

UNIVERSIDADE FEDERAL DE SANTA CATARINA

CENTRO TECNOLÓGICO

GRADUAÇÃO EM CIÊNCIAS DA COMPUTAÇÃO

José Daniel Alves do Prado

**Título**: Números Primos

Florianópolis/SC

2024

José Daniel Alves do Prado

**Título**: Números Primos

Trabalho individual 03 Números Primos submetido ao curso Graduação em Ciências da Computação da Universidade Federal de Santa Catarina como requisito parcial para avaliação da disciplina INE5429 – Segurança em Computação.

Orientador(a): Prof. Dr. Jean Everson Martina e Profa. Dra. Thaís Bardini Idalino.

Florianópolis/SC

2024

**SUMÁRIO**

[1 Descrição e Implementação dos Algoritmos Escolhidos 6](#_Toc167826931)

[1.1 Algoritmo Blum Blum Shub (BBS) 6](#_Toc167826932)

[1.2 Algoritmo Linear Congruential Generator (LCG) 7](#_Toc167826933)

[2 geração de números PSEUDO-aleatórios 10](#_Toc167826934)

[3 Comparação dos Algoritmos 13](#_Toc167826935)

[3.1 Tempo de Geração 13](#_Toc167826936)

[3.2 Análise de Complexidade 14](#_Toc167826937)

[3.3 Conclusão 14](#_Toc167826938)

[4 Algoritmo Miller-Rabin 15](#_Toc167826939)

[4.1 Análise de Complexidade do Algoritmo Miller-Rabin 17](#_Toc167826940)

[5 Algoritmo de Primalidade de Fermat 20](#_Toc167826941)

[5.1 Análise de Complexidade do Algoritmo de Fermat 22](#_Toc167826942)

[6 Resultado DOS TESTES DE PRIMALIDADE 23](#_Toc167826943)

[7 ANÁLISE DO RESULTADO PARA OS ALGORITMOS DE PRIMALIDADE 37](#_Toc167826944)

[7.1 TEMPOS DE GERAÇÃO 37](#_Toc167826945)

[7.2 GRÁFICOS 37](#_Toc167826946)

[7.3 CONCLUSÃO 39](#_Toc167826947)

[8 Referências 40](#_Toc167826948)

Identificação

**NOME**

1. José Daniel Alves do Prado

**MATRÍCULA**

1. 20103689

**TURMA**

1. 07208

**ENDEREÇO ELETRÔNICO**

1. [daniel.prado@grad.ufsc.br](mailto:daniel.prado@grad.ufsc.br)

**CONFIGURAÇÃO DO NOTEBOOK**

Modelo do hardware: Positivo Bahia - VAIO VJFE49F11X-B0111H

Memória: 32,0 GiB

Processador: AMD® Ryzen 5 5500u with radeon graphics × 12

Gráficos: RENOIR (renoir, LLVM 15.0.7, DRM 3.54, 6.5.0-35-generic)

Capacidade de disco: SSD 256,1 GB

Nome do SO: Ubuntu 22.04.4 LTS

Tipo do SO: 64 bits

Versão do GNOME: 42.9

Sistema de janelas: Wayland

**LINK DO GITHUB**

<https://github.com/jdanprad0/INE5429-Seguranca-em-Computacao/tree/trabalho-individual-3>

# Descrição e Implementação dos Algoritmos Escolhidos

Para este relatório, foi escolhido dois algoritmos geradores de números pseudo-aleatórios: Blum Blum Shub (BBS) e Linear Congruential Generator (LCG). Foi utilizado Python para a implementação devido à sua simplicidade e familiaridade.

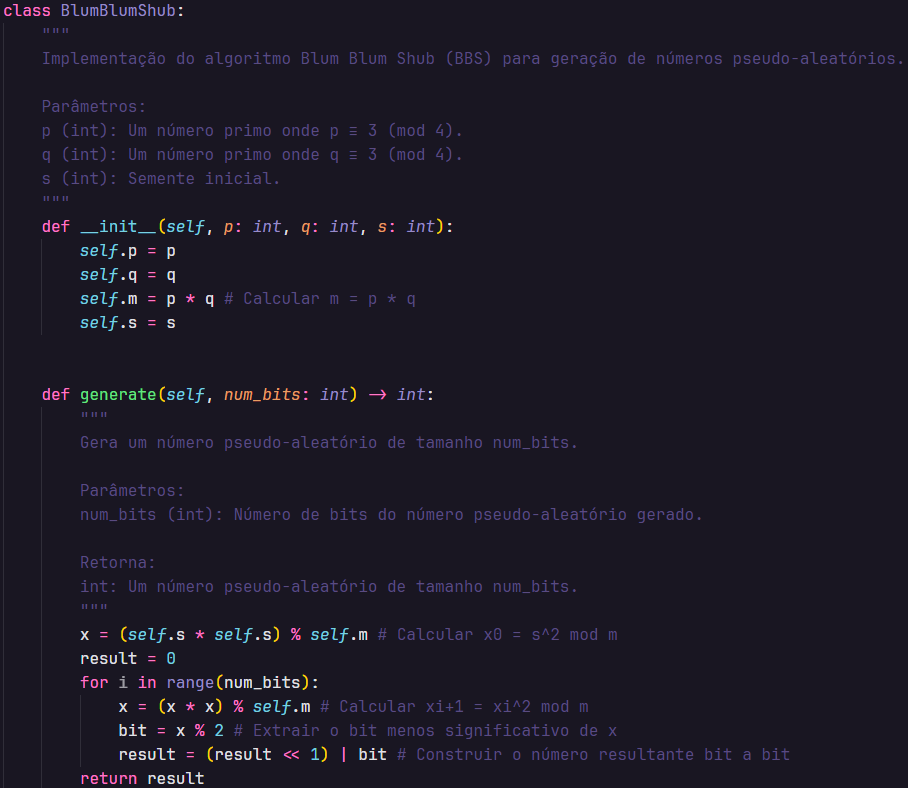
## Algoritmo Blum Blum Shub (BBS)

O Blum Blum Shub é um gerador de números pseudo-aleatórios criptograficamente seguro, introduzido por Lenore Blum, Manuel Blum e Michael Shub em 1986. Ele se baseia em propriedades de números primos e quadrados, e sua segurança se deriva da dificuldade de fatoração de grandes números [1].

Passos do Algoritmo [3]:

1. Escolha dois grandes números primos e onde .
2. Calcule .
3. Escolha um inteiro (semente) inicial, em que onde .
4. Calcule .
5. Para gerar um bit, calcule e o bit gerado é . O resultado da operação é composto pelo quadrado do resultado anterior já calculado.

Implementação em Python:



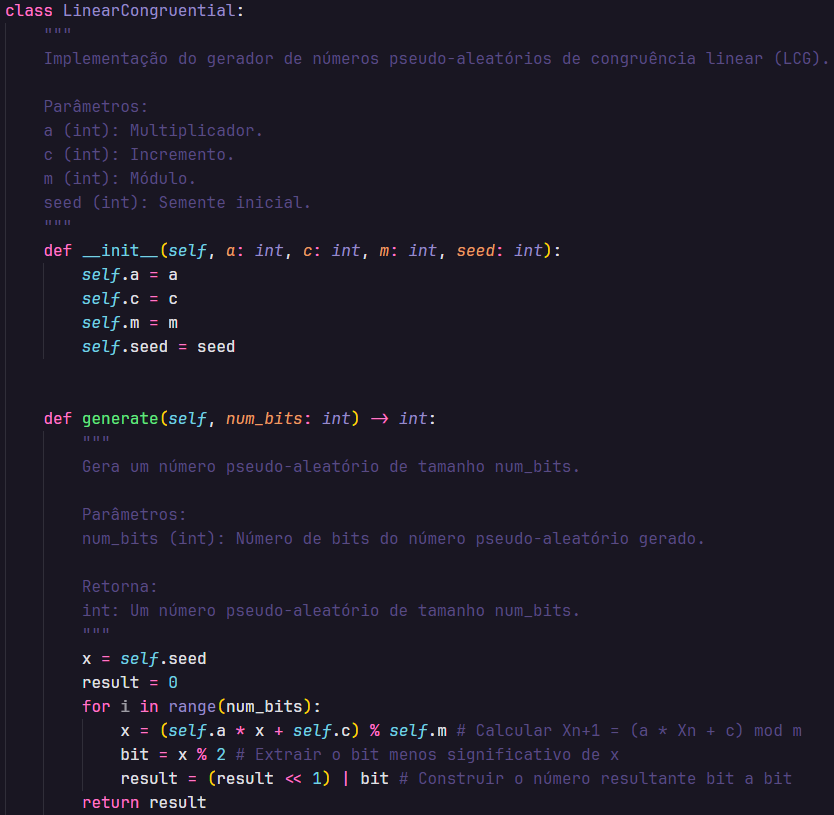
## Algoritmo Linear Congruential Generator (LCG)

O LCG é um dos métodos mais simples e populares para gerar sequências de números pseudo-aleatórios. A fórmula básica é:

Passos do Algoritmo [4]:

1. Escolha o módulo .
2. Escolha o multiplicador .
3. Escolha o incremento .
4. Escolha um valor para a semente inicial .
5. Para gerar um bit, calcule e o bit gerado é . O resultado da operação é composto pelo resultado anterior já calculado, multiplicado pelo seu respectivo multiplicador adicionado ao incremento.

