

UNIVERSIDADE FEDERAL DE SANTA CATARINA
CENTRO TECNOLÓGICO
DEPARTAMENTO DE INFORMÁTICA E ESTATÍSTICA
SISTEMAS DE INFORMAÇÃO

Ruan Luiz Molgero Lopes (20100866)
Victor dos Santos Camargo (21203408)
Enzo da Rosa Brum (22100613)

Implementação do Algoritmo Fatoração de Shor - Relatório Completo

Florianópolis
2024

1 INTRODUÇÃO

A criptografia está presente nas mais diversas ações no mundo digital. Um dos algoritmos de criptografia mais utilizados atualmente é o RSA, que baseia sua segurança na dificuldade de fatorar números inteiros grandes (COBB, 2021). Ele é amplamente utilizado em diversas aplicações, desde o comércio eletrônico até a comunicação segura entre dispositivos. No entanto, sua segurança pode estar ameaçada pelo surgimento da computação quântica.

Com a capacidade de realizar cálculos exponencialmente mais rápidos que os computadores clássicos devido a fenômenos quânticos como superposição e emaranhamento (HASNAINISTZ, 2023), a computação quântica representa uma revolução tecnológica com o potencial de transformar diversos setores. Contudo, essa mesma capacidade representa uma grave ameaça à segurança da informação, pois os algoritmos quânticos podem quebrar com relativa facilidade certos algoritmos de criptografia atualmente utilizados. Algoritmos como o RSA, que são considerados seguros nos computadores clássicos, podem ser decifrados em minutos por um computador quântico (BTQ Technologies, 2023).

Diante desse cenário, um algoritmo que pode ser utilizado para explorar criptografias baseadas na dificuldade de fatoração de números grandes é o Algoritmo de Shor, desenvolvido por Peter Shor em 1994. Esse algoritmo, quando executado em um computador quântico, é capaz de fatorar números inteiros em tempo polinomial (RAVURI, 2021), se mostrando uma ferramenta altamente eficaz para decifrar algoritmos como o RSA.

O presente trabalho tem por objetivo fazer o estudo e implementação do algoritmo de Shor, implementando-o inicialmente em sua versão clássica e posteriormente em sua versão quântica.

2 DESENVOLVIMENTO

O algoritmo de Shor é um dos mais notáveis algoritmos quânticos, pois demonstra a superioridade da computação quântica em resolver problemas normalmente difíceis na computação clássica. Utilizado para a fatoração de números inteiros, esse algoritmo combina conceitos da matemática clássica com os fenômenos da mecânica quântica. “Baseia-se na relação entre o problema de fatoração e o problema de localização de ordem, para o qual existe uma aceleração quântica” (AMICO, 2019, tradução nossa).

Para uma implementação que suporta seu uso em computadores quânticos, o algoritmo utiliza a Transformada de Fourier Quântica Semiclássica (AMICO, 2019) para encontrar o período de uma função.

2.1 IDEIA CLÁSSICA

Dado $N = n.q$ número inteiro formado pela multiplicação de n e q (números primos grandes), escolha arbitrariamente um número $g < N$. Se g for relativamente primo à N , sabe-se que g faz parte do grupo multiplicativo \mathbb{Z}_n .

Seja p a ordem de g em \mathbb{Z}_n , podemos reduzir o problema de fatorar N em nq ao problema de encontrar a ordem de um g qualquer por meio da seguinte equação (SHOR, 1999):

$$g^p \equiv 1 \pmod{n}$$

Que pode ser convertida para:

$$g^p = m * N + 1$$

Somando -1 em ambos os lados:

$$g^p - 1 = m.N$$

Trabalhando o lado esquerdo chega-se em:

$$(g^{p/2} + 1)(g^{p/2} - 1) = m.N$$

Tem-se então uma equação no formato $A.B = m.N$, que é o desejado inicialmente, 2 fatores que multiplicados resultam em N , à exceção de um fator m . $(g^{p/2} \pm 1)$ são os "chutes" g melhorado. Como estamos lidando com $m.N$ e não apenas N , A e B podem não ser exatamente fatores de N , mas sim da forma:

$$(a.n)(b.q) = m.N$$

onde a e b são escalares multiplicando fatores de N . Mas, isso não se torna um problema pois temos o algoritmo de Euclides para encontrar o fator em comum, e com isso quebrar a chave N do RSA.

