010).

Definição 10. (Determinante) Dada uma matriz quadrada $A_{n\times n}$, o determinante de A é um número real que satisfaz as seguintes propriedades:

1. o determinante da matriz identidade I é igual a 1;
2. o sinal do determinante de uma matriz muda quando há uma troca de linhas;
3. o determinante de uma matriz é linear em cada linha separadamente.

A propriedade 3 da Definição 10 pode ser descrita em duas partes:

(a) $$\det \begin{pmatrix} a_{11}+b_{11} & a_{12}+b_{12} & \dots & a_{1n}+b_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \dots & a_{nn} \end{pmatrix}$$

$$= \det \begin{pmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \dots & a_{nn} \end{pmatrix} + \det \begin{pmatrix} b_{11} & b_{12} & \dots & b_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \dots & a_{nn} \end{pmatrix},$$

ou seja, somando-se um vetor linha à primeira linha de uma matriz, o seu determinante será igual à soma do determinante da matriz original com o determinante da matriz obtida pela substituição da primeira linha da matriz original pelo vetor linha, mantendo-se as demais linhas inalteradas.

(b) $$\det \begin{pmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ ca_{i1} & ca_{i2} & \dots & ca_{in} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \dots & a_{nn} \end{pmatrix} = c \cdot \det \begin{pmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ ca_{i1} & ca_{i2} & \dots & ca_{in} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \dots & a_{nn} \end{pmatrix},$$

isto é, multiplicar uma das linhas de uma matriz por um número real c implica que o seu determinante é igual a c vezes o determinante da matriz original.

Observe que uma consequência imediata da parte (b) acima é que, se uma matriz possui uma linha nula, então seu determinante é igual a zero.

Algumas fórmulas para o cálculo de determinantes são bem conhecidas, como é o caso dos determinantes de matrizes 2×2.

Teorema 5. (Determinante de matrizes 2×2) Dada uma matriz

$$A = \begin{pmatrix} a & b \\ c & d \end{pmatrix},$$

temos que $\det\ A = ad - bc$

Demonstração: Para comprovar isto, vamos verificar as 3 propriedades da definição de determinantes:

1. Para a matriz $I = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$, temos que $\det\ I = 1 \cdot 1 - 0 \cdot 0 = 1$. Portanto, a propriedade 1 é válida.

2. Caso as linhas sejam trocadas, obteremos a matriz $B = \begin{pmatrix} c & d \\ a & b \end{pmatrix}$. Seu determinante será
$$\det\ B = cb - ad = -(ad - bc) = -\det\ A$$

3. Vamos verificar que o determinante é linear na 1ª linha.

a) $\det \begin{pmatrix} a+e & b+f \\ c & d \end{pmatrix} = (a+e)d - (b+f)c$

$= ad - bc + ed - cf = \det \begin{pmatrix} a & b \\ c & d \end{pmatrix} + \det \begin{pmatrix} e & f \\ c & d \end{pmatrix}$

b) $\det \begin{pmatrix} ka & kb \\ c & d \end{pmatrix} = kad - kbc = k(ad - bc) = k \cdot \det \begin{pmatrix} a & b \\ c & d \end{pmatrix}$

□

O caso de determinantes de matrizes 3×3 é apresentado no Teorema 6.

Teorema 6. (Regra de Sarrus) Dada uma matriz $A = \begin{pmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{pmatrix}$, temos que

$\det\ A = a_{11}a_{22}a_{33} + a_{12}a_{23}a_{31} + a_{13}a_{21}a_{32} - a_{31}a_{22}a_{13} - a_{32}a_{23}a_{11} - a_{33}a_{21}a_{12}$.

Demonstração: De fato,

1. $\det\ I = \det \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} = 1 \cdot 1 \cdot 1 + 0 \cdot 0 \cdot 0 + 0 \cdot 0 \cdot 0 - 0 \cdot 1 \cdot 0 - 0 \cdot 0 \cdot 1 - 1 \cdot 0 \cdot 0 = 1.$

2. Dada a matriz

$$A = \begin{pmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{pmatrix},$$

há $3! = 6$ maneiras de permutar as suas linhas. Vamos permutar a primeira linha com a segunda. Os outros casos são análogos.

$$\begin{aligned} \det A &= a_{21}a_{12}a_{33} + a_{22}a_{13}a_{31} + a_{23}a_{11}a_{32} - a_{31}a_{12}a_{23} - a_{32}a_{13}a_{21} - a_{33}a_{11}a_{22} \\ &= -(a_{11}a_{22}a_{33} + a_{12}a_{23}a_{31} + a_{13}a_{21}a_{32} - a_{31}a_{22}a_{13} - a_{32}a_{23}a_{11} - a_{33}a_{21}a_{12}) \\ &= -\det\ A, \end{aligned}$$

ou seja, o determinante da matriz A muda de sinal quando trocamos duas linhas.

3. Provemos agora a linearidade do determinante com relação à primeira linha. Seja

$$B = \begin{pmatrix} a_{11}+b & a_{12}+c & a_{13}+d \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{pmatrix}$$

Daí,

a)

$$\begin{aligned}\det B &= (a_{11}+b)a_{22}a_{33} + (a_{12}+c)a_{23}a_{31} + (a_{13}+d)a_{21}a_{32} \\ &\quad - a_{31}a_{22}(a_{13}+d) - a_{32}a_{23}(a_{11}+b) - a_{33}a_{21}(a_{12}+c) \\ &= (a_{11}a_{22}a_{33} + a_{12}a_{23}a_{31} + a_{13}a_{21}a_{32} - a_{31}a_{22}a_{13} - a_{32}a_{23}a_{11} - a_{33}a_{21}a_{12}) \\ &\quad + (ba_{22}a_{33} + ca_{23}a_{31} + da_{21}a_{32} - a_{31}a_{22}d - a_{32}a_{23}b - a_{33}a_{21}c) \\ &= \det \begin{pmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{pmatrix} + \det \begin{pmatrix} b & c & d \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{pmatrix}.\end{aligned}$$

b) $\det \begin{pmatrix} ka_{11} & ka_{12} & ka_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{pmatrix}$

$$\begin{aligned}&= ka_{11}a_{22}a_{33} + ka_{12}a_{23}a_{31} + ka_{13}a_{21}a_{32} - a_{31}a_{22}ka_{13} - a_{32}a_{23}ka_{11} - a_{33}a_{21}ka_{12} \\ &= k(a_{11}a_{22}a_{33} + a_{12}a_{23}a_{31} + a_{13}a_{21}a_{32} - a_{31}a_{22}a_{13} - a_{32}a_{23}a_{11} - a_{33}a_{21}a_{12}) \\ &= k \cdot \det \begin{pmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{pmatrix}.\end{aligned}$$

□

Para o cálculo de determinantes de matrizes de ordens superiores, utiliza-se o Teorema 7, enunciado e demonstrado por Laplace. Contudo, é necessário antes algumas definições.

Definição 11. Seja $A_{n\times n}$. Seja M_{ij} a submatriz $(n-1)\times(n-1)$ obtida pela eliminação da i-ésima linha e da j-ésima coluna de A. O determinante de M_{ij} é chamado determinante menor de a_{ij}. O cofator b_{ij} de a_{ij} é definido como

$$b_{ij} = (-1)^{i+j} \cdot \det(M_{ij})$$

Teorema 7. (Laplace) Seja A uma matriz $n\times n$. Então, para cada linha i, com $1 \leq i \leq n$, temos:

$$\det\ A = a_{i1}b_{i1} + a_{i2}b_{i2} + \cdots a_{in}b_{in},$$

ou ainda, para cada coluna j, com $1 \leq j \leq n$,

$$\det\ A = a_{1j}b_{1j} + a_{2j}b_{2j} + \cdots a_{nj}b_{nj}.$$

A demonstração foge do escopo deste trabalho e será omitida.

Existe um outro método para calcular determinantes de ordens superiores. Este método, atribuído a Gauss, se baseia no escalonamento de matrizes. Diz-se que uma matriz está na forma escalonada reduzida por linhas quando ela cumpre os critérios da Definição 12.

Definição 12. (Forma Escalonada Reduzida por Linhas) Uma matriz $A_{m\times n}$ está na forma escalonada reduzida por linhas se ela satisfaz as seguintes propriedades:

1. todas as linhas nulas, se existirem, ocorrem abaixo de todas as linhas não-nulas;
2. o primeiro elemento diferente de zero a partir da esquerda de uma linha não-nula é igual a 1. Este elemento é chamado de um inicial desta linha;
3. para cada linha não-nula, o um inicial aparece à direita e abaixo dos uns iniciais das linhas precedentes;
4. se uma coluna contém o um inicial, então todos os outros elementos naquela coluna são iguais a zero.

Para transformar uma matriz qualquer em uma matriz escalonada reduzida por linhas, fazemos uso de operações elementares.

Definição 13. Uma operação elementar nas linhas de uma matriz $A_{m\times n}$ é uma das seguintes operações:

1. permuta de duas linhas de A;
2. multiplicação de uma linha de A por uma constante real não-nula;
3. adição de m vezes a i-ésima linha de A à j-ésima linha de A;

Outra definição importante é a de equivalência de duas matrizes com relação a estas operações elementares.

Definição 14. (Equivalência por Linhas) Dizemos que uma matriz $A_{m\times n}$ é equivalente por linhas a uma matriz $B_{m\times n}$ quando B pode ser obtida aplicando-se uma sequência finita de operações elementares nas linhas de A.

As matrizes escalonadas reduzidas por linhas fazem parte de uma classe mais geral de matrizes, chamada matrizes triangulares.

Definição 15. (Matriz Triangular) Uma matriz triangular superior é uma matriz na qual todos os elementos abaixo da diagonal principal são iguais a zero. Analogamente, chama-

se matriz triangular inferior a qualquer matriz na qual os elementos acima da diagonal principal sejam iguais a zero.

É sempre possível transformar uma dada matriz em uma matriz triangular. Isto é garantido pelo Teorema 8.

Teorema 8. Toda matriz é equivalente por linhas a uma matriz triangular.

Demonstração: Mostraremos como transformar uma matriz em uma matriz triangular superior. O caso de matriz triangular inferior é análogo. Dada uma matriz

$$A = \begin{pmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1} & a_{m2} & \cdots & a_{mn} \end{pmatrix},$$

temos que transformá-la em uma matriz escalonada reduzida fazendo uso apenas de operações elementares.

Começamos identificando o primeiro elemento não-nulo da primeira coluna. Tal elemento é chamado pivô. Encontrado tal elemento, permutamos a linha onde ele se encontra com a primeira.

Assim, sem perda de generalidade, suponhamos que a permuta já foi realizada e que o elemento a_{11} é não-nulo. Temos que fazer com que todos os elementos abaixo dele na primeira coluna sejam iguais a zero. Isto pode ser feito substituindo-se, para todo $2 \leq i \leq n$, cada elemento a_{ij} por $a_{ij} - a_{1j} \cdot \dfrac{a_{i1}}{a_{11}}$, com $1 \leq j \leq n$. Após fazer esta operação, o primeiro elemento da r-ésima linha será zero, para todo $2 \leq r \leq n$. Chegamos a uma nova matriz

$$B = \begin{pmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ 0 & b_{22} & \cdots & b_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ 0 & b_{m2} & \cdots & b_{mn} \end{pmatrix}$$

Agora, olhando apenas da segunda linha em diante, identificamos o primeiro elemento não-nulo da segunda coluna, o qual será o novo pivô. Caso ele não esteja na segunda linha, permutamos a linha onde ele se encontra com a segunda. Em seguida, para todo $3 \leq i \leq n$ cada elemento b_{ij} será substituído por $b_{ij} - b_{2j}\dfrac{b_{i2}}{b_{22}}$, com $2 \leq j \leq n$. Após fazer esta operação, o segundo elemento da p-ésima linha será zero para todo $3 \leq p \leq n$.

Chegamos, portanto, a uma nova matriz

$$C = \begin{pmatrix} a_{11} & a_{12} & a_{13} & \cdots & a_{1n} \\ 0 & b_{22} & b_{23} & \cdots & b_{2n} \\ 0 & 0 & c_{33} & \cdots & c_{3n} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & c_{m3} & \cdots & b_{mn} \end{pmatrix}$$

Repetindo este processo para todas as linhas restantes, chegamos à matriz

$$T = \begin{pmatrix} a_{11} & a_{12} & a_{13} & \cdots & a_{1n} \\ 0 & b_{22} & b_{23} & \cdots & b_{2n} \\ 0 & 0 & c_{33} & \cdots & c_{3n} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & t_{nn} \end{pmatrix},$$

que é uma matriz triangular superior, obtida de A apenas por meio de operações elementares.

