CLEBER JOSÉ BATISTA JUNIOR

O ALGORITMO AKS PARA VERIFICAR PRIMALIDADE EM TEMPO POLINOMIAL

LAVRAS - MG
2010

CLEBER JOSÉ BATISTA JUNIOR

O ALGORITMO AKS PARA VERIFICAR PRIMALIDADE EM TEMPO POLINOMIAL

Monografia de graduação apresentada ao Departamento de Ciência da Computação da Universidade Federal de Lavras como parte das exigências do curso de Ciência da Computação para obtenção do título de Bacharel em Ciência da Computação.

Orientador:

Dr. Joaquim Quinteiro Uchôa

LAVRAS - MG
2010

CLEBER JOSÉ BATISTA JUNIOR

O ALGORITMO AKS PARA VERIFICAR PRIMALIDADE EM TEMPO POLINOMIAL

Monografia de graduação apresentada ao Departamento de Ciência da Computação da Universidade Federal de Lavras como parte das exigências do curso de Ciência da Computação para obtenção do título de Bacharel em Ciência da Computação.

Aprovada em ____ de _____ de _____

_____

Prof. Dr. Cláudio Fabiano Motta Toledo

_____

Prof. Pós Dr. Lucas Monteiro Chaves

_____

Prof. Dr. Joaquim Quinteiro Uchôa

Orientador

LAVRAS - MG
2010

A meu pai Cleber

A minha mãe Darly.

A minha irmã Juliana.

A minha namorada Denise

A minha república, escola da vida, Arame Farpado.

Aos meus amigos

DEDICO.

AGRADECIMENTOS

Agradeço a Deus pela inteligência, força e sabedoria nos momentos mais difíceis.

Agradeço a minha família, meu pai Cleber, minha mãe Darly e minha irmã Juliana, eles que são a estrutura da minha vida, a maior importância da minha vida, por todo apoio e paciência destinados a mim nos momentos mais difíceis.

Agradeço à minha namorada Denise, que superou durante 4 anos à distância, o nosso namoro, demonstrando total apoio, amor e carinho.

Ao meu Professor Orientador Joaquim, que me proporcionou toda informação e conhecimento necessários à execução deste trabalho, além do empenho e dedicação para a conclusão do mesmo.

Agradeço a minha república, Arame Farpado, que foi a escola da minha vida, que jamais deixou me abater ou cair perante os problemas. Eles que foram pais, mães, irmãos e amigos durante essa longa jornada.

Agradeço a todos meus tios e primo, em especial ao meu tio Humberto, tio Elimar, tia Bárbara e tia Eliana, que sempre me trataram como um filho.

Aos meus primos, Custela e Marcelo e ao meu grande amigo, irmão que não tive, Rabikó.

A todos amigos que estiveram comigo durante esta jornada e a todas bandas que participei durante todos esses anos.

RESUMO

A criptografia de chave pública é um método de encriptação muito utilizado hoje em dia, e estudos profundos foram feitos para o aperfeiçoamento da segurança desse sistema. Entretanto, essa ciência precisa ser constantemente aprimorada devido ao avanço da velocidade dos computadores, os quais estão se tornando cada vez mais capazes de fazer uma criptoanálise eficiente em um tempo relativamente curto. Um fator importante para garantir a segurança de um cifrário é o tamanho das chaves, que são números primos. Atualmente dispõe-se de algoritmos probabilísticos que acusam se um número é primo com baixíssimo percentual de erro, em tempo polinomial. O AKS é o primeiro algoritmo determinístico a executar este teste em tempo polinomial. O objetivo deste trabalho é entender porque este novo algoritmo AKS não quebrou a segurança de criptografias baseadas em números primos. Serão implementados alguns dos algoritmos mais importantes de verificação de primalidade e também o AKS, todos em C/C++. Ao final, será levantado um estudo a respeito do impacto a segurança, causado pelo AKS.

Palavras-chave**:** Algoritmo, AKS, Monte-Carlo, criptografia, número-primo, complexidade.

ABSTRACT

The public-key cryptography is a method of encryption very using nowadays, and relevant studies were made for development of security of this system. However, this science need to be continuously improved, because the fast development speed of computers, with this situation the computers can be able to execute the criptoanalyze fastly. An important point to secure the security of one encrypted message it's size of key, that are prime numbers. Nowadays there are probabilistic algorithms that execute in polynomial time and accuse if a number is prime with lowest percentual of mistake. The AKS is the first deterministic algorithmic to execute this test in polynomial time. The goal of this work is to understand why this new AKS didn't break the security of the cryptography based on prime numbers. Some of the most important algorithmic of certification for prime numbers will be implemented and also for the AKS, all of them in C/C++. At the end, a study about the impact of security caused by the AKS will be raised.

Keywords***:*** Algorithm, AKS, Miller-Rabin, cryptography, prime-number, complexity.

LISTA DE FIGURAS

FIGURA 1 O ALGORITMO MONTE-CARLO 22
FIGURA 2 O ALGORITMO AKS 23
FIGURA 3 O NOVO ALGORITMO AKS 26
FIGURA 4 PRIMEIRO PASSO DO ALGORITMO AKS 28
FIGURA 5 SEGUNDO PASSO DO ALGORITMO AKS. 28
FIGURA 6 TERCEIRO PASSO DO ALGORITMO AKS 29
FIGURA 7 ALGORITMO PARA IMPLEMENTAÇÃO DO MONTE-CARLO 30
FIGURA 8 ALGORITMO PARA IMPLEMENTAÇÃO DO CRIVO DE ERATÓSTENES 31
FIGURA 9 GRÁFICO DO TEMPO DE EXECUÇÃO DO ALGORITMO AKS 33
FIGURA 10 GRÁFICO DO TEMPO DE EXECUÇÃO DO ALGORITMO MONTE-CARLO 35
FIGURA 11 GRÁFICO DO TEMPO DE EXECUÇÃO DO CRIVO DE ERATÓSTENES 37

LISTA DE TABELAS

TABELA 1 LISTA DE ENTRADAS E TEMPO DE PROCESSAMENTO DO ALGORITMO AKS 34
TABELA 2 LISTA DE ENTRADAS E TEMPO DE PROCESSAMENTO DO ALGORITMO MONTE-CARLO 36
TABELA 3 LISTA DE ENTRADAS E TEMPO DE PROCESSAMENTO DO CRIVO DE ERATÓSTENES..... 38

SUMÁRIO

1 INTRODUÇÃO 1
2 REFERENCIAL TEÓRICO 4
2.1 CONCEITOS BÁSICOS 4
2.2 INTRODUÇÃO À TEORIA DOS NÚMEROS 7
2.3 CRIPTOGRAFIA 10
2.4 CRIPTOGRAFIA RSA 12
2.5 INTRODUÇÃO À COMPLEXIDADE DE ALGORITMOS 15
2.6 O PEQUENO TEOREMA DE FERMAT 18
2.7 O CRIVO DE ERATÓSTENES 19
2.8 O TESTE DE MONTE-CARLO 20
2.9 O ALGORITMO AKS 22
3 METODOLOGIA 27
3.1 PROCEDIMENTO METODOLÓGICO 27
3.2 IMPLEMENTAÇÕES 28
3.2.1 IMPLEMENTAÇÃO DO ALGORITMO AKS 28
3.2.2 IMPLEMENTAÇÃO DO ALGORITMO MONTE-CARLO 29
3.2.3 IMPLEMENTAÇÃO DO CRIVO DE ERATÓSTENES 30
3.3 TESTES 31
4 RESULTADOS 33
4.1 AKS 33
4.2 MONTE-CARLO 35
4.3 CRIVO DE ERATÓSTENES 37
4.4 DISCUSSÃO E RESULTADOS 39
5 CONCLUSÃO 42
6 REFERÊNCIAS BIBLIOGRÁFICAS 45

1.INTRODUÇÃO

Até onde se tem conhecimento, o conceito de número primo foi introduzido na Grécia Antiga. Uma das primeiras publicações a respeito dos números primos são os Elementos de Euclides, escrito por volta de 300 a.C.. Segundo a definição 11 do Livro VII dos Elementos, de acordo com Euclides:

Um número primo é aquele que é medido apenas pela unidade.

(EUCLIDES, 1944)

Para Coutinho (2004), traduzindo para os dias de hoje, a expressão 'medido por' traduz-se como 'divisível por um número menor do que ele próprio'. Assim, um número é primo se não é divisível por nenhum número menor que ele, exceto 1. Mais adiante, de acordo com o livro de Euclides na definição 13, encontra-se a definição de número composto:

Um número composto é aquele que é medido por algum número.

Há um ponto curioso aqui porque esta última definição parece incluir a anterior, afinal, segundo Euclides, um número primo é medido pela unidade. A verdade é que, para os gregos daquela época, a unidade não era considerada um número.

Com a evolução dos estudos sobre números primos, esta acabou por tomar uma direção diferente e tornou-se cada vez mais algébrica, além de retomar também para outra área: a Criptografia.

O uso da Criptografia é muito antigo e vem desde épocas em que o homem sentia necessidade de camuflar informações para que pessoas sem autorização não tivessem acesso a estas informações. Com o passar dos anos foram desenvolvidos diversos algoritmos de encriptação, de forma a esconder informações sigilosas. Um dos mais conhecidos métodos de encriptação atualmente, o RSA, é baseado em um par de chaves p e q, ambos números primos. Essas chaves p e q são a chave de segurança do RSA e caso sejam

conhecidas p e q, toda segurança da cifragem da mensagem estará comprometida.

Algoritmos usados para descobrir a primalidade de um número, por outro lado, são estudados desde a Grécia antiga. O Crivo de Erástones (aproximadamente 240 a.C.) é o algoritmo mais antigo que se tem conhecimento e que funciona corretamente para todos os números primos (COUTINHO, 2004). No entanto, a sua complexidade de tempo é exponencial. Até 2002, esse problema era classificado como pertencente à classe NP (polinomial não-determinístico).

Atualmente, dispõe-se de algoritmos probabilísticos que afirmam se um número é primo em tempo polinomial (classe P) com baixa percentagem de erro. O algoritmo AKS, de 2002, elaborado por um grupo de estudiosos indianos, é o primeiro algoritmo não-probabilístico a resolver esse problema em tempo polinomial. Sobre este algoritmo foi dito: "Este algoritmo é belo" (POMERANCE, 2003), "É o melhor resultado de que ouvi falar nos últimos dez anos" (GOLDWASSER, 2002), "Uma razão para a excitação dentro da comunidade matemática é que, não só este algoritmo resolve um problema de há muitos anos, como também o faz de uma maneira brilhantemente simples" (LEYLAND, 2002).