Naturalmente, para g ser um bom “chute” e a fatoração ocorrer corretamente, precisamos que duas condições sejam cumpridas:

1. A ordem de g deve ser um número par
2. $(g^{p/2} + 1)$ ou $(g^{p/2} - 1)$ deve ser fator de N .

2.2 USO DA COMPUTAÇÃO QUÂNTICA NO ALGORITMO DE SHOR

Como mostrado na seção anterior, o problema da fatoração de um número N pode ser reduzido ao problema de achar a ordem de um número g no grupo multiplicativo \mathbb{Z}_N .

Contudo, mesmo com tal redução, computadores clássicos ainda não são capazes de encontrar a ordem de g eficientemente para um N suficientemente grande (SHOR, 1999). Nesse sentido, podemos utilizar a computação quântica para acelerar isso consideravelmente.

Para utilizar computação quântica na resolução desse problema, precisaremos reduzir o problema de encontrar a ordem de um número para o problema de encontrar o período no qual uma sequência de termos se repete. Isso pode ser entendido da seguinte maneira:

Imagine uma sequência de números g^0, g^1, \dots, g^p tal que:

$$g^0 \equiv 1 \pmod{N}$$

$$g^1 \equiv ? \pmod{N}$$

...

$$g^p \equiv 1 \pmod{N}$$

Devido ao $(\text{mod } N)$, sabe-se que os restos do lado direito da equação são periódicos, de modo que o valor observado em um termo g^i da sequência será observado novamente em um termos g^{i+kp} , onde k é uma constante e p é o período no qual a sequência se repete. Nesse sentido, também podemos dizer que p é a ordem de g em \mathbb{Z}_N , visto que, dado um g^0 igual à 1, g^p com toda certeza será congruente a 1.

Para realizar a computação descrita na seção anterior utilizando os princípios da mecânica quântica, que permitem explorar a superposição e a interferência para resolver problemas de fatoração de forma eficiente, é necessário seguir alguns passos fundamentais

1. Preparação de um estado de superposição quântico que vai de 1 até $N-1$ ($|1\rangle + |2\rangle + \dots + |N-1\rangle$)
2. O estado anterior então passa por um circuito que calcula g^p para cada elemento da superposição gerando como saída uma nova superposição ($|1, g^1\rangle + |2, g^2\rangle + \dots + |N-1, g^{N-1}\rangle$)

3. Novamente a superposição passa por um circuito que retorna os restos do elementos da superposição anterior frente à N ($|1, r_1\rangle + |2, r_2\rangle + \dots + |N-1, r_{N-1}\rangle$)
4. Mede-se o estado, ao fazer isso ocorre o colapso do estado quântico até então e obtêm-se uma medida de resto qualquer
5. O estado colapsado conterá então apenas os elementos referentes ao resto obtido na medida, além disso a frequência p estará "escondida" neste estado, visto que cada elemento está a uma distância p de seu vizinho.
6. Utiliza-se então a versão quântica da transformada de Fourier para aferir o valor de $1/p$ (distância entre elementos do estado quântico anterior).
7. Calcula-se p fazendo a inversa de $1/p$

O pseudo algoritmo anterior faz uso da superposição, propriedade intrínseca de sistemas quânticos, propriedade essa que permite ao algoritmo "olhar" milhares de possibilidades possíveis ao mesmo tempo (algo que seria impossível num computador clássico), obtendo assim em poucos passos o valor de p que pode ser substituído em

$$(g^{p/2} + 1)(g^{p/2} - 1) = m.N$$

a fim de encontrar os fatores N .

