Relatório Final - Testes de Primalidade

Gervásio Protásio dos Santos Neto[*†]

5 de abril de 2016

Iniciação Científica em Criptografia
Orientador: Prof. Dr. Routo Terada

Instituto de Matemática e Estatística - IME USP
Rua do Matão 1010
05311-970 Cidade Universitária, São Paulo - SP

*Número USP: 7990996 e-mail: gervasio.neto@usp.br

†Com bolsa de Iniciação Científica do Conselho Nacional de Desenvolvimento Científico e Tecnológico.

Sumário

1 Introdução

G.H. Hardy, matemático de Cambridge, mentor do indiano Srinivasa Ramanujan e um dos maiores matemáticos do século XX, tinha uma opinião sobre Teoria dos Números insustentável atualmente. Autor de resultados importantes na área, julgava-a abstrata, sem aplicabilidade em atividades humanas corriqueiras, interessante apenas por sua beleza. Ele fala em seu livro sobre Estética Matemática, *A Defesa de um Matemático* (original: *A Mathematician's Apology*):

"[...] tanto Gauss e matemáticos menores podem estar justificados de se regozijar que há uma ciência - Teoria dos Números - [...] cujo próprio distanciamento das atividades humanas ordinárias deve mantê-la gentil e limpa."

Contudo, a partir principalmente das décadas de 1960 e 1970, o campo da Criptografia foi se tornando um contra-exmplo para a crença de Hardy. O Algorimo de Encriptação de Rabin, o Algoritmo RSA, o Protocolo de Troca de Chaves de Diffie-Hellman, o Criptossistema de ElGamal são todos exemplos de aplicações de Álgebra e Teoria dos Números.

Alguns dos resultados mais importantes de Teoria dos Números, no que diz respeito à Criptografia, se relacionam com números primos. Esse conjunto infinito de números possuem diversas aplicações na área. Por exemplo, são fundamentais para o RSA e para Criptossistemas de Curvas Elípticas (CCEs), ambos amplamente utilizados para encriptação de dados e dependendentes de primos para funcionarem corretamente e proverem algum tipo de segurança.

Então, dada a importância que números primos têm, é natural que busque-se formas rápidas e eficientes de encontrá-los. Mais especificamente, é interessante que, dado um número, sejamos capazes de rapidamente determinar se trata-se ou não de um número primo. Com tal finalidade foram desenvolvidos diversos algoritmos (testes de primalidade) que, fundamentando-se em resultados da Teoria dos Números, são capazes de determinar se um número é primo (ou se o é com alta probabilidade).

Este projeto de Iniciação Científica, com apoio do Conselho Nacional de Desenvolvimento Científico e Tecnológico (CNPq), destina-se ao estudo e implementações de diversos testes de primalidade, procurando também desenvolver o arcabouço matemático que prova sua correção, a realização da análise de seu consumo de tempo e a busca de possíveis ideias e heurísticas para melhorar seu funcionamento.

No restante deste relatório descreveremos melhor o objetivo do projeto, as formas de implementação dos testes, os resultados obtidos, os problemas encontrados durante o desenvolvimento e e as conclusões obtidas ao longo do processo.

2 Objetivos

Como citado na Introdução, o projeto tem como tema Testes de Primalidade e se propõe a:

- Estudar a fundamentação teórica que possibilita alguns testes importatantes;
- Realizar a análise de sua eficiência;
- Implementá-los para chegar a algumas conclusões sobre as reais eficácias e aplicabilidades do teste;
- Quando relevante, tentar propor novas ideias, heurísticas ou formas de implementação que possibilitem seu melhoramento

Os testes que escolhidos o foram por corresponder a algum dos seguintes critérios: relevância teórica, aplicabilidade prática, importância histórica.

Ignorou-se o teste trivial de divisões sucessivas e começou-se por procurar entender um Teste baseado no Teorema de Wilson [2]. Foi escolhido pois o teorema no qual se fundamenta é um resultado clássico de Aritmética Modular. Tal teste, apesar de meritar um pouco mais de atenção, ainda possui pouca profundidade e eficiência.

Um grande foco foi dado para o entendimento de um teste probabilístico, o teste de Miller-Rabin [1], baseado na inversão do Pequeno Teorema de Fermat. Sua correção é demonstrada por resultados profundos de Teoria dos Números e o algoritmo mostra-se eficiente.

O Teste AKS [4] também foi estudado por ter grande importância teórica, sendo o primeiro teste de primalidade determinístico e em tempo polinomial no número de bits do número (ou seja, polinomial em $O(\log n)$. Sua implementação foi feita e surtiu resultados interessantes.

Por fim, buscamos avaliar a aplicabilidade de curvas elípticas a testes de primalidade, principalmente o protocolo de autenticação proposto por Goldwasser e Kilian [5].

Não obstante, pretendemos também realizar um paralelo com a área de Complexidade Computacional e entender sob a óptica dessa ciência os principais problemas e algoritmos relacionados com os testes de primalidade.

3 Metodologia

O projeto foi inicialmente desenvolvido da seguinte forma:

Primeiro, foram selecionados os testes a serem estudados. Foram escolhidos o Teste de Wilson, o Teste de Miller-Rabin, o AKS, o Teste de Goldwasser-Kilian [3]. Cada teste será descrito com mais detalhes nas seções que seguem.

Uma vez que os testes a serem estudados foram selecionados, buscou-se entender os principais teoremas que os fundamentam e provar resultados importantes relacionados à aplicabilidade deles.

Com a fundamentação matemática completa, buscou-se implementar alguns destes testes como programas executáveis e medir empiricamente sua eficiência. Escolheu-se fazê-lo em linguagem C, dado o baixo *overhead* dela. Os códigos desenvolvidos encontram-se no Apêndice [1].

Todos os teste foram realizados em um notebook MacBook Air, 1.7 GHz Intel Core i5, memória de 4GB 1333 MHz DDR3, sistema operacional OS X Mavericks.

Com posse de resultados empíricos sobre a eficiência do algoritmo, confrontou-se os dados reais com o desempenho previsto em uma análise formal deste.

Por fim, reuniram-se conclusões sobre eficiência e aplicabilidade dos algoritmos, problemas e dificuldades de implementação e propõe-se ideias que poderiam levar a sua melhoria.

Vale mencionar que um teste utilizado para melhor entender a riqueza do campo, mas não estudado com profundidade foi o Teste de Lucas-Lehmer para primos de Mersenne. Tal teste é o utilizado pelo projeto *Great Internet Mersenne Prime Search* para achar novos primos de Mersenne e foi o responsável por achar o maior número primo conhecido atualmente.

Nas partes seguintes deste relatório irá-se descrever cada um dos testes escolhidos, os alicerces matemáticos destes, a experiência de implementá-los, os resultados sobre sua eficiência e as conclusões finais sobre eles.

[1] Tais códigos podem também ser encontrados em https://github.com/gnsantos/primality

4 Testes

4.1 Teste de Wilson

4.1.1 Fundamentação Matemática

O Teste de Wilson baseia-se no Teorema homônimo e que afirma:

Teorema 1. *p é primo* $\iff (p-1)! \equiv -1 \bmod p$.

Demonstração. ($\rightarrow$) Se p for um número primo, sabe-se que $\mathbf{Z}_p$ é um corpo. Portanto, todo elemento não nulo possui um inverso. Tem-se ainda que o inverso de 1 é 1 e o de -1 é -1 (isso se dá pois em $\mathbf{Z}_p$ a equação $a^2 \equiv 1 \bmod p$ possui apenas duas raízes).
Dessa forma, ao calcularmos $(p-1)! \bmod p$, podemos associar cada número ao seu inverso. Teríamos: $1.(a_1.{a_1}^{-1}) \ldots (a_{\frac{p-1}{2}}.{a_{\frac{p-1}{2}}}^{-1}).(-1) = 1.1. \ldots 1.(-1) =$ -1.