Tendo em conhecimento este algoritmo AKS da classe P, torna-se evidente a possibilidade de descobrir números primos grandes rapidamente. Como a segurança da criptografia é baseada em números primos, fica a seguinte questão: Seria o algoritmo AKS um problema para a segurança computacional, visto que este encontra números primos em tempo polinomial?

Desta maneira, esse trabalho tem como objetivo inspecionar o impacto do algoritmo AKS perante os sistemas de criptografia baseados em números primos, avaliando seu desempenho e eficácia no processo de verificação de primalidade. Entre os objetivos específicos, deve-se verificar o desempenho do

AKS perante um conjunto de números específicos escolhidos para teste. Também é objetivo específico comparar o desempenho e resultados do algoritmo AKS com algoritmos clássicos, como o Monte-Carlo e o Crivo de Eratóstenes.

Finalmente, a partir deste comparativo, será avaliado o resultado dos algoritmos, analisando o desempenho de cada um na verificação da primalidade de um número, avaliando o real impacto do AKS para a segurança de criptografias baseadas em números primos.

Desta forma, serão tratados os principais algoritmos de descoberta de números primos conhecidos hoje em comparação com o AKS. Um amplo referencial teórico dará estrutura para o bom entendimento do trabalho. No Capítulo 2 serão abordados diversos conceitos de aritmética modular, teoria dos números, criptografia, complexidade de algoritmos e também serão analisados os algoritmos AKS, Monte-Carlo e o Crivo de Eratóstenes. Estes algoritmos foram implementados na linguagem C/C++.

No Capítulo 3, estará contida toda metodologia do trabalho e como os testes serão realizados. No Capítulo 4, serão mostrados os resultados. Os algoritmos em questão serão adaptados para este trabalho, testados numa mesma máquina para análise dos seus desempenhos e gerado um gráfico comparativo. Para concluir, no Capítulo 5 será abordado o impacto destes algoritmos de descoberta de primalidade para os algoritmos de criptografia baseados em números primos. Também será comentado o critério de escolha dos números para testes, o desempenho dos algoritmos e suas complexidades .

2. REFERENCIAL TEÓRICO

2.1 CONCEITOS BÁSICOS

Durante grande parte do século XX, os estudos sobre os conceitos de primalidade, eram praticamente tratados como problemas secundários em Matemática. Essa situação só mudou a partir da década de 70, através de dois fatores principais. O primeiro foi o surgimento da criptografia RSA, extremamente eficiente e que necessita de números primos extremamente grandes. O segundo foi o surgimento de computadores com capacidade de processamentos cada vez maiores, possibilitando assim aplicar testes de altíssimo custo computacional.

Um importante estudioso que iniciou a busca pela descoberta dos números primos foi Euclides. Ele viveu em Alexandria, e trabalhou na grande Biblioteca, um dos mais importantes centros de pesquisa da época, mas coube a outro matemático de Alexandria chamado Erastóstenes, o mérito de descobrir o primeiro procedimento prático para achar números primos. O primeiro método conhecido de busca de números primos é conhecido por *Crivo de Erastóstenes.* Nicômaco, um matemático que viveu por volta de 100 d.C., descreveu o método de Erastóstenes para descoberta de primos da seguinte maneira:

> *O método para obtê-los (os números primos) é chamado por Erastóstenes uma peneira (crivo), porque tomamos os números ímpares misturados de maneira indiscriminada e, por este método, como se fosse pelo uso de um instrumento ou peneira, separamos os primos ou indecomponíveis dos secundários ou compostos.*
>
> *(NICÔMACO apud COUTINHO, 2004)*

Portanto, o *Crivo de Erastóstenes* atua como uma peneira que retém os números primos, porém deixa passar os números compostos.

Em 1801, um jovem estudioso chamado C. F. Gauss, com apenas 24 anos, publicava uma das mais importantes obras matemáticas do século, a *Disquisitiones Arithmeticae*. Foi neste livro que Gauss deu a primeira demonstração satisfatória do Teorema da Fatoração Única para inteiros, que define a congruência módulo um inteiro positivo. O interesse de Gauss pela teoria dos números começou muito cedo. Aos 14 anos, Gauss escreveu no verso de uma folha de logaritmos a seguinte expressão:

Primzahlen unter a (=∞) a/la.

Coutinho (2004) traduz esta expressão obtendo o seguinte significado: A quantidade de primos (positivos) menores que um inteiro $a>0$ se aproxima de $a/\ln a$ quando a fica muito grande.

Para entender o significado deste resultado é necessário retomar à Grécia Antiga. Na proposição 20 do Livro IX dos Elementos, Euclides prova que existe uma infinidade de primos. Levando isto em conta, temos que o estoque de primos menores que um inteiro positivo n aumenta à medida que n aumenta. Isto nos dá um número progressivamente maior de primos. Analisando este significado, é de espera que haja intervalos cada vez maiores formados por números compostos entre dois primos consecutivos. Considerando $\mu(x)$ a quantidade de números primos positivos menores ou iguais a um número $x > 0$, isto sugere que $\mu(x)$ cresça cada vez mais devagar à medida que x aumenta.

Apesar de tantas descobertas, nem Gauss, nem seus antecessores foram capazes de dar demonstrações para essas suposições. O primeiro resultado matematicamente provado sobre a distribuição dos primos só viria com a publicação do artigo *Sur lês nombres premiers,* do matemático russo P.L. Chebyshev (1852).

No século 17, Fermat publicou o “Pequeno Teorema de Fermat” e este vem sendo ponto de partida para vários algoritmos eficientes de descoberta de primalidade. De acordo com Fermat, citado por Coutinho (2004), se p é um

primo, então $a^{p-1} - 1$ é divisível por *p*, sempre que *a* for um inteiro que não é divisível por *p*, ou seja:

$$a^p \equiv a \pmod{p} \therefore a^{p-1} \equiv 1 \pmod{p}$$

M. Artjuhov (1967), citado por Coutinho (2004), escreveu um dos testes que se tornaram mais populares deste então. Foi originalmente publicado em russo e é um aperfeiçoamento de outro, já conhecido desde o século XVII.

O teste mais antigo consistia em observar que, se descobrirmos um número $0 < a < n$ tal que $n^{a-1} - 1$ não é divisível por *n*, então *n* é composto. Alguns matemáticos acreditaram que a recíproca deste problema fosse verdadeira. Se fosse assim, bastaria testar se 2^{p-1} é divisível por *p* para provar que um número n>2 é primo, no entanto isto é falso (COUTINHO, 2004, p.6).

Segundo o mesmo Coutinho (2004), o teste de Artjuhov foi redescoberto dez anos depois por J. Selfridge, e popularizado pelos trabalhos de G. Miller e M. Rabin. Hoje em dia, o teste de Artjuhov é a base da maioria dos métodos "rápidos" pelos quais os sistemas de computação algébrica certificam que um número é primo, no entanto, o teste não garante a primalidade de um número, ele apenas certifica que há uma probabilidade *P* de que o número seja primo. Contudo, o teste pode ser refinado de modo a fazer *P* tão pequeno quanto se deseje, ainda que haja um custo computacional maior.

Vistos estes conceitos, surge agora com uma questão primordial na busca de primalidade, que é quanto custa descobrir se um dado número é primo. Do ponto de vista do século XX encontram-se dois grupos de buscas. O primeiro é formado pelos "testes rápidos" que determinam se um determinado número é "provavelmente" primo, como os testes de Artjuhov e Rabin. O segundo é formado pelo grupo dos testes lentos, mas que por sua vez, afirma com certeza a primalidade de um determinado número, como o Crivo de Eratóstenes.

Para Coutinho (2004) o estudo da complexidade dos algoritmos, que tomou forma com os trabalhos de J. Hartmanis e R. E. Stearns (1965) permitiu

categorizar estes dois tipos de testes de uma maneira mais conceitual. Assim, de um lado encontram-se algoritmos pertencentes ao teste de Rabin, cujo tempo de execução é uma função polinomial do tamanho da entrada. Do outro, existem os testes cujo tempo de execução é uma função exponencial da entrada.

Até meados de 2002, a situação a respeito da busca de primalidades era basicamente essa: números "provavelmente" primos com um custo relativamente baixo ou números primos corretos a um altíssimo custo. O cenário desta busca mudara completamente quando em agosto de 2002, um grupo de estudiosos indianos formado por, *Manindra Agrawal, Neeraj Kayal e Nitin Saxena* divulgou um artigo na web, chamado *Primes is in P*, que continha um algoritmo que provava se determinado número era primo e o tempo de execução deste algoritmo era uma função polinomial de entrada, ou seja, em P.

Não tardou para que o algoritmo fosse modificado, melhorado e generalizado. Passados meses da divulgação do artigo na web, já havia versões do algoritmo, que utilizam apenas resultados elementares, mesmo no que diz respeito à análise do custo do algoritmo. Coutinho (2004) cita que há o entendimento de que este é o primeiro de uma nova classe de algoritmos, conhecidos como algoritmos ciclotômicos ou do tipo AKS.

2.2 INTRODUÇÃO À TEORIA DOS NÚMEROS

Um número inteiro, *n*, maior do que 1 diz-se primo se os seus divisores positivos são apenas *n* e 1, ou seja, *n* não tem fatores próprios. Portanto *n* é primo se for um número natural com exatamente dois divisores. O Teorema Fundamental da Aritmética afirma que: "Todo número inteiro *n*, maior do que 1, pode ser escrito de maneira única (a menos da ordem dos fatores) como um produto de primos".