□

O processo descrito na demonstração do Teorema 8 é conhecido como método de escalonamento de Gauss-Jordan.

Após a matriz ser transformada em uma matriz triangular superior, o determinante será dado pelo produto dos elementos da diagonal principal, o que é garantido pelo Teorema 9.

Teorema 9. O determinante de uma matriz triangular é igual ao produto dos elementos da diagonal principal.

Demonstração: Vamos verificar que as três propriedades da definição de determinantes são satisfeitas.

1. Na matriz identidade

$$I_n = \begin{pmatrix} 1 & 0 & 0 & \cdots & 0 \\ 0 & 1 & 0 & \cdots & 0 \\ 0 & 0 & 1 & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & 1 \end{pmatrix},$$

todos os elementos da diagonal principal são iguais a 1. Portanto, $\det I_n = 1$.

2. Considere uma matriz triangular

$$T = \begin{pmatrix} a_{11} & a_{12} & a_{13} & \cdots & a_{1n} \\ 0 & a_{22} & a_{23} & \cdots & a_{2n} \\ 0 & 0 & a_{33} & \cdots & a_{3n} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & a_{nn} \end{pmatrix}.$$

Sem perda de generalidade, vamos permutar as linhas 1 e 2. Os outros casos são análogos. Após a permuta, obtemos a matriz

$$T_1 = \begin{pmatrix} 0 & a_{22} & a_{23} & \cdots & a_{2n} \\ a_{11} & a_{12} & a_{13} & \cdots & a_{1n} \\ 0 & 0 & a_{33} & \cdots & a_{3n} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & a_{nn} \end{pmatrix}$$

Substituindo cada elemento a_{1j} por $a_{1j} - a_{2j}$, temos:

$$T_2 = \begin{pmatrix} -a_{11} & a_{22} - a_{12} & a_{23} - a_{13} & \cdots & a_{2n} - a_{1n} \\ a_{11} & a_{12} & a_{13} & \cdots & a_{1n} \\ 0 & 0 & a_{33} & \cdots & a_{3n} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & a_{nn} \end{pmatrix}$$

Agora, somamos a primeira linha à segunda, substituindo cada elemento a_{2j} por $a_{2j} + a_{1j}$, donde obtém-se a matriz

$$T_3 = \begin{pmatrix} -a_{11} & a_{22} - a_{12} & a_{23} - a_{13} & \cdots & a_{2n} - a_{1n} \\ 0 & a_{22} & a_{23} & \cdots & a_{2n} \\ 0 & 0 & a_{33} & \cdots & a_{3n} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & a_{nn} \end{pmatrix}$$

A matriz T_3 é uma matriz triangular superior. Seu determinante é dado por:

$$\det T_3 = -a_{11}a_{22}a_{33}\cdots a_{nn} = -\det T$$

3. Somando o vetor $\begin{pmatrix} b_{11} & b_{12} & \cdots & b_{1n} \end{pmatrix}$ à matriz T, obtemos a matriz

$$T_1 = \begin{pmatrix} a_{11}+b_{11} & a_{12}+b_{12} & a_{13}+b_{13} & \cdots & a_{1n}+b_{1n} \\ 0 & a_{22} & a_{23} & \cdots & a_{2n} \\ 0 & 0 & a_{33} & \cdots & a_{3n} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & a_{nn} \end{pmatrix}$$

T_1 é uma matriz triangular superior. Seu determinante é dado por:

$$\begin{aligned} \det T_1 &= (a_{11}+b_{11})a_{22}\cdots a_{nn} = a_{11}a_{22}\cdots a_{nn} + b_{11}a_{22}\cdots a_{nn} \\ &= \det \begin{pmatrix} a_{11} & a_{12} & a_{13} & \cdots & a_{1n} \\ 0 & a_{22} & a_{23} & \cdots & a_{2n} \\ 0 & 0 & a_{33} & \cdots & a_{3n} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & a_{nn} \end{pmatrix} + \det \begin{pmatrix} b_{11} & b_{12} & b_{13} & \cdots & b_{1n} \\ 0 & a_{22} & a_{23} & \cdots & a_{2n} \\ 0 & 0 & a_{33} & \cdots & a_{3n} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & a_{nn} \end{pmatrix} \end{aligned}$$

Multiplicando-se a r-ésima linha de T por uma constante k, obtemos a matriz triangular superior

$$T_2 = \begin{pmatrix} a_{11} & a_{12} & a_{13} & a_{14} & \cdots & a_{1n} \\ 0 & a_{22} & a_{23} & a_{24} & \cdots & a_{2n} \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & ka_{rr} & \cdots & ka_{rn} \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & \cdots & a_{nn} \end{pmatrix}$$

donde $\det T_2 = a_{11}a_{22}\cdots ka_{rr}\cdots a_{nn} = k \cdot \det T_2$

□

2 Metodologia

Neste capítulo iremos descrever a metodologia utilizada para a escolha dos algoritmos a serem analisados, para a escolha dos livros didáticos e para a implementação dos algoritmos, bem como a contagem das operações.

2.1 Escolha dos Algoritmos

Os problemas escolhidos foram o do cálculo de potências com expoente natural, o do cálculo do máximo divisor comum de dois números naturais e o do cálculo de determinantes de matrizes $n \times n$. A escolha desses problemas se deu pelo fato de existirem mais de um algoritmo com uso difundido entre professores e alunos para a solução dos mesmos. O trabalho tem como objetivo, em primeiro lugar, a comparação entre esses algoritmos do ponto de vista da análise de complexidade assintótica. Logo, não seria possível fazer esta comparação no caso de haver somente um algoritmo para a resolução de um problema.

No caso da potenciação, há dois algoritmos amplamente utilizados: o da potenciação por definição e o da potenciação por meio de divisão e conquista. Já para o problema do máximo divisor comum, utiliza-se o algoritmo por decomposição em fatores primos ou o algoritmo de Euclides. Por fim, para o cálculo de determinantes, em geral, aprendemos na escola a utilizar fórmulas fechadas para os casos de matrizes 2×2 e 3×3 (regra de Sarrus). Já para determinantes de ordens superiores, comumente faz-se uso do cálculo por meio de cofatores. Alternativamente, podemos usar um algoritmo de escalonamento e produto dos elementos da diagonal principal.

Estes problemas estão presentes em toda a matemática escolar. Por isso, consideramos de suma importância fazer uma comparação de tais algoritmos, para que possamos escolher o ensino daqueles que forem mais eficientes. Estas escolhas não se devem apenas a uma melhoria do cálculo escrito. Há uma tendência crescente por se adotar o ensino de programação básica no currículo escolar. Sendo assim, a compreensão de um algoritmo não passa somente por sua construção, mas também pela sua análise de eficiência.

2.2 Escolha e Verificação dos Livros Didáticos

Para a escolha dos livros didáticos, fizemos uma pesquisa no banco de dados da Secretaria de Estado de Educação do Distrito Federal (SEEDF) e verificamos quais as três Coordenadorias Regionais de Ensino (CRE) com maior número de alunos. Os dados estão disponíveis em (FEDERAL, 2017).

Tabela 1 – Coordenadorias Regionais de Ensino (CRE) com maior número de matrículas – Ensino Fundamental II (6º ao 9º ano)

CRE	**Matrículas**
Ceilândia	22.868
Planaltina	11.519
Plano Piloto	10.895

Tabela 2 – Coordenadorias Regionais de Ensino (CRE) com maior número de matrículas – Ensino Médio

CRE	**Matrículas**
Ceilândia	13.195
Taguatinga	9.409
Plano Piloto	8.273

Feito isto, procuramos, em cada regional de ensino, as 5 escolas com o maior número de matrículas nos anos finais do ensino fundamental e no ensino médio. Os dados estão disponíveis em (FEDERAL, 2018a) e em (FEDERAL, 2018b).

Tabela 3 – Escolas com maior número de matrículas no Ensino Fundamental II – CRE Ceilândia

Escola	**Matrículas**
CEF 25 de Ceilândia	1.632
CEF 27 de Ceilândia	1.506
CEF 28 de Ceilândia	1.169
CEF 26 de Ceilândia	1.059
CEF 07 de Ceilândia	1.052

Tabela 4 – Escolas com maior número de matrículas no Ensino Fundamental II – CRE Planaltina

Escola	**Matrículas**
CEF 04 de Planaltina	1.173
CED Dona América Guimarães	1.020
CEF 01 de Planaltina	999
CED Pompílio Marques de Souza	930
CED Stella dos Cherubins Guimarães Trois	907

Em seguida, pesquisamos quais os livros adotados em matemática no 6º ano nas escolas das Tabelas 3, 4 e 5 e no 2º ano do ensino médio nas escolas das Tabelas 6, 7 e 8.

De posse dos livros, analisamos cuidadosamente os exercícios e os algoritmos de potências e máximo divisor comum apresentados nos livros de 6º ano e os exercícios e algoritmos de determinantes nos livros de 2º ano.

Tabela 5 – Escolas com maior número de matrículas no Ensino Fundamental II – CRE Plano Piloto

Escola	**Matrículas**
CEF Polivalente	1.103
CEF CASEB	994
CEF Athos Bulcão	735
CEF 07 de Brasília	723
CEF 01 do Cruzeiro	699

Tabela 6 – Escolas com maior número de matrículas no Ensino Médio – CRE Ceilândia

Escola	**Matrículas**
CEM 02 de Ceilândia	2.132
CEM 12 de Ceilândia	1.583
CED 15 de Ceilândia	1.484
CEM 04 de Ceilândia	1.399
CED 06 de Ceilândia	1.246

Tabela 7 – Escolas com maior número de matrículas no Ensino Médio – CRE Taguatinga

Escola	**Matrículas**
CEM Ave Branca	2.414
CEM Taguatinga Norte	1.525
CED EIT	1.377
CEM 03 de Taguatinga	1.399
CED 04 de Taguatinga	1.246

Tabela 8 – Escolas com maior número de matrículas no Ensino Médio - CRE Plano Piloto

Escola	**Matrículas no Ensino Fundamental II**
CEM Setor Leste	1.636
CEM Elefante Branco	1.582
CEM Setor Oeste	1.156
CEM Paulo Freire	904
CED GISNO	716

Os livros de ensino fundamental adotados pelas escolas das Tabelas 3, 4 e 5 são Bianchini (2016), Dante (2016b), Chavante (2016), Andrini e Vasconcelos (2015), Silveira (2016) e Centúrion e Jakubovic (2015).

Todos os livros apresentam somente o cálculo da potenciação por definição. Na verdade, apenas Silveira (2016) tem uma seção onde são apresentadas as propriedades da potenciação. Com relação ao cálculo do máximo divisor comum, os livros Centúrion e Jakubovic (2015) e Chavante (2016) mostram como calcular o máximo divisor comum somente via definição. Já os demais livros apresentam também o algoritmo da decomposição

em fatores primos. Nenhum deles apresenta o algoritmo de Euclides.

Com relação ao ensino médio, os livros adotados pelas escolas das Tabelas 6, 7 e 8 são Dante (2016a), Iezzi et al. (2016) e Leonardo (2016).

Em todos eles é apresentado somente o cálculo de determinantes de matrizes 2×2 e 3×3 utilizando as conhecidas fórmulas dos Teoremas 5 e 6. Curiosamente, Leonardo (2016) mostra um recurso de *softwares* de planilhas eletrônicas que calcula determinantes de matrizes 2×2, mas não comenta sobre como seriam as programações de fórmulas nestes tipos de *softwares*.

2.3 Problemas Escolhidos

Nesta seção, descreveremos os problemas selecionados para este trabalho e os algoritmos ensinados na escola para a solução dos mesmos. A escolha dos problemas se deu de acordo com a existência de mais de um algoritmo para os mesmos e que pudessem ser ensinados nos ensinos fundamental e médio. Os problemas escolhidos foram: a potenciação com base e expoente naturais, o máximo divisor comum entre dois números naturais e o determinante de matrizes quadradas.

Em cada uma das subseções, escreveremos os algoritmos em linguagem de pseudocódigo e, em seguida, faremos uma análise do tempo gasto para a execução do algoritmo em função do tamanho da instância. Conforme discutido no Capítulo 1, iremos considerar que cada operação do mesmo tipo leva um tempo constante para ser executada.

Na Subseção 2.3.1, descrevemos os algoritmos para a potenciação. Iremos comparar um algoritmo que calcula uma potenciação por definição com outro que calcula utilizando propriedades da potenciação. Na Subseção 2.3.2, serão abordados os algoritmos para a determinação do MDC entre dois números naturais. Faremos a comparação de um algoritmo que calcula o máximo divisor comum por decomposição dos números em fatores primos com o algoritmo de Euclides. Por fim, na Subseção 2.3.3, faremos uma análise de dois algoritmos para o cálculo de determinantes: um deles que calcula por meio de cofatores e do teorema de Laplace e outro que calcula por meio do escalonamento de Gauss-Jordan.

