1.1. Exemplo de Aplicação do Algoritmo de Shor

Vamos mostrar a aplicação prática do algoritmo de Shor para o problema da fatoração de números inteiros. Veremos um exemplo completo de como utilizar o algoritmo para fatorar o número $N = 15$. Naturalmente, esse número é pequeno e muitos dos detalhes técnicos serão omitidos visto que seu pleno entendimento demandaria uma exposição mais longa e aprofundada sobre o assunto, o que está além do escopo deste minicurso. A Figura 1.1 mostra a entrada e as saídas do algoritmo para esse problema.

Figura 1.1: Algoritmo de Shor para Fatoração de Números Inteiros

A ideia adotada por Shor foi a seguinte: dada a função exponenciação modular definida abaixo com $a < N$ e $gcd(a,N) = 1$:

$$f(x) = a^x \ mod \ N \qquad a,N \in \mathbb{Z}_+^* \tag{1}$$

O período r dessa função é o menor inteiro não nulo que satisfaz:

$$a^r \equiv 1 \ (mod \ N) \tag{2}$$

Se pudermos computar o período de forma eficiente, então poderemos também fatorar de forma eficiente [Shor 1994]. A Figura 1.2 mostra um exemplo de função periódica do tipo $f(x) = a^x \ mod \ N$ para $a = 5$ e $N = 21$:

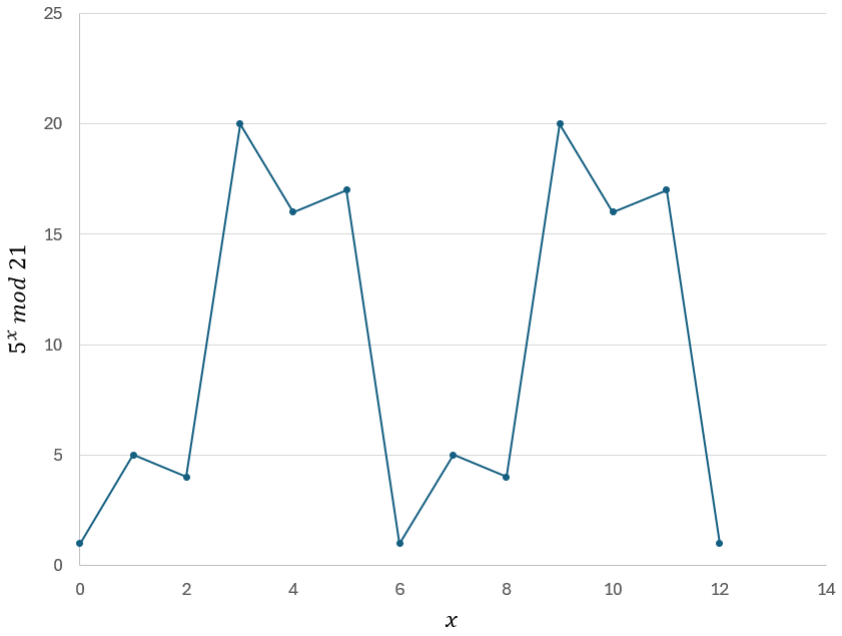

Figura 1.2: Exemplo de função periódica

Para o caso trivial $x = 0$ temos $5^0 \ mod \ 21 = 1$. Porém, esse caso não nos interessa. Note que a próxima ocorrência que satisfaz essa relação para $x \neq 0$ é $x = 6$. Dessa forma, o período é $r = 6$. É justamente essa variável r que precisamos encontrar para conseguirmos efetuar a fatoração.