Implementação em Python:



Previsibilidade do LCG:

Embora o LCG seja fácil de implementar e rápido, ele tem algumas limitações significativas em termos de segurança e previsibilidade. As principais razões para sua previsibilidade são:

* **Linearidade:** A fórmula básica do LCG é linear, o que significa que a sequência de números gerada não possui uma complexidade suficiente para evitar a previsão. Dado um número suficiente de valores da sequência, é possível resolver o sistema de equações lineares para determinar os parâmetros e .
* **Período Limitado:** O LCG pode gerar no máximo valores distintos antes de começar a repetir a sequência. Para muitos LCGs práticos, o valor de é uma potência de 2 (por exemplo, ), o que limita o período máximo e pode levar a ciclos curtos.

# geração de números PSEUDO-aleatórios

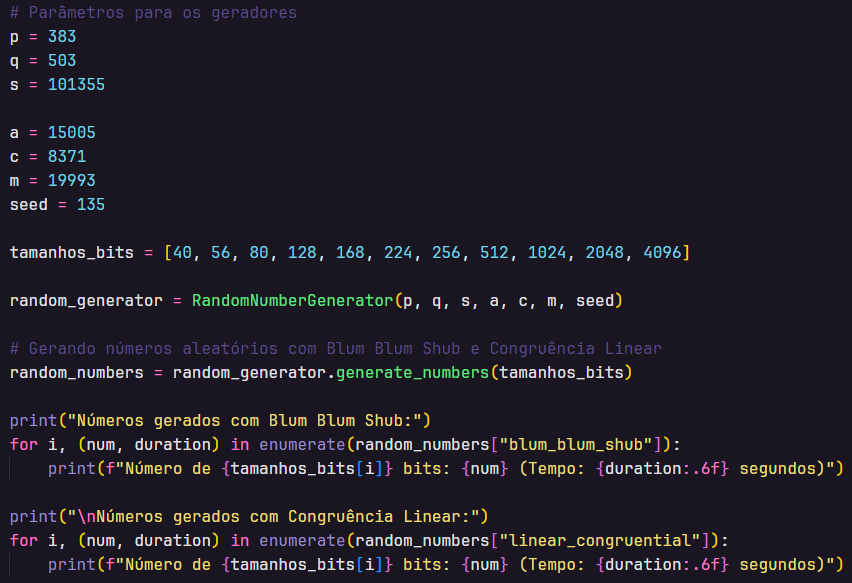
A seguir, implementamos a geração de números de diferentes tamanhos (em bits) e medimos o tempo necessário para cada um dos algoritmos.

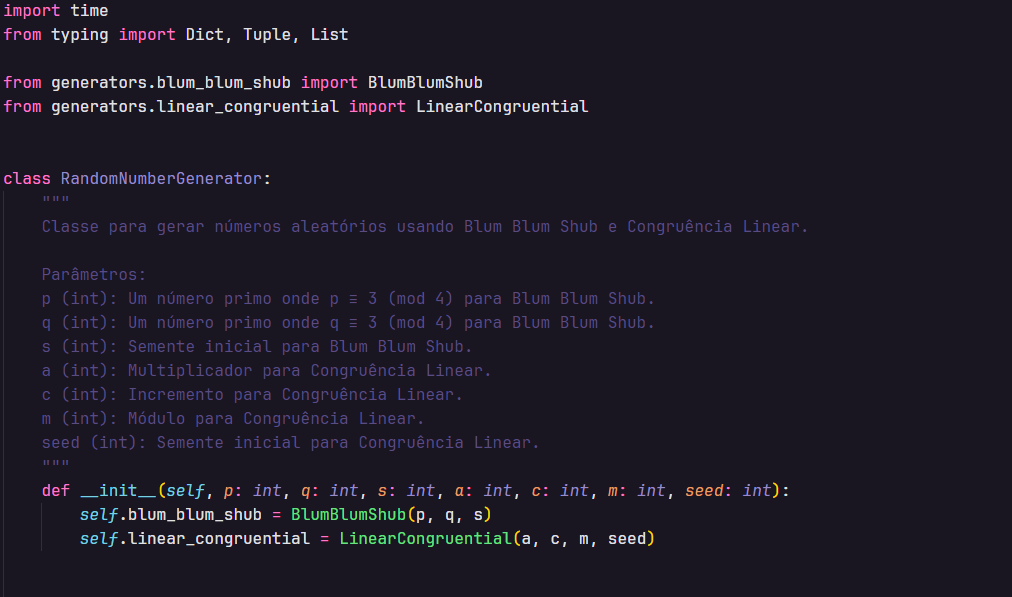
Parâmetros para BBS [3]:

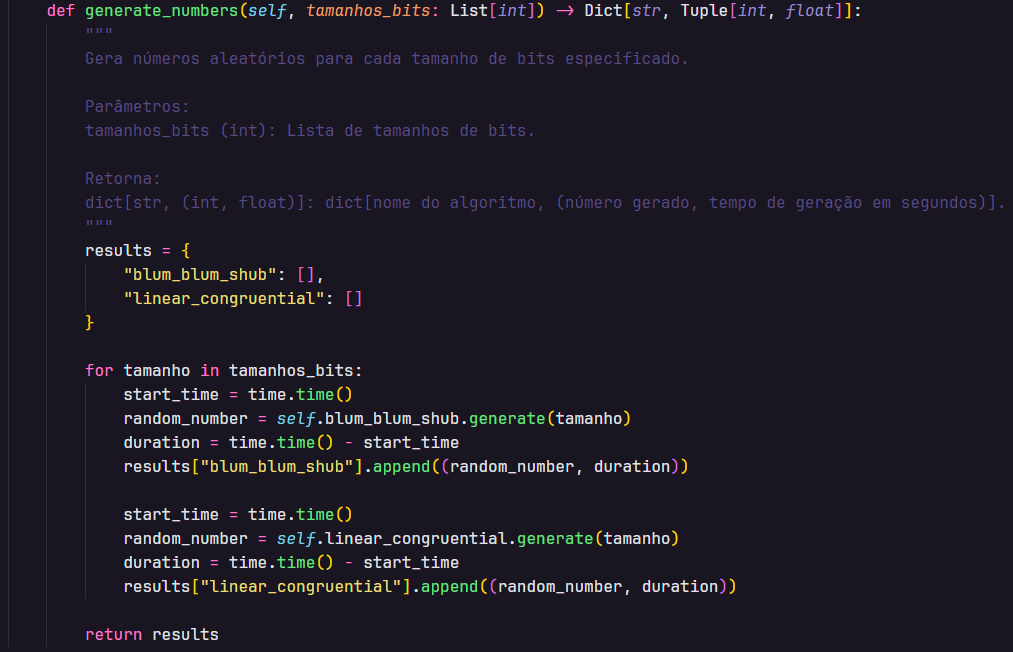
Parâmetros para LCG [10]:

Tamanhos de bits: 40, 56, 80, 128, 168, 224, 256, 512, 1024, 2048, 4096

Implementação em Python:







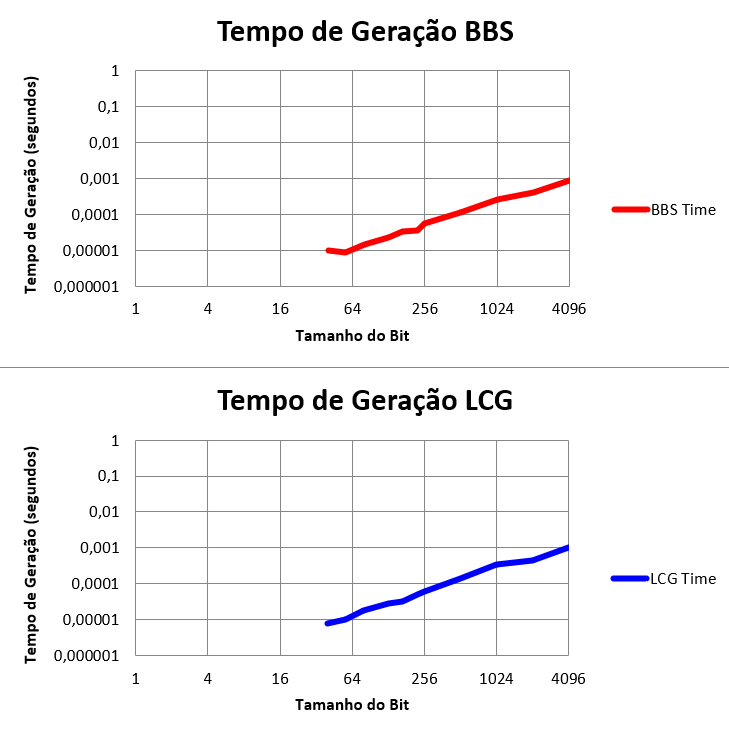
Resultados:

|  |  |  |
| --- | --- | --- |
| Algoritmo | Tamanho do Número (bits) | Tempo pra Gerar (segundos) |
| Blum Blum Shub | 40 | 0.000010 |
| Blum Blum Shub | 56 | 0.000009 |
| Blum Blum Shub | 80 | 0.000015 |
| Blum Blum Shub | 128 | 0.000023 |
| Blum Blum Shub | 168 | 0.000033 |
| Blum Blum Shub | 224 | 0.000037 |
| Blum Blum Shub | 256 | 0.000055 |
| Blum Blum Shub | 512 | 0.000113 |
| Blum Blum Shub | 1024 | 0.000256 |
| Blum Blum Shub | 2048 | 0.000406 |
| Blum Blum Shub | 4096 | 0.000891 |
| LCG | 40 | 0.000008 |
| LCG | 56 | 0.000010 |
| LCG | 80 | 0.000018 |
| LCG | 128 | 0.000029 |
| LCG | 168 | 0.000031 |
| LCG | 224 | 0.000050 |
| LCG | 256 | 0.000062 |
| LCG | 512 | 0.000136 |
| LCG | 1024 | 0.000346 |
| LCG | 2048 | 0.000436 |
| LCG | 4096 | 0.001013 |

# Comparação dos Algoritmos

## Tempo de Geração

Ambos os algoritmos geraram números pseudo-aleatórios de todos os tamanhos especificados com sucesso. Os tempos de geração foram bastante rápidos para ambos, mas o Blum Blum Shub apresentou tempos ligeiramente menores nos casos apresentados. No entanto, são valores muito baixos, a diferença não chega a ser destoante. O que nos leva a induzir que essa diferença está interligada ao tamanho dos parâmetros escolhidos e as operações entre si.



## Análise de Complexidade

* Blum Blum Shub (BBS): A complexidade do BBS é pois cada bit adicional requer uma operação de módulo que depende do tamanho dos números primos e .
* Linear Congruential Generator (LCG): A complexidade do LCG também é , já que cada iteração para gerar um bit envolve uma operação aritmética modular simples. O LCG não é adequado para aplicações criptográficas devido à sua previsibilidade, portanto, sua qualidade é baseada na escolha de seus parâmetros, e para teste, foram utilizados valores conforme sugeridos por PRESS, William H [5] após estudos.

## Conclusão

Neste relatório, foi implementado e comparado dois algoritmos geradores de números pseudo-aleatórios: Blum Blum Shub e Linear Congruential Generator. O BBS apresentou tempos de geração ligeiramente melhores e oferece maior segurança criptográfica em comparação ao LCG. Dependendo da aplicação e dos requisitos de segurança, um algoritmo pode ser mais adequado do que o outro.