($\leftarrow$) Provemos que se n é um número composto, então $(p-1)! \neq -1 \bmod p$.
Observa-se que para o menor composto, 4, tem-se: $3! \equiv 2 \bmod 4$. Para demais compostos tem-se dois possíveis casos:

1. $n = ab$, $a \neq b$; $2 \leq a < b \leq (n-2)$: Nesse caso tem-se $(n-1)! = 1. \ldots a. \ldots b. \ldots (n-1)$, logo $n|(n-1)! \rightarrow (n-1)! \equiv 0 \bmod n$
2. $n = p^2$, para algum primo $p > 2$: Nesse caso, temos que $2p < p^2 = n$. Então, $(n-1)! = 1. \ldots q. \ldots 2q. \ldots (n-1)$. Logo, $n|(n-1)!$. Portanto, $(n-1)! \equiv 0 \bmod n$.

□

4.1.2 Algoritmo Resultante

Criar um algoritmo que retira sua correção do teorema de Wilson é imediato. Basta estabelecer uma função que calcula o fatorial modular de um número e então checar se $(n-1)! \equiv -1 \bmod n$. Temos:

Input: $n \in \mathbf{Z}$

1. $x \leftarrow (n-1)! \bmod n$
2. Se $x \equiv -1 \bmod n$ **retornar PRIMO**
3. Senão, **retornar COMPOSTO**

4.1.3 Eficiência e aplicabilidade do algoritmo

O teste de Wilson, apesar de interessante do ponto de vista teórico e de possuir uma implementação simples e rápida, não é viável como um teste de primalidade na realidade.

O cálculo do fatorial é uma sub-rotina demorada e que demora $O(n)$ execuções. Mas temos que ter em mente que o comprimento da entrada considerada é $O(\log n)$, o tamanho binário do número sendo testado. Sobre esse ponto de vista, o teste de Wilson possui consumo de tempo exponecial no tamanho de

sua entrada.

Dados empíricos corroboram o resultado analítico. Na prática, o algoritmo mostrou-se demasiadamente lento, levando em média meio minuto para decidir a primalidade de números relativamente pequenos (mais especificamente de $2^{31} - 1$, o maior inteiro que é suportado pelo formanto **long int** do C, e um primo de Mersenne).

Um possível benefício deste teste é que, diferentemente de testes como o de Miller-Rabin, ele é determinístico. Contudo, quando testes probabilísticos (discutidos a diante) podem ter precisão arbitrária e existem testes polinomiais determinísticos (AKS), esse benefício não justifica o uso do Teste de Wilson.

4.2 Teste de Miller-Rabin

4.2.1 Fundamentação Matemática

O teste de Miller-Rabin é um algoritmo probabilístico para determinação da primalidade de um dado número. Escolheu-se estudá-lo, e não o (também probabilístico) teste de Solovay-Strassen [2], pois este é menos eficiente, com probabilidade de erro de 1/2 (o dobro da do teste de Miller-Rabin).

O teste baseia-se essencialmente no inverso do Pequeno Teorema de Fermat. Este diz:

Teorema 2. ***(Pequeno Teorema de Fermat)*** *Se p é um número primo, então $\forall a \in \mathbf{Z}_p$, $p \nmid a$, $a^{p-1} \equiv 1 \bmod p$.*

Demonstração. Seja $a \in \mathbb{Z}_p$, $a \neq 0$. Como $\mathbb{Z}_p$ é um corpo, o conjunto $\{a, 2a, \ldots, (p-1)a\}$ é um permutação de $\{1, 2, \ldots, (p-1)\}$.

Sendo assim, temos que $a.2a.\ldots.(p-1)a \equiv 1.2.\ldots.(p-1) \bmod p$. Assim, $a^{p-1}(p-1)! \equiv (p-1)! \bmod p$. Pelo **Teorema 1**, isso significa que $-a^{p-1} \equiv -1 \bmod p$. Ou, de forma mais clara, $a^{p-1} \equiv -1 \bmod p$.

□

Então, um teste baseado na inversa desse teorema escolheria um número em $\mathbb{Z}_p$ e verificaria se o resultado módulo p é congruente a 1. Contudo, existem números que "enganam este teste".

Definição 1. *Um inteiro n é dito um* ***pseudoprimo na base b*** *se n é composto, mas $\exists b \in \mathbb{Z}$ tal que $b^{n-1} \equiv 1 \bmod n$.*

Alguns resultados sobre pseudoprimos são:

Lema 1. *Se n é um pseudoprimo nas bases b_1 e b_2 então n é pseudoprimo nas bases $b_1 b_2$ e ${b_1}^{-1}$.*

Demonstração. Temos ${b_1}^{n-1} \equiv 1 \bmod n$ e ${b_2}^{n-1} \equiv 1 \bmod n$. Então, ${b_1}^{n-1}{b_2}^{n-1} \equiv 1 \bmod n$. Logo, $(b_1 b_2)^{n-1} \equiv 1 \bmod n$. Portanto, n é um pseudoprimo na base $b_1 b_2$.

Temos também que $b_1 {b_1}^{-1} \equiv 1 \bmod n$. Então, $(b_1 {b_1}^{-1})^{n-1} \equiv 1 \bmod n$. Como $(b_1 {b_1}^{-1})^{n-1} \equiv b_1^{n-1}({b_1}^{-1})^{n-1} \equiv ({b_1}^{-1})^{n-1} \bmod n$. Isso implica que $({b_1}^{-1})^{n-1} \equiv 1 \bmod n$. Logo, n é um pseudoprimo na base ${b_1}^{-1}$.

□

Teorema 3. *Seja n um número composto. Se existe ao menos um $b \in \mathbb{Z}_n$ tal que $b^{n-1} \neq 1 \bmod n$, então n* ***não*** *será um pseudoprimo em pelo menos metade das possíveis bases.*

Demonstração. Sejam $\{b_1, b_2, \ldots, b_s\}$ as bases nas quais n é um pseudoprimo. Seja c um base na qual n não é um pseudoprimo. Então n também não pode ser pseudoprimo na base cb_i, pois , pelo **Lema 1**, isso implicaria que n é um pseudoprimo na base $cb_i.{b_i}^{-1} = c$, o que não é o caso (por hipótese).

Portanto, n não é pseudoprimo nas bases $\{cb_1, cb_2, \ldots, cb_s\}$. Ou seja, n não é um pseudoprimo para, pelo menos, o mesmo número de bases que o aceitam como pseudoprimo. □

O **Teorema 3** parece ser uma boa garantia de que o teste baseado apenas no inverso do **Teorema 1**, mesmo falhando para alguns casos, falha em no máximo metade deles e poderia ser utilizado como um teste probabilístico de precisão moderada. Contudo, exitem números que são pseudoprimos para todas as bases.

Definição 2. *Um **número de Charmichael** é um inteiro que é um pseudoprimo para toda base $b \in \mathbb{Z}$.*

Um número de Charmichael, n, seria então sempre capaz de enganar completamente um teste que simplemente observa para alguns $b \in \mathbb{Z}$ se $b^{n-1} \equiv 1 \bmod n$ e seria declarado primo. E apesar de raros, os números de Charmichael são infinitos [7].

Precisa-se, então, de um teste mais preciso para que os resultados de Fermat possam ser utilizados na determinação de primalidade.

O que o teste de Miller-Rabin faz é valer-se do fato de que, pelo Teorema de Legendre, ± 1 são as únicas raízes de $x^2 \equiv 1 \bmod m$, onde $m = p^s$, para $s \in \mathbb{Z}$ e p primo.