A prova de Araújo (2003) é a seguinte: Se n é primo, não há nada a provar. Se n é composto, seja $p_1 > 1$ o menor dos divisores de n. Como n é composto, n = a.b, com a, b inteiros e maiores do que 1. Portanto, D = {divisores de n, maiores do que 1}. Seja p_1, pelo princípio da boa ordem, o menor elemento de D. Queremos agora mostrar que p_1 é primo. Suponhamos, por redução ao absurdo, que p_1 é composto. Então $p_1 = c.d$ com $c, d > 1$ e $c, d < p_1$. Como $p_1 \mid n$, temos $n = p_1.h = c.d.h$ e $c \mid n$ com $c < p_1$. Isto contraria a escolha de p_1 como o menor divisor maior do que 1 de n. Portanto p_1 é primo. Temos então $n = p_1.n_1$ com $n_1 < n$. Aplicando o mesmo argumento a n_1, obtemos um divisor primo p_2 de n_1. Agora temos $n = p_1.p_2.n_2$ e $n_2 < n_1$. Se $n_2 = 1$, concluímos a prova, se não, prosseguimos desta forma e obtemos as sequências $p_1, p_2, ... p_k$ e $n_1 > n_2 > ... > n_k = 1$ de primos e inteiros, respectivamente. Quanto a unicidade, suponhamos que $n = p_1p_2....p_k = q_1q_2....q_t$, com $p_1, p_2, ... p_k, q_1, q_2, ... q_t$ primos. Como os fatores de um e outro lado da igualdade são primos, então os fatores devem ser iguais, provando a unicidade.

As diferentes demonstrações da existência de uma infinidade de números primos não são construtivas e não dão qualquer indicação sobre o enésimo número primo. Essas demonstrações não indicam precisamente quantos números primos existem inferiores a um determinado número n.

Paulo Ribenboim (2001) diz que seria possível estimar essa quantidade de números primos inferiores a esse número n, porém, a distribuição de números primos em alguns intervalos tem comportamento aleatório.

Estudiosos conseguiram analisar profundamente esses números e obter conhecimento extremamente importante a respeito dos números primos, de forma a escrever o Teorema dos Números Primos, teorema este que demonstrou importante resultado sobre a distribuição dos números primos. Através desse teorema, é possível estimar quantos primos existem até um inteiro x, ou seja, descreve a distribuição dos primos. Uma descrição mais detalhada deste teorema

pode ser visto em Bateman e Diammond, (2004). O resultado deste teorema foi primeiramente demonstrado independentemente por dois matemáticos franceses Jacques Hadamard e Charles Jean de La Vallée-Poussin e foi extremamente importante para a Matemática, pois era possível analisar a primalidade de um número.

Para Simon Singh (1998), citado por Cláudio Ferreira (2008), entre os tópicos mais avançados da teoria dos números, podem ser selecionados dois, que são particularmente interessantes: o último teorema de Fermat e o teorema dos números primos. À medida que avançamos na seqüência dos números inteiros, os primos tornam-se cada vez mais raros. Isso levanta uma questão muito importante: o conjunto dos números primos seria finito ou infinito?

Um resultado conhecido na parte central dos *Elementos de Euclides* que lida com as propriedades dos números, responde esta questão como sendo infinita. Na proposição 20, Euclides explica uma verdade simples, porém fundamental sobre os números primos: existe um número infinito deles. Uma simples demonstração desta notação está demonstrada abaixo.

Suponha, por absurdo, que o número de primos seja finito e sejam *p1, p2, p3 ...., pn* os primos. Seja *N* o número tal que

$$N = \prod_{i=1}^{n} p_i + 1,$$ onde $\prod$ denota o produtório.

se *N* é um número primo, é necessariamente diferente dos primos $p_1, p_2, p_3 ...., p_n$ pois sua divisão por qualquer um deles tem resto 1. Por outro lado, se *N* é composto, existe um número primo *q* tal que *q* é divisor de *N*. Mas obviamente $q \neq p_1, p_2, p_3 ...., p_n$. Logo, existe um novo número primo. Este processo pode ser repetido indefinidamente, logo, há um número infinito de números primos.

Chris K. Caldwell (2010) disponibiliza em seu site uma lista com os maiores números primos descobertos na atualidade. Atualmente, o maior primo conhecido possui 12.978.189 dígitos.

Pierre de Fermat (1601-1665) conhecido por seus diversos teoremas descobriu que todo número primo da forma 4n + 1, tal como 5, 13, 29, 37 etc., é a soma de dois quadrados, como por exemplo:

$5 = 1^2 + 2^2$;

$13 = 2^2 + 3^2$;

$29 = 2^2 + 5^2$, etc...

Atualmente são conhecidos dois grupos de números primos: 4n + 1 que sempre podem ser escritos na forma $(x^2 + y^2)$ e 4n – 1 que nunca podem ser escritos na forma $(x^2 + y^2)$.

Tratando-se de números primos, é sempre perigoso fazer generalizações baseadas apenas em observações. Um exemplo destas observações é que 31, 331, 3.331, 33.331, 333.331, 3.333.331 e 33.333.331 são primos, mas 333.333.331 não é, pois é fatorável como 17 x 19.607.843.

Um olhar mais atento na distribuição dos números primos revela que não há regularidade na distribuição. Podem ser encontrados longos buracos nessa distribuição ou possíveis generalizações falsas.

Essa irregularidade na distribuição dos números primos é uma das razões de não existir uma fórmula matemática que produza todos números primos. A fórmula $x^2 - x + 41$, por exemplo, fornece primos quando x está entre 0 e 40. É fácil observar que para $x = 41$ a fórmula resulta em 41^2 que não é primo. De fato, em 1752 Goldbach provou que não há uma expressão polinomial em x com coeficientes inteiros que possa fornecer primos para todos os valores de x (Weisstein, 2009).

2.3 CRIPTOGRAFIA

Desde tempos antigos, buscava-se formas eficientes de comunicação, de comandar exércitos e governar países. A importância de não revelar segredos e

estratégias às forças inimigas motivou o desenvolvimento de códigos e cifras, ou técnicas para mascarar uma mensagem, possibilitando apenas ao destinatário ler o conteúdo. As nações passaram a criar departamentos para elaborar códigos, por outro lado, surgiram os decifradores de códigos, criando uma corrida intelectual.

As diversas formas e utilidades dadas aos códigos ao longo do tempo mostram a presença fundamental da Matemática na evolução de tal teoria. E evolução é um termo bem apropriado, já que todo código sempre está sob o ataque dos decifradores. Ao longo da história, os códigos decidiram o resultado de diversas batalhas. À medida que a informação se torna cada vez mais valiosa, o processo de codificação de mensagens tem um papel cada vez maior na sociedade.

> *Já se falou que a Primeira Guerra Mundial foi a guerra dos químicos, devido ao emprego, pela primeira vez, do gás mostarda e do cloro, que a Segunda Guerra Mundial foi a guerra dos físicos devido à bomba atômica. De modo semelhante se fala que uma Terceira Guerra Mundial seria a guerra dos matemáticos, pois os matemáticos terão o controle sobre a próxima grande arma de guerra, a informação. Os matemáticos têm sido responsáveis pelo desenvolvimento dos códigos usados atualmente para a proteção das informações militares. E não nos surpreende que os matemáticos também estejam na linha de frente da batalha para tentar decifrar esses códigos.*
>
> *(SINGH, 2007, p.13).*

Através dos anos o conceito de Criptografia evoluiu bastante. Sistemas de segurança baseados em criptografia estão cada dia mais comuns. Um dos

sistemas mais conhecidos nos dias de hoje é o sistema de criptografia RSA e abordaremos um pouco sobre ele neste capítulo.

2.4 CRIPTOGRAFIA RSA

Um dos algoritmos mais seguros de encriptação de informações atual originou-se dos estudos de Ronald Rivest, Adi **S**hamir e Leonard **A**dleman, um trio de Matemáticos brilhantes que mudaram a história da Criptografia. O princípio do algoritmo é construir chaves públicas e privadas utilizando números primos. Uma chave é uma informação restrita que controla toda a operação dos algoritmos de criptografia. No processo de codificação uma chave é quem dita a transformação do texto puro (original) em um texto criptografado.

A chave privada é uma informação pessoal que permanece em posse da pessoa - não publicável. A chave pública é uma informação associada a uma pessoa que é distribuída a todos. Uma analogia amplamente conhecida no meio acadêmico através de diversos sites da internet é a transmissão de mensagens entre dois personagens, aqui chamados de Alice e Bob.

Alice e Bob são personagens fictícios que precisam trocar mensagens seguras sem interceptação. O algoritmo RSA permite essa troca segura de mensagens pela utilização de chaves públicas e privadas: Alice cria seu par de chaves (uma pública e outra privada) e envia sua chave pública para todos, inclusive Bob; Bob escreve sua mensagem para Alice. Após escrita, Bob faz a cifragem do texto final com a chave pública de Alice, gerando um texto criptografado; Alice recebe o texto criptografado de Bob e faz a decifragem utilizando a sua chave privada.

O procedimento é realizado com sucesso porque somente a chave privada de Alice é capaz de decifrar um texto criptografado com a sua chave pública. É importante destacar que se a chave pública de Alice for aplicada sobre

o texto criptografado não se obterá a mensagem original de Bob. Dessa forma, mesmo que a mensagem seja interceptada é impossível decifrá-la sem a chave privada de Alice.

Conforme mencionado, o algoritmo RSA é baseado na construção de chaves públicas e privadas utilizando números primos. Inicialmente devem ser escolhidos dois números primos quaisquer *P e Q*. Quanto maior o número escolhido mais seguro será o algoritmo. A título de exemplificação, serão escolhidos números primos pequenos, para permitir um acompanhamento de todo o processo de cifragem e decifragem.

P = 17

Q = 11

A seguir são calculados dois novos números *N e Z* de acordo com os números *P e Q* escolhidos:

N = P * Q

Z = (P - 1) * (Q - 1)

No caso obtêm-se como resultado:

N = 17 * 11 = 187

Z = 16 * 10 = 160

Agora se define um número *D* que tenha a propriedade de ser primo em relação à *Z*. No caso, opta-se pela escolha:

D = 7

De posse desses números começa o processo de criação das chaves públicas e privadas. É necessário encontrar um número *E* que satisfaça a seguinte propriedade:

(E * D) mod Z = 1 ,

Se forem feitos os testes com 1, 2, 3... tem-se:

E = 1 => (1 * 7) mod 160 = 7

E = 2 => (2 * 7) mod 160 = 14

E = 3 => (3 * 7) mod 160 = 21

...

E = 23 => (23 * 7) mod 160 = 1

...

E = 183 => (183 * 7) mod 160 = 1

...

E = 343 => (343 * 7) mod 160 = 1

...

E = 503 => (503 * 7) mod 160 = 1

...

Logo até o momento os números 23, 183, 343, 503 satisfazem a propriedade indicada. Para efeito de simplificação de cálculos, será tomado como referência:

E = 23.