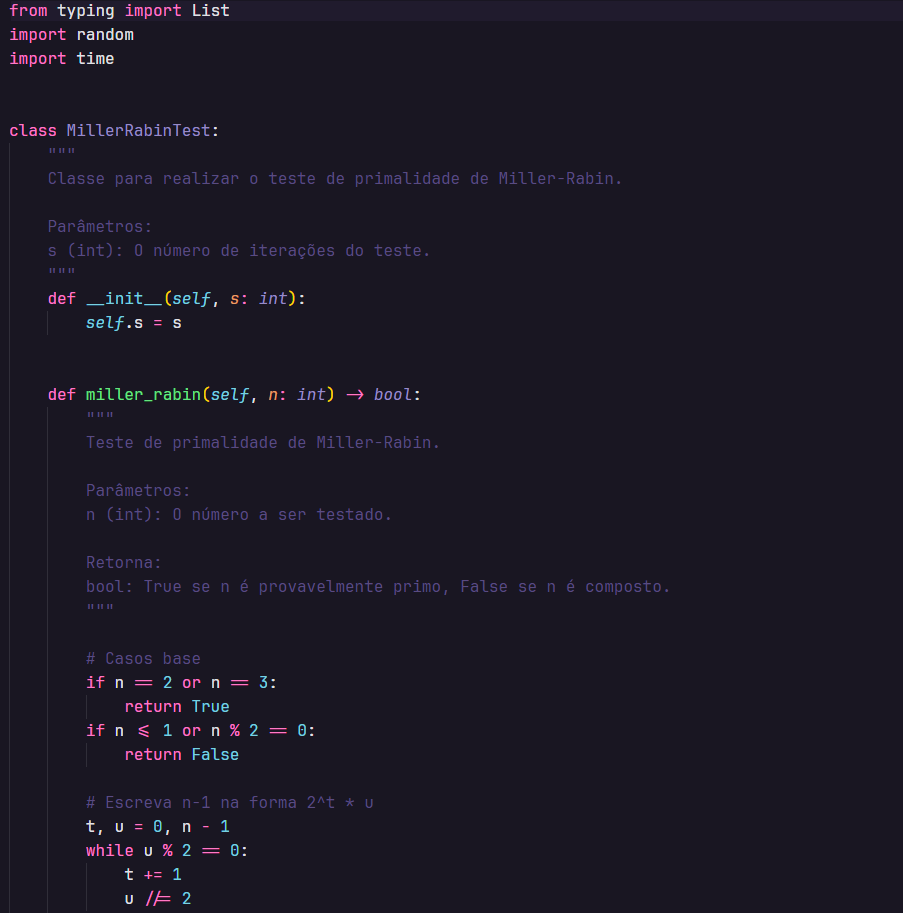
# Algoritmo Miller-Rabin

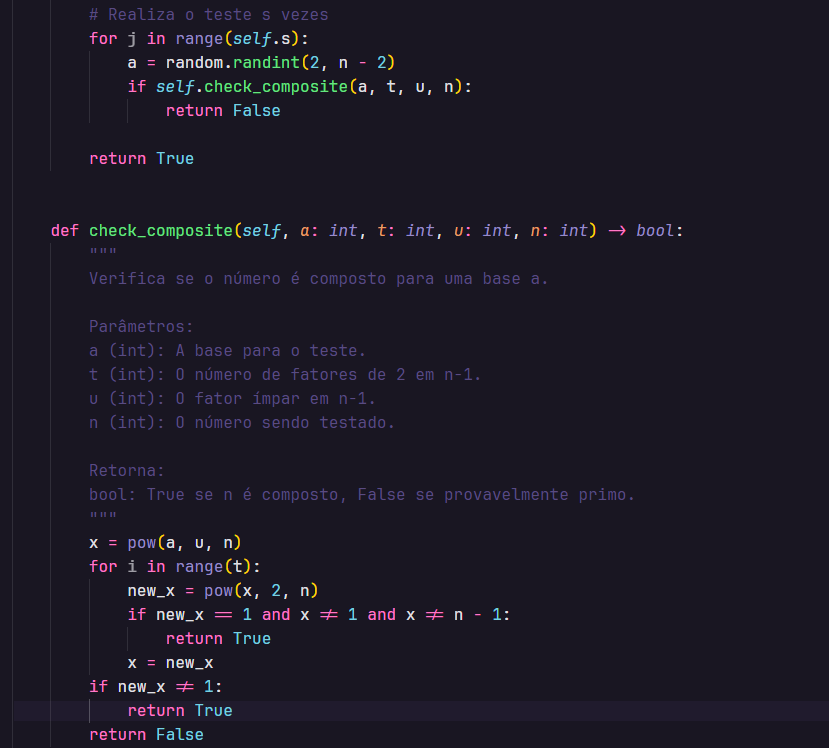
O algoritmo Miller-Rabin é um teste probabilístico para verificar se um número é primo. A ideia central é que, para um número ser primo, ele deve satisfazer certas propriedades aritméticas. O teste baseia-se em propriedades dos números em relação a uma base escolhida aleatoriamente.

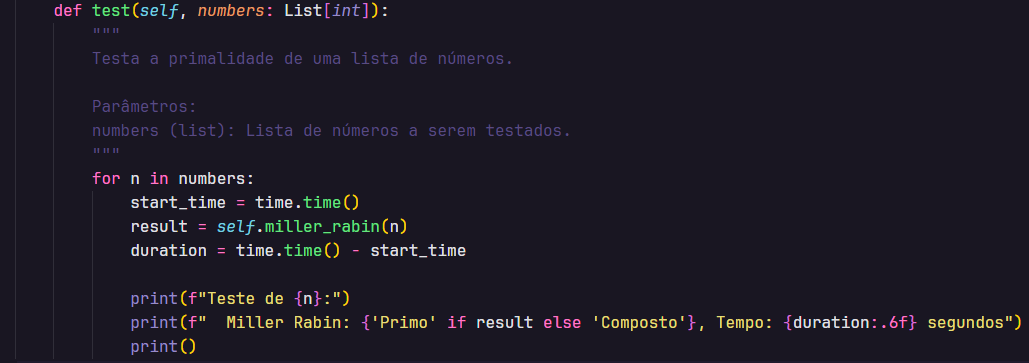
Passos do Algoritmo [6]:

1. **Decomposição do Número:** Dado um número ímpar , escreva na forma , onde é um número ímpar.
2. **Escolha de Bases:** Escolha bases aleatórias para gerar onde .
3. **Verificação:** Para cada base :
   1. Calcule .
   2. Para de 0 a :
      1. Calcule . Se e e , é composto.
   3. Se é composto.
   4. Se nenhuma das condições acimas forem satisfeitas, pode ser primo.
4. **Conclusão:** Se passar em todas as iterações, é provavelmente primo; caso contrário, é composto.

Código em Python:

****

****

****

## Análise de Complexidade do Algoritmo Miller-Rabin

O algoritmo Miller-Rabin é um teste probabilístico para verificar a primalidade de um número. A análise de complexidade leva em consideração tanto o tempo de execução quanto a eficiência do algoritmo em termos de operações necessárias para concluir o teste.

1. **Decomposição do Número na Forma :**
   1. Este passo envolve fatorar para encontrar e .
   2. A fatoração em é feita dividindo respectivamente por até que o resultado seja ímpar.
   3. No pior caso, isso requer operações de divisão, já que o número de divisões é proporcional ao número.
2. **Escolha das Bases e Verificações:**
   1. Para cada uma das iterações, escolhemos uma base aleatória no intervalo .
   2. O cálculo de usa exponenciação modular, que pode ser feita em usando o método de exponenciação rápida [7].
   3. A exponenciação requer multiplicações modulares. Cada multiplicação modular pode ser realizado em .
   4. Portando, a complexidade de .
   5. Como é no máximo , então a complexidade é .
3. **Verificação de** 
   1. Este passo é repetido vezes. Cada atribuição é uma multiplicação modular, que tem complexidade .
   2. No pior caso, este passo é realizado vezes, onde .
4. **Iteração Total:**
   1. Como realizamos iterações, a complexidade total para as verificações é .
   2. Simplificando, a complexidade por iteração é então a complexidade total é .
5. **Conclusão:**
   1. A complexidade se dá pelas exponenciações modulares que são feitas e pelo número de iterações escolhido.
   2. [6] Cormen menciona que a complexidade do algoritmo Miller-Rabin, com sendo um número de bits, exige operações aritméticas e operações de bits, pois não exige assintoticamente mais trabalho que operações modulares.

# Algoritmo de Primalidade de Fermat

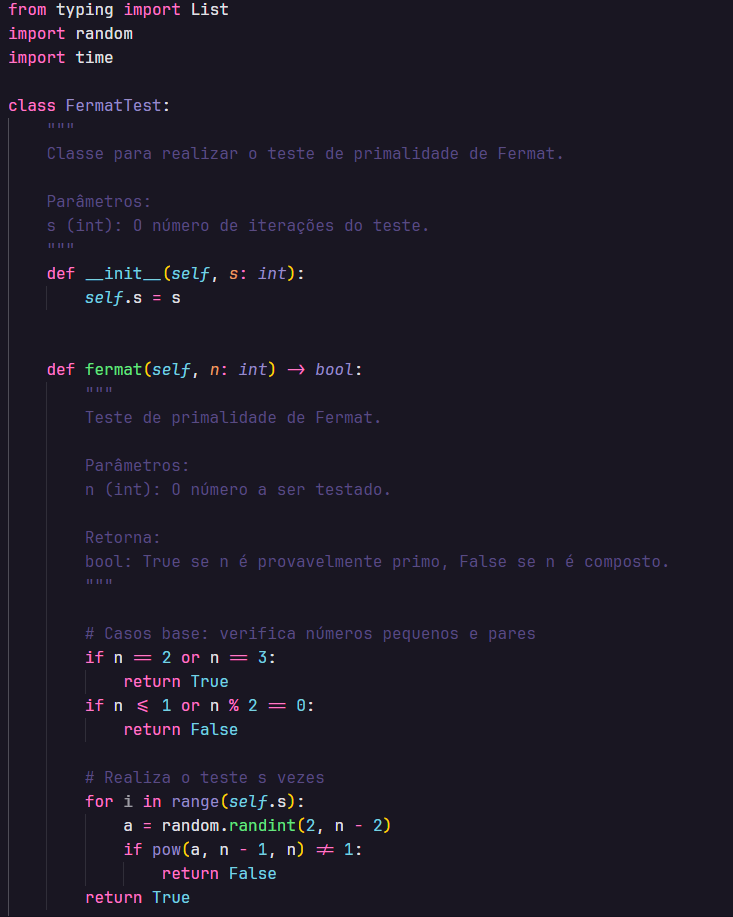
O Teste de Primalidade de Fermat é um método probabilístico para verificar se um número é primo. Baseia-se no pequeno teorema de Fermat, que afirma que se é um número primo, então

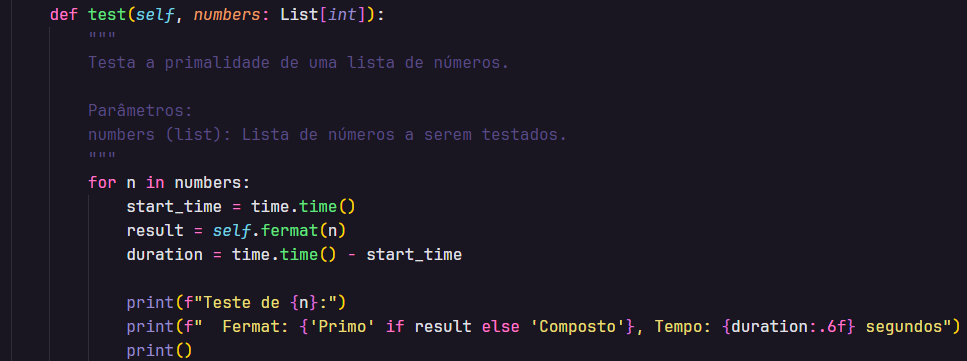
O Teste de Primalidade de Fermat foi escolhido pois é um dos algoritmos mais simples para verificar a primalidade de um número. De tal modo, é interessante ver como ele se comporta comparado com um algoritmo mais complexo como é o de Miller-Rabin.