2.3.1 Potenciação

O primeiro algoritmo que iremos analisar é o da potenciação. O Algoritmo 1 computa uma potenciação nos moldes da Definição 4.

No Algoritmo 1, para calcular qualquer potência, partimos do número natural 1. Em seguida, multiplicamo-o pela base a. O resultado é, então, multiplicado por a, e assim sucessivamente. Este procedimento é iterado um número de vezes igual ao expoente n.

Algoritmo 1 Potenciação por definição

Entrada: dois números naturais a e n
Saída: um número natural b tal que $a^n = b$

```
1: b = 1
2: for i = 1 do n
3:     b = b · a
4: end for
5: return b
```

Neste algoritmo, portanto, o número de multiplicações feitas é exatamente igual a n. O laço `for` da linha 2 é iterado n vezes, cada vez com um tempo constante c. Portanto, a função que descreve o tempo de execução do Algoritmo 1 é dada por

$$f(n) = cn = \Theta(n)$$

Um algoritmo alternativo para a potenciação é apresentado no Algoritmo 2. Ele utiliza como base de construção as propriedades da potenciação, descritas na Seção 1.4.

Algoritmo 2 Potenciação por divisão e conquista

Entrada: dois números naturais a e n
Saída: um número natural b tal que $a^n = b$

```
1: function POT2(a, n)
2:     if n = 1 then
3:         return a
4:     else if n ≡ 0 mod 2 then
5:         return POT2(a, n/2)^2
6:     else
7:         return POT2(a, (n−1)/2)^2 · a
8:     end if
9:     return b
10: end function
```

O Algoritmo 2 calcula uma potenciação por meio de uma recorrência. Se o expoente for igual a 1, o valor da potência é o próprio a, que é a base da potenciação. Caso contrário, separa-se o cálculo de acordo com a paridade de n, o expoente da potenciação: se n for par, faz-se a potenciação $\left(a^{\frac{n}{2}}\right)^2$. Caso contrário, faz-se a potenciação $\left(a^{\frac{n-1}{2}}\right)^2 a$.

Observe que, a cada iteração do algoritmo, o expoente se reduz pela metade. Logo, após k iterações, teremos que o expoente n_k será menor que $\frac{n}{2^k}$. Assim, em algum momento, teremos $n_k = 1$ (pois a sequência $a_k = \frac{n}{2^k} \to 0$ quando $k \to \infty$). Isto ocorre quando $k = \log_2 n$. Daí, se cada multiplicação leva um tempo constante c_1 para ser executada e cada divisão um tempo c_2, então a função que dá o tempo de execução do Algoritmo 2 é

$$f(n) = (c_1 + c_2) \log_2 n = \Theta(\log n)$$

2.3.2 Máximo Divisor Comum

Na escola, aprendemos alguns métodos para obter o MDC entre dois números. O método da decomposição em fatores primos está descrito no Algoritmo 3.

Algoritmo 3 MDC por decomposição em fatores primos

Entrada: Dois naturais a e b, com $a \geq b$ e $P = \{p_i : p_i$ é o i-ésimo número primo$\}$.
Saída: Um número natural d tal que $MDC(a, b) = d$.

1: $d = 1$
2: $i = 1$
3: **while** $a > 1$ **and** $b > 1$ **and** $p_i \leq b$ **do**
4: **if** $a \equiv 0 \mod p_i$ **and** $b \equiv 0 \mod p_i$ **then**
5: $d = d \cdot p_i$
6: $a = \frac{b}{p_i}$
7: $b = \frac{b}{p_i}$
8: **else if** $a \equiv 0 \mod p_i$ **then**
9: $a = \frac{a}{p_i}$
10: **else if** $b \equiv 0 \mod p_i$ **then**
11: $b = \frac{b}{p_i}$
12: **else**
13: $i = i + 1$
14: **end if**
15: **end while**
16: **return** d

O Algoritmo 3 calcula o MDC (a, b) como um produto de números primos. Inicialmente, o algoritmo considera o valor de $d =$ MDC (a, b) como sendo 1. Na primeira iteração, a e b são divididos por $p_1 = 2$. Se ambas as divisões forem exatas, d é multiplicado por 2 e a e b tem seus valores atualizados para $\frac{a}{2}$ e $\frac{b}{2}$, respectivamente. Caso a divisão seja exata apenas para a, apenas o valor de a será atualizado e d permanece igual a 1. O mesmo ocorre se a divisão for exata apenas para b. Caso a divisão seja inexata para ambos, o algoritmo passa a dividir a e b pelo próximo primo, que é $p_2 = 3$. O algoritmo prossegue enquanto p_i for menor ou igual a b.

Suponha que $a = 2^{\alpha_1} \cdot 3^{\alpha_2} \cdot \ldots \cdot p_k^{\alpha_k}$ e $b = 2^{\beta_1} \cdot 3^{\beta_2} \cdot \ldots \cdot p_j^{\beta_j}$, com $a \geq b$, onde p_i representa o i-ésimo número primo e $\alpha_i \geq 0$ e $\beta_i \geq 0$. O Teorema Fundamental da Aritmética garante que tal representação sempre é possível para $b \geq 2$ (GONçALVES, 2006). Aplicando o logaritmo em base 2 dos dois lados de cada igualdade, temos que

$$\log_2 a = \alpha_1 \cdot \log_2 2 + \alpha_2 \cdot \log_2 3 + \ldots + \alpha_k \cdot \log_2 p_k \geq \alpha_1 + \alpha_2 + \ldots + \alpha_k$$

e

$$\log_2 b = \beta_1 \cdot \log_2 2 + \beta_2 \cdot \log_2 3 + \ldots + \beta_j \cdot \log_2 p_j \geq \beta_1 + \beta_2 + \ldots + \beta_j$$

pois $\log_2 n \geq 1$ para todo $n \geq 2$. Assim, $\alpha_1+\alpha_2+\ldots+\alpha_k \leq \log_2 a$ e $\beta_1+\beta_2+\ldots+\beta_j \leq \log_2 b$.

O Algoritmo 3 faz, no mínimo, $2 \cdot \pi(b)$ testes, dividindo os números a e b por todos os primos menores ou iguais a b. Além disso, se a e b são escritos como na decomposição acima, pode acontecer de serem feitos $\gamma_1 = \max\{\alpha_1, \beta_1\}$ testes para o primeiro primo, $\gamma_2 = \max\{\alpha_2, \beta_2\}$ testes para o segundo primo, e assim sucessivamente. Daí, o número de divisões realizadas seria, na pior das hipóteses, $2 \cdot (\gamma_1 + \gamma_2 + \ldots + \gamma_n) \cdot \pi(b)$, onde $\gamma_i = \max\{\alpha_i, \beta_i\}$ e $n = \max\{k, j\}$. Como $\gamma_1+\gamma_2+\ldots+\gamma_n \leq (\alpha_1+\beta_1)+(\alpha_2+\beta_2)+\ldots+\alpha_k+\beta_j \leq \log_2 a + \log_2 b \leq 2\log_2 a$ então, se cada divisão leva um tempo constante c para ser executada, temos que o tempo de execução do Algoritmo 3 é limitado superiormente pela função

$$f(n) = c \cdot 4 \cdot \log_2 a \cdot \pi(b) = O(\log a \cdot \pi(b))$$

Observe que a análise feita acima coloca um limite superior para o tempo de execução do Algoritmo 3. Por este motivo escrevemos $f(n)$ como uma função $O(\log(a)\pi(b))$, e não $\Theta(\log(a)\pi(b))$.

Podemos calcular o MDC utilizando outro algoritmo. O algoritmo de Euclides para a determinação do MDC entre dois números é conhecido desde a idade antiga e, surpreendentemente, é mais eficiente que o da decomposição em fatores primos. O Algoritmo 4 descreve um procedimento computacional que executa o algoritmo de Euclides para a determinação do MDC entre dois números.

Algoritmo 4 Algoritmo de Euclides Estendido

Entrada: dois números naturais a e b, com $a \geq b$
Saída: um número natural d tal que MDC $(a, b) = d$

```
1: if b = 0 then
2:     d = a
3: else
4:     r = a % b
5:     d = MDC (b, r)
6: end if
7: return d
```

O Algoritmo 4 se baseia no fato de que MDC (a, b) = MDC (b, r), onde r é o resto da divisão de a por b (HEFEZ, 2014). Em cada iteração do algoritmo, a é dividido por b, obtendo-se um resto r. O número b é, então, dividido por r e as definições de a e b são atualizadas para b e r, respectivamente.

Seja r_i o resto da i-ésima iteração do Algoritmo 4. Ora, o Teorema 4 garante que $r_1 < \frac{a}{2}$. Na segunda iteração do algoritmo, $r_2 < \frac{b}{2}$. Na terceira e quarta iterações do algoritmo, tem-se $r_3 < \frac{r_1}{2}$ e $r_4 < \frac{r_2}{2} < \frac{b}{4}$, e assim sucessivamente. Observe que, a cada duas iterações do algoritmo, o resto cai pela metade. Logo, após $2k$ iterações, teremos que $r_{2k} < \frac{b}{2^k}$. Como a sequência (r_n) tende para zero quando n tende para infinito, em

algum momento devemos ter $r_k < 1$, momento este em que o algoritmo para. Isto ocorre quando $b < 2^{2k}$, isto é, quando $k > \dfrac{\log_2 b}{2}$.

O raciocínio acima, explicitado também em (SCHEINERMAN, 2011), mostra que são necessárias $\dfrac{\log_2 b}{2}$ divisões para que o Algoritmo 4 dê o MDC entre a e b. Se cada divisão leva um tempo constante c para ser executada, temos que a função que dá o tempo de execução do Algoritmo 4 é dada por

$$f(b) = c \cdot \frac{\log_2 b}{2} = \Theta(\log b)$$

2.3.3 Determinantes

É comum, ao aprendermos determinantes, aplicarmos métodos diferentes de cálculo para cada tamanho de matriz quadrada. O caso 2×2 é o mais simples. O Algoritmo 5 é um procedimento computacional que executa a fórmula obtida no Teorema 5 da Seção 1.6.

Algoritmo 5 Matrizes 2×2

Entrada: uma matriz $A = \begin{pmatrix} a & b \\ e & f \end{pmatrix}$
Saída: um número real d tal que $\det\ A = d$
1: $d = a \cdot f - b \cdot e$
2: **return** d

O Algoritmo 5 traduz uma fórmula fechada. A contagem de operações é, portanto, fixa. Uma matriz 2×2 tem sempre 4 elementos e faremos sempre 2 multiplicações a um tempo constante c_1 e 1 adição a um tempo constante c_2, totalizando 3 operações, ou seja, a função que dá o tempo de execução do algoritmo 5 é constante, conforme mostra a função abaixo:

$$f(n) = 2c_1 + c_2 = \Theta(1)$$

Para o caso de matrizes 3×3, em geral utiliza-se a regra de Sarrus, descrita no Teorema 6. O Algoritmo 6 descreve um procedimento computacional para se calcular o determinante de uma matriz 3×3 por meio da regra de Sarrus.

Algoritmo 6 Determinantes de matrizes 3×3

Entrada: uma matriz $A = \begin{pmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{pmatrix}$
Saída: um número real d tal que $d = \det\ A$
1: $d = a_{11}a_{22}a_{33} + a_{12}a_{23}a_{31} + a_{13}a_{21}a_{32} - a_{31}a_{22}a_{13} - a_{32}a_{23}a_{11} - a_{33}a_{21}a_{12}$
2: **return** d

Novamente, por ser uma fórmula fechada, a regra de Sarrus é sempre um procedimento operacional com um número de operações fixo. Neste caso, sabemos que uma

matriz 3×3 tem 9 elementos. Para aplicar a regra de Sarrus, fazemos 12 multiplicações a um tempo constante c_1 e 5 adições a um tempo constante c_2. Logo, a função que dá o tempo de execução do Algoritmo 6 é igual a

$$f(n) = 12c_1 + 5c_2 = \Theta(1)$$

Para o cálculo de determinantes de ordens quaisquer, temos duas possibilidades: podemos usar o teorema de Laplace (Teorema 4 da Seção 1.6), que é o método dos cofatores, ou aplicar o escalonamento de Gauss para transformar a matriz em uma matriz triangular e, em seguida, multiplicar os números da diagonal principal. O Algoritmo 7 descreve um procedimento computacional para o cálculo do determinante através de cofatores e o Algoritmo 8, um procedimento para o cálculo por meio do escalonamento de Gauss.