Com esse processo definem-se as chaves de encriptação e desencriptação: para encriptar: utilizar *E e N* - esse par de números será utilizado como chave pública, e para desencriptar: utilizar *D e N* - esse par de números utilizado como chave privada. As equações são:

Texto criptografado = (Texto original ^ E) mod N

Texto original = (Texto criptografado ^ D) mod N

Seja a necessidade de se encaminhar uma mensagem bem curta de forma criptografada, como o número 4, por exemplo, tendo por base as chaves aqui estabelecidas. Para criptografar:

Texto original = 4

Texto criptografado = (4 ^ 23) mod 39

Texto criptografado = 70.368.744.177.664 mod 39

Texto criptografado = 64

Para desencriptar:

Texto recebido = 64

Texto original = (64 ^ 7) mod 39

Texto original = 4.398.046.511.104 mod 39

Texto original = 4

A questão das escolhas dos números primos envolvidos é fundamental para o algoritmo. Por essa razão escolhem-se números primos gigantescos para garantir que a chave seja inquebrável.

2.5 INTRODUÇÃO Á COMPLEXIDADE DE ALGORITMOS

De acordo com (Menezes, A; Oorschot, Van; Vanstone, S, 2009) o ramo da Ciência da Computação que estuda a complexidade de algoritmos busca definir maneiras de se classificar um problema computacional de acordo com os recursos necessários para a resolução do mesmo. Complexidade não deve depender de um modelo computacional específico e sim de vários recursos como memória, tempo de execução, processamento, etc. De modo geral, o recurso mais focado é o tempo de execução.

Uma analogia seria pensar em um algoritmo com um programa de computador, escrito em uma linguagem de programação específica que assume uma variável de entrada e para com uma variável de saída.

Tratando-se de algoritmos, é sempre interessante buscar o mais eficiente, isto é, mais rápido e com custo computacional mais baixo. Geralmente, é difícil calcular exatamente o tempo gasto na realização de operações por um certo algoritmo, assim, trabalha-se com aproximações que normalmente derivam do estudo assintótico do custo de um algoritmo, isto é, o crescimento do tempo de realização das operações de um algoritmo em função do crescimento dos valores de entrada.

No estudo de primalidade dos números, o tempo não é medido em segundos ou minutos e sim pelo número de operações elementares necessárias à execução do processo. No estudo da complexidade, os problemas geralmente são classificados como sendo das classes P, NP, NP-Completo e NP-Difícil.

Diz-se que determinado problema é da classe P, se no pior dos casos, o seu custo é da forma $O(n^k)$, onde n é o tamanho dos dados de entrada e k é constante. Algoritmos de custo polinomial são considerados eficientes enquanto algoritmos de custo exponencial são considerados ineficientes. Se um problema computacional pode ser resolvido em tempo polinomial, este é chamado de tratável. Quando surge um problema computacional qualquer, a primeira questão é: existe algum algoritmo "rápido" para resolver o problema?

A classe de Complexidade P é a classe de problemas que podem ser resolvidos em tempo polinomial. A classe NP é a classe de problemas de decisão, em que a resposta é sim ou não, que podem ser verificados em tempo polinomial (SANDERSON, 2010). Suponha que temos que resolver um problema em que os dados da entrada são dados por um determinado inteiro *n*. Se este problema está na classe P, o tempo necessário para resolvê-lo cresce de forma não mais do que polinomial. Este problema seria considerado "tratável", pois o tempo de resolução não cresce absurdamente com a dimensão do problema.

Um exemplo clássico de problema da classe P é a multiplicação de inteiros. A multiplicação de dois inteiros com *n* algarismos exige no máximo n^2 somas e multiplicações de algarismos.

Problemas da classe NP são problemas onde o tempo de execução cresce mais rapidamente do que qualquer polinômio em função de *n*, tornando-se um problema difícil ou por vezes, intratável. Para Sanderson (2010), um problema pertence à classe NP se e somente se, existe um verificador para o problema que execute em tempo polinomial[...]. Um algoritmo que resolva qualquer problema

da Classe NP em tempo polinomial não é conhecido nem foi provado que há um limite inferior não-polinomial para qualquer um dos problemas dessa classe.

Não-determinismo, significa que há um oráculo que "escolhe o caminho certo" entre n opções para verificar uma instância do problema em tempo polinomial, onde *n* é o tamanho da entra de L. É como se para n opções, uma máquina abstrata se multiplicasse em n outras máquinas e indicasse o "caminho certo" imediatamente (SANDERSON, 2010). Ziviani (2005) explica que um algoritmo não-determinista contém operações cujo resultado não é unicamente definido, ainda que limitado a um conjunto específico de possibilidades.

Problemas NP-completos são bastante conhecidos. O mais famoso é o problema do caixeiro-viajante: *Dado um conjunto de n cidades e respectivas distâncias, qual é o percurso ótimo para que o caixeiro viajante visite todas as cidades com a menor distância percorrida?*

Imagine que temos um computador capaz de fazer 1 bilhão de operações por segundo. Se nosso número de cidades for $n = 20$, o computador precisará de apenas 19 operações para definir uma única rota, sendo capaz de calcular $10^9 / 19 = 53$ milhões de rotas por segundo. Essa imensa velocidade não é nada comparado as 19! rotas que precisará calcular. O cálculo para 20 cidades demoraria 73 anos, considerando este computador que consegue executar 1 bilhão de operações por segundo.. Para se ter uma dimensão do problema e do quanto ele cresce a medida que *n* aumenta, se considerarmos $n = 15$ cidades, o tempo gasto para o cálculo seria de 20 minutos, enquanto $n = 25$ demoraria 470 milhões de anos.

Se alguém construir um algoritmo que resolva um único problema NP-completo em tempo polinomial, esse algoritmo pode ser traduzido para qualquer problema NP, assim todos problemas NP seriam na verdade P, confirmando assim que P = NP. Este problema, ainda sem solução, é de uma importância

extrema para a criptografia, porque a maioria dos sistemas criptográficos de chave pública são baseados em problemas NP.

2.6 O PEQUENO TEOREMA DE FERMAT

O pequeno Teorema de Fermat é um teorema relacionado aos números primos e tem sido o passo inicial de vários algoritmos de descoberta de primalidade como o Monte-Carlo e o próprio AKS. Este teorema faz uma análise de relação entre um número primo *p* e um número qualquer *a.*

Teorema: *Seja p um número primo. Se p não divide a, então* $a^{p-1} \equiv 1 \pmod{p}$.

Demonstração. Seja o conjunto de valores *a, 2a, 3a,...,(p - 1)a*. Sabemos que, pelo fato de que *p* não divide *a, (a; p) = 1* e, portanto, nenhum dos números deste conjunto é divisível por *p*. Além disso, temos que, se $aj \equiv ak \pmod{p}$, então $j \equiv k \pmod{p}$, ou seja, todos eles são incongruentes módulo *p* e, portanto, podemos estabelecer uma relação biunívoca entre os "*aj*", *j* = 1, 2,..., *p* - 1 e o conjunto 1, 2, 3,..., (*p* - 1), em termos de congruência, isto é, cada um dos termos do primeiro conjunto é congruente a um diferente do segundo. Deste argumento, segue a seguinte igualdade:

$$a(2a)(3a)...(p - 1)a \equiv 1.2.3...(p - 1) \pmod{p}$$

ou seja $a^{p-1}(p - 1)! \equiv (p - 1)! \pmod{p}$. Da lei do cancelamento, e do fato de que $((p - 1)!, p) = 1$, segue que

$$a^{p-1} \equiv 1 \pmod{m}.$$

2.7 O CRIVO DE ERATÓSTENES

Eratóstenes foi um dos grandes matemáticos da Antiguidade e um de seus grandes marcos foi encontrar a medida da circunferência da Terra com o valor mais próximo do atual. Ele é reconhecido pelos seus contemporâneos como um grande sábio, através das suas grandes contribuições nas mais diversas áreas de conhecimento.

Foi astrônomo, historiador, filósofo e matemático. Seu maior feito na Matemática foi o Crivo de Eratóstenes, um algoritmo para encontrar todos os números primos menores que um determinado número. O Crivo de Eratóstenes, consiste em dispor os números naturais de 1 a *n*, em ordem crescentes em um quadro e ir eliminando, por etapas, os números que não são primos da seguinte maneira:

Inicialmente, risca-se o 1, que não é primo.

Em seguida, risca-se todos os múltiplos de 2, exceto o 2 que é primo;

Depois de riscar todos os múltiplos do primeiro primo (no caso o 2) passe para o próximo primo, o número 3 e risque todos os múltiplos do 3 e assim sucessivamente.

Ao observar este método de marcação de primos, é notável que alguns números são marcados mais de uma vez, pois são múltiplos de mais um primo, como o número 30, múltiplo de 2, 3 e 5. Por isso, é importante saber em que primo *p* podemos parar, para evitar operações desnecessárias.

Para encontrar esse *p* ideal usa-se a seguinte expressão $2 \leq p \leq \sqrt{n}$. Através desta expressão é possível observar que para *n*=50, por exemplo, é possível parar no primo 7, pois $2 \leq 7 \leq \sqrt{50}$, ou seja, para determinar todos os números primos menores que 50 basta riscar os múltiplos de 2, 3, 5 e 7.

2.8 O TESTE DE MONTE-CARLO

O teste de Monte Carlo é um método estatístico utilizado em simulações estocásticas com diversas aplicações em áreas como a física, matemática e biologia. O método tem sido usado principalmente como forma de obter aproximações numéricas de funções complexas. O teste é uma forma de resolver problemas usando números aleatórios. O método explora as propriedades estatísticas dos números aleatórios para assegurar que o resultado correto é computado da mesma maneira que num jogo de cassino para se certificar de que a "casa" sempre terá lucro.

Para resolver um problema através do método de Monte Carlo, é usada uma série de tentativas aleatórias. A precisão do resultado final depende em geral do número de tentativas. Esse equilíbrio entre a precisão do resultado e o tempo de computação é uma característica extremamente útil dos métodos de Monte Carlos. Se uma solução aproximada for suficiente, o teste de Monte-Carlo pode ser bastante rápido.

De acordo com (HAMMERSELEY, 1964) o nome "Monte Carlo" surgiu durante o projeto Manhattan na Segunda Guerra Mundial. Apesar de despertar a atenção apenas nesta época, a lógica do método já era conhecida há bastante tempo.