Passos do algoritmo [9]:

1. **Escolha de bases:** escolha bases aleatórias onde [8].
2. **Verificação:** Para cada base :
   1. Calcule . Se , então é primo, do contrário, é composto.
3. **Conclusões:** Se passar em todas as iterações, é provavelmente primo; caso contrário, é composto.

Código em Python:

****

****

## Análise de Complexidade do Algoritmo de Fermat

1. **Escolha das Bases e Verificações:**
   1. Para cada uma das iterações, escolhemos uma base no intervalo [8].
   2. O cálculo de usa exponenciação modular, que pode ser feita em usando o método deexponenciação rápida [7].
   3. A exponenciação requer multiplicações modulares. Cada multiplicação modular pode ser realizado em .
   4. Portando a complexidade de é .
   5. Como é no máximo , então a complexidade é
2. **Iteração Total:**
   1. Como realizamos iterações, a complexidade total para as verificações é de .
3. **Conclusão:**
   1. A complexidade se dá pelas exponenciações modulares que são feitas e pelo número de iterações escolhido.

# Resultado DOS TESTES DE PRIMALIDADE

Todos os números obtidos foram validados por ambos os algoritmos, ou seja, os dois algoritmos de teste de primalidade respondiam “Primo” para o número gerado em todas as situações. Mais detalhes podem ser encontrados no arquivo output.txt no GitHub.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Algoritmo Gerador** | **Algoritmo Verificador** | **Tamanho do Número (bits)** | **Tempo Para Gerar (s)** | **Número Gerado** |
| Blum Blum Shub | Miller-Rabin | 40 | 0.000604 | 227877182887 |
| Blum Blum Shub | Miller-Rabin | 56 | 0.000750 | 30613074601800277 |
| Blum Blum Shub | Miller-Rabin | 80 | 0.001014 | 651432684564531651056533 |
| Blum Blum Shub | Miller-Rabin | 128 | 0.005069 | 255005855825846520748527576384406687331 |
| Blum Blum Shub | Miller-Rabin | 168 | 0.008927 | 21635907438847625183021732508294938488579834789477 |
| Blum Blum Shub | Miller-Rabin | 224 | 0.021098 | 21829146326494738484793623778111811550763244173127943046489196798937 |
| Blum Blum Shub | Miller-Rabin | 256 | 0.022905 | 49488524816903844853526848168458758216625260329202956597982717682969560125529 |
| Blum Blum Shub | Miller-Rabin | 512 | 0.125956 | 8459281068608022457068937218309746694066760600722862589293125551953621554079359164092431506642215012374686344611856086774303473664482377036573416673829251 |
| Blum Blum Shub | Miller-Rabin | 1024 | 0.350132 | 99914484872593646208632632091632995383385157632501607150655672859346193966945723918889699918880139508536843886410128925638720694731913500065585606280090203274650624397240161092417696996396891104871839902610140634847777073478508212059939665541252042991142286155721108640022009571724569856022779677410870027077 |
| Blum Blum Shub | Miller-Rabin | 2048 | 13.229811 | 28471106502594212738199680286732092625460283074372138717509779328703378766687193663517763496539930067949278315444020541504660054909689770689133316806528328007501962052512108420059671006221200394204035333275789303961004772657836290700214100091897657833228816810504071307506965362004203737982655265966353388579292194373753068813729558288430166357463416415229695501005497950669726919360684206714949417106061862479534144213764962505258498747815124941757004118458894754791010222808443410361314851240514043078446605082948011751564256796877443963338484630439006513125316367429615819961979033485640031287582080924716670071393 |
| Blum Blum Shub | Miller-Rabin | 4096 | 327.818829 | 319810385505400763375900583762309931672242425636491473610515026438142828520014892911123505915414635069457459812842106599064362962341864735257018645414850450954831826246186680201534739404992863699487406471043625084338430619221343384333300642715636201379938467829181800663144596564685337985514335094387174078695853232459032631114708970358235539975853910474342091042181304823813732329458424678527394694569044769036718140431830996637767769583610163903455184467080727531979547625500839813168331354504207666644849135917440253493261025047905135696481684444741538635463551348438053488025004006369288889478671568353115291084176041321067440056695061840734732675767792536658756244772320264981506203748924456209889176298850191879724166203979669665570771382342767412413876355087523082008316320690373739569777227487160147100208178348771481235522679418920677006972738422173827759590529778916900188814096072902040788618994432516173306617990893441990449950257529350485846558356579081878349176664123210655980511307188400808597495778374741570705430373374437632465845628944487840479455978953689925477694590965267642196312533863715947527522051387490968357172502170469200566135207823722988261282937895824832047592429909958503577597701023775697421077040879 |
| Blum Blum Shub | Fermat | 40 | 0.000541 | 227877182887 |
| Blum Blum Shub | Fermat | 56 | 0.000673 | 30613074601800277 |
| Blum Blum Shub | Fermat | 80 | 0.000945 | 651432684564531651056533 |
| Blum Blum Shub | Fermat | 128 | 0.005650 | 255005855825846520748527576384406687331 |
| Blum Blum Shub | Fermat | 168 | 0.009552 | 21635907438847625183021732508294938488579834789477 |
| Blum Blum Shub | Fermat | 224 | 0.021561 | 21829146326494738484793623778111811550763244173127943046489196798937 |
| Blum Blum Shub | Fermat | 256 | 0.025817 | 49488524816903844853526848168458758216625260329202956597982717682969560125529 |
| Blum Blum Shub | Fermat | 512 | 0.124287 | 8459281068608022457068937218309746694066760600722862589293125551953621554079359164092431506642215012374686344611856086774303473664482377036573416673829251 |
| Blum Blum Shub | Fermat | 1024 | 0.349625 | 99914484872593646208632632091632995383385157632501607150655672859346193966945723918889699918880139508536843886410128925638720694731913500065585606280090203274650624397240161092417696996396891104871839902610140634847777073478508212059939665541252042991142286155721108640022009571724569856022779677410870027077 |
| Blum Blum Shub | Fermat | 2048 | 13.232701 | 28471106502594212738199680286732092625460283074372138717509779328703378766687193663517763496539930067949278315444020541504660054909689770689133316806528328007501962052512108420059671006221200394204035333275789303961004772657836290700214100091897657833228816810504071307506965362004203737982655265966353388579292194373753068813729558288430166357463416415229695501005497950669726919360684206714949417106061862479534144213764962505258498747815124941757004118458894754791010222808443410361314851240514043078446605082948011751564256796877443963338484630439006513125316367429615819961979033485640031287582080924716670071393 |