Algoritmo 7 Determinantes por cofatores

Entrada: Uma matriz quadrada $A_{n\times n}$, com $n \in N$
Saída: Um número real d tal que $d = \det\ A$

1: **function** DET(A, n)
2: **if** $n = 1$ **then**
return a_{11}
3: **else**
4: $d = 0$
5: **for** $j = i$ **do** n
6: $i = 1$
7: $M_{ij} = \begin{pmatrix} a_{11} & a_{12} & \cdots & a_{1(j-1)} & a_{1(j+1)} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2(j-1)} & a_{2(j+1)} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots & \vdots & \cdots & \vdots \\ a_{(i-1)1} & a_{(i-1)2} & \cdots & a_{(i-1)(j-1)} & a_{(i-1)(j+1)} & \cdots & a_{(i-1)n} \\ a_{(i+1)1} & a_{(i+1)2} & \cdots & a_{(i+1)(j-1)} & a_{(i+1)(j+1)} & \cdots & a_{(i+1)n} \\ \vdots & \vdots & \ddots & \vdots & \vdots & \cdots & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{n(j-1)} & a_{n(j+1)} & \cdots & a_{nn} \end{pmatrix}$
8: **end for**
9: $d = (-1)^{i+j} a_{ij} \cdot$ DET$(M_{ij}, n-1)$
10: **return** d
11: **end if**
12: **end function**

No Algoritmo 7, começamos pela linha 1. Nesta linha, há n cofatores. Como eles serão multiplicados pelos determinantes menores, teremos n multiplicações. Os determinantes menores são determinantes de matrizes $(n-1) \times (n-1)$. Para cada determinante menor, inicia-se com a 1ª linha. Ela terá $n-1$ cofatores, o que implica que serão feitas $n-1$ multiplicações. Logo, o total de multiplicações, por enquanto, é $n \cdot (n-1)$. Prosseguindo com este raciocínio, temos que o número de multiplicações para o cálculo de um determinante por este método é igual a

$$n(n-1)(n-2)\cdots 3 \cdot 2 \cdot 1 = n!$$

Com relação ao número de adições, observe que, se matriz tem dimensão 1, não há adições a fazer. O número de adições é, portanto, igual a zero. No caso de matrizes 2×2, há 2 determinantes 1×1 com zero adições cada, mais uma adição entre eles. Para matrizes 3×3, há 3 determinantes 2×2, cada um com 1 adição, mais duas adições entre os resultados. Prosseguindo com este raciocínio, podemos construir a Tabela 9.

Tabela 9 – Quantidade de adições do Algoritmo 7

Tamanho da matriz	**Quantidade de adições**
1×1	$a_1 = 0$
2×2	$a_2 = 2 \cdot a_1 + 1$
3×3	$a_3 = 3 \cdot a_2 + 2$
4×4	$a_4 = 4 \cdot a_3 + 3$
$\vdots$	$\vdots$
$n \times n$	$a_n = n \cdot a_{n-1} + n - 1$

Observe que, para determinar a quantidade de adições realizadas pelo Algoritmo 7, chegamos à recorrência

$$\begin{cases} a_1 = 0 \\ a_n = n \cdot a_{n-1} + n - 1 \end{cases}$$

Lema 1. A solução da recorrência

$$\begin{cases} a_1 = 0 \\ a_n = n \cdot a_{n-1} + n - 1 \end{cases},$$

se $n \geq 1$ é $a_n = n! - 1$.

Demonstração: por indução sobre n. Para $n = 1$, é válida, pois $a_1 = 1! - 1 = 0$.

Supondo a recorrência válida para algum $n \in \mathbb{N}$, temos

$$a_{n+1} = (n+1)a_n + n = (n+1)(n! - 1) + n = (n+1)n! - n - 1 + n = (n+1)! - 1$$

Pelo princípio da indução, a solução $a_n = n! - 1$ é válida para todo n natural.

□

A discussão acima mostra que, se cada multiplicação for feita a um tempo c_1 e cada adição a um tempo c_2, então a função que dá o tempo de execução do Algoritmo 7 é dada por

$$f(n) = c_1 n! + c_2(n! - 1) = \Theta(n!)$$

O Algoritmo 8 apresenta um método alternativo para o cálculo de determinantes de matrizes de ordem n, baseado no escalonamento de Gauss.

Algoritmo 8 Determinantes por escalonamento

Entrada: Uma matriz quadrada $A_{n\times n}$, com $n \in N$
Saída: Um número real d tal que $d = \det\ A$

1: **for** $k = 1$ **do** n
2: $m = k$
3: **while** $a_{mk} = 0$ **do**
4: $m = m + 1$
5: **end while**
6: **if** $m > n$ **then**
return 0
7: **end if**
8: swap(A_m, A_k)
9: **for** $i = k + 1$ **do** n
10: **for** $j = k$ **do** n
11: $a_{ij} = a_{ij} - \left(\frac{a_{ik}}{a_{kk}}\right) \cdot a_{kj}$
12: **end for**
13: **end for**
14: **end for**
15: $d = 1$
16: **for** $i = 1$ **do** n
17: $d = d \cdot a_{ii}$
18: **end for**
19: **return** d

No Algoritmo 8, iniciamos pelo elemento da 1ª linha e da 1ª coluna da matriz A. Caso ele seja zero, permutamos com a 1ª linha com a 2ª, e assim sucessivamente, até que apareça um elemento não-nulo. Observe que se todos os elementos forem nulos, o determinante será zero. Para cada linha i, com $2 \leq i \leq n$, substituímos o elemento a_{ij} por $a_{ij} - a_{1j} \cdot \frac{a_{i1}}{a_{11}}$.

O processo descrito acima faz com que todos os elementos da 1ª coluna abaixo da 1ª linha se tornem nulos. Repetindo o procedimento para a 2ª linha, zeramos todos os elementos da 2ª coluna abaixo da 2ª linha, e assim por diante, até que a matriz A é transformada em uma matriz triangular superior. Finalmente, multiplicamos os elementos da diagonal principal, obtendo um número real d, que é o determinante de A.

No Algoritmo 8, como cada linha possui n elementos, a substituição $a_{ij} \leftarrow a_{ij} - a_{1j} \cdot \left(\frac{a_{i1}}{a_{11}}\right)$ na primeira iteração do laço while tem um custo de uma divisão a um tempo constante c_1, uma multiplicação a um tempo constante c_2 e uma adição a um tempo constante c_3. Logo, em cada linha são feitas $3n$ operações, no máximo, a um tempo constante c para algum $c \in \mathbb{R}$. Como esta operação é feita em $n - 1$ linhas, temos um

custo de tempo da ordem de

$$3n(n-1)\cdot c,$$

com $c \in \mathbb{R}$.

Na segunda iteração do laço while, as operações para cada elemento são feitas $n-1$ vezes, ao longo de $n-2$ linhas. O tempo gasto é, portanto, uma função da forma

$$3(n-1)(n-2)\cdot c,$$

com $c \in \mathbb{R}$

Logo, o custo de tempo total é dado pelo somatório

$$\begin{aligned} f(n) &= 3c[n(n-1)+(n-1)(n-2)+\cdots+2\cdot 1] = 3c\sum_{k=1}^{n}(n-k)[n-(k-1)] \\ &= 3c\sum_{k=1}^{n}(n^2-2nk+n+k^2-k) \\ &= 3c\left[n^2\sum_{k=1}^{n}1-(2n+1)\sum_{k=1}^{n}k+n\sum_{k=1}^{n}1+\sum_{k=1}^{n}k^2\right] \end{aligned}$$

Usando a conhecida fórmula

$$\sum_{k=1}^{n}k^2 = \frac{n(n+1)(2n+1)}{6},$$

, a qual pode ser consultada em (LIMA et al., 2006), temos que o custo das operações é:

$$\begin{aligned} f(n) &= 3c\left[n^3-(2n+1)\frac{n(n+1)}{2}+n^2+\frac{n(n+1)(2n+1)}{6}\right] \\ &= 3c\left[\frac{6n^3-6n^3-6n-3n^2-3n+6n^2+2n^3+n^2+2n^2+n}{6}\right] \\ &= c\frac{2n^3+6n^2-8n}{2} = c(n^3+3n^2-4n) \end{aligned}$$

Portanto,

$$f(n) = \Theta(n^3).$$

Este é o custo para triangularizar a matriz A. Em seguida, faz-se apenas $n-1$ multiplicações com os números da diagonal principal. Como este número é acrescentado ao número anterior, ainda tem-se a função custo do cálculo do determinante por escalonamento da ordem de n^3.

3 Resultados

Neste capítulo, iremos aplicar os algoritmos em exemplos conhecidos, de modo a comparar sua eficiência. Aplicaremos tanto em exemplos extraídos de livros didáticos quanto em exemplos numéricos com números de ordem elevada. Esta última análise tem o propósito de mostrar o comportamento assintótico dos algoritmos à medida que cresce o tamanho da amostra.

3.1 Algoritmos de Potenciação

O primeiro algoritmo a ser analisado é o da potenciação. Vamos testá-lo primeiramente em um exemplo didático, extraído de (IMENES; LELLIS, 2012), com adaptações.

Exemplo 1. Calcule o valor da potência 4^3.

Solução Temos que $a = 4$ e $n = 3$.

De acordo com o Algoritmo 1, seguem-se os seguintes passos.

1. iniciamos com o valor 1;
2. em seguida, multiplicamo-no por 4, obtendo $1 \cdot 4 = 4$;
3. multiplicamos o resultado por 4, obtendo $4 \cdot 4 = 16$;
4. multiplicamos o resultado por 4, obtendo $16 \cdot 4 = 64$.

Após fazer as multiplicações, obtemos a resposta: 64. O número de multiplicações realizadas utilizando este algoritmo é igual 3.

Agora, aplicando o Algoritmo 2:

1. separamos a potência em $4^2 \cdot 4^1$;
2. atribui-se $4^1 = 4$;
3. depois, calculamos $4^2 = 4 \cdot 4 = 16$;
4. por último, utilizamos os valores de 4^2 e 4^1 para concluir o cálculo, obtendo $4^2 \cdot 4^1 = 16 \cdot 4 = 64$.

Embora os dois métodos sejam bastante parecidos, aplicando o Algoritmo 2 executamos apenas 2 multiplicações.

□

Para comparar os dois algoritmos de forma mais transparente, utilizamos um sorteador de bases e expoentes. Em linguagem Python, escrevemos os dois algoritmos. Em seguida, inserimos um comando de geração aleatória de números naturais nos intervalos fechados [1, 10], [1, 100], [1, 1 000] e [1, 10 000]. Para cada intervalo, o código sorteou 100 valores para a base e o expoente dentro dos limites do intervalo e, partindo desses valores, calculou-se as potenciações. Por fim, o código incluiu um contador do número de multiplicações realizadas por cada algoritmo. O código retorna como valores de saída um par ordenado (x, y), onde x representa o limite superior do intervalo e y, o número médio de multiplicações feitas considerando as 100 amostras. Os códigos em Python podem ser consultados no Apêndice 1.

Por exemplo, no intervalo [1, 10], o programa sorteou 100 valores para a base e o expoente. Após calcular as 100 potências, o programa contou o número de multiplicações realizadas em cada uma e calculou a média aritmética desses valores. No nosso experimento, o número médio de multiplicações foi 5,02 para o Algoritmo 1 e 2,65 para o Algoritmo 2. A Tabela 10 mostra o número médio de multiplicações realizadas por cada algoritmo em cada intervalo, com 100 amostras por intervalo.

Tabela 10 – Número médio de multiplicações para os Algoritmos 1 e 2

Intervalo	**Algoritmo** 1	**Algoritmo** 2
$[1, 10]$	5,02	2,65
$[1, 100]$	47,36	6,97
$[1, 1000]$	507,06	11,92
$[1, 10000]$	5.717,72	16,71

A Tabela 10 mostra, de forma evidente, que o Algoritmo 2, que calcula a potenciação por divisão e conquista, utilizando propriedades de potência, é muito mais eficiente que o Algoritmo 1, que calcula uma potenciação por definição. Por exemplo, para expoentes variando de 1 a 100, o Algoritmo 1 fez, em média, 47,36 multiplicações para calcular cada potenciação, contra 6,97 multiplicações no Algoritmo 2. Observe que mesmo para instâncias de valor baixo, isto é, no intervalo [1, 10], já há uma grande diferença entre o número médio de multiplicações efetuadas pelos dois algoritmos, já que o Algoritmo 1 faz, em média, o dobro de multiplicações em relação ao Algoritmo 2.