Algorithm 1 Algoritmo de Shor para Fatoração de Números Inteiros

Entrada: $N \in \mathbb{Z}_+^*$, ímpar e composto da forma $N = pq$ com p e q primos
Saídas: $\{p, q\}$ ou $Erro$

```
a ←R Z*+ | 1 < a < N                          ▷ Etapa 1: pré-processamento
p ← gcd(a,N)
if p ≠ 1 then
    q ← N/p
    return {p,q}
end if
r ← x > 0 | a^x ≡ 1(mod N)                   ▷ Etapa 2: encontrando o período
if r é par then                               ▷ Etapa 3: pós-processamento
    y ← (a^(r/2) + 1) mod N
    p ← gcd(y,N)
    if p ≠ 1 then
        q ← N/p
        return {p,q}
    else
        return Erro
    end if
else
    return Erro
end if
```

O algoritmo de Shor, mostrado no Algoritmo 1, é composto de basicamente três etapas. Na primeira delas, que pode ser vista como um pré-processamento, devemos encontrar um valor adequado para a variável a. Na segunda delas tentamos encontrar o período r e na terceira fazemos o pós-processamento para encontrar os fatores p e q. A primeira e a terceira etapas podem ser realizadas de forma eficiente com computadores clássicos, sendo a segunda etapa reservada para o computador quântico. Por isso, muitas vezes o algoritmo de Shor é tratado como um algoritmo híbrido com a parte efetivamente acelerada por computadores quânticos sendo a tarefa de encontrar o período da função. Em todo caso, o algoritmo de Shor refere-se às três etapas em conjunto e não apenas à segunda, visto que um algoritmo pode ser descrito como combinações de outros algoritmos mais simples. Ele também poderia ser completamente executado em computadores quânticos, porém com custos computacionais que poderiam eventualmente ser maiores para as etapas inicial e final, algo que pode não ter grandes impactos dependendo do custo de integração com computadores clássicos para realizar a execução híbrida.

Vejamos a aplicação direta do algoritmo, considerando a obtenção do período por inspeção, para fatorar o número $N = 15$. Na primeira etapa, escolhemos $a = 13$ pois $1 < 13 < 15$ e $gcd(13,15) = 1$. Na segunda etapa, construímos a função $f : \mathbb{N} \to \mathbb{Z}_N$ tal

que $f(x) = a^x \ mod \ N \rightarrow f(x) = 13^x \ mod \ 15$. A Tabela 1.1 mostra o cálculo de $f(x)$ para alguns valores de x.

Tabela 1.1: Encontrando o período

x	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**
$f(x)$	1	13	4	7	1	13	4	7

O menor valor $x > 0$ tal que $13^x \ mod \ 15 = 1$ é $x = 4$. Então, $r = 4$. Por ser um número par, podemos continuar os cálculos, indo para a terceira etapa e fazendo $p = (13^{4/2} + 1) \ mod \ 15 \equiv 5 \ (mod \ 15)$. Verificamos que $p \neq 1$, então $q = N/p = 15/5 = 3$ e a saída do algoritmo será $\{p, q\} = \{3, 5\}$.

Antes de efetivamente vermos como implementar a segunda etapa do algoritmo de Shor, na qual tentaremos encontrar o período r utilizando um circuito quântico, precisamos aprender um pouco mais sobre dois outros algoritmos quânticos que serão essenciais para seu funcionamento.

1.1.1. Transformada Quântica de Fourier

Geralmente, pensamos nos fenômenos ao nosso redor como sendo funções do tempo. Porém, muitas vezes é útil pensar sobre eles como funções de frequências. As transformações do domínio do tempo para o da frequência e vice-versa são chamadas, respectivamente, de Transformada de Fourier (FT – *Fourier Transform*) e Transformada Inversa de Fourier (IFT – *Inverse Fourier Transform*). A transformada de Fourier permite analisar de forma adequada funções não periódicas, extraindo comportamentos periódicos subjacentes. Ela possui diversas formulações dentro da computação clássica, sendo utilizada em diferentes aplicações de engenharia e ciência no geral (processamento de sinais, remoção de ruídos, compressão de dados, etc). Apesar de geralmente aplicarmos as transformações a funções dependentes do tempo, podemos aplicá-la a funções de outros tipos de variáveis (ex: variáveis espaciais) e elas podem ser tanto contínuas (atuando sobre dados contínuos) quanto discretas (atuando sobre dados discretos).