Isso nos leva a uma outra classe de pseudoprimos:

Definição 3. *Seja n um número ímpara composto tal que $n = 2^s t + 1$, com t ímpar. Seja $b \in \mathbb{Z}_n$, $b \neq 0$. Dizemos que n é um **pseudoprimo forte na base b** se*

1. $b^t \equiv 1 \bmod n$, *ou;*
2. $\exists r$, $0 \leq r < s$, *tal que* $b^{2^r} t \equiv -1 \bmod n$

O que essa definição implica é que pseudoprimos fortes são números compostos capazes de enganar um teste que busca por raízes não triviais da unidade módulo n.

Algo que é fácil observar é que o conjunto de pseudoprimos fortes de uma dada base está contido no conjunto de pseudoprimos dessa base. Ou seja:

Teorema 4. *Se $n = 2^s t + 1$ é um pseudoprimo forte na base b, então n é um pseudoprimo na base n.*

Demonstração. Se n é um pseudoprimo forte para a base b temos duas possibilidades:

1. $\underline{b^t \equiv 1 \bmod n}$
   Nesse caso, podemos elevar os dois lados da congruência modular ao quadrado. O lado direito mantem-se sempre 1, já o lado esquerdo muda após cada quadratura. Após a primeira será b^{2t}, após a segunda será $b^{2^2} t$ e assim sucesivamente até a s-ésima quadratura em que teremos $b^{2^s} t = b^{n-1}$. Isso significa que $b^{n-1} \equiv 1 \bmod n$, logo n é um pseudoprimo na base b. s

2. $b^{2^r}t \equiv -1 \bmod n$, para algum $0 \leq r <$
   Nesse caso podemos novamente quadrar sucessivamente os dois lados da congruência. Após a primeira iteração teremos a congruência $b^{2^{r+1}}t \equiv 1 \bmod n$, com $r+1 \leq s$. Repetimos isso até obtermos do lado direito $b^{2^s}t = b^{n-1}$, que será então congruente a 1. Logo, n será um pseudoprimo na base b.

□

A melhora obtida de usar-se a **Definição 3** no lugar da **Definição 1** é que não existem análogos dos Números de Charmichael para pseudoprimos fortes. Na verdade, um dado ímpar composto n será pseudoprimo forte em relativamente poucas bases $0 < b < n$.

O seguinte teorema (cuja prova encontra-se em [3]) nos dá uma cota superior para o número de possíveis bases nas quais um dado inteiro composto pode ser um pseudoprimop forte.

Teorema 5. *Se n é um número ímpar composto, então é um pseudoprimo forte em no máximo $(n-1)/4$ bases b, $0 < b < n$.*

O que o **Teorema 5** nos diz é que, se n é composto, então será pseudoprimo forte em no máximo 25% das possíveis bases. Logo, se temos que $n = 2^s t + 1$ e que existe um $0 < b < n$ tal que $b^t \equiv 1 \bmod n$ ou $b^{2^r}t \equiv -1 \bmod n$, então a probabilidade de n *não* ser primo (ou seja, *ser* um pseudoprimo forte na base b) é menor ou igual a 1/4.

De posse destes resultados, somos capazes de construir o algoritmo que constitui o teste de Miller-Rabin.

4.2.2 Algoritmo

O algoritmo é como segue (aqui em pseudo-código. Uma implementação em C encontra-se no Apêndice):

Entrada: $n, k \in \mathbb{Z}$, n ímpar, $n, k > 0$.
Saída: COMPOSTO (Pr = 1) ou PRIMO (Pr $\geq 1 - (\frac{1}{4})^k$)

1. $n - 1 = 2^s t$
2. **para**: *teste* $= 1, \ldots, k$ **faça**:
   (a) Escolher $a \in (2, n-2)$
   (b) $x_0 = a^t \bmod n$; $x_1 = (r_0)^2 \bmod n$
   (c) **para**: $j = 1, \ldots, s$ **faça**:
      i. **se** $x_j \equiv 1 \bmod n$ e $x_{j-1} \not\equiv \pm 1 \bmod n$: **retornar** COMPOSTO
      ii. $x_{j+1} = (x_j)^2 \bmod n$
   (d) **se** $x_s \neq 1$: **retornar** COMPOSTO
3. **retornar** PRIMO

Essencialmente, o algoritmo aplica quadraturas sucessivas em busca de uma raiz não trivial da unidade. Se uma é encontrada, sabe-se com certeza que o número de entrada não é primo. Caso não se tenha achado uma raiz não trivial, pelo **Teorema 5**, sabe-se que n é primo com probabilidae 3/4 (pois é um pseudoprimo na base escolhida com probabilidade 1/4).

4.2.3 Relação com Complexidade

Definição 4. *A classe de complexidade **BPP** (bounded-error probabillistic polynomial time) é definida como a classe dos problemas de decisão para os quais existem soluções algorítmicas polinomiais não-determinísticas que produzem a resposta errada com probabilidade inferior à $\frac{1}{3}$ para qualquer instância.*

O Algoritmo é polinomial em k (a precisão do teste) e no tamanho da entrada [1].

Por si só, o Teste de Miller-Rabin colocaria o problema de decisão da primalidade de um número na classe BPP, definida anteriormente. Dado que no máximo $\frac{1}{4}$ das possíveis bases poderiam ser testemunhas da primalidade de um número composto, pelo **Teorema 4**, a probabilidade de erro do algoritmo (com uma única iteração) é de $p \leq \frac{1}{4}$.

Vale notar que, caso valha a Hipótese de Riemann Estendida, é possível provar que pode-se ter certeza da primalidade de um número checando-se apenas uma quantidade finita de bases. Mais especificamente, seria possível provar que um número composto n falha o teste em pelo menos uma base $b < 2\log n$. [2][3]

4.2.4 Implementação e Resultados

O teste de Miller-Rabin mostrou-se bastante eficiente em sua implementação feita em linguagem C.

Foram testadas diversas possíveis entradas para o algoritmo para avaliar sua eficácia. Foram usadas dois tipos de entradas:

1. **Individuais:** um único número foi passado para o programa, que calculou, com probabilidade padrão de 1 - $\frac{1}{2^{30}}$, se esse número era primo ou não.

   Foram então estudadas as médias do tempo necessário para chegar a essa conclusão. Para primos da ordem de 10^9 (os maiores suportados pelo **long int** do C), o tempo médio foi em torno de 61,7 nanossegundos.

   Os tempos para números compostos foram negligenciáveis. O que percebeu-se foi que, na prática, o algoritmo determinava rapidamente (logo nas primeiras iterações) se um número era composto e demorava-se apenas para primos (uma vez que nestes casos, o algoritmo não era interrompido, tendo todos seus passos realizados).

2. **Sequências:** nesse caso, foi passada ao programa uma lista de números, e desejava-se saber quantos primos havia na lista (e quais eram eles). As médias de tempo foram plotadas e observou-se o que já se tinha visto

no caso das entradas individuais: o consumo de tempo do algorítmo, por menor que fosse, ainda era significantemente maior quando um número era primo do que quando era composto.

4.3 Algoritmo AKS

O AKS foi desenvolvidos pelos indianos M. Agrawal, N. Kayal e N. Saxena e publicado no artigo de 2002, *PRIMES is in P* [4]. O objetivo do algoritmo não é ser extremamente eficiente, mas sim demonstrar que é possível demonstrar a primalidade de um número de forma determinística em tempo polinomial.

4.3.1 Fundamentação Matemática

O seguinte resultado é a base da correção do AKS. Sua prova encontra-se em [4].

Teorema 6. *p é primo $\iff \forall a, p \geq 2$, tais que mdc(a,p) = 1, vale $(x+a)^p \equiv x + a^p \bmod p$.*

A ideia é que é construído um polinômio $p(x)$ relacionado com n [1] e então, para todo a de um conjunto predeterminado (provado em [4] ter cardinalidade no máximo $\sqrt{\varphi(r)} \log n$), checa-se a igualdade estabelecida pelo **Teorema 5**.