Estes resultados estão de acordo com a análise de complexidade feita no Capítulo 3. Lá, verificamos que o custo operacional do Algoritmo 1 era uma função $\Theta(n)$, enquanto a do Algoritmo 2, $\Theta(\lg n)$. Deste modo, com expoentes variando de 1 a 100, temos 50 como valor médio. Daí, o número médio de multiplicações para o Algoritmo 1 seria 50 e, do Algoritmo 2, $\lg 50 \approx 5,64$.

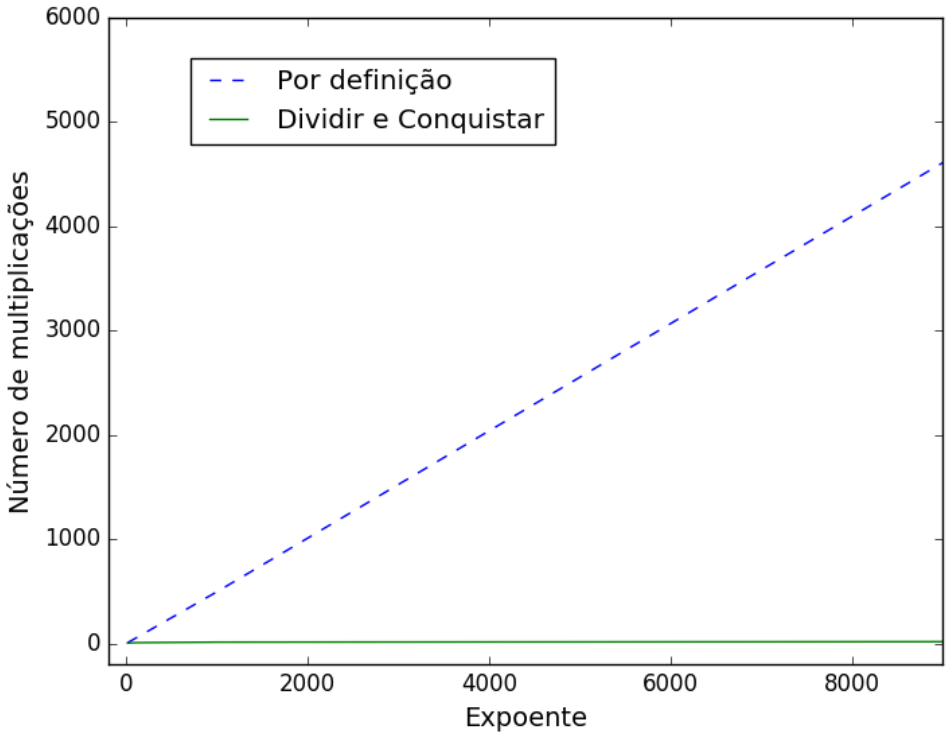

Figura 1 – Algoritmos de potenciação

A Figura 1 compara os dois experimentos realizados com o código Python para os Algoritmos 1 e 2. O gráfico mostra o número de multiplicações em função do expoente n nos dois casos. Observe que, dada a escala utilizada, o gráfico para o Algoritmo 2 é praticamente uma reta horizontal, o que mostra sua enorme vantagem em termos de custo operacional.

3.2 Algoritmos de MDC

Nesta seção, iremos comparar os algoritmos para o cálculo do MDC, apresentados na Seção 3.2. Vamos começar com um exemplo retirado do livro didático (BARROSO; MATHEUS; NANI, 2010), com adaptações.

Exemplo 2. Calcule o MDC entre 180 e 150.

Solução Aplicando o Algoritmo 3 com $a = 180$, $b = 150$ e $n = 6$, temos:

1. o teste $a > 1$ e $b > 1$ é feito sempre no início da cada iteração. Caso um dos dois seja igual a 1, o algoritmo pára e dá a resposta. Nenhum dos dois é igual a 1. Logo, podemos executar o laço while, dividindo a e b pelo menor número primo, $p_1 = 2$, obtendo $\frac{180}{2} = 90$ e $\frac{150}{2} = 75$;

2. prosseguimos, dividindo 90 e 75 por 2. $\frac{90}{2} = 45$, mas 75 não é divisível por 2. (Observe que $75 \equiv 1 \mod 2$);

3. é necessário dividir 45 mais uma vez por 2, o que não é possível, pois $45 \equiv 1 \mod 2$;

4. agora, passamos a dividir ambos os números pelo próximo primo, que é $p_2 = 3$, obtendo $\frac{45}{3} = 15$ e $\frac{75}{3} = 25$;

5. prosseguimos com a divisão por 3. $\frac{15}{3} = 5$, mas 25 não é divisível por 3. ($25 \equiv 1 \mod 3$);

6. a divisão de 5 por 3 não é possível, pois $5 \equiv 2 \mod 3$. Daí, passamos a dividir pelo próximo primo, $p_3 = 5$, obtendo $\frac{5}{5} = 1$ e $\frac{25}{5} = 5$;

7. um dos números a ou b é 1. Logo, o MDC $(180, 150)$, será o resultado da multiplicação de todos os números primos que dividiram ambos a e b, logo MDC $(180, 150) = 2 \cdot 3 \cdot 5 = 30$;

8. portanto, MDC $(180, 150) = 30$.

Contando as operações, realizamos 12 divisões (entre exatas e não exatas) e 2 multiplicações, totalizando 14 operações. Se considerarmos apenas o número de divisões, foram 12.

Agora vamos aplicar o Algoritmo de Euclides (Algoritmo 4).

1. primeiramente, faz-se o teste para saber se um dos dois números é zero. Como nenhum dos dois é zero, prosseguimos com o algoritmo como se segue;

2. dividimos 180 por 150, de modo a obter o quociente e o resto, obtendo $180 = 1 \cdot 150 + 30$;

3. agora, faz-se o MDC entre 150 e 30. Como nenhum dos dois é zero, dividimos 150 por 30, obtendo $150 = 5 \cdot 30 + 0$;

4. agora, faz-se o MDC entre 30 e 0. Como um dos dois é zero, conclui-se que o MDC $(180, 150) = 30$.

Contando o número de operações neste algoritmo, temos apenas 2 divisões.

□

Semelhantemente ao que fizemos para a potenciação, escrevemos um código em Python para calcular o MDC (a, b) por meio dos Algoritmos 3 e 4 e contar o número de divisões realizadas. Os intervalos de amostragem foram do tipo $[1, 10^k]$, com $k \in \{1, 2, \ldots, 12\}$. Em cada intervalo, foram geradas 100 amostras com a e b variando aleatoriamente no intervalo. Em cada amostra, registrou-se o número de divisões realizadas e, em seguida, calculou-se o número médio de divisões realizadas.

Para o Algoritmo 3, necessitou-se de uma lista de primos. Utilizamos uma lista com os primeiros 1 milhão de primos. A Tabela 11 mostra o número médio de divisões realizadas por cada algoritmo em cada intervalo, com 100 amostras por intervalo.

Tabela 11 – Número médio de divisões para os Algoritmos 3 e 4

Intervalo	Algoritmo 3	Algoritmo 4
$[1, 10]$	8,95	1,73
$[1, 10^2]$	29,01	3,94
$[1, 10^3]$	111,84	5,81
$[1, 10^6]$	23.125,36	12,23
$[1, 10^9]$	112.620,67	17,58
$[1, 10^{12}]$	144.298,78	23,62

Novamente, os dados obtidos no experimento mostram uma enorme diferença entre os algoritmos. Observando a Tabela 11, vemos que o Algoritmo 4 é muito mais eficiente que o Algoritmo 3. Por exemplo, para números aleatórios no intervalo [1, 10^6], o número médio de divisões realizadas pelo Algoritmo 4 é 12,23 contra 23.125,36 do Algoritmo 3.

A Figura 2 compara os Algoritmos 3 e 4 com base nos experimentos realizados em linguagem *Python*. O gráfico mostra o número de divisões em função da ordem de grandeza dos números a e b. O eixo horizontal está na escala logarítmica, para facilitar a leitura. Observe que o gráfico relativo ao Algoritmo 4 é praticamente uma reta horizontal, o que mostra sua vantagem em termos de custo operacional.

É importante notar que a escala dos eixos na Figura 2 pode levar a falsas conclusões. Aparentemente, para valores de a e b da ordem de 10^2 não há muita diferença entre os algoritmos. Contudo, enquanto o Algoritmo 3 efetua em média 29 divisões, aproximadamente, o Algoritmo 4 efetua pouco menos de 4, o que representa 7 vezes mais divisões no Algoritmo 3 do que no Algoritmo 4. Analisando os livros didáticos adotados nas escolas consultadas, vemos que a maioria absoluta dos exemplos utilizados para o cálculo de máximo divisor comum são de números no intervalo $[1, 10^3]$, onde o Algoritmo 4 já se mostra muito mais eficiente.

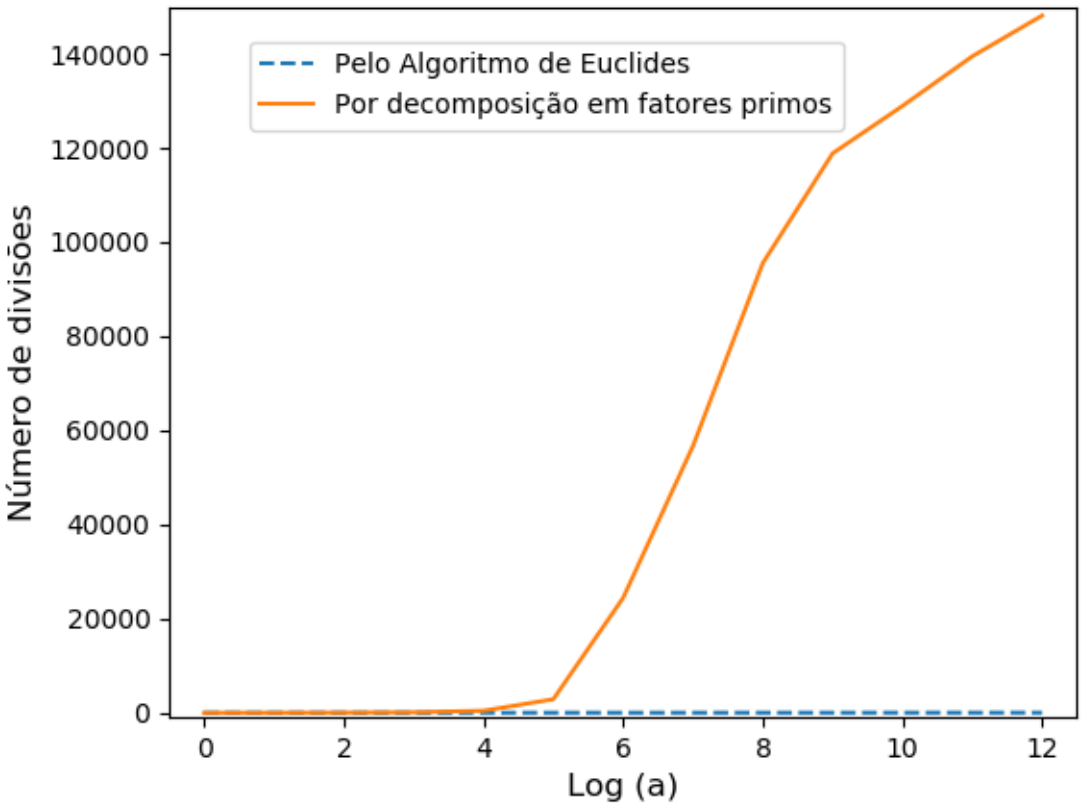

Figura 2 – Algoritmos de máximo divisor comum

3.3 Algoritmos de Determinantes

Nesta seção, iremos aplicar os algoritmos desenvolvidos para calcular determinantes. Iniciaremos com os determinantes de matrizes de ordem 2, com um exemplo do livro didático (IEZZI et al., 2011), com adaptações.

Exemplo 3. Calcule o determinante da matriz $A = \begin{pmatrix} 1 & -1 \\ 2 & 2 \end{pmatrix}$

Solução Primeiramente, aplicaremos o Algoritmo 5. Basta substituir os valores na fórmula.

$$\det\ A = \begin{pmatrix} 1 & -1 \\ 2 & 2 \end{pmatrix} = 1 \cdot 2 - 2 \cdot (-1) = 2 + 2 = 4$$

Por se tratar de uma fórmula fechada, o número de operações realizadas é sempre igual a 3: duas multiplicações e uma adição. Se observarmos apenas o número de multiplicações, foram duas.

Pelo Algoritmo 7, começamos com a primeira linha da matriz. Temos, então, que $M_{11} = 2$ e $M_{12} = 2$.