2.3 PASSOS DO ALGORITMO

Como detalhado por (BEAUREGARD, 2003), o algoritmo de Shor se resume à esses passos:

1. Se N for par, retorne o fator 2.
2. Determine, de forma clássica, se $N = p^q$ para $p \geq 1$ e $q \geq 2$. Se for o caso, retorne o fator p (isso pode ser feito em tempo polinomial).
3. Escolha um número aleatório g tal que $1 < g \leq N-1$. Usando o algoritmo de Euclides, determine se $\gcd(g, N) > 1$. Se for o caso, retorne o fator $\gcd(g, N)$.
4. Use o algoritmo quântico de busca de ordem definido na seção 2.2 para encontrar a ordem r de $a \bmod N$.
5. Se r for ímpar ou, se r for par mas $g^{r/2} \equiv -1 \pmod{N}$, volte para o passo (iii). Caso contrário, compute $\gcd(g^{r/2} - 1, N)$ e $\gcd(g^{r/2} + 1, N)$. Verifique se algum desses valores é um fator não trivial de N . Se for, retorne o fator. Senão, volte ao passo (iii)

2.4 EXEMPLO DO ALGORITMO EM FUNCIONAMENTO

Seja $N = 15$.

1. N não é par, seguimos para o próximo passo
2. N não é uma potência de um primo existente, logo seguimos para o próximo passo
3. Seja $g = 2$.
4. É fácil ver que a ordem r de g é 4 ($2^4 \equiv 1 \pmod{15}$), mas imaginemos que isso tenha sido encontrado de maneira quântica

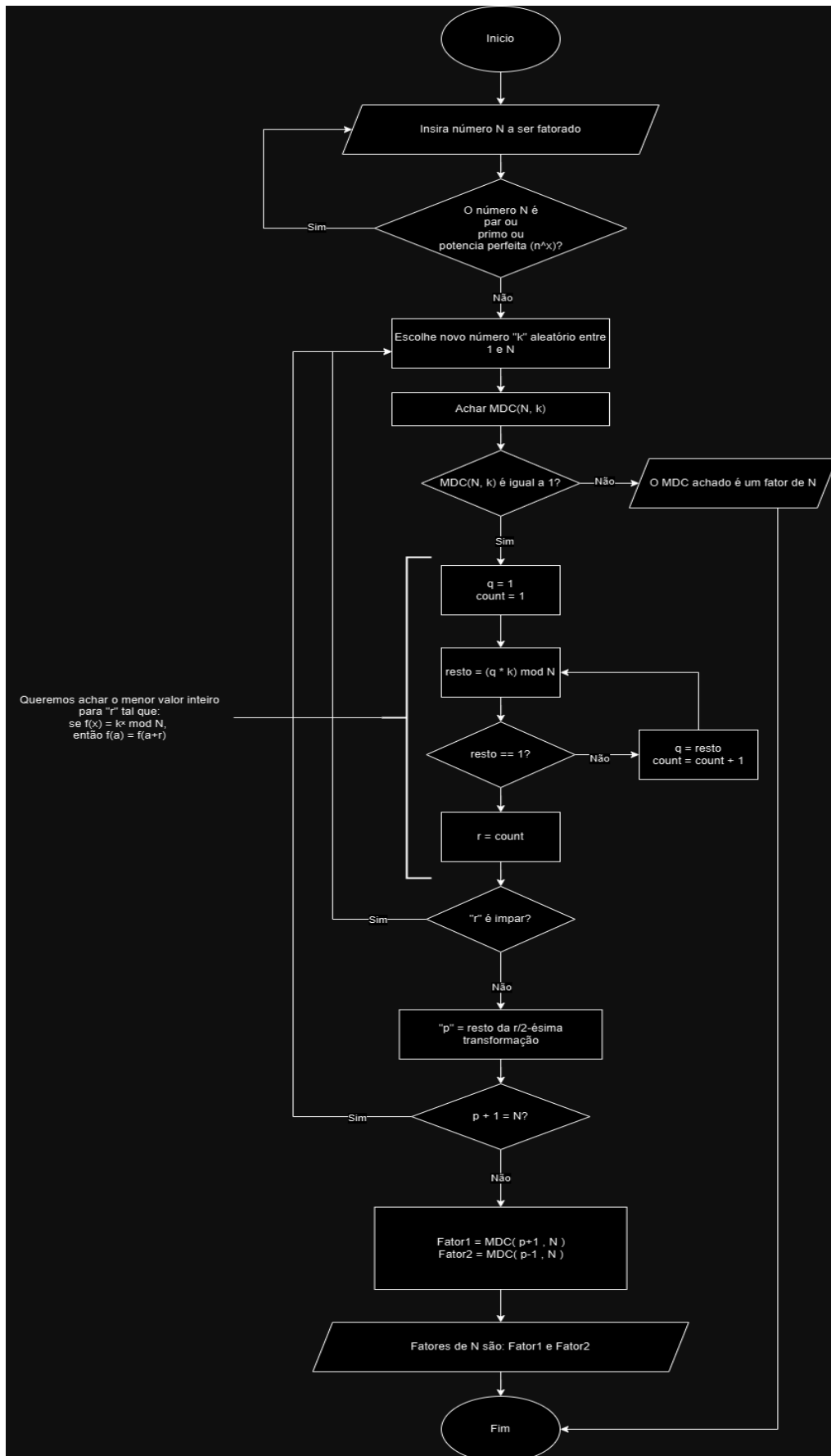
5. Como r é par e $2^{4/2} \not\equiv -1 \pmod{15}$, temos $\gcd(3, 15) = 3$ e $\gcd(5, 15) = 5$ e descobrimos os fatores de 15.