Se essa igualdade for validada em todos os testes, conclui-se que n é primo. Caso algum teste falhe, n é composto.

4.3.2 Algoritmo

Segue o algoritmo, como descrito em [4]:

Entrada: $n > 1$
Saída: COMPOSTO ou PRIMO

1. **se** $n = a^b, a \in \mathbb{N}, b > 1$ **return** COMPOSTO
2. r = min $\{m | o_m(n) > (\log n)^2\}$
3. **se** $1 < mdc(a, n) < n$ **return** COMPOSTO
4. **se** $n \leq r$ **return** PRIMO
5. **para** $a = 1, \ldots, \lfloor \sqrt{\varphi(r)} \log n \rfloor$
   (a) **se** $(x-a)^n \not\equiv (x^n - a) \bmod (n, x^r - 1)$ **return** COMPOSTO
6. **return** PRIMO

4.3.3 Relação com Complexidade

Como diz no título do artigo em que apareceu pela primeira vez, o AKS é responsável por colocar o problema de decisão da primalidade de um número em **P**. Onde:

Definição 5. ***P*** *é a classe dos problemas de decisão que podem ser decididos em tempo polinomial determinístico.*

A análise do algoritmo feita em [4] revela que seu consumo de tempo é de $O(\log n^{21/2})$, sendo que esse tempo poderia ser melhorado caso valessem algumas conjecturas não provadas em Teoria dos Números.

Como citado em [4], uma versão alterada do algoritmo, devido à H. Lenstra, C. Pomerance tem tempo de execução de $O(\log n^{15/2})$. [10]

4.3.4 Implementação e Eficiência

Esse teste também foi implementando na linguagem C. Contudo, o algoritmo traduzido do pseudo-código provido por [4] mostrou-se bastante ineficiente se comparado com o teste de Miller-Rabin (apesar de apresentar uma grande melhora em relação ao teste de Wilson ou o teste trivial de divisões sucessivas).

Então, modificou-se a implementação buscando melhorar a eficiência.

A primeira mudança feita foi parar de se realizar o primeiro passo do algoritmo (como descrito na seção **4.3.2**).

Feito isso, procurou-se um limite superior diferente para o laço do passo 5 do algoritmo, pois o cálculo de $\varphi(n)$ é uma atividade computacionalmente custosa.

Para tentar resolver este aparente obstáculo, testou-se as seguintes soluções:

- Criou-se um conjunto de pares ordenados $(x, \varphi(x))$, que então foi plotado e ao qual foi aplicado um algoritmo de *curve fitting* por meio do aplicativo GNUPlot do Linux. Então usou-se a função retornada pelo *curve fitting* no limite superior. Contudo, não observou-se mudanças significativas no tempo de execução.

  Devido a natureza esparsa da função $\varphi(x)$, as soluções dadas pelos algorimos do GNUPlot eram demasiadamente aproximados, logo, ineficientes para melhoras sensíveis.

  A Figura 1 mostra a distribuição de $\varphi(x)$ e a Figura 2 mostra reta apontada pelo algoritmo de curve fitting como a melhor aproximação. A **Figura 2** foi obtida por meio do *Wolfram|Alpha*, que por sua vez é feito em *Mathematica*.

- Inspirado por desigualdades que envolvem a função $\varphi(x)$ e pelo Teorema dos Números Primos, foram testadas as *bounds* $\frac{x}{\log x}$ e $\frac{x}{log(logx)}$.

  Contudo, estas foram ligeiramente menos eficientes que a limite superior original (mas por pouco, em média 1 ou 2 nanossegundos). Teoriza-se que, apesar de não termos mais que calcular $\varphi(n)$, o fato das novas bounds serem significantemente maiores que a anterior equilibrava o ganho de tempo e fazia com que seu uso não trouxesse ganhos de eficiência.

A última mudança feita levou à diminuição do consumo de tempo do algoritmo, à custa de introduzir aleatoridade nele. No passo **5(a)**, ao invés de checar

uma igualdade de polinômios, escolhe-se um valor aleatório para x e verifica-se se a igualdade se mantem.

Conclui-se que, apesar de relevante do ponto de vista teórico, principalmente no que diz respeito o campo da Complexidade Computacional, o AKS não é prático para aplicações reais, ainda sendo bem mais lento que o teste de Miller-Rabin (cerca de 2700 vezes, segundo dados empíricos).

4.4 Teste de Goldwasser-Kilian

4.4.1 Fundamentação Matemática

Definição 6. *A equação* $y^2 = x^3 + ax + b \bmod n$, *onde* $a, b \in \mathbb{Z}_n$ *é chamada* ***Equação de Weiestrass****.*

Definição 7. *Se n da equação de Weiestrass for primo, o conjunto de pares ordenados (x,y), que a satisfazem, juntamente com o ponto no infinito O, formam uma Curva Elíptica E sobre o corpo finito* $\mathbb{F}_n$. *Denota-se* $E(\mathbb{F}_n)$.

A Teoria de Curvas Elípticas é bastante rica. Contudo, nem todos seus resultados nos são relevantes no presente trabalho. A seguir, vamos apresentar algumas definições e resultados que nos ajudaram em nossos propósitos.

Definição 8. *Sejam* $P_1 = (x_1, y_1)$ *e* $P_2 = (x_2, y_2)$ *pontos da curva elíptica E definida sobre um corpo K. Podemos obter um terceiro ponto* $P_3 = (x_3, y_3)$ *que também está em E a partir de* P_1 *e* P_2. P_3 *é dita sua* ***soma****. As fórmulas para realizar a* ***adição*** *são:*

1. *Se* $x_1 \neq x_2$*:*

$$x_3 = (\frac{y_2 - y_1}{x_2 - x_1})^2 - (x_2 + x_1)$$
$$y_3 = \frac{y_2 - y_1}{x_2 - x_1}.(x_1 - x_3) - y_1$$

2. *Se* $x_1 = x_2$ *e* $y_1 \neq y_2$

$$x_3 = y_3 = \infty \textit{(ou seja, o ponto resultante é o ponto no infinito)}$$

3. *Se* $x_1 = x_2$ *e* $y_1 = y_2 \neq 0$

$$x_3 = \frac{3x_1^2 + A}{2y_1} - 2x_1$$
$$y_3 = \frac{3x_1^2 + A}{2y_1}.(x_1 - x_3) - y_1$$

4. P_2 *é o ponto no infinito*

$$P_3 = P_1$$

De posse desta operação, temos o seguinte teorema, facilmente verificável e cuja prova encontra-se em [13]:

Teorema 7. *Os pontos de uma curva elíptica E com a operação de adição da* ***Definição 8*** *formam um grupo abeliano com identidade O e o inverso de um ponto* $P = (x, y)$ *sendo* $-P = (x, -y)$

Uma vez que podemos somar um ponto da curva a si mesmo, conseguimos definir a multiplicação de pontos de uma curva elíptica por inteiros da seguinte forma:

Definição 9. *Seja $k \in \mathbb{Z}$. Então $kP = P + P + \cdots + P$ (k vezes), se $k > 0$. Se $k < 0$, $kP = (-P) + (-P) + \cdots + (-P)$ ($|k|$ vezes).*

Uma observação relevante para Criptografia é que se temos P e k podemos dobrar sucessivamente k e calcular kP em tempo $O(\lg k)$. Contudo, dado kP e P, descobrir k é dificl. Essa é a base do **Problema do Logarítmo Discreto** sobre cuvas elípticas, que fundamenta criptossistemas de curvas elípticas.