Logo, pela fórmula, tem-se que $\det\ A = (-1)^{1+1} \cdot 1 \cdot \det\ M_{11} + (-1)^{1+2} \cdot (-1) \cdot \det\ M_{12} = (-1)^2 \cdot 1 \cdot 2 + (-1)^3 \cdot (-1) \cdot 2 = 2 + 2 = 4$.

Contando as operações, foram feitas 2 multiplicações e 1 adição, totalizando 3 operações. Se observarmos apenas o número de multiplicações, foram 2.

Agora vamos utilizar o escalonamento de Gauss, descrito no Algoritmo 8. Como o elemento a_{11} é não-nulo, o escolhemos como pivô. Em seguida, multiplicamos a linha 1 por $\dfrac{-a_{21}}{a_{11}}$ e somamos o resultado à linha 2, substituindo-a pelos valores obtidos. Assim, os elementos da linha 2 passam a ser:

$$a_{21} = a_{11} \cdot \left(\frac{-a_{21}}{a_{11}}\right) + a_{21} = 1 \cdot \left(\frac{-2}{1}\right) + 2 = 0$$

e

$$a_{22} = a_{12} \cdot \left(\frac{-a_{21}}{a_{11}}\right) + a_{22} = -1 \cdot \left(\frac{-2}{1}\right) + 2 = 4$$

Isto é, obtemos a matriz triangular superior

$$\begin{pmatrix} 1 & -1 \\ 0 & 4 \end{pmatrix}$$

Agora, o determinante é dado pelo produto dos elementos da diagonal principal, ou seja, $\det\ A = 1 \cdot 4 = 4$

Fazendo a contagem de operações, vemos que foram utilizadas 3 multiplicações e 2 adições, num total de 5 operações. Se contarmos apenas o número de multiplicações, foram 3.

□

Para analisar a aplicação dos algoritmos em matrizes 3×3, utilizaremos no exemplo 4 um problema do livro (DANTE, 2014), com adaptações.

Exemplo 4. Aplicando a regra de Sarrus, calcule o determinante da matriz

$$A = \begin{pmatrix} 3 & 2 & -1 \\ 5 & 0 & 4 \\ 2 & -3 & 1 \end{pmatrix}$$

Solução:

Primeiro vamos obedecer ao comando do exemplo e utilizar a regra de Sarrus, descrita no Algoritmo 6, para calcular o determinante. Basta substituir os valores na fórmula.

$$\begin{aligned}\det\ A &= \begin{pmatrix} 3 & 2 & -1 \\ 5 & 0 & 4 \\ 2 & -3 & 1 \end{pmatrix} \\ &= 3 \cdot 0 \cdot 1 + 2 \cdot 4 \cdot 2 + (-1) \cdot 5 \cdot (-3) - 2 \cdot 0 \cdot (-1) - (-3) \cdot 4 \cdot 3 - 1 \cdot 5 \cdot 2 \\ &= 0 + 16 + 15 - 0 + 36 - 10 = 57\end{aligned}$$

Por se tratar de uma fórmula fechada, o número de multiplicações e adições é fixo, sendo 12 multiplicações e 5 adições, totalizando 17 operações. Observando apenas o número de multiplicações, são 12 (estamos aqui contando também as adições e multiplicações por zero).

Agora, usando o Algoritmo 7, escolhe-se uma linha da matriz aleatoriamente, digamos, a segunda. Temos, então, que:

$$M_{21} = \begin{pmatrix} 2 & -1 \\ -3 & 1 \end{pmatrix}; M_{22} = \begin{pmatrix} 3 & -1 \\ 2 & 1 \end{pmatrix} \text{ e} M_{23} = \begin{pmatrix} 3 & 2 \\ 2 & -3 \end{pmatrix}$$

Apenas para efeito de organização, calcularemos os determinantes de M_{21}, M_{22} e M_{23} separadamente, inserindo-os posteriormente na fórmula. Sem perda de generalidade, escolheremos a primeira linha de cada uma das matrizes para o cálculo dos determinantes.

$$\begin{aligned}\det M_{21} &= (-1)^2 \cdot 2 \cdot 1 + (-1)^3 \cdot (-1) \cdot (-3) = -1 \\ \det M_{22} &= (-1)^2 \cdot 3 \cdot 1 + (-1)^3 \cdot (-1) \cdot 2 = 5 \\ \det M_{23} &= (-1)^2 \cdot 3 \cdot (-3) + (-1)^3 \cdot 2 \cdot 2 = -13\end{aligned}$$

Feito isto, aplicamos a fórmula explicitada no Algoritmo 7.

$$\begin{aligned}\det\ A &= (-1)^{2+1} \cdot 5 \cdot \det\ M_{21} + (-1)^{2+2} \cdot 0 \cdot \det\ M_{22} + (-1)^{2+3} \cdot 4 \cdot \det\ M_{23} \\ &= (-1) \cdot 5 \cdot (-1) + 1 \cdot 0 \cdot 5 + (-1) \cdot 4 \cdot (-13) = 5 + 52 = 57\end{aligned}$$

Contando o número de operações, temos 6 multiplicações e 5 adições, totalizando 11 operações. Se contarmos apenas o número de multiplicações, são 6.

Aplicando o Algoritmo 8, temos que o primeiro elemento da primeira linha é não-nulo. Logo, multiplicamos a primeira linha por $\frac{-a_{i1}}{a_{11}}$ e somamos o resultado com a linha i, com $2 \leq i \leq 3$, substituindo os resultados na linha i. Temos que cada a_{ij} será:

$$
\begin{aligned}
a_{21} &= -\frac{5}{3} \cdot 3 + 5 = 0 \\
a_{22} &= -\frac{5}{3} \cdot 2 + 0 = -\frac{10}{3} \\
a_{23} &= -\frac{5}{3} \cdot (-1) + 4 = \frac{17}{3} \\
a_{31} &= -\frac{2}{3} \cdot 3 + 2 = 0 \\
a_{32} &= -\frac{2}{3} \cdot 2 - 3 = -\frac{13}{3} \\
a_{33} &= -\frac{2}{3} \cdot (-1) + 1 = \frac{5}{3}
\end{aligned}
$$

A matriz passa a ser

$$
\begin{pmatrix}
3 & 2 & -1 \\
0 & -\frac{10}{3} & \frac{17}{3} \\
0 & -\frac{13}{3} & \frac{5}{3}
\end{pmatrix}
$$

.

Agora, a segunda linha será multiplicada por $\frac{\frac{13}{3}}{-\frac{10}{3}} = -\frac{13}{10}$ e somada com a terceira linha, substituindo os resultados na terceira linha. Logo:

$$
\begin{aligned}
a_{32} &= -\frac{10}{3} \cdot \left(-\frac{13}{10}\right) - \frac{13}{3} = 0 \\
a_{33} &= -\frac{17}{3} \cdot \left(-\frac{13}{10}\right) + \frac{5}{3} = -\frac{57}{10}
\end{aligned}
$$

Donde obtemos a matriz

$$
\begin{pmatrix}
3 & 2 & -1 \\
0 & -\frac{10}{3} & \frac{17}{3} \\
0 & 0 & -\frac{57}{10}
\end{pmatrix}
$$

.

A matriz acima é triangular superior. Deste modo:

$$
\det A = 3 \cdot \left(-\frac{10}{3}\right) \cdot \left(-\frac{57}{10}\right) = 57
$$

Para o Algoritmo 8, realizamos 10 multiplicações, 8 adições e 1 divisão, totalizando 19 operações. Contando apenas o número de multiplicações, foram 10.

□

Agora chegou a vez de aplicar os algoritmos no cálculo de determinantes de ordens superiores. Embora seja pouco comum a sua aplicação em sala de aula do ensino médio, encontra-se nos livros didáticos exercícios envolvendo o cálculo de determinantes de matrizes 4×4. Vamos comparar o algoritmo por escalonamento com o algoritmo por cofator. o exemplo 5 encontra-se no livro (PAIVA, 2010), com adaptações.

Exemplo 5. Calcule o determinante da matriz

$$A = \begin{pmatrix} 1 & 2 & -1 & 0 \\ 2 & 0 & 1 & 4 \\ 0 & -3 & -1 & 2 \\ 2 & 0 & 1 & 3 \end{pmatrix}$$

Solução:

Vamos aplicar o Algoritmo 7. Escolhendo a primeira linha da matriz, temos que:

$$M_{11} = \begin{pmatrix} 0 & 1 & 4 \\ -3 & -1 & 2 \\ 0 & 1 & 3 \end{pmatrix}$$

$$M_{12} = \begin{pmatrix} 2 & 1 & 4 \\ 0 & -1 & 2 \\ 2 & 1 & 3 \end{pmatrix}$$

$$M_{13} = \begin{pmatrix} 2 & 0 & 4 \\ 0 & -3 & 2 \\ 2 & 0 & 3 \end{pmatrix}$$

$$M_{14} = \begin{pmatrix} 2 & 0 & 1 \\ 0 & -3 & -1 \\ 2 & 0 & 1 \end{pmatrix}$$

Para não gerar confusão, vamos usar a letra N para denotar os determinantes menores das matrizes acima. Em cada uma delas, escolheremos a primeira linha para a determinação dos menores e cofatores. No caso da matriz M_{11}, tem-se que:

$$N_{11} = \begin{pmatrix} -1 & 2 \\ 1 & 3 \end{pmatrix}$$

$$N_{12} = \begin{pmatrix} -3 & 2 \\ 0 & 3 \end{pmatrix}$$

$$N_{13} = \begin{pmatrix} -3 & -1 \\ 0 & 1 \end{pmatrix}$$

Escolhendo agora a segunda linha para a determinação dos menores e cofatores de cada matriz, temos que:

$$\det N_{11} = (-1)^3 \cdot 1 \cdot 2 + (-1)^4 \cdot 3 \cdot (-1) = -2 - 3 = -5$$
$$\det N_{12} = (-1)^3 \cdot 0 \cdot 2 + (-1)^4 \cdot 3 \cdot (-3) = -9$$
$$\det N_{13} = (-1)^3 \cdot 0 \cdot (-1) + (-1)^4 \cdot 1 \cdot (-3) = -3$$

Deste modo, temos que:

$$\det M_{11} = (-1)^2 \cdot 0 \cdot (-5) + (-1)^3 \cdot 1 \cdot (-9) + (-1)^4 \cdot 4 \cdot (-3) = 9 - 12 = -3$$

Repetindo o procedimento acima para as outras matrizes, temos que:

$$\det M_{12} = 2$$
$$\det M_{13} = 6$$
$$\det M_{14} = 0$$

Portanto,

$$\det A = (-1)^2 \cdot 1 \cdot (-3) + (-1)^3 \cdot 2 \cdot 2 + (-1)^4 \cdot (-1) \cdot 6 + (-1)^5 \cdot 0 \cdot 0 = -3 - 4 - 6 = -13$$

Utilizando o Algoritmo 7, realizamos 24 multiplicações e 23 adições, totalizando 47 operações. Se contarmos apenas o número de multiplicações, foram 24.

Pelo Algoritmo 8, o primeiro elemento da primeira linha é não nulo. Multiplicamos a primeira linha por $-\frac{a_{i1}}{a_{11}}$ para todo i, com $2 \leq i \leq 4$, e somamos o resultado à linha i, substituindo-a pelos valores encontrados. Assim:

$a_{21} = -2 \cdot 1 + 2 = 0$; $a_{22} = -2 \cdot 2 + 0 = -4$; $a_{23} = -2 \cdot (-1) + 1 = 3$; $a_{24} = -2 \cdot 0 + 4 = 4$;

Repetindo o procedimento acima para as outras linhas, temos que a terceira linha permanece inalterada, pois o primeiro elemento já é nulo e:

$a_{41} = 0$; $a_{42} = -4$; $a_{43} = 3$; $a_{44} = 3$;

Assim, obtemos a matriz

$$A_1 = \begin{pmatrix} 1 & 2 & -1 & 0 \\ 0 & -4 & 3 & 4 \\ 0 & -3 & -1 & 2 \\ 0 & -4 & 3 & 3 \end{pmatrix}$$

A próxima iteração do laço *for* multiplicará a segunda linha por $-\frac{3}{4}$, somando-a com a terceira linha, substituindo os resultados na própria linha 3. Daí:

$a_{32} = -\frac{3}{4} \cdot (-4) - 3 = 0$; $a_{33} = -\frac{3}{4} \cdot 3 - 1 = -\frac{13}{4}$; $a_{34} = -\frac{3}{4} \cdot 4 + 2 = -1$;

A operação de multiplicar a segunda linha por -1 e somar com a quarta linha, substituindo os valores na linha 4 resulta em:

$a_{42} = 0$; $a_{43} = 0$; $a_{41} = -1$;

Como consequência, obtemos a matriz

$$A_2 = \begin{pmatrix} 1 & 2 & -1 & 0 \\ 0 & -4 & 3 & 4 \\ 0 & 0 & -\frac{13}{4} & -1 \\ 0 & 0 & 0 & -1 \end{pmatrix}$$

A matriz A_2 é uma matriz triangular superior. Seu determinante é igual ao determinante de A. O algoritmo é concluído com a multiplicação

$$d = 1 \cdot (-4) \cdot \left(-\frac{13}{4}\right) \cdot (-1) = -13$$

Aplicando o Algoritmo 8, realizamos 17 multiplicações e 14 adições, totalizando 31 operações. Contando apenas as multiplicações, foram 17.