3 EXPERIMENTO

Para o experimento, será realizada uma implementação do Algoritmo de Shor na linguagem de programação Python. Mesmo sendo um algoritmo utilizável por computadores quânticos, ele também pode ser implementado em computadores clássicos com bibliotecas que simulam a computação quântica, sem aproveitar dos ganhos de computação gerados pela computação quântica.

Uma visão geral do algoritmo que será implementado pode ser vista no fluxograma da imagem 1, também disponível no [link](#), que busca trazer uma ideia geral de como funcionará o fluxo de execução. Ele é baseado no algoritmo já descrito até então e no passo a passo descrito sobre o algoritmo no trabalho de RAKHADE, 2020, que pode vir a necessitar adaptações conforme ocorre o desenvolvimento.

Imagem 1: Fluxograma



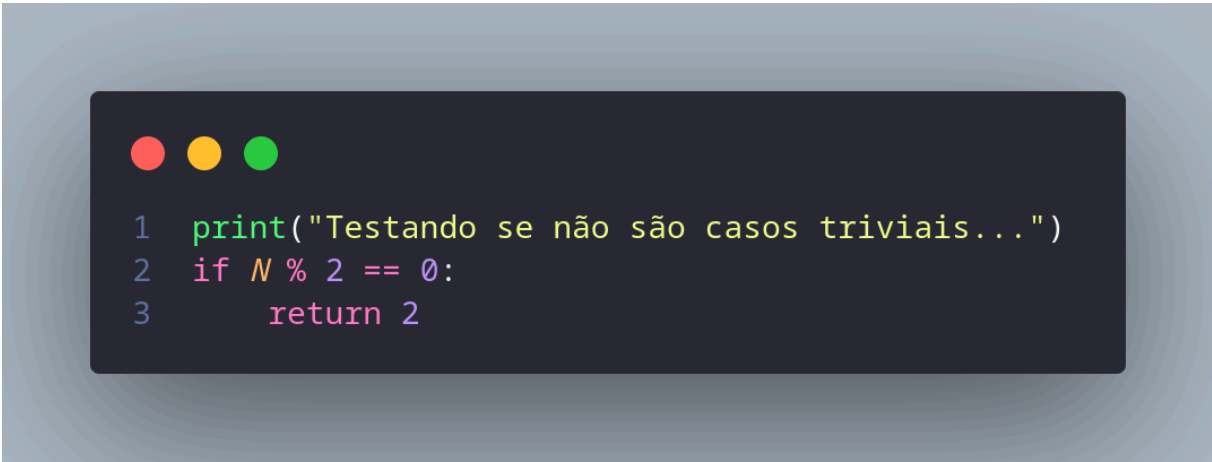
Fonte: do autor

4 IMPLEMENTAÇÃO

Para esta sessão optou-se por fazer a implementação do algoritmo na sua versão clássica inicialmente, assim como foi descrito no fluxograma presente na seção 3, a versão quântica começou a ser desenvolvida mas devido a algumas dificuldades, não chegou a ser completada.

4.1 A CHECAGEM

Após o recebimento de N , deve-se checar se o número é par ou é uma potência perfeita (a^b). A primeira checagem é trivial, faz-se o módulo com 2 e checa-se o resto 0, já a segunda existe um algoritmo



```
1 print("Testando se não são casos triviais...")
2 if N % 2 == 0:
3     return 2
```

4.2 GARANTIR QUE N NÃO É POTÊNCIA DE UM PRIMO

Aqui, verifica-se em tempo polinomial se N é a potência de um primo. Para fazer isso, é suficiente checar a e -ésima raiz de N para todos os valores entre 2 e $\text{floor}(\log_2(N))$. Como $2^{\log_2(N)} \leq N$, sabe-se que $p^{\log_2(N) + 1} > N$ para qualquer primo p