Definição 10. *Um **endomorfismo** é um isomorfismo de grupos $\alpha : E(K) \to E(K)$ tal que existem funções racionais R_1 e R_2 tais que:*

$$\alpha(x, y) = (R_1(x, y), R_2(x, y))$$

Temos que se α é um endomorfismo, por ser um isomorfismo teremos

$$\alpha(x, -y) = \alpha(-(x, y)) = -\alpha(x, y)$$

Algumas contas (detalhadas em [13]) nos levam a concluir que podemos reescrever as funções R_1 e R_2 da definição de α como

$$r_1 = p(x)/q(x)$$
$$r_2 = u(x).y/v(x)$$

respectivamente. Nota-se que $p(x), q(x), u(x)$ e $v(x)$ são polinômios.

Uma informação importante sobre um endomorfismo α é seu grau, denotado por $\deg(\alpha)$. Isto é:

Definição 11. *Se α for um endomorfismo não trivial (isto não, não mapear todos os pontos para o elemento neutro do grupo) temos que o **grau** de α é:*

$$deg(\alpha) = \max\{deg(p(x)), deg(q(x))\}$$

Definição 12. *Um endomorfismo α é dito **separável** se a derivada r_1' não for identicamente nula, ou seja, se nem $p'(x)$ ou $q'(x)$ forem identicamente nulos.*

É necessário notar que nos corpos $\mathbb{Q}, \mathbb{R}, \mathbb{C}$ todas as funções racionais não constantes possuem primeira derivada não nula. Contudo, em um corpo finito $\mathbb{F}_q$ temos que a derivada de um polinômio $p(x)$ é identicamente nula se $p(x) = cx^q$, para alguma constante c.

A seguir, vamos definir uma classe particularmente útil de endomorfismos.

Definição 13. *Seja $\mathbb{F}_q$ um corpo finito com fecho algébrico $\boldsymbol{F}$ e*

$$\phi_q : \boldsymbol{F} \to \boldsymbol{F}$$
$$x \to x^q$$

Seja E uma curva elíptica sobre $\mathbb{F}_q$. Temos que ϕ_q age sobre um ponto $(x, y) \in E$ da seguinte forma:

$$\phi_q(x, y) = (x^q, y^q) \text{ e } \phi_q(O) = O$$

*ϕ_q é chamado de **Endomorfismo de Frobenius**.*

Lema 2. *Seja E uma curva elíptica definida sobre $\mathbb{F}_q$ e $(x, y) \in E(\boldsymbol{F})$. Então:*

1. $\phi_q(x, y) \in E(\boldsymbol{F})$
2. $\phi_q(x, y) = (x, y)$ *se, e somente se,* $(x, y) \in E(\mathbb{F}_q)$

Demonstração. Como q ou é um número primo ou é a potência do primo que é a característica do corpo, temos que $(a + b)^q = a^q + b^q$. Elevando-se os dois lados da equação de Wiertrass a q e usando esse fato, juntamente com o fato de que $a^q = a \iff a \in \mathbb{F}_q$, concluímos que $\phi_q(x, y) = (x^q, y^q) \in E(\mathbf{F})$.

Além dsso, temos que:

$$\phi_q(x, y) = (x, y) \iff x^q = x \text{ e } y^q = y \iff x, y \in \mathbb{F}_q \iff (x, y) \in E(\mathbb{F}_q)$$

□

Lema 3. *ϕ_q não é separável e deg(ϕ_q) = q.*

Demonstração. No caso de ϕ_q, temos que os polinômios $p(x)$ e $q(x)$ que caracterizam $r_1(x, y)$ (como definidos anteriormente) são tais que $p(x)/q(x) = x^q$. Então temos que $p(x) = x^q$ e $q(x) = 1$. Logo, $\deg(\phi_q) = deg(\mathrm{p(x)}) = q$.

Além disso, temos que $p'(x) = qx^{q-1}$, que é identicamente nulo em $\mathbb{F}_q$), logo ϕ_q não é separável. □

Definição 14. *Seja α um endomorfismo. Então o* ***núcleo*** *do endomorfismos é*

$$Ker(\alpha) = \{P \in E : P \neq O \text{ e } \alpha(P) = O\}$$

Teorema 8. *Seja α um endomorfismo não trivial da curva elíptica E sobre $\boldsymbol{F}$. Então, se α é separável, deg(α) = |Ker(α)| e se α não for separável, deg(α) > |Ker(α)|.*

Demonstração. Seja $\alpha(x, y) = (r_1(x, y), yr_2(x, y))$ um endomorfismo com $r_1 = p(x)/q(x)$. Se α é separável, temos que $r_1' \neq 0$. Logo, $p'q - pq' \neq 0$.

Seja $S := \{x \in \mathbf{F} : (p'q - pq')(x).q(x) = 0\}$ e $r_1(S) = \{x \in \mathbf{F} : \exists x' \in S, x = r_1(x)\}$.

Seja $(a, b) \in E(\mathbf{F})$ um ponto tal que:

1. $a \neq 0$, $b \neq 0$ e $(a, b) \neq O$.
2. $\deg(\ p(x) - aq(x)\ ) = \deg(\alpha)$.
3. $a \notin r_1(S)$.
4. $(a, b) \in \text{Im } \alpha$

Como $p'q - pq' \neq 0$, temos que S será finito. Contudo Im α é infinito, pois $r_1(x)$ assi,e infinitos valores e para cada valor de x existirá um (x, y). Então, deve existir um par a, b que satisfaz nossas exigências.

Vamos mostrar que existem exatamente $\deg(\alpha)$ pontos (x_1, y_1) tais que $\alpha(x_1, y_1) = (a, b)$.

Temos que para um dado ponto vale:

$$a = r_1(x1) = \frac{p(x_1}{q(x_1}$$
$$b = r_2(x_1).y_1$$

Como $(a, b) \neq O$, temos que $r_2(x_1)$ estará definido, então y_1 será univocamente determinado. Logo, para contar o número de pares ordenados, basta contar o número de possíveis valores distintos para x_1.

Temos que $p(x) - aq(x) = 0$ tem $\deg(\alpha)$ raizes (contando multiplicidade). O que queremos mostrar é que o polinômio não possuirá raízes múltiplas. Então, suponhamos por contradição que x_0 é uma raiz múltipla. Teremos:

$$p(x_0) - aq(x_0) = 0, \qquad p'(x_0) - aq'(x_0) = 0$$
$$pq' = aqq', \qquad p'q = aqq'$$
$$pq' = p'q \rightarrow p'(x_0)q(x_0) - p(x_0)q'(x_0) = 0$$

Logo, $x_0 \in S$, o que implica que $a \in r_1(S)$, contrariando nossa escolha de a. Logo, $p(x) - aq(x) = 0$ tem $\deg(\alpha)$ raizes distintas.

Já que há $\deg(\alpha)$ pontos (x, y) tais que $\alpha(x, y) = (a, b)$, temos que $|\mathrm{Ker}(\alpha)| = \deg(\alpha)$.

Se o endomorfismo não for separávelm o polinômio $p' - aq'$ é identicamente nulo, então $p - aq$ tem raízes múltipla, logo, possuirá menos que $\deg(\alpha)$ soluções distintas, nos dando que $|\mathrm{Ker}(\alpha)| < \deg(\alpha)$.

□

Definição 15. *Sejam α, β endomorfismos e a, b inteiros. Então definimos $a\alpha + b\beta$ como o endomorfimos que mapeia um ponto $P \in E$ a $a\alpha(P)$ somado a $b\beta(P)$. Como multiplicação por inteiros é um endomorfismo, se α ou β corresponderem a multplicação por algum inteiro k escrevemos apenas $a\alpha + k$.*

Teorema 9. *Ker(ϕ_q - 1) = E($\mathbb{F}_q$).*

O endomorfismo definido equivale a aplicar o endomorfismo de Frobenius a um ponto $P \in E$ e então somar o resultado ao inverso $-P$ de P.