O teste de Monte-Carlo aplicado à descoberta de números primos é conhecido como teste de Miller-Rabin. Este teste é um algoritmo não-determinístico de busca de primalidade em tempo polinomial, ou seja, determina a probabilidade de um número qualquer *p* ser primo. No teste de Monte-Carlo, escolhe-se um número aleatório *r* no intervalo [2,p-1] e aplica-se dois testes:

1. $r^{p-1} \neq 1 \pmod{p}$
2. para algum inteiro k, $1 < \text{mdc}(r(p-1)/2k - 1, p) < p$

O primeiro teste é baseado no Pequeno Teorema de Fermat, enquanto o segundo procura achar um fator comum entre dois números, sendo um deles *p*, o que implicaria *p* ser composto. Se *r* for aprovado em ambos os testes, então *p* é composto. Mais da metade dos números do intervalo estão neste caso.

De acordo com Braga (2002), se este número *r* não passar no teste, então *p* pode ser primo com probabilidade de 50%. Agora, escolhe-se um novo *r* e aplica-se o teste de Monte-Carlo novamente. Se o novo *r* também não passar no teste, a probabilidade de que *p* seja primo sobre para 75%, ou seja, P(m) = Sum($1/2^i$, i=1.m) onde *m* é a quantidade de valores de *r* que não passaram no teste até o momento.

A escolha do número *r* na execução deste teste é de extrema importância, visto que, para um mesmo número *n* e diferentes escolhas de *r*, é possível analisar que a descoberta varia muito quanto à quantidade de testes aplicados. Felizmente, para 10 iterações em que um número *p* não passe no teste, temos 99,9% de certeza de que *p* é primo.

Na figura 1, é possível visualizar um algoritmo mais completo de Miller-Rabin.

r is odd such that $n-1=2^s r$
for $i \leftarrow 1$ **to** t
do {
  $a \leftarrow$ random integer such that $2 \leq a \leq n-2$
  $y \leftarrow a^r \pmod{n}$
  if $y \neq 1$ **and** $y \neq n-1$
    then {
      $j \leftarrow 1$
      while $j \leq s-1$ **and** $y \neq n-1$
        do {
          $y \leftarrow y^2 \pmod{n}$
          if $y = 1$
            then return $(COMPOSITE)$
          $j \leftarrow j+1$
        }
    }
  if $y \neq n-1$
    then return $(COMPOSITE)$
}
return (PROBABLY PRIME)

Figura 1 - O algoritmo Monte-Carlo

Atualmente, apesar de ser um teste probabilístico, o teste de Monte-Carlo (Miller-Rabin) é um dos testes mais usados para se verificar a primalidade de um número.

2.9 O ALGORITMO AKS

O algoritmo AKS é um algoritmo desenvolvido por um grupo de indianos onde é possível identificar a primalidade de um número em tempo polinomial (AGAWAL, KAYAL, SAXENA, 2002). É um pequeno algoritmo de 14 linhas onde se tem um teste determinístico em P.

No teste de Monte-Carlo, podemos observar que se testarmos todas as possibilidades de *r*, podemos afirmar se um número é primo. No entanto, isso é inviável para *p* grande. Logo, se fosse possível achar um sub-conjunto de valores para *r* tal que, aplicados ao teste nos fornecesse uma resposta certa, então

teríamos um teste determinístico em P. E é exatamente isso que faz o algoritmo AKS, apresentado na figura 2.

Input: Integer $n > 1$
1. if ($n = a^b$ with $b > 1$) then output COMPOSITE;
2. $r := 2$;
3. while ($r < n$) {
4. if (gcd(n,r) is not 1) then output COMPOSITE;
5. if (r is prime greater than 2) then {
6. let q be the largest factor of r-1;
7. if ($q > 4\text{sqrt}(r)\log n$) and not($n^{(r-1)/q} = 1 \pmod r$) then
8. break;
9. $r := r + 1$;
10. }
11. for $a := 1$ to $2\text{sqrt}(r)\log n$ {
12. if not($(x-a)^n = (x^n-a) \pmod{x^r-1, n}$)
then output COMPOSITE;
13. }
14. output PRIME;

Figura 2 - O algoritmo AKS

As linhas de 1 a 10 apresentadas na figura 2 funcionam como um filtro para descartar os valores que não influem na decisão de primalidade, dados certos princípios algébricos já conhecidos. Da linha 11 em diante que se trata devidamente a primalidade do número. Na linha 12, tem-se a aplicação do Pequeno Teorema de Fermat. Em síntese, o algoritmo primeiro escolhe um *r* primo para obter *q*, o maior fator primo de *r-1*, tal que este *r* delimita um intervalo onde certamente haverá um fator primo de *n* se este for composto. Em

seguida, o algoritmo testa a congruência para $a \in [1, 2r^{1/2}\log(n)]$, uma quantidade de testes realizável em tempo polinomial no pior caso. Este é justamente o avanço feito por este algoritmo, explicado no artigo *Primes is in P* (AGRAWAL et al., 2006).

O teste de primalidade AKS é baseado na seguinte equivalência:

$$(x-a)^n \equiv (x^n - a) \pmod{n},$$

a qual é verdadeira somente se *n* é primo. Isto é uma generalização do pequeno teorema de Fermat estendido para polinômios e pode ser facilmente provado usando o teorema binomial juntamente com o fato de que:

$$\binom{n}{k} \equiv 0 \pmod{n}$$

para todo $0 < k < n$ se *n* é primo.

Esta inequação constitui um teste de primalidade por si só, verificando isto em tempo exponencial. Além disso, o AKS faz uso da relação de equivalência:

$$(x-a)^n \equiv (x^n - a) \pmod{n, x^r - 1},$$

a qual pode ser verificada em tempo polinomial. Contudo enquanto todos os primos satisfazem esta equivalência, alguns números compostos também o fazem. A prova da correção consiste em mostrar que existe um conveniente menor *r* e um conveniente conjunto menor de inteiros *A* tal que se a equivalência que assegura que para todo *a* em *A* então *n* deva ser primo.

O algoritmo para o teste de primalidade de algum inteiro *n* consiste em duas partes. O primeiro passo gira em torno de encontrar um número primo conveniente $r = kq + 1$, tal que:

- P(r-1) = q onde P(x) é o maior fator primo de x,
- $q \geq 4\sqrt{r}\log_2(n)$,
- $n^k \not\equiv 1 \pmod{r}$.

Durante este passo, é importante confirmar que *n* não é divisível por nenhum primo $p \leq r$. Se ele é divisível, o algoritmo poderá terminar imediatamente para avisar que *n* é composto.

No segundo passo, um número de testes são realizados para se testar a equivalência de dois polinômios no campo $(x-a)^n \equiv (x^n - a) \pmod{n, x^r - 1}$. Para todos os inteiros positivos com $a \leq 2\sqrt{r}\log_2(n)$, então *n* é garantidamente primo. Em todos os outros casos ele é composto.

Como em qualquer tipo de algoritmo, a principal preocupação é estabelecer dois fatos: provar que o algoritmo está correto e estabelecer seu tempo de complexidade assintótico. Isto pode ser encontrado e estabelecido em um limite superior da sua magnitude, e depois, a demonstração do teste de equidade múltiplo na segunda parte do algoritmo, que verifica $(x-a)^n \equiv (x^n - a) \pmod{n, x^r - 1}$ é suficiente para garantir se *n* é primo ou composto.

Nos meses seguintes a descoberta do algoritmo AKS, novas variantes já apareceram como mostrado por Lenstra *(2002),* Weisstein *(2010),* Berrizbeitia *(2003)*, dentre outras, as quais melhoraram a velocidade do algoritmo em várias ordens de magnitude. (CRANDALL E PAPADOPOULOS, *2003)* referem-se a uma classe AKS de algoritmos, definida em seus trabalhos como "*On the implementation of AKS-class primality tests*".

Também em março de 2003, *Agrawal*, *Kayal* e *Saxena* colocaram na internet uma revisão do seu texto. O novo texto contém aperfeiçoamentos feitos por *Hendrik Lenstra Jr.*(2002) no algoritmo AKS, e já possui uma melhora significativa no seu desempenho. A figura 3 mostra o novo código.

Input: integer $n > 1$.
1. If $(n = a^b$ for $a \in \mathcal{N}$ and $b > 1)$, output COMPOSITE.
2. Find the smallest r such that $o_r(n) > \log^2 n$.
3. If $1 < (a, n) < n$ for some $a \leq r$, output COMPOSITE.
4. If $n \leq r$, output PRIME.[1]
5. For $a = 1$ to $\lfloor \sqrt{\phi(r)} \log n \rfloor$ do
   if $((X + a)^n \neq X^n + a \pmod{X^r - 1, n})$, output COMPOSITE;
6. Output PRIME;

Figura 3 - O novo algoritmo AKS.

A idéia por detrás deste novo algoritmo é a mesma do anterior, porém a procura do *r* "conveniente" está muito melhorada. Esta nova idéia está muito mais rápida. Na figura 3 é possível analisar na linha 2 a nova proposta para encontrar *r* e na linha 5 a nova verificação de congruência.

Os novos passos são essencialmente:

- Linha 1: Decidir se *n* é uma potência perfeita de um número natural; se for então n é composto.
- Linha 2: Encontrar o *r* "conveniente";
- Linha 3: Calcular o máximo divisor comum de *a* e *n*, para $a \leq r$, se for maior do que 1 então *n* é composto;
- Linha 4: Se *n* for menor ou igual a *r*, *n* é primo.
- Linha 5: Verifica a congruência $(x - a)^n$ $(x^n - a)$ (mod $x^r - 1$, *n*), para todo *a* que satisfaça a condição da estrutura de controle for, se existir a congruência, *n* é composto.
- Linha 6: Retorna *n* primo.