A Transformada Quântica de Fourier (QFT – *Quantum Fourier Transform*) é a versão quântica [Coppersmith 2002] da Transformada Discreta de Fourier (DFT – *Discrete Fourier Transform*), sendo aplicada sobre as amplitudes de probabilidade dos estados quânticos. A QFT faz parte de diversos algoritmos quânticos, em particular do algoritmo de estimativa quântica de fase (QPE – *Quantum Phase Estimation*) e do algoritmo de Shor. Ela realiza a transformação entre duas bases: a base computacional (base Z, formada pelos estados $|0\rangle$ e $|1\rangle$) e a chamada base de Fourier (base X, formada pelos estados $|+\rangle$ e $|-\rangle$). A porta Hadamard é considerada uma QFT de único qubit, mapeando os estados $|0\rangle$ e $|1\rangle$ da base Z para os estados $|+\rangle$ e $|-\rangle$ da base X, conforme vimos anteriormente. De forma geral, os estados de múltiplos qubits representados na base computacional possuem estados correspondentes na base de Fourier (vimos casos particulares até aqui). Sua operação inversa, denotada $\text{QFT}^\dagger$, realiza a transformação da base X para a base Z. Ou seja, um determinado estado $|\psi\rangle$ na base computacional terá um estado correspondente na base de Fourier e vice-versa. Sendo $|\widetilde{\psi}\rangle$ o estado na base X correspondente ao estado $|\psi\rangle$ na base Z, temos:

$$QFT|\psi\rangle = |\widetilde{\psi}\rangle \quad , \quad QFT^{\dagger}|\widetilde{\psi}\rangle = |\psi\rangle \tag{3}$$

Vimos até aqui como codificar números na representação binária usando os estados da base computacional, seguindo basicamente o mesmo processo que fazemos ao contarmos em binário. Nesse caso, a contagem dos números acontece através de diferentes frequências de *bit-flips* (rotações de π radianos em torno do eixo x) aplicados ao estado de cada qubit, com os qubits mais significativos apresentando frequências menores do que os qubits menos significativos. Na base de Fourier, podemos fazer a codificação dos números utilizando fases relativas presentes nos estados dos qubits através de diferentes rotações em torno do eixo Z, com cada qubit possuindo uma frequência de rotação diferente. Cada número determina o ângulo que o estado de cada qubit é rotacionado em torno do eixo z de forma a codificá-lo de maneira única (por exemplo, se $x = 11$ na base decimal é o número que queremos codificar e $n = 4$ é o número de qubits no registrador que vai representar esse número na base X, o qubit menos significativo receberá uma fase $(x/2^n)2\pi = (11/16)2\pi = 11\pi/8$; cada qubit seguinte recebe uma fase que é o dobro da anterior). Vejamos um exemplo nas Figuras 1.3 e 1.4 para as codificações do número 11 $(0b1011)$ nas bases computacional e de Fourier.