Corolário 1. *|E($\mathbb{F}_q$)| = deg(ϕ_q - 1).*

Demonstração. Segue da aplicação dos resultados dos teoremas 8 e 9. □

Teorema 10. *$deg(a\alpha + b\beta) = a^2 deg(\alpha) + b^2 deg(\beta) + ab(deg(\alpha + \beta) - deg(\alpha) - deg(\beta)$*

Corolário 2. *Sejam $r, s \in \mathbb{Z}$ tais que mdc(s,q) = 1. Seja $a = q + 1 - |E(\mathbb{F}_q)|$. Então $deg(r\phi_q - s) = r^2 q + s^2 - rsa$.*

Demonstração. Aplicação direta do Teorema 10. □

Todos as definições e resultados apresentados até o presente momento objetivavam provar o próximo teorema. Nota-se que os resultados que foram enunciados mas não provados tem demonstrações em [13].

Teorema 11. ***(Teorema de Hesse)*** $|q + 1 - |E(\mathbb{F}_q)|| \leq 2\sqrt{q}$.

Demonstração. Temos que $deg(r\phi_q - s) \geq 0$, para $r, s \in \mathbb{Z}$. Então, pelo Corolário 2, temos:

$$r^2q + s^2 - rsa \geq 0$$
$$q(\frac{r}{s})^2 + 1 - a(\frac{r}{s}) \geq 0$$

O conjunto dos números racionais r/s tais que $\text{mdc}(s, q) = 1$ é denso em $\mathbb{R}$. Logo, $\forall x \in \mathbb{R}$:

$$qx^2 - ax + 1 \geq 0$$

Então o descriminante $\Delta = a^2 - 4q$ será menor ou igual a 0:

$$a^2 - 4q \leq 0$$
$$|a| \leq 2\sqrt{q}$$
$$|q - 1 + |E(\mathbb{F}_q)|| \leq 2\sqrt{q}$$

□

Para aplicações de curvas elípticas em primalidade é muitas vezes necessário saber qual é a ordem do grupo formado pelos pontos da curva sobre um corpo finito (que equivale a perguntar quantos pontos no corpo satisfazem a equação da curva). Assim, o Teorema de Hesse é relevante pois nos dá um intervalo para os possíveis valores.

Agora vamos apresentar resultados diretamente relacionados a primalidade. Os seguintes resultados relevante foram retirados de [3],[5], [8] e [13].

Lema 4. ***(Critério de Pocklington)*** *Seja n um inteiro positivo. Se houver um primo p, $p > \sqrt{n}-1$, tal que $p|(n-1)$; e um inteiro a tal que $a^{n-1} \equiv 1 \bmod n$ e $mdc(a^{(n-1)/p} - 1, n) = 1$; então n é primo.*

Demonstração. Vamos assumir por absurdo que n não é primo. Então, $\exists q$ primo, $q < \sqrt{n}$, tal que $q|n$. Então $q - 1 < \sqrt{n} - 1 < p \rightarrow p > q - 1$.

Como p é primo e $q - 1 < p$, temos que $mdc(q - 1, p) = 1$. Sendo assim, p é inversível no anel $\mathbb{Z}_{q-1}$. Logo, $\exists u \in \mathbb{Z}_{q-1}$ tal que $up \equiv 1 \bmod q - 1$.

Sendo assim, $a^{(n-1)/p} \equiv a^{up(n-1)/p} \equiv a^{u(n-1)} \bmod q$. Mas por hipótese, temos que $a^{n-1} \equiv 1 \bmod n$, isso implica que $a^{n-1} \equiv 1 \bmod q$, que por sua vez implica que $a^{u(n-1)} \equiv 1 \bmod q$. Este raciocínio nos leva a concluir que $a^{(n-1)/p} \equiv 1 \bmod q$.

Contudo, isso significa que $mdc(a^{(n-1)/p} - 1, n) = q$, uma contradição. Logo, n deve ser um número primo. □

O resultado do Lema acima por si só poderia constituir um teste de primalidade, provido que fôssemos capazes de achar um primo p adequado. Contudo, caso tal primo não seja imediato, a utilidade de tal teste seria reduzida.

O teste baseado em curvas elípticas fundamenta-se em uma versão análogo do **Lema 4**.

Teorema 12. *Seja n um inteiro positivo. Seja E o conjunto de pares ordenados que satisfazem a equação de Weiestrass. Seja também m um inteiro. Suponhamos que um primo q é tal que $q|m$ e $q > (n^{1/4} + 1)^2$. Se $\exists P \in E$ tal que $mP = O$; e $(m/q)P \neq O \in E$, então n é primo.*

Demonstração. Suponhamos por contradição que n não é primo. Então existem um primo $p < \sqrt{n}$ tal que $p|n$.. Seja E' a curva elíptica dada pela mesma equação que define E, mas tomada módulo p e seja m' a ordem do grupo de E'.

Pelo Teorema de Hesse, temos que

$$m' \leq 2\sqrt{p} + 1 + p = (\sqrt{p} + 1)^2 \leq (n^{1/4} + 1) < q$$

Logo, $\text{mdc}(m', q) = 1$e exite u tal que $uq \equiv 1 \bmod m'$.

Seja $P' \in E'$ o ponto P considerado módulo p. Então, em E', teremos que:

$$\begin{aligned}(m/q)P' &= uq(m/q)P' \\ &= umP' \\ &= O\end{aligned}$$

(uma vez que mP' é obitdo da mesma forma que mP, mas em mod p no lugar de mod n)

Contudo, isto contradiz a hipótese de que $(m/q)P \neq O \bmod n$. Contudo, se repetirmos esse procedimento tomando as operações mod p, teremos que $(m/q)P' \neq O$. □

A analogia do **Teorema 12** e do **Lema 4** podem não ser imediatas, então compete a explicação:

- O m descrito no enunciado do teorema deve ser igual ao número de pontos da curva (a cardinalidade de E), dessa forma, o resto do teorema pode funcionar corretamente.

  No **Teorema 12**, o inteiro m desempenha o mesmo papel de $n - 1$ no **Lema 4**. Isso se dá porque, se n for primo, ambos são iguais às ordens de seus respectivos grupos multiplicativos. Ou seja, $n - 1$ é a ordem do grupo $\mathbb{Z}_n$ e m é a ordem do grupo E, no caso de n primo.

4.4.2 Algoritmo

A seguir, será delineado o ideia do algortimo em pseudo-código.

1. Escolhe-se 3 inteiros aleatórios $a, x, y \in \mathbb{Z}_n$.
2. Define-se $b = y^2 - x^3 - ax$. O ponto P = (x,y) deverá pertencer a curva E, definida pela equação $y^2 = x^3 + ax + b \bmod n$ e será escolhido de forma aleatória.
3. Obtem-se o inteiro m através de um método de contagem de pontos, como o Algoritmo de Schoof [11][12][13] ou o Algoritmo *Baby Step, Giant Step.* [13]
4. No caso da primalidade de n, m representa *exatamente* o número de pontos da curva.
5. Encontra-se um primo provável q tal que $m = cq$, para algum $c \in \mathbb{Z}$. Se não formos capazes de achar tal q, voltamos ao passo 1.
6. Computa-se mP e cP. Neste momento, temos as seguintes possibilidades:
   (a) Se $mP \neq O$, n é composto.
   (b) Se $cP = O$, recomeça-se o algoritmo.
   (c) Se $mP = O$ e $cP \neq O$, o Teorema 12 nos diz que n é primo, provido que q é primo.
7. Como a primalidade de n agora depende da primalidade de q, aplica-se o algoritmo para determinar a primalidade de q. Para um q suficientemente pequeno podemos usar o teste direto de procura exaustiva por um fator não trivial.