3. METODOLOGIA

3.1 PROCEDIMENTO METODOLÓGICO

A maior parte do estudo foi desenvolvida através de pesquisas em materiais já existentes e houve uma menor parte de programação em laboratório para teste e análise dos algoritmos estudados. Foram estudados os conceitos teóricos e a fase de programação foi toda em laboratório, cedido pela própria Universidade Federal de Lavras (UFLA), para análise e comparação dos testes. Após todo levantamento de referencial teórico, foram programados os algoritmos de descoberta de primalidade Monte-Carlo, o Crivo de Eratóstenes e o AKS nas linguagens C/C++. Ocorreram os mesmos testes para comparar os desempenhos com diferentes tipos de entradas. Todos os códigos foram rodados em um computador Pentium D 3.4 com 4 MB de cache e 3 GB de memória Ram. Após os resultados obtidos, foram gerados gráficos para análise dos mesmos a fim de se obter respostas sobre o desempenho de cada um. Em síntese, as etapas deste trabalho foram:

- Estudo dos principais conceitos de aritmética modular;
- Busca pelos principais algoritmos de descoberta de primalidade;
- Pequena análise do custo destes algoritmos;
- Estudo, análise e implementação dos algoritmos Monte-Carlo, Crivo de Eratóstenes e AKS;
- Análise do impacto dos algoritmos de descoberta de primalidade na segurança computacional.

3.2 IMPLEMENTAÇÕES

3.2.1 IMPLEMENTAÇÃO DO ALGORITMO AKS

Para a implementação do AKS, conhece-se uma divisão do algoritmo em três passos de acordo com Kumar (2007). No primeiro passo, o algoritmo checa se o número de entrada está na forma a^b (Figura 4).

$$
\begin{aligned}
&P \leftarrow 2^{\lfloor B(n)/b \rfloor} \\
&\textbf{for } b \leftarrow 2 \textbf{ to } \log n \\
&\quad \textbf{do} \begin{cases} a = n^{1/b} \\ \textbf{if } n = \lfloor a + 1/2 \rfloor^b \\ \quad \textbf{then return } (\textbf{ true }) \end{cases} \\
&\textbf{return } (\textbf{ false })
\end{aligned}
$$

Figura 4 - Primeiro passo do algoritmo AKS

No segundo passo, o algoritmo busca $n^{1/b}$. (Figura 5).

$$
\begin{aligned}
&P \leftarrow 2^{\lfloor B(n)/b \rfloor} \\
&\textbf{while} \\
&\quad \textbf{do} \begin{cases} Q = \lfloor (b-1)p \rfloor + \lfloor \frac{(n/p)^{b-1}}{b} \rfloor \\ \textbf{if } Q > P \\ \quad \textbf{then return } (P) \\ P \leftarrow Q \end{cases}
\end{aligned}
$$

Figura 5 - Segundo passo do algoritmo AKS

Não é conhecido nenhum algoritmo de tempo polinomial que retorna o maior fator para um determinado número. Se a entrada é um *b*-bit inteiro, não é conhecido algum algoritmo que fatore um inteiro em tempo $O(b^k)$ para qualquer

constante *k*. No entanto, existem algoritmos publicados que são mais rápidos que $o(a^b)$ para qualquer número *a* maior que 1.

O terceiro passo é visto na figura 6 e é usado para obter o maior fator primo de *n*.

$$
\begin{array}{l}
\textbf{if } n < 2 \\
\quad \textbf{then return } (1) \\
r \leftarrow n \\
\textbf{if } r = 0 \pmod 2 \\
\quad \textbf{then} \begin{cases} p \leftarrow 2 \\ \textbf{while } r = 0 \pmod 2 \\ \quad \textbf{do } r \leftarrow r \div 2 \end{cases} \\
\textbf{for } i \leftarrow 3 \textbf{ to } r \\
\quad \textbf{do} \begin{cases} \textbf{if } r = 0 \pmod i \\ \quad \textbf{then} \begin{cases} p \leftarrow i \\ \textbf{while } r = 0 \pmod i \\ \quad \textbf{do } r \leftarrow r \div i \end{cases} \end{cases} \\
\textbf{return } (p)
\end{array}
$$

Figura 6 - Terceiro passo do algoritmo AKS

3.2.2 IMPLEMENTAÇÃO DO ALGORITMO MONTE-CARLO

Dado um número n a ser testado, o algoritmo escolhe um número $a<n$, de forma aleatória e verifica se $a^{n-1} = 1 \pmod n$. Se a equação não valer para um certo *a*, então *n* não é primo. *O* algoritmo da implementação do Monte-Carlo é visto na Figura 7.

$$
\begin{array}{l}
r \text{ is odd such that } n-1=2^s r \\
\textbf{for } i \leftarrow 1 \textbf{ to } t \\
\quad \textbf{do} \left\{ \begin{array}{l}
a \leftarrow \text{random integer such that } 2 \leq a \leq n-2 \\
y \leftarrow a^r \pmod{n} \\
\textbf{if } y \neq 1 \textbf{ and } y \neq n-1 \\
\qquad \textbf{then} \left\{ \begin{array}{l}
j \leftarrow 1 \\
\textbf{while } j \leq s-1 \textbf{ and } y \neq n-1 \\
\qquad \textbf{do} \left\{ \begin{array}{l}
y \leftarrow y^2 \pmod{n} \\
\textbf{if } y = 1 \\
\quad \textbf{then return } (COMPOSITE) \\
j \leftarrow j+1
\end{array} \right.
\end{array} \right. \\
\textbf{if } y \neq n-1 \\
\quad \textbf{then return } (COMPOSITE)
\end{array} \right. \\
\textbf{return } (\text{PROBABLY PRIME})
\end{array}
$$

Figura 7 - Algoritmo para implementação do Monte-Carlo

3.2.3 IMPLEMENTAÇÃO DO CRIVO DE ERATÓSTENES

O funcionamento do Crivo de Eratóstenes já foi explicada no tópico 2.7. Para implementar o Crivo de Eratóstenes, foi usado um algoritmo de acordo com a figura 8.

```
1.  /* Primeiro passo */
2.  recebe valorLimite

3.  /* Segundo passo */
4.  raiz ← √valorLimite

5.  /* Terceiro passo */
6.  para i ← 2 até valorLimite
7.  vetor[i] ← i
8.  fim-para

9.  /* Quarto passo */
10. para i ← 2 até raiz
11. se vetor[i] = i
      a.  imprima "O número " i " é primo."
      b.  para j ← i+i até valorLimite, de i e i
            i.  vetor[j] ← 0
      c.  fim-para
12. fim-se
13. fim-para
```

Figura 8 - Algoritmo para implementação do Crivo de Eratóstenes

3.3 TESTES

Para analisar o desempenho dos algoritmos, foram definidos alguns critérios, a fim de se comparar os resultados.

Os números escolhidos são todos primos, com algumas exceções para comprovar casos particulares. Inicialmente, foi gerada uma lista com números primos de até 7 dígitos. Para cada um dos números escolhidos, foi feito o teste de descoberta de primalidade pelos três algoritmos trabalhados e anotado o tempo gasto para confirmar se o número era primo.

Optou-se por rodar os testes com apenas números primos (com uma única exceção, pois, no algoritmo AKS foi testado um número não primo), pois, o caso de afirmação de que um número é primo, é o pior caso, ou seja, com o maior custo computacional.

Como exceções, encontramos o caso do AKS, onde foi colocado um número não primo para evidenciar a facilidade do AKS em eliminar estes números. Foi escolhido um número alto propositalmente, mostrando a queda de tempo gasto para confirmar a não primalidade de um número.

Outra exceção ocorrida foi no caso do Monte-Carlo. Por ser um algoritmo extremamente eficiente, foi necessário inserir números maiores nos testes, para que este retornasse algum custo computacional significante. Utilizando apenas os números escolhidos, o Monte-Carlo se mostrou extremamente eficiente e confirmou a primalidade de quase todos os números com tempo de execução de zero milissegundo, sendo necessários outros testes com números maiores, para forçar algum valor significante no tempo de execução.

Todos os tempos de execução foram realizados pela própria aplicação em que se executava os algoritmos. Antes de iniciar os cálculos, armazenava-se o tempo inicial em uma variável e fazia-se o mesmo após o término dos cálculos. Após obter esses valores, subtraia-se o tempo inicial do final, obtendo assim, o tempo gasto para o cálculo da operação.

4. RESULTADOS

4.1 AKS

Na figura 9, temos o resultado dos testes do algoritmo AKS. O eixo x representa o tamanho do número de entrada *n*. O eixo y representa o tempo de execução em msec. O ponto baixo no gráfico, referente ao ponto x=2001967 mostra o tempo de execução para um número não primo, provando a eficiência do AKS em excluir números que não são primos.

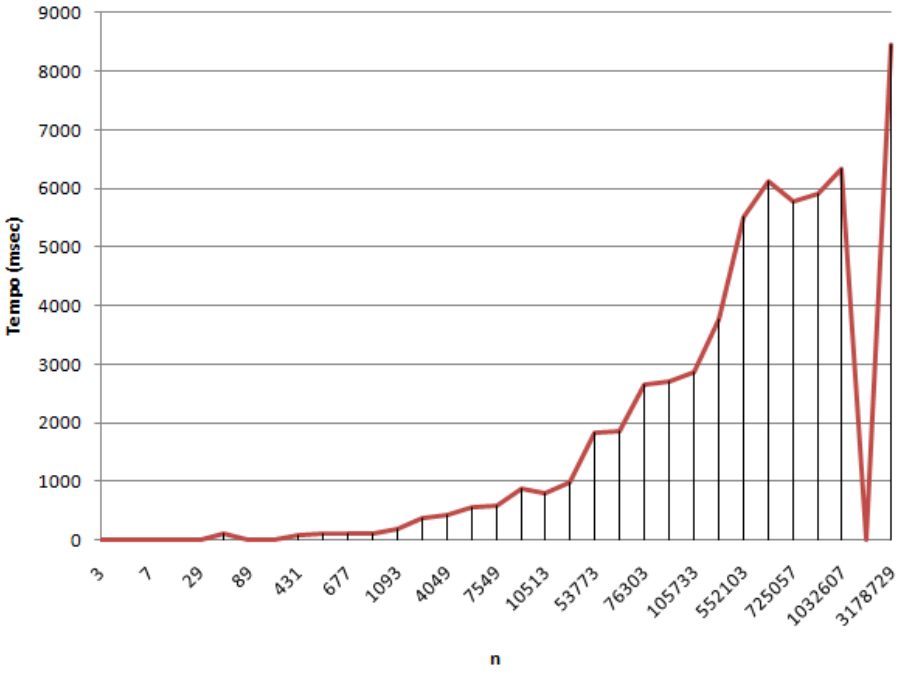

Figura 9 - Gráfico do tempo de execução do algoritmo AKS

LISTA DE ENTRADAS E TEMPO DE PROCESSAMENTO DO ALGORITMO AKS

n	**Tempo de execução em msec**	**n**	**Tempo de execução em msec**
5	0	9907	876
7	0	10513	789
13	0	21517	987
29	0	53773	1830
53	101	63521	1855
89	12	76303	2666
101	10	99053	2697
431	98	105733	2859
523	123	208891	3768
677	100	552103	5524
751	104	601819	6127
1093	195	725057	5777
3457	376	944701	5908
4049	431	1032607	6340
6317	573	2001967	1
7549	592	3178729	8452

Tabela 1.