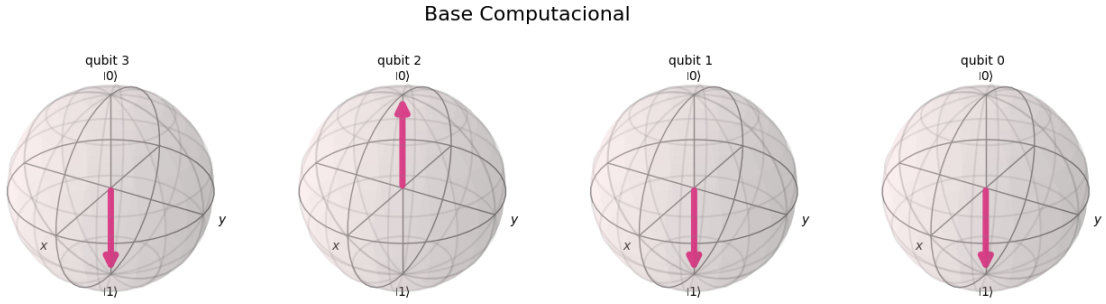

Figura 1.3: Estado $|1011\rangle$

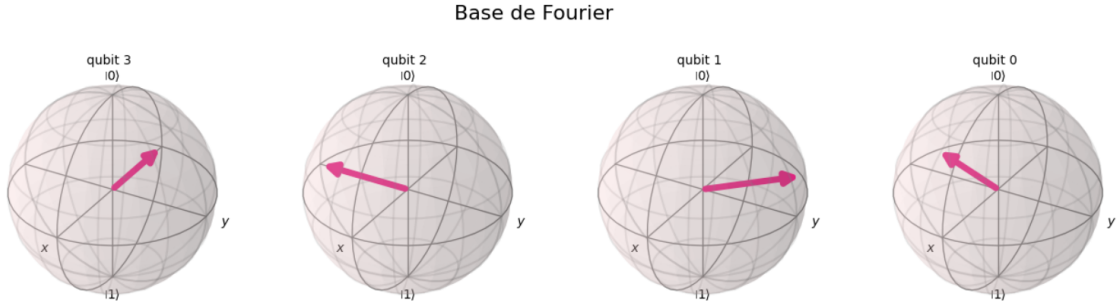

Figura 1.4: Estado $|\widetilde{1011}\rangle$

A QFT é um operador unitário e atuando sobre um estado da base computacional, seu efeito é representado pela transformação mostrada na Equação 4, com $N = 2^n$, n correspondendo ao número de qubits do sistema, i é a unidade imaginária e j é a representação decimal do estado quântico:

$$|j\rangle \rightarrow \frac{1}{\sqrt{N}} \sum_{k=0}^{N-1} e^{2\pi i jk/N} |k\rangle \tag{4}$$

De formal geral, seu efeito sobre um estado que corresponde a uma superposição qualquer dos estados da base computacional pode ser escrito como:

$$\sum_{j=0}^{N-1} x_j |j\rangle \rightarrow \sum_{k=0}^{N-1} y_k |k\rangle \tag{5}$$

Ou seja, em linhas gerais, transformamos uma superposição em outra. Esse é um detalhe importante: quando transformamos um estado quântico representado em termos da base computacional para um representado na base de Fourier, como podemos expressar os estados $|+\rangle$ e $|-\rangle$ em função dos estados $|0\rangle$ e $|1\rangle$, teremos novamente uma superposição em termos da base computacional mas com amplitudes de probabilidade diferentes, visto que é justamente sobre elas que a QFT atua. Por fim, a partir da Equação 4, podemos expressar o novo estado em termos do produto tensorial dos estados de cada qubit após a aplicação da QFT ao estado $|x\rangle$ como:

$$QFT|x\rangle = \frac{1}{\sqrt{2^n}} \bigotimes_{k=1}^{n} \left(|0\rangle + e^{\frac{2\pi i x}{2^k}} |1\rangle \right) \tag{6}$$

Expandindo a expressão, notamos como o qubit mais significativo apresenta maior frequência de rotação (observar o denominador menor na fase relativa) enquanto o qubit menos significativo apresenta frequência menor (denominador maior):

$$QFT|x\rangle = \frac{1}{\sqrt{2^n}} \left(|0\rangle + e^{\frac{2\pi i x}{2}} |1\rangle \right) \otimes \left(|0\rangle + e^{\frac{2\pi i x}{2^2}} |1\rangle \right) \otimes \cdots \otimes \left(|0\rangle + e^{\frac{2\pi i x}{2^n}} |1\rangle \right) \tag{7}$$