O algoritmo pode ser usado para certificar rapidamente (tempo polinomial) a primalidade de um número. Caso a última etapa do algoritmo seja utilizada em todos os possíveis valores de q, teremos um algoritmo recursivo que será chamada $O(\log n)$ vezes. O certificado devolvido é uma tupla (n, a, b, m, q) que pode ser usada para rapidamente verificar que o número n é primo através da aplicação do Teorema 12.

Além disso, como o algoritmo também certifica a primalidade do primo q utilizado no teste de n, isso significa que ele acabará por certificar não somente o primo original, mas uma sequência de primos menores, que antes eram apenas possivelmente primos.

4.4.3 Testes, Resultados e Problemas

Para testar a eficiência do algoritmo usou-se uma implementação deste feita em SAGE (*System for Algebra and Geometry Experimentation*), uma plataforma de computação matemática que é uma alternativa grátis e open-source a softwares como Maple e MATLAB. A implementação em questão foi feita por Georg Hahn e encontra-se em `http://trac.sagemath.org/attachment/ticket/10562/ecpp.py`, sendo distribuida sob licença GNU Public License (GPL).

O algoritmo mostrou-se, em geral, satisfatoriamente rápido. Foram rodados testes para verificar o tempo médio necessário para decidir a primalidade de um número (as entradas eram números primos aleatórios) e para verificar o certificado de primalidade. Para números da ordem de 10^{30} observou-se em média 2.5 segundos para obter o certificado de primalidade; caso já se tivesse um certificado, este era testado com tempo médio de 0.078 segundos. Com entradas da ordem de 10^{40} observou-se um tempo médio de 42.95 segundos para obtenção de um certificado e 0.448 segundos para a verificação de um.

Para fins de comparação, também obtivemos o tempo médio necessário para a função *is_prime*, que é padrão do SAGE, levava para decidir a primalidade de números nas ordens de grandeza de 10^{30} e 10^{40}. Para entradas da ordem de 10^{30} o tempo médio para testar a primalidade foi 0.0078 segundos. Já para entradas da ordem de 10^{30}, a media foi 0.0152 segundos.

Também portou-se a implementação do Teste de Miller-Rabin, anteriormente feita em C, para SAGE. As mudanças foram mínimas, apenas adequação de sintaxe e melhor utilização de recursos já disponíveis. Novamente, realizou-se testes para avaliar o tempo médio tomado pelo algoritmo. Observou-se que para confirmar com probabilidade $1 - 2^{-30}$ de que um número de ordem de 10^{30} é primo levou-se em média 0.0083 segundos. Para números de ordem de 10^{40}, a média foi de 0.0113 segundos.

Com base nestes números, somos levados a concluir que, apesar de não ser tão eficiente quanto testes mais estabelecidos de primalidade (como o de Miller-Rabin), o teste de Goldwasser-Kilian ainda consegue ser satisfatoriamente rápido. Além disso, é um teste determinístico que nos dá um certificado de primalidade que pode ser testado rapidamente, ambas vantagens sobre o teste de Miller-Rabin.

O que temos, do ponto de vista teórico, é que o algoritmo é penalizado na etapa 3 (a contagem de pontos). Apesar de conhecer-se o Algoritmo de Schoof ([12]) para contar os pontos de uma curva sobre um corpo finito, tal algoritmo possui difícil implementação e baseia-se em resultados profundos de Teoria dos Números e curvas elípticas. Além disso, implementações eficientes, mesmo que polinomiais, ainda possuem consumo de tempo de $O((\log n)^5)$. Algoritmos anteriores ao de Schoof tinham consumo de tempo exponecial, portanto, tornam-se impráticos para validar a primalidade de números relevantes.

A.O.L Atkin e F. Morain propuseram um teste de primalidade baseado em curvas elípticas que é uma adaptação do de Goldwasser-Kilian [8]. No algoritmo

revisado, tem-se mais cuidado na escolha da curva que será utilizida para que se possa conhecer de antemão quantos pontos ela terá. Uma implementação desta versão do algoritmo também está presente no código em SAGE que utilizamos, contudo, não foram realizados experimentos com ele.

5 Trabalhos futuros

Uma vez encerrada a segunda etapa do projeto, com o estudo dos resultados pertinentes aos testes baseados em curvas elípticas e a avaliação de suas implementações, perspectivas para o futuro incluem:

- Pesquisar e entender versões determinísticas do teste de Miller-Rabin que se valem de resultados ainda não provados em Teoria dos Números (mais especificamente, relacionados à Hipótese de Riemann);
- Estudar sobre a classe especial de curvas elípticas usadas por Morain *et al.* para contornar o problema de contar pontos em curvas elípticas sobre corpos finitos.
- Comparar possíveis ganhos de eficiência se deixarmos de ter que contar os pontos curva, ou sej,a comparar implementações da versão clássica proposta por Goldwasser e Kilian com a proposta por Morain e Atkin.

6 Conclusão

Testes de primalidade e outras formas de autenticar a primalidade de um número são essenciais na Criptografia moderna, que depende de primos para seu funcionamento correto.

Representam também uma interessante área de intersecção entre Computação e Matemática, sendo assim uma boa forma de desenvolver as áreas em conjunto.

Contudo, trata-se de um campo complexo, com diversos conceitos de difícil apreensão e que precisa ser tratado com cuidado. Sua inerente relação com problemas computacionalmente difíceis fazem com que eficiência seja algo difícil de ser obtido, pelo menos de forma imediata.

E é por isso que estudos na área são importantes. Atualmente, nossas formas de verificação de primos, enquanto satisfatórias, poderiam ser melhoradas. Trata-se de uma área importante (e antiga) e cujo estudo pode trazer muito para o desenvolvimento não somente da Criptografia e Teoria dos Números, mas da própria Matemática.

Durante o desenvolvimento deste projeto confrontei alguns dos conceitos, teoremas e algoritmos mais complexos que já havia encontrado e foi uma experiência instigante e estimulante, mesmo que em muitos momentos difícil.

7 Apêndices

7.1 Códigos

7.1.1 Exponenciação Modular

```
unsigned long modExp(unsigned long m, unsigned long e,
unsigned long n){

  unsigned long temp = 1;
  int j;
  int b;
  for(j = log2(e); j >= 0; j--){
    temp = (temp*temp)%n;
    b = (e & (1 << j)) >> j;
    if(b == 1)
      temp = (temp*m)%n;
  }
  return temp;
}
```

7.1.2 Adiquirir parametros t e s

```
void getParameters(int* t, long* c,unsigned long n){
  int x = n-1, aux = 0;
  while(x%2 == 0){
    x/=2;
    aux++;
  }
  *t = aux;
  *c = (n-1)/(pow(2,aux));
}
```

7.1.3 Teste de Miller-Rabin

```
Boolean millerRabinTest(unsigned long n, int s){
  int i,j, t=0, aux;
  unsigned long c = 0, a = 0, r0, r1;

  if((n % 2 == 0 && n != 2) || n == 1 )
    return FALSE;
  if(n == 2) return TRUE;

  getParameters(&t, &c, n);

  srand(time(NULL));
  for(i = 0; i < s; i++){
    a = rand()%(n-4) + 2;
    r0 = modExp(a, c, n);
    r1 = (r0 * r0)%n;
    for(j= 0; j < t; j++){
```

```
      if(r1 == 1 && r0 != 1 && r0 != n-1)
return FALSE;
      aux = r1;
      r1 = (r1*r1)%n;
      r0 = aux;
    }
    if(r1 != 1) return FALSE;
  }
  return TRUE;
}
```