| Blum Blum Shub | Fermat | 4096 | 327.880555 | 319810385505400763375900583762309931672242425636491473610515026438142828520014892911123505915414635069457459812842106599064362962341864735257018645414850450954831826246186680201534739404992863699487406471043625084338430619221343384333300642715636201379938467829181800663144596564685337985514335094387174078695853232459032631114708970358235539975853910474342091042181304823813732329458424678527394694569044769036718140431830996637767769583610163903455184467080727531979547625500839813168331354504207666644849135917440253493261025047905135696481684444741538635463551348438053488025004006369288889478671568353115291084176041321067440056695061840734732675767792536658756244772320264981506203748924456209889176298850191879724166203979669665570771382342767412413876355087523082008316320690373739569777227487160147100208178348771481235522679418920677006972738422173827759590529778916900188814096072902040788618994432516173306617990893441990449950257529350485846558356579081878349176664123210655980511307188400808597495778374741570705430373374437632465845628944487840479455978953689925477694590965267642196312533863715947527522051387490968357172502170469200566135207823722988261282937895824832047592429909958503577597701023775697421077040879 |
| Linear Congruential | Miller-Rabin | 40 | 0.001323 | 346028905727 |
| Linear Congruential | Miller-Rabin | 56 | 0.002710 | 3949291432650937 |
| Linear Congruential | Miller-Rabin | 80 | 0.002890 | 418045946593530905965129 |
| Linear Congruential | Miller-Rabin | 128 | 0.003456 | 197631754590221660197139318512004370103 |
| Linear Congruential | Miller-Rabin | 168 | 0.015765 | 106114640430759987135880057196769434133553973845861 |
| Linear Congruential | Miller-Rabin | 224 | 0.019281 | 10520265214584507099285208134265786403433700225124378856914842408043 |
| Linear Congruential | Miller-Rabin | 256 | 0.035845 | 25781976137606568767524762621046834646435149573681735932906611543975970177031 |
| Linear Congruential | Miller-Rabin | 512 | 0.073406 | 840352179727084542636529542363130918252402189936734610768771544213036816722814568615681736294186917812259680217300703581636169336510490808195717265753069 |
| Linear Congruential | Miller-Rabin | 1024 | 1.059104 | 43426398017777611592234604520331259145671613766052813476094420435492721886356441807953990407543606011225822743560032006624459979169156012714746680562873927089139398219655133116097699546630927566884940259721984557659533155173837588104768850183872363308024426964530067687737562670006941790447213047207198119271 |
| Linear Congruential | Miller-Rabin | 2048 | 17.540953 | 22349512090823111846821587360807426518262002059325264028927250850946869081767080414417531418156605454991450411203776952650272783433231952890159796886601660095646433646282382158350525876197373185321737240901292957096552781269952468218156479141267921196912478202753884208708360032146946866822785482727663515351306239840140474458774168880750146703258923715174385659517554549408471212976785208276876658165710995253295656127237594155977621019379753335140948704442188709462056379742919881682392533327994851045703371938112008046240563045910134153478016325315771509132947436602340622280300749792467147684099568101827942000177 |
| Linear Congruential | Miller-Rabin | 4096 | 382.107086 | 465963050408088428948853807295774144625553332264856755959355910156433816045541148463785657766247939313033144437702623226576585630502842797264746562560543977900282367443940061732214596125350210348036722027693043007303531830955213164275565678622228649380653335079963568537042040500850234886511433735459694679401563084875087499339116689052389610300523228130441327326310629695382823562916936498104723176573011792999836790368185666605597336652395072835845612963411698044037924250071433117403250214331914868005884170421896806633591569776162706528151333480155404652292668431234379515660310524617144514612500113686298780465021489008540692962026667568807866978172381991697202071364205845236547015743487164703397285855183998197716382352968762694866645535579079670002199521669167694852060691628372200004626548308831324136043981000129959181003542350449212857745812556804506610830022989029347040254466348395473364629064708535864593208473330753060897240250620171460705744479347337948228203391843503349848345945936230799470443954751559374478327910100388553395138678549919761241304470032356585429269401371906982566183694034392653034968871765407708163361804346000000049525539114345183593405650953080397762798685556169242246799128704189318541071946729 |
| Linear Congruential | Fermat | 40 | 0.001156 | 346028905727 |
| Linear Congruential | Fermat | 56 | 0.002379 | 3949291432650937 |
| Linear Congruential | Fermat | 80 | 0.002622 | 418045946593530905965129 |
| Linear Congruential | Fermat | 128 | 0.003284 | 197631754590221660197139318512004370103 |
| Linear Congruential | Fermat | 168 | 0.014839 | 106114640430759987135880057196769434133553973845861 |
| Linear Congruential | Fermat | 224 | 0.018411 | 10520265214584507099285208134265786403433700225124378856914842408043 |
| Linear Congruential | Fermat | 256 | 0.034730 | 25781976137606568767524762621046834646435149573681735932906611543975970177031 |
| Linear Congruential | Fermat | 512 | 0.073168 | 840352179727084542636529542363130918252402189936734610768771544213036816722814568615681736294186917812259680217300703581636169336510490808195717265753069 |
| Linear Congruential | Fermat | 1024 | 1.060179 | 43426398017777611592234604520331259145671613766052813476094420435492721886356441807953990407543606011225822743560032006624459979169156012714746680562873927089139398219655133116097699546630927566884940259721984557659533155173837588104768850183872363308024426964530067687737562670006941790447213047207198119271 |
| Linear Congruential | Fermat | 2048 | 17.528493 | 22349512090823111846821587360807426518262002059325264028927250850946869081767080414417531418156605454991450411203776952650272783433231952890159796886601660095646433646282382158350525876197373185321737240901292957096552781269952468218156479141267921196912478202753884208708360032146946866822785482727663515351306239840140474458774168880750146703258923715174385659517554549408471212976785208276876658165710995253295656127237594155977621019379753335140948704442188709462056379742919881682392533327994851045703371938112008046240563045910134153478016325315771509132947436602340622280300749792467147684099568101827942000177 |
| Linear Congruential | Fermat | 4096 | 382.418324 | 465963050408088428948853807295774144625553332264856755959355910156433816045541148463785657766247939313033144437702623226576585630502842797264746562560543977900282367443940061732214596125350210348036722027693043007303531830955213164275565678622228649380653335079963568537042040500850234886511433735459694679401563084875087499339116689052389610300523228130441327326310629695382823562916936498104723176573011792999836790368185666605597336652395072835845612963411698044037924250071433117403250214331914868005884170421896806633591569776162706528151333480155404652292668431234379515660310524617144514612500113686298780465021489008540692962026667568807866978172381991697202071364205845236547015743487164703397285855183998197716382352968762694866645535579079670002199521669167694852060691628372200004626548308831324136043981000129959181003542350449212857745812556804506610830022989029347040254466348395473364629064708535864593208473330753060897240250620171460705744479347337948228203391843503349848345945936230799470443954751559374478327910100388553395138678549919761241304470032356585429269401371906982566183694034392653034968871765407708163361804346000000049525539114345183593405650953080397762798685556169242246799128704189318541071946729 |