4.2 MONTE-CARLO

Na figura 10 temos a representação dos testes do algoritmo Monte-Carlo. O eixo x representa o tamanho do número de entrada *n*. O eixo y representa o tempo de execução em msec. Com exceção aos casos com *n* pequeno, o algoritmo de Monte-Carlo apresenta melhor desempenho em relação ao AKS.

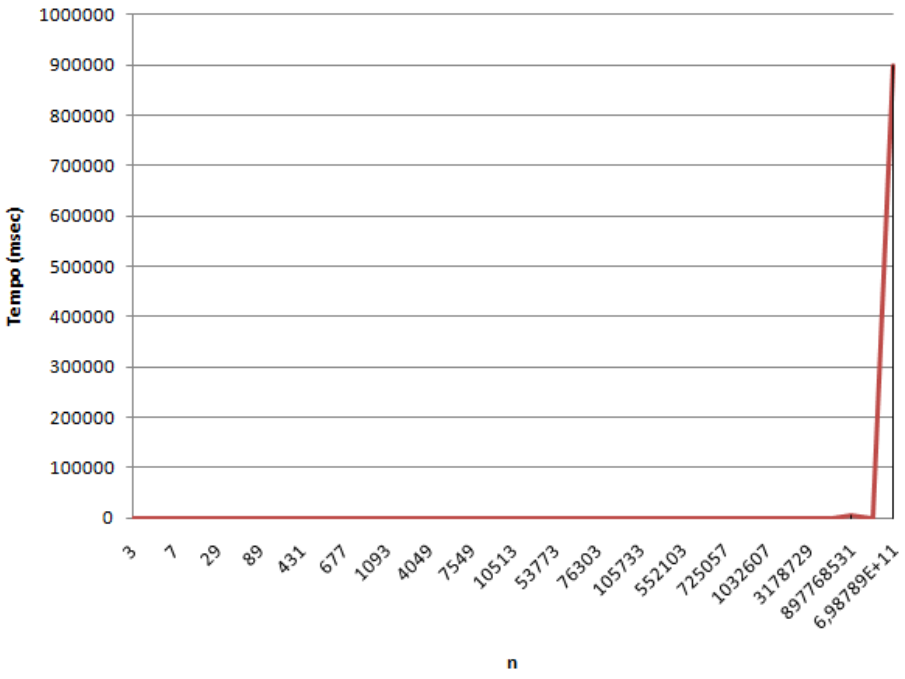

Figura 10 - Gráfico do tempo de execução do algoritmo Monte-Carlo

LISTA DE ENTRADAS E TEMPO DE PROCESSAMENTO DO ALGORITMO MONTE-CARLO

n	**Tempo de execução em msec**	**n**	**Tempo de execução em msec**
5	0	21517	0
7	0	53773	0
13	0	63521	0
29	0	76303	0
53	0	99053	2
89	0	105733	0
101	0	208891	0
431	0	552103	0
523	0	601819	0
677	0	725057	1
751	0	944701	0
1093	0	1032607	4
3457	0	2001967	0
4049	0	3178729	1
6317	0	888777653	41
7549	0	897768531	5400
9907	0	99876789121	37
10513	0	698789258931	Superior a 900000

Tabela 2.

4.3 CRIVO DE ERATÓSTENES

Na figura 11 temos a representação dos testes do algoritmo do Crivo de Erastótenes. O eixo x representa o tamanho do número de entrada *n*. O eixo y representa o tempo de execução em msec. Maiores detalhes acerca dos gráficos estão no apêndice, com todos valores usados para teste e seus respectivos resultados. Em casos com *n* relativamente pequeno, o Crivo de Eratóstenes se mostra bem mais eficiente que o AKS, porém, para *n* grande, com 5 dígitos ou mais, o AKS já se mostra bastante superior.

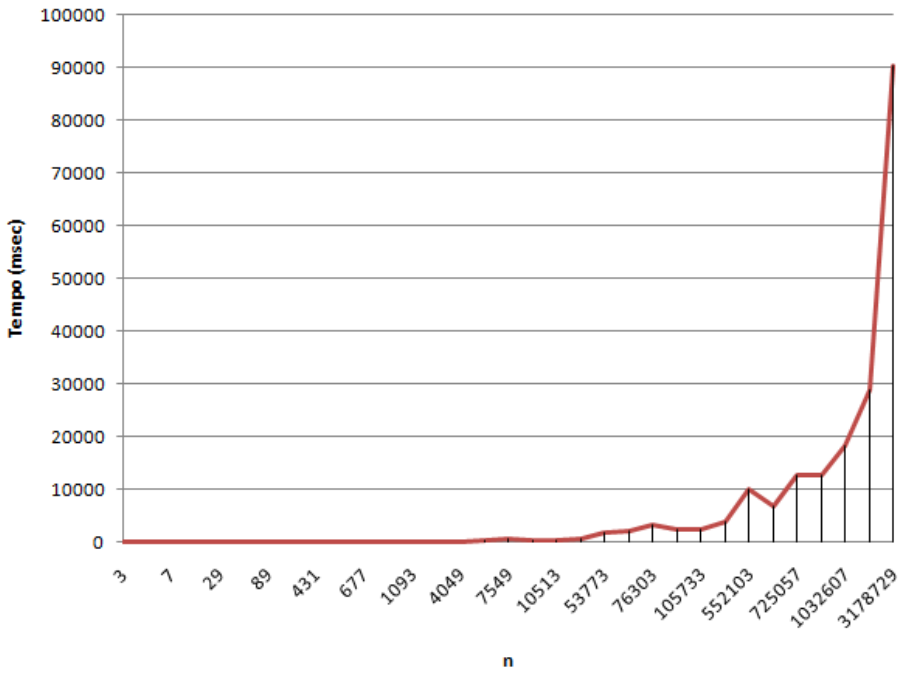

Figura 11 - Gráfico do tempo de execução do Crivo de Eratóstenes

LISTA DE ENTRADAS E TEMPO DE PROCESSAMENTO DO CRIVO DE ERATÓSTENES

n	**Tempo de execução em msec**	**n**	**Tempo de execução em msec**
5	7	9907	275
7	5	10513	389
13	6	21517	661
29	8	53773	1913
53	8	63521	2183
89	10	76303	3199
101	11	99053	2546
431	23	105733	2280
523	25	208891	3801
677	32	552103	9925
751	33	601819	6933
1093	44	725057	12659
3457	178	944701	12839
4049	144	1032607	18307
6317	278	2001967	28847
7549	547	3178729	90481

Tabela 3.

4.4 DISCUSSÃO E RESULTADOS

Através dos resultados obtidos, é necessário interpretar a diferença dos tempos de execução para os diferentes algoritmos.

Analisando os gráficos a priori, é visto o desempenho excelente do Monte-Carlo em relação aos demais. É possível notar, que o algoritmo começa a demonstrar certo esforço computacional, apenas com números acima de onze dígitos, diferentemente do algoritmo AKS, que com números de três dígitos apenas, já tem um custo computacional semelhante ao do Monte-Carlo. O Crivo se mostra bem eficiente para números pequenos, porém, à medida que os números vão crescendo, o esforço computacional também vai aumentando muito. Note pelo gráfico, que num mesmo intervalo de números de sete dígitos executados pelo Crivo de Eratóstenes, há um aumento de 50000 milissegundos, o que implica que para números extremamente grandes, esse tempo seja totalmente impraticável e inviável.

No caso do AKS foi forçado o cálculo de um número não-primo, para demonstrar a sua velocidade em eliminar números que não sejam primos. É possível analisar, que apesar do crescimento polinomial do tempo de execução para a descoberta de números primos, ao se deparar com um número não-primo, o custo computacional cai muito, tendendo a zero. Ao analisarmos o gráfico do Monte-Carlo, vemos o quanto o tempo do cálculo pode crescer para números extremamente grandes, porém, ainda se mostra bem mais viável sua utilização do que o algoritmo AKS.

Como visto anteriormente, o Monte-Carlo, apesar de ser probabilístico, nos mostra com 99,9% de certeza se um número é primo ou não. Apesar de ser uma probabilidade, com certeza esse algoritmo é melhor de se usar, pois seu custo computacional é muito inferior ao AKS.

Para entendermos melhor esse custo, basta analisarmos nas listas de entradas, o tempo de execução do número 3.178.729. Enquanto no AKS este número foi verificado em 8.452 milissegundos, no Monte-Carlo ele é verificado em 1 milissegundo, contra 90.481 milissegundos do Crivo de Eratóstenes. É notável citar, que o Crivo foi testado apenas por ser um algoritmo clássico da descoberta de primalidade, mas que já se tem conhecimento sobre o seu alto custo para números maiores.

O Monte-Carlo é tão mais eficiente, que para termos resultados nessa amostra, foi necessário aumentar em muito os valores testados por esse algoritmo, para poder retornar algum tempo considerável. Observe que o algoritmo começa a demonstrar esforço computacional apenas com números superiores a nove dígitos.

Para evidenciarmos melhor o caso, basta compararmos a complexidade dos três algoritmos. A complexidade do algoritmo AKS é de $O(\log^{21/2} n)$ (AGRAWAL et al., 2006). O Crivo de Eratóstenes tem uma complexidade $O(n(\log n)(\log\log n))$ (BECK, 2010), enquanto o algoritmo Monte-Carlo tem complexidade $O(\log n)$ (BATEMAN, DIAMOND, 2004).

Note que a complexidade do algoritmo AKS é dez vezes superior a do Monte-Carlo, enquanto a do Crivo de Eratóstenes é n vezes superior.

Com estes resultados em mãos, podemos inspecionar o impacto da segurança nos algoritmos de criptografia baseados em números primos. Apesar de o algoritmo AKS ser um algoritmo pertencente à classe P, ele não demonstra perigo aos sistemas de criptografia baseados em números primos, como o RSA, visto que, sua complexidade ainda é aproximadamente dez vezes superior à complexidade do algoritmo Monte-Carlo, que é um dos algoritmos mais utilizados para descoberta de primalidade nos dias de hoje.