7.1.4 Teste de Miller-Rabin - Versão SAGE

```
def getParameters(n):
  t = n-1
  s = 0
  while Mod(t,2)==0:
       s = s + 1
       t = t/2

  return Integer(s),Integer(t)

def millerRabinTest(n,k):
  if((n % 2 == 0 and n != 2) or n == 1 ):
    return False

  if(n == 2):
   return True

  [s,t] = getParameters(n)

  for i in range(k):
    a = Integer(floor(100000000000*random())%(n-4) + 2)
    r0 = power_mod(a, t, n)
    r1 = (r0 * r0)%n
    for j in range(s):
      if (r1 == 1 and r0 != 1 and r0 != n-1):
return False
      aux = r1
      r1 = (r1*r1)%n
      r0 = aux

    if(r1 != 1):
         return False

  return True
```

7.1.5 Fatorial Modular

```
unsigned long factModN(unsigned long n, unsigned long m){
  unsigned long i, x = 1;
  for(i = 1; i <= n; i++)
    x = (x*i)%m;
  return x;
}
```

7.1.6 Teste de Wilson

```
Boolean wilsonTest(unsigned long n){
  unsigned long f = factorialModN(n-1,n);
  if(f == n-1) return TRUE;
  return FALSE;
}
```

7.1.7 Obtem ordem do elemento

```
static unsigned long multOrderMod(unsigned long n, unsigned long r){
  unsigned long i, t;
  for(i = 1; i < r; i++){
    t = modExp(n,i,r);
    if(t == 1) break;
  }
  return i;
}
```

7.1.8 Obtem elemento com menor ordem

```
static unsigned long multOrderMod(unsigned long n, unsigned long r){
  unsigned long i, t;
  for(i = 1; i < r; i++){
    t = modExp(n,i,r);
    if(t == 1) break;
  }
  return i;
}
```

7.1.9 AKS

```
Boolean aksTestClassic(unsigned long n){
  int i, a, b, r, x;
  srand(time(NULL));
  r = smallestMultOrder(n);
  for(i = 1; i <= r; i++){
    b = gcd(i,n);
    if(b > 1 && b < n)
      return FALSE;
  }
  if(n <= r) return TRUE;
  for(a = 1; a <= floor( sqrt(phi(r)) * log2(n) ); a++){
    x = rand();
    if(modExp(x+a,n,n) != ((modExp(x,n,n) + a) % n) ) return FALSE;
  }
  return TRUE;
}
```

7.1.10 Código para avaliação de desempenho

```
#include <stdlib.h>
#include <stdio.h>
#include <unistd.h>
#include <time.h>
#include "rabinFunctions.h"

int main(int argc, char* argv[]){
  char c, *outputName, *inputName;
  FILE* input;
  FILE* output;
  Boolean runtime = TRUE, single = FALSE, result;
  unsigned long p = 2;
  int runs = 30, primes = 0, composites = 0;
  clock_t start, end;
  double elapsed = 0;

  while( (c = getopt(argc, argv, "t:f:o:r:y")) != -1){
    switch(c){
    case 'f':
      inputName = optarg;
      break;
    case 't':
      p = atoi(optarg);
      single = TRUE;
      break;
    case 'o':
      outputName = optarg;
      break;
    case 'y':
      runtime = FALSE;
      break;
    case 'r':
      runs = atoi(optarg);
      break;
    }
  }

  output = fopen(outputName, "w");
  input = fopen(inputName, "r");

  if(!single){
    if(input == NULL){
      puts("Unable to open file for reading.");
      help();
    }
    while(fscanf(input, "%lud", &p) != EOF){\

      start = clock();
```

```
      result = millerRabinTest(p, runs);

      end = clock();

      elapsed += (double)(end - start)/CLOCKS_PER_SEC;

      if(result){
if(!output) fprintf(stdout,
     "%ld is a prime.\n", p);

else fprintf(output, "%ld is a prime.\n", p);
primes++;
      }
      else{
if(!output) fprintf(stdout,
     "%ld is composite.\n", p);

else fprintf(output, "%ld is composite.\n", p);
composites++;
      }
    }

    fprintf(stdout, "There were %d composites and %d
      primes.\n"composites, primes);

    if(runtime) fprintf(stdout, "TIME: %f (reading
    and printing times not included)\n", elapsed);

  }

  else{
    start = clock();
    result = millerRabinTest(p, runs);

    end = clock();

    elapsed = (double)(end - start)/CLOCKS_PER_SEC;

    if(result){
      if(!output)fprintf(stdout,"%ld is a prime.\n",p);

      else fprintf(output, "%ld is a prime.\n", p);
    }
    else{
      if(!output) fprintf(stdout,"%ld is composite.\n",p);
      else fprintf(output, "%ld is composite.\n", p);
    }

    if(runtime) fprintf(stdout, "TIME: %f (reading
```

```
    and printing times not included)\n", elapsed);

  }

  fclose(input);
  fclose(output);

  return 0;
}
```

7.2 Figuras

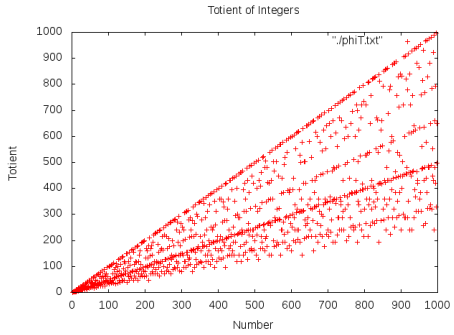

Figura 1: Distribuição de $\varphi(n)$, $n < 10^3$.

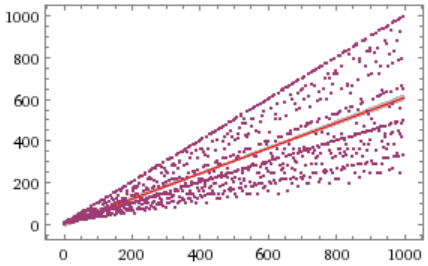

Figura 2: Distribuição de $\varphi(n)$, $n < 10^3$. A linha cheia representa uma reta obtida por Regressão Linear.

8 Bibliografia

Referências

[1] Routo Terada, *Segurança de Dados*. Blucher, São Paulo, Segunda Edição, 2008.

[2] E. Kranakis. *Primality and Criptography*. Wiley, 1986.

[3] N. Koblitz. *A Course in Number Theory and Criptography*. Springer-Verlag, 1994.

[4] M. Agrawal, N. Kayal, N. Saxena. *PRIMES is in P. Annals of Mathematics*, 160 (2):781–793, 2002

[5] S. Goldwasser, J. Kilian. *Almost all primes can be quickly certified. Proc. 18th Annual ACM Symposium on Theory of Computing*, :316–329, 1986.

[6] François Morain. *Pseudoprimes: A Survey Of Recent Results*, *Proc. Eurocode '92* , Springer (1993), pp. 207–215. 1993.

[7] W. R. Alford and Andrew Granville and Carl Pomerance. *There are infinitely many Carmichael numbers*, *ANN. OF MATH*, 1982, 140, 703–722.

[8] A. O. L. Atkin and F. Morain. *Elliptic Curves And Primality Proving*, *AMS Mathematics of Computation*, 61, 29–68, 1993.

[9] Rafael Dantas de Castro, Ricardo Dahab e Augusto Jun Devegili. *Introdução à Segurança Demonstrável. VII Simpósio Brasileiro em Segurança da Informação e de Sistemas Computacionais* , SBC, 103 – 153. 2007.

[10] H. Lenstra, C. Pomerance. Primality testing with gaussian periods. Private communication. Março, 2003.

[11] S. Goldwasser, J. Kilian. *Primality Testing Using Elliptic Curves. Journal of the ACM*, Vol 46, No. 4, Julho 1999, pp. 450-472.

[12] Gregg Musiker. Schoof's Algorithm for Counting Points on $E(\mathbb{F}_q)$. Dezembro, 2005.

[13] Washington, Lawrence C. *Elliptic Curves: Number Theory and Criptography Chapman & Hall/CRC, 2nd Edition, 2008*