```
1 # 2. Caso onde  $N = a^b$ 
2  $n = N.bit\_length()$  # limite superior para b
3  $y = \text{int}(\log_2(N))$ 
4 for b in range(2, n+1):
5      $x = y/b$ 
6      $u1 = \text{int}(2^{**x})$ 
7      $u2 = u1 + 1$ 
8
9     if  $u1^{**b} == N$  and isPrime(u1):
10         return u1, ' $a^{**b}$ '
11     elif  $u2^{**b} == N$  and isPrime(u2):
12         return u2, ' $a^{**b}$ '
```

4.3 GARANTIR QUE N NÃO É PRIMO

Com a chamada da função isPrime, baseada em Miller-Rabin, verificamos que N é primo. Caso seja, o retornamos.



```
1 if isPrime( $N$ ):
2     return  $N$ , "N é primo"
```

4.4 ENCONTRAR A ORDEM R PARA UM INTEIRO ALEATÓRIO NO INTERVALO [2, N-1]

Por fim, caso nenhuma das condições anteriores seja alcançada, deve-se encontrar a ordem de um número aleatório g e usar tal informação para descobrir um fator de N se possível. Esse passo é repetido até que um fator seja encontrado.

```
1  while True:
2      # 3. Caso onde gcd(a, N) > 1
3      x = randint(2, N-1)
4      print(f"Número aleatório selecionado: {x}")
5      g = gcd(x, N)
6      if g > 1:
7          # O MDC achado não é um fator de N
8          print("Encontrado pelo GCD.")
9          return g, 'Encontrado pelo GCD.'
10
11     r, restos = classic_order_finding(N, x)
12     print(f"r = {r}")
13
14     if (r % 2) != 0:
15         print("É ímpar. Vamos tentar outro...")
16         continue
17
18     print("Calculando p...")
19
20     p = restos[int((r-1)/2)]
21
22     print(f"p = {p}")
23
24     if (p + 1) == N:
25         continue
26
27     fator1 = gcd(p + 1, N)
28     fator2 = gcd(p - 1, N)
29
30     print(f"Fatores são: {fator1} e {fator2}")
31
32     return fator1, fator2, "Fatores encontrados com sucesso!"
```