Quando o algoritmo AKS foi publicado, certo caos foi causado, pois, se o algoritmo realmente descobrisse números primos em tempo polinomial

"viável", o sistema de criptografia RSA, por exemplo, estaria acabado. Sites da internet e salas de bate-papo citaram o evento da descoberta desse algoritmo, como o "fim do mundo", "caos geral" ou "toda segurança iria por água abaixo", caso esse algoritmo realmente tivesse a eficiência que se esperava. O que se pode constatar, é que apesar de ser um algoritmo polinomial determinístico, o polinômio da complexidade do AKS ainda é um polinômio muito grande, o que implica em resultados extremamente demorados e inviáveis.

5. CONCLUSÃO

Este projeto foi realizado em um esforço de implementar e observar os algoritmos de teste de primalidade. Neste caso, trabalhamos com um algoritmo determinístico, o AKS, um algoritmo de teste probabilístico, o Monte-Carlo e o mais antigo algoritmo de descoberta de números primos que se tem conhecimento, o Crivo de Eratóstenes. Formas mais eficientes de implementação dos algoritmos com certeza existem, mas ressalvo a dedicação em busca de uma análise de resultados de comparação de desempenho, e não em obter algoritmos mais eficientes.

O algoritmo AKS é o único algoritmo determinístico para teste de primalidade em tempo polinomial, sendo o resultado obtido com uma confiança de 100%. No caso deste trabalho, a implementação do AKS foi baseada em cima de um código [28] já existente e adaptada para as necessidades do mesmo. No processo de execução do algoritmo, verificou-se que, apesar do resultado encontrado, ele é extremamente ineficiente. Embora o algoritmo seja provado matematicamente determinístico e o tempo de execução estar em tempo polinomial, o tempo de execução ainda é extremamente insatisfatório. A complexidade decorre da necessidade de avaliar esta expressão:

$$(x-a)^n \neq (x^n - a)$$

Se *n* é grande, precisamos de um algoritmo muito eficiente para avaliar a expressão acima. Esta é a idéia central do algoritmo, porém, sem dúvida o algoritmo AKS é de grande interesse acadêmico.

No entanto, a implementação utilizada, mostra que esse algoritmo leva um tempo considerável no cálculo dos testes de igualdade polinomial. Para um número de 9 dígitos ele leva vários minutos enquanto que o algoritmo de Monte-Carlo dá o seu veredicto em menos de um segundo com uma probabilidade de erro de $(¼)^{20}$ (de 20 tentativas). Devido a este fato, o algoritmo

AKS ainda não foi de muito uso na indústria da criptografia, que ainda prefere usar o teste de Monte-Carlo, um algoritmo de complexidade *log(n)*.

Tendo esta complexidade do algoritmo, podemos ver que o custo de se descobrir um número primo não é tão alto, porém, ele não representa risco para os principais sistemas de criptografia. Isso acontece, porque a segurança de sistemas de criptográficos mais conhecidos como o RSA, está em fatorar *n*, que é produto de dois primos p e q, imensos, e não em descobrir números primos. A fatoração de um número imenso, é extremamente difícil e qualquer aproximação que possa ser usada, chega a ser tão difícil quanto a fatoração.

Rafael T. de Souza Jr. (2010), mostra que para se retratar melhor a complexidade de se fatorar um número, basta analisar um dos algoritmos mais eficientes de fatoração que tem complexidade O(exp (sqrt (ln(n) ln(ln(n))))) onde ln denota o logaritmo natural e "exp" seu inverso. Se considerarmos um número *n* de 200 bits, e assumirmos que seja feito um passo por microssegundo, esse algoritmo demoraria alguns dias para fatorar o número, porém, se considerarmos um *n* de 664 bits (200 dígitos decimais), teríamos o resultado em alguns bilhões de anos.

Conhecendo agora a complexidade da fatoração de um número, fica evidente o porque os algoritmos de descoberta de primalidade não afetam os sistemas de criptografia baseados em números primos, afinal, a segurança está em fatorar o *n*, produto de *p* e *q* primos, e não em descobrir *p* e *q*.

Este projeto e a implementação nos deu uma oportunidade de estudar alguns aspectos da Teoria dos Números e Algoritmos da teoria dos números, com uma ênfase na aritmética modular. A partir dos resultados obtidos, observa-se a necessidade de trabalhos futuros nesta área, contribuindo principalmente na otimização do algoritmo AKS tanto quanto no surgimento de novos algoritmos determinísticos, capazes de descobrirem números primos em tempo polinomial e com mais eficiência. A necessidade de novos estudos nessa área é iminente, e

este trabalho é apenas um esboço do que ainda está por vir com novos estudos e trabalhos.

6. REFERÊNCIAS BIBLIOGRÁFICAS

Agrawal, Manindra; Kayal, Neeraj; Saxena, Nitin. **Primes is in P**. Indian Institute of Technology Kanpur, Índia, http://www.cse.iitk.ac.in/users/manindra/algebra/primality_v6.pdf versão 6, 2006.

Araújo, Vítor. **Aulas teóricas de Teoria dos Números**. 2003.

Bateman, Paul T.; Diamond, G. Arold. **Analytic Number Theory**. World Scientific Publishing, 2004

Beck, George. **Sieve of Eratosthenes**. The Wolfram Demonstrations Project. Publicado em http://demonstrations.wolfram.com/SieveOfEratosthenes/. Consultado em 25/05/2010.

Bernstein, B. **An exposition of the Agrawal-Kayal-Saxena primality proving theorem**. www.cr.yp.to/paper.html. Consultado em 18/06/2009.

Bernstein, Daniel. **Deterministic Polynomial-Time Primality Tests**. 2002.

Bernstein, Daniel. **Proving Primality in Essentially Quartic Expected**. 2003.

Berrizbeitia, Pedro. **Sharpening Primes is in P for a Large Family of Numbers.** 2003.

Bhattacharje, R; Pandey, P. **Primality testing**. 2004.

Bornemann, Folkmar. **Primes is in P – a Breakthrough for Everyman.** 05/2003.

Braga, Bruno da Rocha. **Algoritmo AKS – Primalidade de um número em tempo polinomial**. Universidade Federal do Rio de Janeiro, Rio de Janeiro, 2002.

Campello, Antonio Carlos; Leal, Isabel. **Teoria Aritmética dos Números e Criptografia RSA.** 2007.

Cerdeira, Orestes. **Umas coisitas de Grafos e uma Introdução Informal à Complexidade Computacional.** 2001.

Choi, Ji-Wung; Kumar, Neeraj. **An Implementation and Comparison of Two Primality Testing Algorithms, AKS and Rabin-Miller in C++ Programming Language**. University of California, Santa Cruz. 2007.

Chris K. Caldwell, 2010. Disponível em http://primes.utm.edu/largest.html. Consultado em 25/05/2010.

Coutinho, S.C. **Números inteiros e criptografia RSA**. Segunda edição, IMPA, 2000. 213 págs.

Coutinho, S.C. **Primalidade em tempo polinomial – Uma introdução ao algoritmo AKS.** Universidade Federal do Rio de Janeiro, 2004.

Crandall, R; Mayer, E; Papadopoulos, J. **The Twenty Fourth Fermat Number is Composite**. 2003.

Esslinger, Bernd; Koy, Henrik; Schneider, Jorg; Clausius, Thorste; Eckert, Claudia. **Cryptool Presentation.** 2008.

Euclides. **Elementos de Geometria.** Versão Latina de Frederico Commandino. 1944.

Ferreira, Cláudio Roberto. Universidade de São Paulo. http://www.ime.usp.br/~cesar/. Consultado em 12/10/2009.

Fouvry, E. **Theoreme de Brun-Titchmarsh; application au theorem de Fermat.**

Gilbert, George. **The Polynomial Time Algorithm for Testing Primality.** 2006.

Hendrik W. Lenstra Jr.: **Primality Testing with Gaussian Periods**. FSTTCS, 2002.

Humphreys, Keith. **Primes ∈ P – the AKS Algorithm**. 2005.

Implementation of AKS algorithm. Publicado em http://gpoulose.home.att.net/gc/src/AKS_cpp.txt. Consultado em 10/11/09.

J. Hartmanis, R. E. Stearns: **On the computational complexity of algorithms**. Trans. Amer. Math. Soc. **117**:285-306, 1965.

Jr, Rafael T. de Souza. **Principais aspectos na segurança de redes de computadores.** Publicado em http://www.redes.unb.br/security/criptografia/rsa/rsa.html. Acessado em 22/06/2010.

Kayal , Neeraj; Saxena , Nitin. **Towards a deterministic polynomial-time test. Technical report, IIT Kanpur, 2002.** www.cse.iitk.ac.in/research/btp2002/primality.html. Consultado em 14/06/2009.

Lentra, H.W; Pomerance, Jr and Carl. **Primality Testing with Gaussian Periods**, 2002.

Menezes, A; Oorschot, Van; Vanstone, S.. **Handbook of Applied Cryptography.** http://www.cacr.math.uwaterloo.ca/hac/. Consultado em 14/11/09.

P. L. Chebyshev: **M´emoire sur les nombres premiers**. Journal de Math. Pures et Appl. 17 (1852), 366-390.

Ribenboim, Paulo. **Números Primos: Mistérios e Recordes.** Coleção Matemática Universitária, IMPA, 2001.

Schneier, Bruce. **Applied Cryptography**. Segunda edição. John Wiley & Sons, Inc, Nova York.

Silveira, Aline Souza da; Faleiros, Antônio Cândido. **Criptografia de chave pública – O papel da aritmética em precisão múltipla**. Instituto Tecnológico da Aeronáutica, 2005.

Singh, Simon. **Fermat's Enigma**. New York: Anchor Books, 1998.

Smid, Michiel. **Primality Testing in Polynomial Time.**

Weisstein, Eric W. **Adleman-Pomerance-Rumely Primality Test**. From *MathWorld* - A Wolfram Web Resource. Disponível em http://mathworld.wolfram.com/Adleman-Pomerance-RumelyPrimalityTest.html. Consultado em 25/05/2010.

Weisstein, Eric W. **Goldbach Conjecture. From MathWorld – A WolframWebResource**. 2009.