□

Para complementar a análise dos algoritmos, escrevemos um código em *Python* para implementar os Algoritmos 7 e 8. O código sorteia aleatoriamente matrizes $n \times n$ com n variando de 1 a 8. Para cada valor de n, foram obtidas dez amostras de matrizes formadas por números inteiros no intervalo $[-10, 10]$. O código contém também instruções para a contagem do número de multiplicações realizadas pelo algoritmo, que retorna um par ordenado (x, y), onde x é o determinante da matriz e y é o número médio de multiplicações realizadas. A Tabela 12 o número médio de multiplicações realizadas por cada algoritmo, com n variando no intervalo [1, 8] com 10 amostras para cada n.

Tabela 12 – Número médio de multiplicações os Algoritmos 7 e 8

n	Algoritmo 7	Algoritmo 8
2	6	2
3	27	8
4	120	20
6	3.708	70
8	207.840	168

Os dados da Tabela 12 mostram, de forma contundente, que o Algoritmo 7 é muito mais eficiente que o Algoritmo 8. Por exemplo, para matrizes 6×6, o número médio de

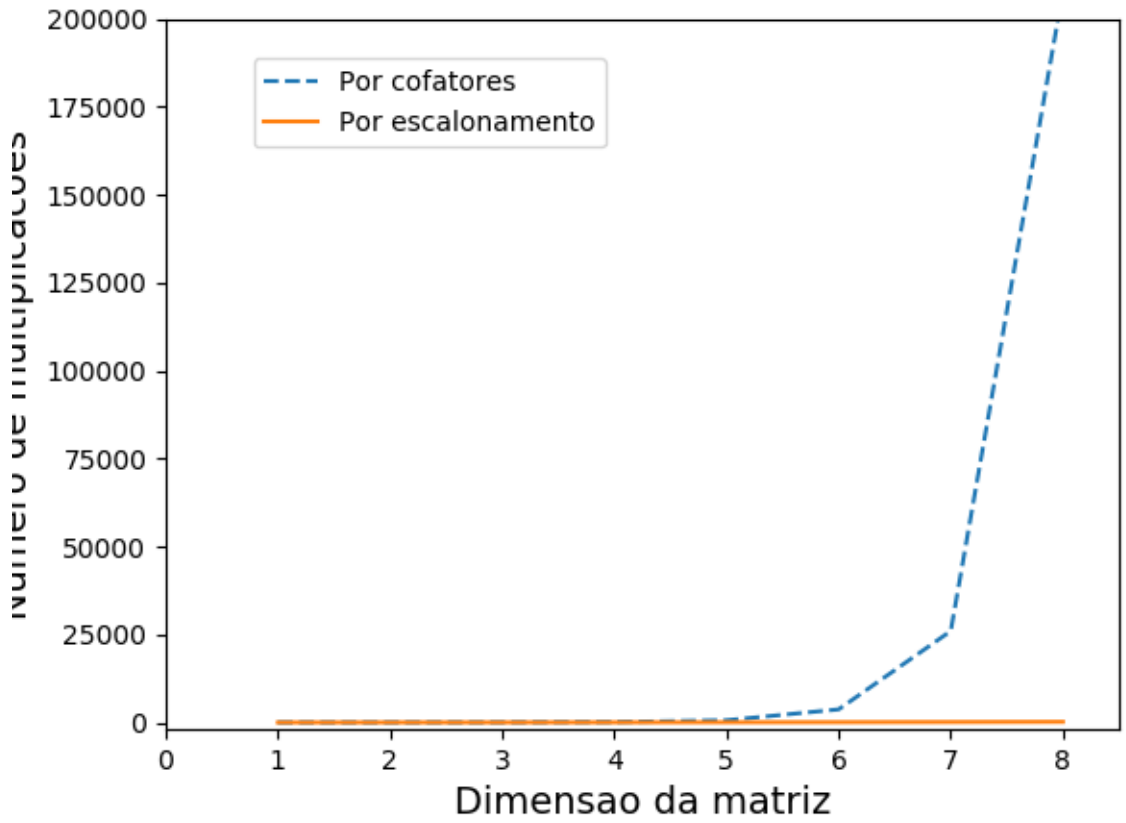

Figura 3 – Algoritmos de determinantes

multiplicações realizadas pelo Algoritmo 7 no experimento realizado foi 3.708 contra 70 do Algoritmo 8.

A Figura 3 compara o desempenho dos Algoritmos 7 e 8. Os gráficos mostram o número de multiplicações realizadas em função do tamanho da matriz no experimento realizado em linguagem *Python*. Novamente, observamos que um dos gráficos, no caso a contagem de multiplicações para o Algoritmo 8, é praticamente uma reta horizontal. Isto siginifica que a taxa de crescimento da quantidade de operações realizadas pelo Algoritmo 8 e, consequentemente, do tempo de execução, com relação ao tamanho da matriz é muito pequeno, se comparado ao do Algoritmo 7, cujo gráfico tem um crescimento acentuado para $n \geq 6$, ou seja, para n suficientemente grande, o Algoritmo 8 faz menos operações que o Algoritmo 7.

4 Considerações Finais

Este trabalho buscou comparar, sob o ponto de vista da análise de complexidade assintótica, os algoritmos disponíveis para a solução de três problemas típicos da matemática escolar: o cálculo de potenciações com base e expoente naturais; o cálculo do máximo divisor comum de dois números naturais e o determinante de matrizes de ordem n. As análises dos algoritmos feitas nos Capítulos 2 e 3 revelaram que o Algoritmo para a potenciação por divisão e conquista é mais eficiente que o Algoritmo por definição, que o Algoritmo para o máximo divisor comum de Euclides é mais que eficiente que o Algoritmo por decomposição em fatores primos e que o Algoritmo para o cálculo de determinantes por escalonamento é mais eficiente que o de cofatores. A pergunta de pesquisa que motivou o trabalho foi: *Os algoritmos ensinados no ensino médio e fundamental para o cálculo de potenciações, máximo divisor comum e determinantes são computacionalmente eficientes?*

Após fazer uma análise dos livros didáticos adotados nas escolas públicas das regionais de ensino do Distrito Federal com maior número de alunos, inferimos que são adotados os algoritmos menos eficientes para a solução dos problemas escolhidos. Os livros apresentam apenas o cálculo de potenciações por definição e do máximo divisor comum por decomposição em fatores primos. Com relação ao cálculo de determinantes, os livros de ensino médio cobrem apenas as matrizes de tamanho 2×2 ou 3×3.

Isto nos leva a concluir que os algoritmos mais eficientes, como o da potenciação por divisão e conquista, o algoritmo de Euclides e o de determinantes por escalonamento, são deixados de lado no ensino destes temas na escola.

Acreditamos não haver empecilho para apresentar tais algoritmos aos alunos. Supondo que um aluno do 6º ano saiba efetuar as quatro operações fundamentais, fazer uma potenciação agrupando os fatores aos pares não representa nenhuma dificuldade extra. Por outro lado, o algoritmo de Euclides exige apenas que a pessoa saiba efetuar uma divisão, registrando resto e quociente. O caso do cálculo de determinantes é mais emblemático. Ora, os alunos, no 2º ano do ensino médio, já aprendem a fazer escalonamento de sistemas. Por que não aplicar o escalonamento para o cálculo de determinantes?

Não queremos de forma alguma ampliar o conteúdo programático baseados apenas em opiniões pessoais. Queremos reforçar que nossa opinião é a de que se deve priorizar os algoritmos mais eficientes. Isto não impede a apresentação dos conteúdos da forma como o professor achar melhor. O conceito a ser apresentado é uma coisa, o algoritmo escolhido para efetuar um cálculo, é outra.

Em relação aos aspectos didáticos e pedagógicos relacionados a este trabalho, não fizemos uma pesquisa aprofundada sobre a temática, deixando em aberto para trabalhos

futuros. Sabemos que as estratégias de ensino muito influenciam o aprendizado dos alunos e não apenas o conteúdo. Portanto, embora tenhamos verificado quais algoritmos são mais eficientes para resolver os problemas abordados, é necessário construir estratégias adequadas para ensiná-los. Contudo, saber qual algoritmo é mais eficiente muda o olhar do professor para a questão, tornando importante a escolha dos mais eficientes.

Para finalizar, entendemos que é importante dar um direcionamento para trabalhos futuros dentro do tema de pesquisa abordado neste trabalho. Uma possibilidade é fazer uma busca por mais algoritmos escolares que possam ser comparados com outros utilizados para o mesmo fim. Poderíamos, inclusive, estudar algoritmos típicos da matemática superior, como os algoritmos numéricos para encontrar a raiz de um polinômio, por exemplo. Uma outra possibilidade, esta bem mais difícil, é tentar escrever um algoritmo para o cálculo de determinantes com complexidade inferior a n^3.

Outro aspecto importante a ressaltar é a da possibilidade de se ensinar lógica de programação ainda na educação básica. Para isso, podemos aproveitar fórmulas que os alunos já utilizam, como a fórmula de Bháskara ou a regra de Sarrus, por exemplo, para escrever programas simples que calculem as raízes reais de uma equação do 2º grau (quando existirem) e o determinante de uma matriz 3×3 com entradas conhecidas ou escrever, em pseudocódigo, algoritmos para problemas contemplados no currículo. Este trabalho pode servir como uma ponte para abordar essas ideias.

Esperamos que este trabalho tenha sido útil para dar um novo olhar para os algoritmos que usamos comumente e para dar ao professor de matemática um embasamento teórico para a escolha dos algoritmos a serem ensinados nos casos da potenciação, máximo divisor comum e determinantes.

Referências

ANDRINI Álvaro; VASCONCELOS, M. J. *Praticando Matemática*. 4ª. ed. São Paulo: Editora do Brasil, 2015. v. 6. Citado na página 41.

BARROSO, J. M.; MATHEUS, A. dos R.; NANI, A. P. S. *Araribá Matemática*. 3ª. ed. São Paulo: Editora Moderna, 2010. v. 6. Citado na página 53.

BIANCHINI, E. *Matemática*. 8ª. ed. São Paulo: Editora Moderna, 2016. v. 6. Citado na página 41.

CENTúRION, M.; JAKUBOVIC, J. *Matemática na Medida Certa*. 1ª. ed. São Paulo: Editora Leya, 2015. v. 6. Citado na página 41.

CHAVANTE, E. *Série Convergências Matemática*. 1ª. ed. São Paulo: Editora SM, 2016. v. 6. Citado na página 41.

CORMEN, T. H. et al. *Algoritmos: teoria e prática*. 3ª. ed. São Paulo: Editora Campus, 2012. Citado 2 vezes nas páginas 23 e 24.

DANTE, L. R. *Matemática*. 1ª. ed. São Paulo: Editora Ática, 2014. Citado na página 57.

DANTE, L. R. *Matemática contexto e aplicações*. 3ª. ed. São Paulo: Editora Ática, 2016. v. 2. Citado na página 42.

DANTE, L. R. *Projeto Teláris Matemática*. 2ª. ed. São Paulo: Editora Ática, 2016. v. 6. Citado na página 41.

DROZDEK, A. *Estrutura de dados e algoritmos em C++*. 1ª. ed. São Paulo: CENGAGE Learning, 2002. Citado na página 25.

ESQUINCA, S. R. A. *Algoritmos: resolução de problemas básicos*. Dissertação (Mestrado) — Universidade Federal de Mato Grosso do Sul, 2014. Citado na página 21.

FEDERAL, G. do D. *Censo Escolar do DF*. 2017. Disponível em: <http://www.cre.se.df.gov.br/ascom/documentos/censo/2017/2017_qd_pub_df_mat_ef_201_cre.pdf>. Citado na página 39.

FEDERAL, G. do D. *Matrículas no Ensino Fundamental*. 2018. Disponível em: <http://www.cre.se.df.gov.br/ascom/documentos/dados/2017/ii_d_escola.pdf>. Citado na página 40.

FEDERAL, G. do D. *Matrículas no Ensino Médio*. 2018. Disponível em: <http://www.cre.se.df.gov.br/ascom/documentos/dados/2017/ii_d_medio_escola.pdf>. Citado na página 40.