O circuito para implementar o algoritmo QFT é mostrado na Figura 1.5. O registrador de entrada $|x\rangle$ representa o estado de entrada ($|\Psi_0\rangle$) sobre o qual será aplicada a transformada, com o estado do qubit mais significativo de índice $n-1$ sendo representado como $|x_{n-1}\rangle$. Iniciando por ele, cada qubit recebe a aplicação de uma porta Hadamard e uma sequência de rotações controladas pelos demais qubits menos significativos em relação ao qubit de referência resultando no estado intermediário ($|\Psi_1\rangle$). A mesma ideia aplica-se ao qubit $n-2$ mas com uma rotação a menos, chegando ao estado ($|\Psi_2\rangle$). Prossegue-se dessa mesma forma para os próximos qubits até chegarmos ao estado ($|\Psi_3\rangle$). O qubit menos significativo do sistema recebe apenas a aplicação da porta Hadamard ao final do circuito chegando em ($|\Psi_4\rangle$), visto que a QFT de único qubit é a própria Hadamard e não temos qubits menos significativos do que ele para serem utilizados como controles. Por fim, portas SWAP são utilizadas para que a ordenação dos estados finais corresponda à definição matemática (se essas forem as últimas etapas antes de uma medição, as trocas podem ser feitas diretamente com os bits do resultado, evitando a aplicação de portas quânticas adicionais, o que pode contribuir para diminuir o tempo de processamento e os erros inseridos pelo uso de cada porta).

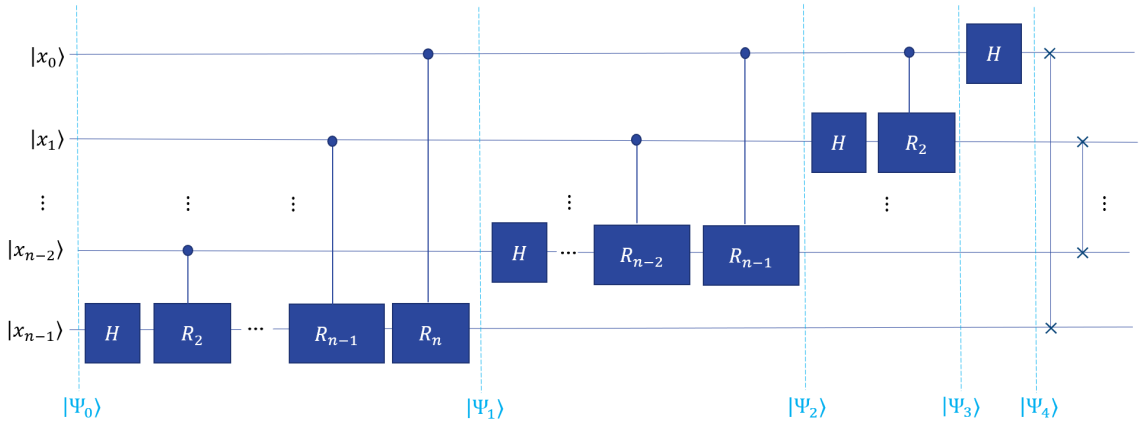

Figura 1.5: Circuito para implementar a QFT

As portas R_k são controladas e podemos implementá-las através da porta $CP(\phi)$ parametrizada adequadamente. Lembrando que a porta $CP(\phi)$ aplica um fator de fase $e^{i\phi}$ quando temos o estado $|11\rangle$ (porta habilitada em $|1\rangle$ e qubit alvo em $|1\rangle$). Por outro lado, os fatores de fase que precisamos na QFT são da forma $e^{2\pi i/2^k}$. Igualando as duas expressões, ficamos com:

$$e^{2\pi i/2^k} = e^{i\phi} \quad \rightarrow \quad \phi = 2\pi/2^k = \pi/2^{k-1} \tag{8}$$

Assim, a forma matricial desse operador será dada pela expressão abaixo parametrizando com k de acordo com os qubits envolvidos:

$$CR_k = CP(\pi/2^{k-1}) = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & e^{i\pi/2^{k-1}} \end{pmatrix} \tag{9}$$

Para inverter esse circuito e fazermos a transformada inversa ($\text{QFT}^\dagger$), basta aplicar as portas na ordem oposta, com o detalhe de que os ângulos adotados nas rotações recebem um sinal negativo. Não mostraremos novamente todo o formalismo para esse caso.