4.4.1 ALGORITMO CLÁSSICO

A função *classic_order_finding* é uma implementação para computadores clássicos que nos ajuda a encontrar o “r”. Esse é o principal trecho que pode ser adaptado para computadores quânticos para usar o potencial da superposição e a interferência.

```
1 def classic_order_finding(N, k):
2     print("Rodando o algoritmo clássico")
3     q = 1
4     count = 1
5     restos = []
6     resto = (q * k) % N # Valor inicial que não importa
7     print(f"Resto: {resto}")
8
9     while resto != 1:
10         restos.append(resto)
11         q = resto
12         count += 1
13         resto = (q * k) % N
14         print(f"Resto: {resto}")
15
16     return count, restos
```

5 CONCLUSÃO

O presente projeto teve como objetivo a implementação do algoritmo de Shor, abordando tanto sua parte clássica quanto quântica. Devido a imprevistos relacionados à implementação da parte quântica, a implementação concentrou-se na etapa clássica do algoritmo.

A parte clássica do algoritmo proporcionou uma análise prática e detalhada do funcionamento do algoritmo, destacando os desafios computacionais envolvidos mesmo em sua versão clássica. Embora o trabalho não tenha alcançado a integração completa com a parte quântica, os resultados obtidos foram satisfatórios. O projeto também mostrou ao grupo a importância do estudo de algoritmos híbridos (clássicos e quânticos) no contexto atual, onde o paradigma quântico está em expansão, mas ainda enfrenta barreiras práticas.

Como trabalho futuro, pensamos em completar a implementação da parte quântica e integração com algum framework (IBM Qiskit ou Amazon Braket) para realizar testes em computadores reais em nuvem.

Assim, o projeto cumpriu seus objetivos principais que eram estudo e implementação do algoritmo de Shor, contribuindo para o entendimento do grupo acerca da nova era que estamos chegando a passos largos.

6 REFERÊNCIAS

COBB, M. **RSA algorithm (Rivest-Shamir-Adleman)**. Disponível em:
<<https://www.techtarget.com/searchsecurity/definition/RSA>>. Acesso em: 3 dez. 2024.

How Far Away Is The Quantum Threat?. **BTQ Technologies**, 2023. Disponível em:
<<https://www.b tq .com/blog/how-far-away-is-the-quantum-threat>>. Acesso em: 3 dez. 2024.

HASNAINISTZ. **Quantum Computing: Why it is so fast?**. Disponível em:
<<https://medium.com/@hasnainistz/quantum-computing-why-it-is-so-fast-ddc93f3dffd>>.
Acesso em: 3 dez. 2024.

RAVURI, C. **A General Implementation of Shor's Algorithm**. Disponível em:
<<https://medium.com/mit-6-s089-intro-to-quantum-computing/a-general-implementation-of-shors-algorithm-da1595694430>>. Acesso em: 3 dez. 2024.

AMICO, M.; SALEEM, Z. H.; KUMPH, M. Experimental study of Shor's factoring algorithm using the IBM Q Experience. **Physical Review A**, v. 100, n. 1, 8 jul. 2019.

RAKHADE, K. **Shor's Algorithm (for Dummies)**. Disponível em:
<<https://kaustubhrakhade.medium.com/shors-factoring-algorithm-94a0796a13b1>>. Acesso em: 3 dez. 2024.

SHOR, P. W. Polynomial-Time Algorithms for Prime Factorization and Discrete Logarithms on a Quantum Computer. **SIAM Review**, v. 41, n. 2, p. 303-332, jan. 1999.

BEAUREGARD, S. Circuit for Shor's algorithm using $2n+3$ qubits. 21 fev. 2003