GONçALVES, A. *Introdução à Álgebra*. 5. ed. Rio de Janeiro: IMPA, 2006. Citado 2 vezes nas páginas 30 e 44.

HEFEZ Ábramo. *Aritmética*. 1ª. ed. Rio de Janeiro: Sociedade Brasileira Matemática, 2014. Citado na página 45.

IEZZI, G. et al. *Matemática*. 5ª. ed. São Paulo: Editora Atual, 2011. Citado na página 56.

IEZZI, G. et al. *Matemática: ciência e aplicações*. 9ª. ed. São Paulo: Editora Saraiva, 2016. v. 2. Citado na página 42.

IMENES, L. M.; LELLIS, M. *Matemática Imenes & Lellis*. 2ª. ed. São Paulo: Editora Moderna, 2012. v. 7. Citado na página 51.

LEONARDO, F. M. de. *Conexões com a matemática*. 3ª. ed. São Paulo: Editora Moderna, 2016. v. 2. Citado na página 42.

LIMA, E. L. *Analise Real*. 9ª. ed. Rio de Janeiro: IMPA, 2007. v. 1. Citado na página 28.

LIMA, E. L. et al. *A Matemática do Ensino Médio*. 6ª. ed. Rio de Janeiro: Sociedade Brasileira de Matemática, 2006. v. 2. Citado na página 50.

PAIVA, M. *Matemática*. 2ª. ed. São Paulo: Editora Moderna, 2010. v. 2. Citado na página 60.

SCHEINERMAN, E. R. *Matemática Discreta*. 2ª. ed. São Paulo: CENGAGE Learning, 2011. Citado na página 46.

SILVEIRA Ênio. *Matemática: Compreensão e Prática*. 3ª. ed. São Paulo: Editora Moderna, 2016. v. 6. Citado na página 41.

STRANG, G. *Álgebra Linear e Suas Aplicações*. 4ª. ed. São Paulo: CENGAGE Learning, 2010. Citado na página 30.

Apêndices

APÊNDICE A – Algoritmos dos experimentos

No apêndice, mostramos os códigos escritos em linguagem Python para executar os algoritmos do Capítulo 3.

A.1 Potenciação

Segue abaixo os códigos em Python para os Algoritmos 1 e 2. Em cada um deles foi definido um contador do número de multiplicações executadas. Além disso, gerou-se, de forma aleatória, uma base fixa $a \in \mathbb{N}$ no intervalo de 1 a 100 e uma amostra com 100 valores aleatórios para o expoente $n \in \mathbb{N}$ nos intervalos [1, 10], [1, 100], [1, 1000] e [1, 10000]. Por fim, para cada intervalo, o programa retornou dois valores: o limite superior do intervalo e o número médio de multiplicações realizadas em cada amostra. Isto nos deu uma base para confirmar as previsões da função de complexidade assintótica e também para comparar os dois algoritmos.

```
1  import random
2  #import matplotlib.pyplot as pl
3
4  def potenciacao(a, n, _):
5
6    x = 1
7    mults = 0
8
9    for i in xrange(n):
10     x = x * a
11     mults += 1
12
13   return (x, mults)
14
15 def potenciacao2(a, n, mults):
16
17   if n == 1:
18     return (a, mults)
19   elif n % 2 == 0:
20     pot, m = potenciacao2(a, n/2, mults)
21     return (pot * pot, m + mults + 1)
22   else:
23     pot, m = potenciacao2(a, (n - 1)/2, mults)
24     return (pot * pot * a, m + mults + 2)
```

```
25
26 limits = [10, 100, 1000, 10000]
27 samples = 100
28 max_a = 1000
29
30 def exp(f):
31   data = []
32
33   for limit in limits:
34
35     total = 0
36
37     for sample in xrange(samples):
38       a = random.randint(1, max_a)
39       n = random.randint(1, limit)
40
41       _, mults = f(a, n, 0)
42       total += mults
43
44     mean = 1.0*total / samples
45     data.append((limit, mean))
46
47   return data
48
49 data1 = exp(potenciacao)
50 data2 = exp(potenciacao2)
51
52 x1 = [x for x, y in data1]
53 y1 = [y for x, y in data1]
54
55 print data1
56 print x1
57 print y1
58
59 x2 = [x for x, y in data2]
60 y2 = [y for x, y in data2]
61
62 print data2
63 print x2
64 print y2
65
66 #pl.plot(x1, y1)
67 #pl.plot(x2, y2)
68 #pl.savefig('teste.png')
69 #pl.show()
```

A.2 Máximo Divisor Comum

A seguir, apresentamos os códigos em *Python* para os Algoritmos 3 e 4. Em cada um deles foi inserido um contador do número de divisões realizadas pelos algoritmos.

```
1  #!/usr/bin/python
2  # -*- coding: utf-8 -*-
3  import sys
4  import math
5  import random
6  import matplotlib.pyplot as plt
7
8
9  def load_primes():
10
11   with open('primes.txt') as f:
12     primes = [int(x) for x in f.readlines()]
13
14   return primes
15
16
17 def MDC(a, b, primes):
18   if b == 0:
19     return (a, 0)
20   else:
21     d, m = MDC(b, a % b, primes)
22     return (d, m + 1)
23
24
25 def MDC2(a, b, primes):
26
27   d = 1
28   i = 0
29   total = 0
30
31   if a == 0:
32     return (b, 0)
33
34   if b == 0:
35     return (a, 0)
36
37   while i < len(primes):
38
39     p = primes[i]
40
41     if p > a and p > b:
42       return (d, total)
43
```

```
44      ra = a % p
45      rb = b % p
46
47      total += 2
48
49      if ra == 0 and rb == 0:
50        a /= p
51        b /= p
52        d *= p
53        total += 2
54      elif ra == 0:
55        a /= p
56        total += 1
57      elif rb == 0:
58        b /= p
59        total += 1
60      else:
61        i += 1
62
63    return (d, total)
64
65
66  def experimento(f, g):
67
68    primes = load_primes()
69    limits = [10 ** x for x in xrange(13)]
70    samples = 100
71
72    data_f = []
73    res_f = []
74
75    data_g = []
76    res_g = []
77
78    for limit in limits:
79
80      inputs = [(random.randint(0, limit), random.randint(0, limit)
       ) for x in xrange(samples)]
81      scenario = [(f, data_f, res_f), (g, data_g, res_g)]
82
83
84      for func, data, res in scenario:
85        total = 0
86
87        for a, b in inputs:
88          if a == 0 and b == 0:
89            a = 1
```

```
90
91      #       print 'calculando mdc({},{}), funcao {}'.format(a, b,
         func)
92              d, divs = func(a, b, primes)
93              total += divs
94              res.append((a, b, d))
95
96          mean = 1.0 * total / samples
97          data.append((limit, mean))
98
99      for i in xrange(len(res_f)):
100       if res_f[i] != res_g[i]:
101         a, b, d = res_f[i]
102         A, B, D = res_g[i]
103         print "Erro no calculo do MDC! a = {}, b = {}, d = {}, A =
         {}, B = {}, D = {}".format(a, b, d, A, B, D)
104         sys.exit(-1)
105
106     return (data_f, data_g)
107
108
109 if __name__ == '__main__':
110
111     data1, data2 = experimento(MDC, MDC2)
112
113     x1 = [math.log10(x) for x, y in data1]
114     y1 = [y for x, y in data1]
115
116     x2 = [math.log10(x) for x, y in data2]
117     y2 = [y for x, y in data2]
118
119     print data1
120     print data2
121
122     euclides=plt.plot(x1, y1, '--', label=u'Pelo Algoritmo de
         Euclides')
123     decomposicao=plt.plot(x2, y2, label=u'Por decomposicao em
         fatores primos')
124     plt.xlabel('Log (a)', fontsize=12)
125     plt.ylabel(u'Numero de divisoes', fontsize=12)
126     plt.legend(bbox_to_anchor=(0.05, 1), loc=2, borderaxespad=1.3)
127     axes = plt.gca()
128     axes.set_xlim([-0.5, 12.5])
129     axes.set_ylim([-1000, 150000])
130     plt.savefig('euclides2.png')
131     plt.show()
```

A.3 Determinantes

Por fim, apresentamos os códigos em *Python* para os Algoritmos 7 e 8. Em cada código foi inserido um contador do número de multiplicações realizadas pelos algoritmos.

```
1  #!/usr/bin/python
2  # -*- coding: utf-8 -*-
3  import sys
4  import math
5  import random
6  import numpy as np
7  import matplotlib.pyplot as pl
8
9
10 def random_matrix(n, limit = 10):
11
12   A = np.zeros((n, n))
13
14   for x in xrange(n):
15     for y in xrange(n):
16       A[x][y] = random.randint(-limit, limit)
17
18   return A
19
20
21 def cofatores(A):
22
23   if A.size == 1:     # Matriz 1 x 1
24     return (A[0][0], 0)
25   else:
26     n, _ = A.shape
27
28     i = 0       # Linha 0
29
30     det = 0
31     mults = 0
32
33     for j in xrange(n):
34
35       B = np.zeros((n - 1, n - 1))
36
37       s = 0
38       r = 0
39
40       for x in xrange(n):
41
42         if x == i:
43           continue
```

```
44
45          for y in xrange(0, n):
46
47            if y == j:
48              continue
49
50            #print 'B =', B
51            #print 'A =', A
52            #print 'i = {}, j = {}, Copiando elemento A[{}][{}]
      para B[{}][{}]'.format(i, j, x, y, s, r)
53            B[s][r] = A[x][y]
54            r += 1
55
56          s += 1
57          r = 0
58
59        d, m = cofatores(B)
60        mults += m
61        det += (1 - 2 * ((i + j) % 2)) * A[i][j] * d
62        mults += 3
63
64      return (det, mults)
65
66
67  def gauss(A):
68
69    n, _ = A.shape
70    total = 0
71
72    for x in xrange(n):
73
74      pivot = -1
75      k = x
76
77      while k < n and pivot == -1:
78        if A[k][x] != 0:
79          pivot = k
80          break
81        else:
82          k += 1
83
84      if pivot == -1:
85        return (0, total)
86
87      if pivot != x:
88
89        for y in xrange(x, n):
```

```
90          A[x][y], A[pivot][y] = A[pivot][y], A[x][y]
91
92        for z in xrange(x + 1, n):
93          for y in xrange(n - 1, x - 1, -1):
94            #print 'Atualizando A[{}][{}] a partir de A[{}][{}], pivo
          A[{}][{}]'.format(z, y, x, y, x, x)
95            A[z][y] -= (A[x][y] * A[z][x])/A[x][x]
96            total += 1
97
98      det = 1
99
100     for x in xrange(n):
101       det *= A[x][x]
102
103     #print 'U =', A
104     return (round(det), total)
105
106
107 def experimento(f, g):
108
109     limits = range(1, 9)
110     samples = 10
111
112     data_f = []
113     res_f = []
114
115     data_g = []
116     res_g = []
117
118     for limit in limits:
119
120       inputs = [random_matrix(limit) for x in xrange(samples)]
121       scenario = [(f, data_f, res_f), (g, data_g, res_g)]
122
123       for func, data, res in scenario:
124         total = 0
125
126         for A in inputs:
127           det, mults = func(A)
128           total += mults
129           res.append((det, A))
130
131         mean = 1.0 * total / samples
132         data.append((limit, mean))
133
134     #for i in xrange(len(res_f)):
135     #  if res_f[i] != res_g[i]:
```

```
136     #     d, A = res_f[i]
137     #     c, B = res_g[i]
138     #     print "Erro no calculo do determinante! d = {}, A = {}, c
          = {}, B = {}".format(d, A, c, B)
139     #     sys.exit(-1)
140
141     return (data_f, data_g)
142
143
144 if __name__ == '__main__':
145
146     random.seed(1)
147
148     data1, data2 = experimento(cofatores, gauss)
149
150     x1 = [x for x, y in data1]
151     y1 = [y for x, y in data1]
152
153     x2 = [x for x, y in data2]
154     y2 = [y for x, y in data2]
155
156     print data1
157     print data2
158
159     cofatores=pl.plot(x1, y1, '--', label='Por cofatores')
160     escalonamento=pl.plot(x2, y2, label='Por escalonamento')
161     pl.xlabel(u'Dimensao da matriz', fontsize=14)
162     pl.ylabel(u'Numero de multiplicacoes', fontsize=14)
163     pl.legend(bbox_to_anchor=(0.05, 1), loc=2, borderaxespad=1.5)
164     axes = pl.gca()
165     axes.set_xlim([0, 8.5])
166     axes.set_ylim([-2000, 200000])
167     pl.savefig('matrizes.png')
168     pl.show()
```