Vamos analisar brevemente o circuito em termos de complexidade computacional. Para construí-lo, precisamos das portas H e CR_k. Para o qubit $n-1$ precisamos de n portas (1 *Hadamard* e $n-1$ CR_k). Para o qubit $n-2$ precisamos de $n-1$ portas porque teremos uma CR_k a menos. Essa contagem segue até o último qubit, onde teremos apenas 1 porta. Dessa forma, a soma total dessas portas forma uma progressão aritmética de razão -1. Assim, a soma dessa PA é dada por:

$$n + (n-1) + \cdots + 2 + 1 = \frac{n(n+1)}{2} \rightarrow O(n^2) \tag{10}$$

Precisaremos também de no máximo $n/2$ portas SWAP ao final do circuito ($O(n)$).

Assim, a complexidade computacional temporal da QFT é dada por:

$$O(n^2) + O(n) \rightarrow O(n^2) \tag{11}$$

O melhor algoritmo clássico para fazer essa transformação sobre 2^n elementos (lembre-se que para n qubits teremos 2^n amplitudes de probabilidade) é o *Fast Fourier Transform* (FFT) com complexidade temporal $O(n \cdot 2^n)$ para um conjunto de $N = 2^n$ elementos.

1.1.2. Estimativa Quântica de Fase

A estimativa quântica de fase (QPE – *Quantum Phase Estimation*) é um dos algoritmos mais fundamentais da computação quântica [Kitaev 1995], sendo utilizado como bloco de construção em diversos outros algoritmos. É utilizado para estimar a fase de um autovalor λ correspondente a um autovetor $\mathbf{v}$ de um operador unitário T. Autovalores e autovetores são nomes especiais que damos, respectivamente, a escalares e vetores que satisfazem a seguinte relação:

$$T\mathbf{v} = \lambda\mathbf{v} \quad (\mathbf{v} \neq \mathbf{0}) \tag{12}$$

Ou seja, o efeito da aplicação do operador T no vetor $\mathbf{v}$ é equivalente a uma multiplicação por escalar (real ou complexo). O vetor então recebe o nome de autovetor (ou autoestado no nosso contexto de estados quânticos) e o escalar o nome de autovalor. Dado um operador unitário U, um estado quântico $|\psi\rangle$ e a seguinte relação $U|\psi\rangle = e^{2\pi i\theta}|\psi\rangle$, onde $|\psi\rangle$ é o autovetor e $e^{2\pi i\theta}$ é o autovalor, o algoritmo QPE pode ser utilizado para estimar o valor de $\theta \in [0,1)$.

O circuito que implementa o algoritmo QPE é mostrado na Figura 1.6. A ideia é explorar a técnica chamada *phase kickback*, na qual operações controladas também podem ter efeitos nos qubits de controle, na forma de modificações de fases em seus estados, através de construções específicas no circuito quântico. Teremos um registrador de entrada com t qubits (dependendo da precisão desejada) chamado *contador*, com todos os qubits inicializados no estado $|0\rangle$, que será responsável por codificar nossa estimativa para a fase. Um registrador *ancilla* (auxiliar) de tamanho adequado será preparado em um autovetor $|\psi\rangle$ do operador U, e nos ajudará a realizar o processo de *phase kickback* através das operações controladas indicadas.

Para que esse efeito possa ocorrer, primeiramente precisamos aplicar portas Hadamard ao contador para colocar todos os qubits em superposições uniformes. Em seguida, cada operação controlada da porta U será realizada um determinado número de vezes para cada qubit do *contador*, com o respectivo qubit de controle recebendo uma fase proporcional a $e^{2\pi i\theta}$. Aplicando essa sequência de portas controladas, conseguiremos codificar uma estimativa para a fase θ como um número entre 0 e $2^t - 1$.