# ANÁLISE DO RESULTADO PARA OS ALGORITMOS DE PRIMALIDADE

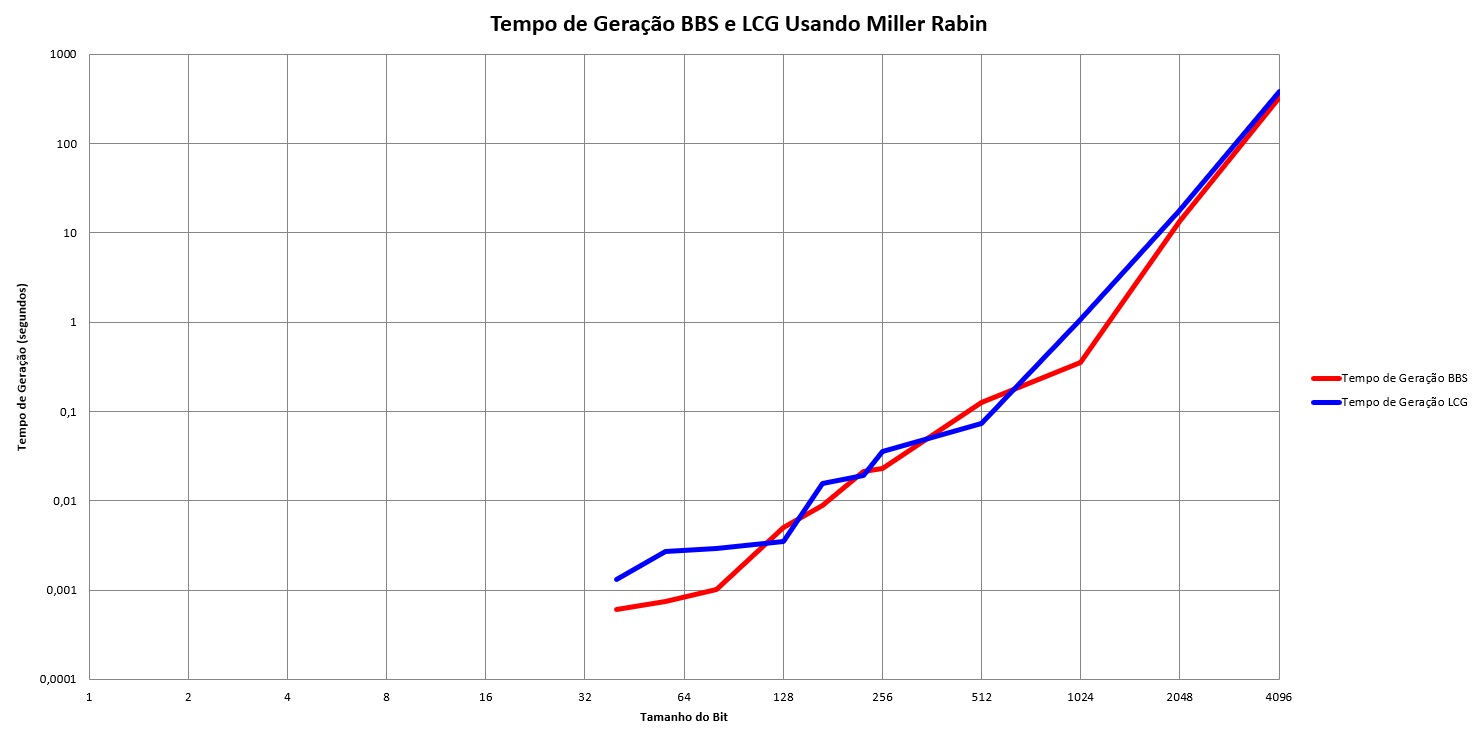
## TEMPOS DE GERAÇÃO

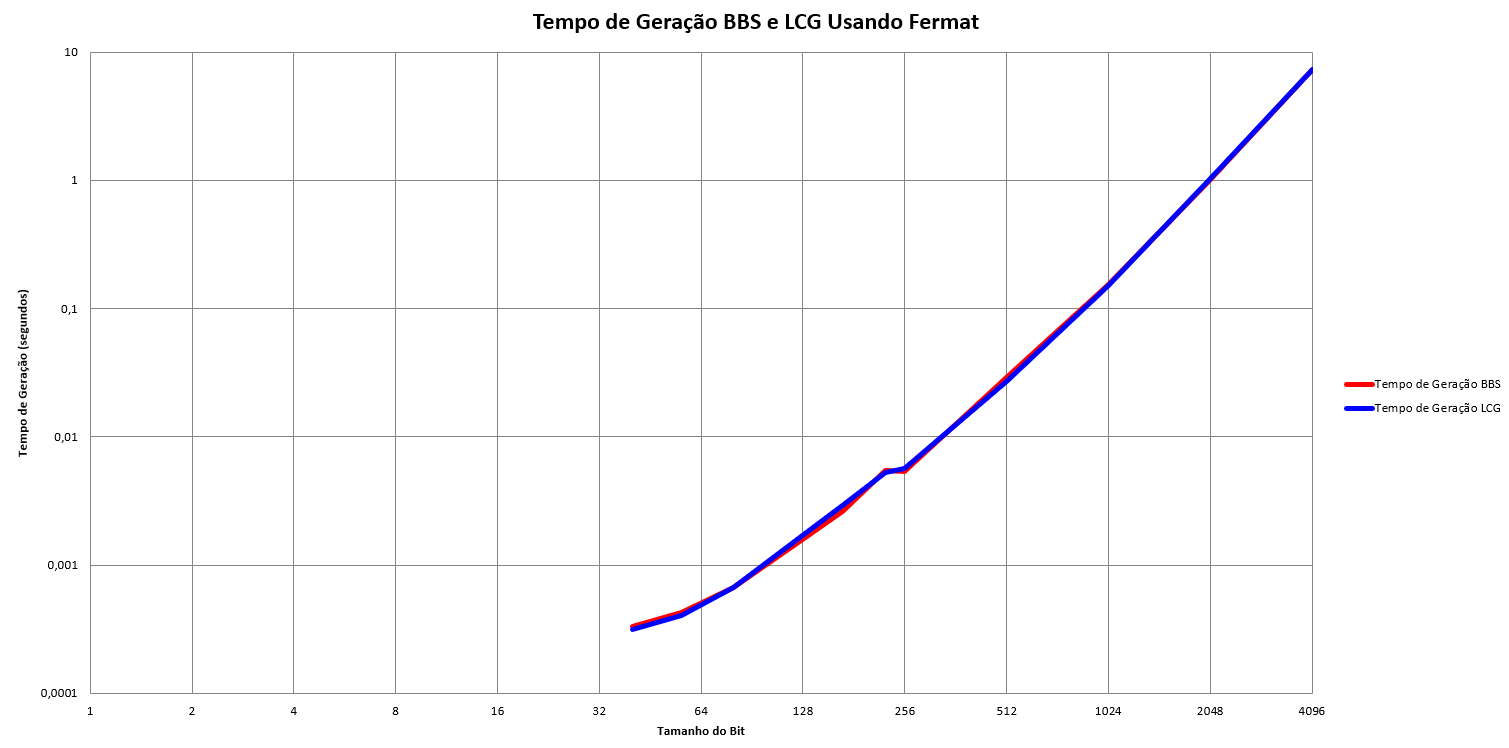
* Blum Blum Shub (BBS): O tempo de geração aumenta de acordo com o aumento do tamanho do bit. Por exemplo, a geração de números primos de 40 bits leva aproximadamente 0.000604 segundos, enquanto para 4096 bits leva cerca de 327.818829 segundos.
* Linear Congruential (LCG): Similar ao BBS, o tempo de geração para LC também aumenta com o tamanho do bit, chegando a 382.107086 segundos com um número de 4096 bits.