Ao fim do processo, uma estimativa para a fase estará codificada no estado do contador seguindo o padrão de representação que vimos no algoritmo QFT, ou seja, na base de Fourier, a não ser por um fator multiplicativo. Como realizamos as medições na base computacional, após a codificação da fase ter sido realizada, precisamos aplicar a $\text{QFT}^\dagger$

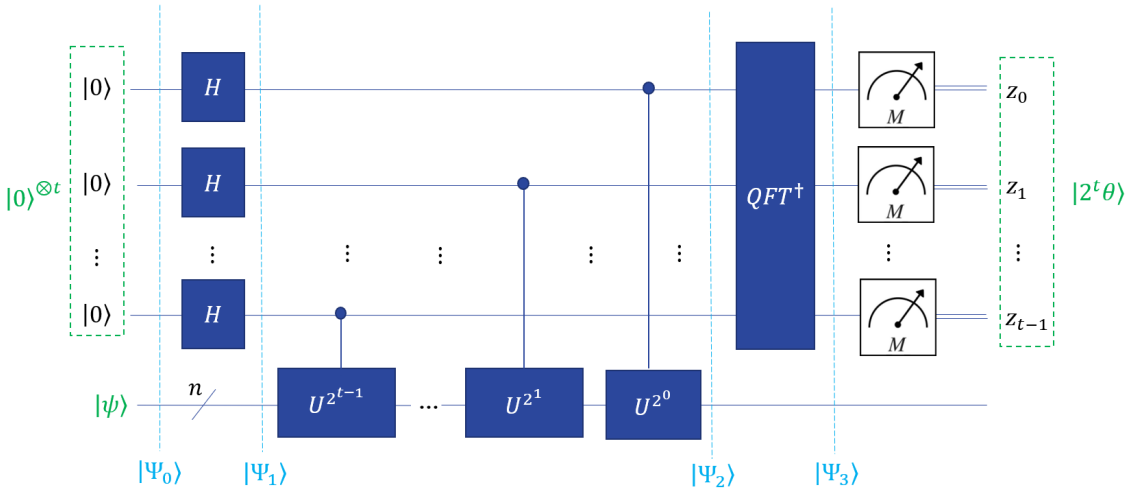

Figura 1.6: Circuito para implementar o QPE

para transformar o estado para a base computacional antes de efetivamente realizar a medição do contador. O valor medido com alta probabilidade ainda não será a estimativa, pois ele estará escalado ($2^t\theta$) pela precisão do *contador*, sendo representado por um dos estados da base computacional e assim associado com a respectiva cadeia de bits. Basta então que dividamos o resultado por 2^t para obtermos a estimativa desejada. Os detalhes de implementação da operação U podemos deixar de lado para nosso exemplo, tratando-a como uma caixa preta.

Vale notar que como $\theta \in [0,1)$ é um número real, nem sempre $2^t\theta$ será um número inteiro, resultando em aproximações que podem variar de acordo com a execução. A partir dos resultados, podemos aproximar nossa solução variando a precisão de acordo com o tamanho do contador. Para construir esse circuito, precisamos da porta H e das portas Controlled-U^{2^j} e QFT†. Na primeira parte temos uma camada de portas H, ou seja, são n portas no total. Já vimos que a QFT (mesma coisa pra QFT†) tem complexidade temporal $O(n^2)$. Se considerássemos cada operador Controlled-U^{2^j} como uma porta quântica parametrizável, teríamos $O(n)$. Porém, se considerarmos que Controlled-U^{2^j} é a repetição de Controlled-U por 2^j vezes, teremos um crescimento exponencial de acordo com o número de qubits do contador.