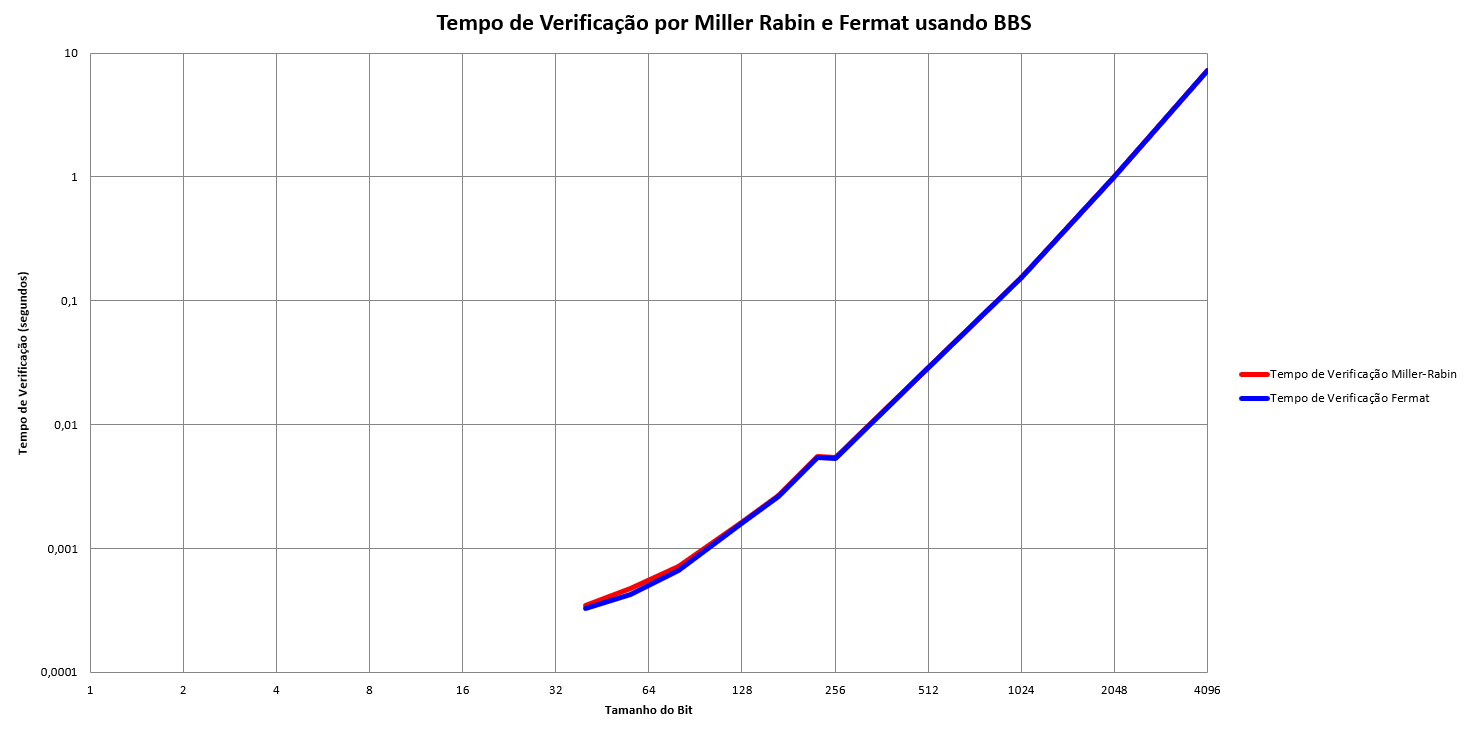
## GRÁFICOS

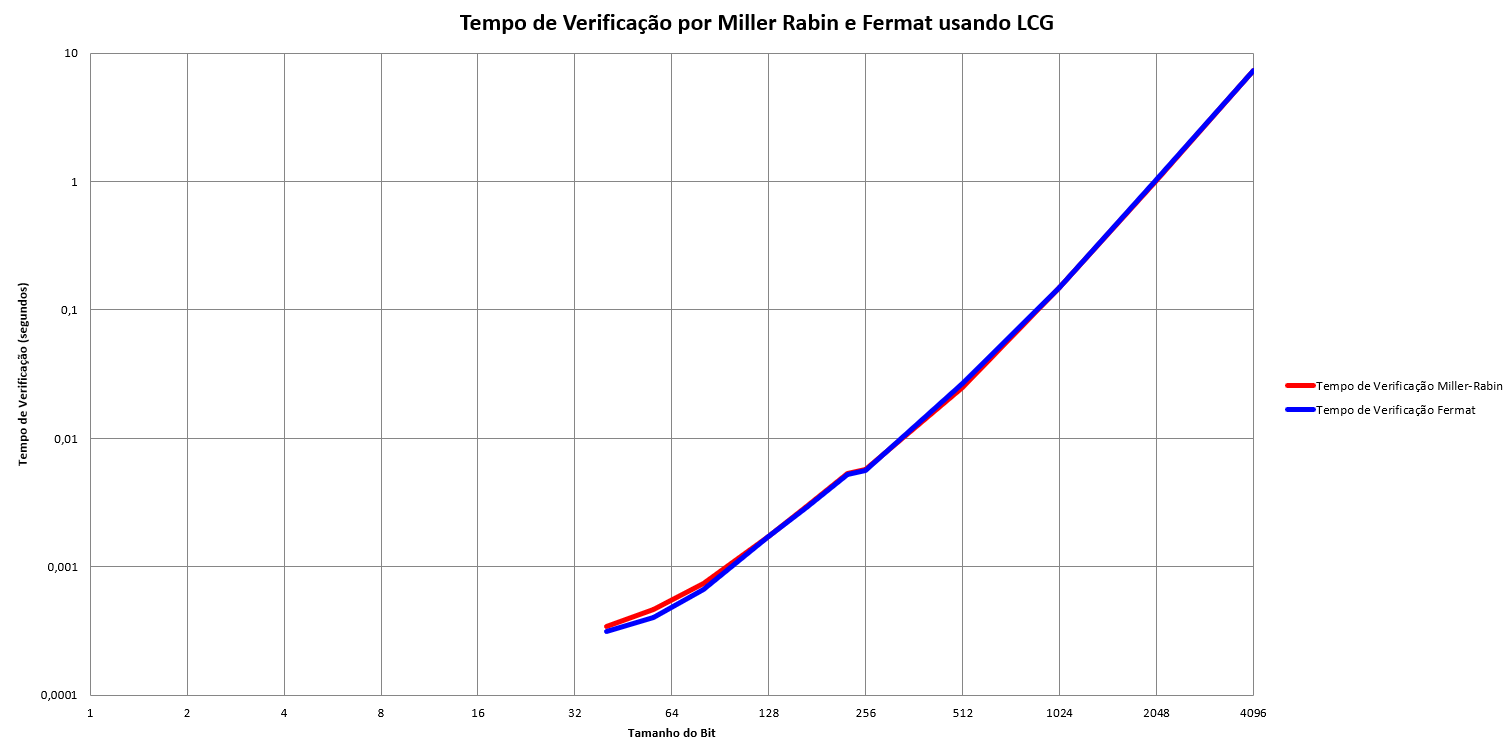
Os gráficos a seguir mostram visualmente os tempos de geração e verificação:

* Tempo de Geração: Comparação dos tempos de geração de número possivelmente primo entre BBS e LCG usando cada um dos algoritmos de verificação de primalidade implementado.
* Tempo de Verificação de Primalidade: Comparação dos tempos de verificação de primalidade de Fermat e Miller-Rabin após número possivelmente primo ser gerado com BBS e LCG.









## CONCLUSÃO

Ao gerar números possivelmente primos utilizando os algoritmos Blum Blum Shub (BBS) e Linear Congruential Generator (LCG) e verificar sua primalidade com os métodos de Fermat e Miller-Rabin, observamos comportamentos interessantes. Quando utilizamos Miller-Rabin para a verificação de primalidade, o algoritmo BBS se destaca na geração de números pequenos e grandes, enquanto para números intermediários, há uma oscilação entre os dois algoritmos. Já ao utilizarmos o método de Fermat para verificar a primalidade, os tempos de geração dos números pseudo-aleatórios por ambos os algoritmos permanecem semelhantes.

Por outro lado, quando analisamos a eficácia dos métodos de Fermat e Miller-Rabin para verificar a primalidade dos números possivelmente primos gerados por BBS ou LCG, os tempos de validação são muito próximos. Isso reflete a eficiência de ambos os algoritmos de validação de primalidade, independentemente do método utilizado. Tanto Fermat quanto Miller-Rabin apresentam a mesma complexidade computacional, como analisado anteriormente. Isso implica que o tempo necessário para verificar se um número é possivelmente primo é similar para ambos os métodos, o que é confirmado pelos resultados e pelos gráficos apresentados.

# Referências

1. BLUM, Lenore; BLUM, Manuel; SHUB, Michael. A simple unpredictable pseudo-random number generator. SIAM Journal on Computing, v. 15, n. 2, p. 364-383, 1986.
2. BLUM Blum Shub. Wikipédia, a enciclopédia livre. Disponível em: https://pt.wikipedia.org/wiki/Blum\_Blum\_Shub. Acesso em: 20 de maio de 2024.
3. STINSON, Douglas; PATERSON, Maura. Cryptography: Theory and Practice. 4ª ed. Boca Raton: CRC Press, 2019.
4. STALLINGS, William. Cryptography and Network Security: Principles and Practice. 7ª ed. Pearson, 2017. Capítulo 8.
5. PRESS, William H.; TEUKOLSKY, Saul A.; VETTERLING, William T.; FLANNERY, Brian P. Numerical Recipes in C: The Art of Scientific Computing. 2ª ed. Cambridge: Cambridge University Press, 1992. Capítulo 7.
6. CORMEN, Thomas H.; LEISERSON, Charles E.; RIVEST, Ronald L.; STEIN, Clifford. Introduction to Algorithms. 4. ed. MIT Press, 2022.
7. Exponentiation by squaring. Wikipédia, a enciclopédia livre. Disponível em: https://en.wikipedia.org/wiki/Exponentiation\_by\_squaring. Acesso em: 20 maio 2024.
8. RABIN, M. O. Probabilistic Algorithm for Testing Primality. Journal of Number Theory, v. 12, n. 1, p. 128-138, 1980.
9. Fermat Primality Test. Disponível em: https://en.wikipedia.org/wiki/Fermat\_primality\_test. Acesso em: 28 May 2024.
10. J. C. Hernndez, A. Ribagorda, P. Isasi, and J. M. Sierra. 2001. Finding near optimal parameters for Linear Congruential pseudorandom number generators by means of evolutionary computation. In Proceedings of the 3rd Annual Conference on Genetic and Evolutionary Computation (GECCO'01). Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1292–1298.