Para alguns casos poderemos construir esses operadores controlados de forma mais eficiente do que simplesmente por repetição (por exemplo com custo $O(n^2)$), apesar de que didaticamente é muito mais simples de entender o funcionamento do circuito dessa forma. Porém, para casos mais gerais, precisamos pagar o preço, criando um compromisso entre custo computacional e precisão da estimativa. Para um abordagem que reduz o número de qubits no contador através de um método iterativo, consultar o trabalho de [O'Loan 2009].

1.1.3. Algoritmo de Shor

A solução desenvolvida por Shor para resolver o problema da fatoração de números inteiros foi aplicar uma estimativa quântica de fase sobre um operador unitário que apresente o seguinte efeito:

$$U|y\rangle \equiv |ay \; mod \; N\rangle \tag{13}$$

Vamos supor que conseguimos construir esse operador em um circuito quântico e analisar as consequências dessa possibilidade. Inicialmente, vamos tentar visualizar o formato de um autoestado desse operador. Começamos com um registrador que possui o estado inicial $|y\rangle = |1\rangle$ (vamos usar a representação decimal dentro dos kets assumindo que temos o número necessário de qubits a nossa disposição).

Aplicamos o operador U sucessivamente, o que terá por resultado a multiplicação do estado desse registrador por *a mod N* sucessivas vezes. Vejamos essa ideia para o nosso exemplo ($a = 13$ e $N = 15$):

$$\begin{aligned}
U^1|1\rangle &= U|1\rangle = |13 \cdot 1 \; mod \; 15\rangle = |13\rangle \\
U^2|1\rangle &= U|13\rangle = |13 \cdot 13 \; mod \; 15\rangle = |4\rangle \\
U^3|1\rangle &= U|4\rangle = |13 \cdot 4 \; mod \; 15\rangle = |7\rangle \\
U^4|1\rangle &= U|7\rangle = |13 \cdot 7 \; mod \; 15\rangle = |1\rangle
\end{aligned} \tag{14}$$

Depois de aplicarmos $r = 4$ vezes o operador U, retornaremos ao estado $|1\rangle$, formando um ciclo. Percebemos que para o caso trivial, quando nenhuma aplicação do operador U é realizada (equivalentemente $r = 0$), o estado que obtemos é o próprio estado $|1\rangle$ enquanto para o caso não trivial, encontramos de fato o período conforme nosso exemplo nos mostrou. Se construirmos uma superposição uniforme dos estados presentes nesse ciclo, obteremos um autoestado do operador U:

$$|\psi_0\rangle = \frac{1}{\sqrt{4}}(|13\rangle + |4\rangle + |7\rangle + |1\rangle) \tag{15}$$

$$\begin{aligned}
U|\psi_0\rangle &= \frac{1}{\sqrt{4}}(U|13\rangle + U|4\rangle + U|7\rangle + U|1\rangle) \\
&= \frac{1}{\sqrt{4}}(|4\rangle + |7\rangle + |1\rangle + |13\rangle) \\
&= |\psi_0\rangle
\end{aligned} \tag{16}$$

Essa ideia nos leva a um autovalor não tão interessante, pois não apresenta o formato ideal que precisamos para aplicar o QPE. No entanto, Shor percebeu que para esse operador, podemos obter uma família de autoestados do seguinte formato:

$$U|u_s\rangle = e^{2\pi i s/r}|u_s\rangle \qquad 0 \leq s \leq r-1 \tag{17}$$

Quando utilizamos o QPE, precisamos de um registrador *ancilla* com um autoestado do operador em questão de forma a obter a estimativa pra fase do respectivo autovalor. Para operadores arbitrários, pode não ser simples construir tais estados, como neste caso onde, para construir esses autoestados, precisaríamos saber r a priori, que é

justamente o que queremos encontrar. Porém, o que Shor notou foi que ao construirmos um estado de superposição uniforme de todos os autoestados desse operador, obteríamos o estado da base computacional $|1\rangle$, que é um estado mais simples de se construir:

$$\frac{1}{\sqrt{r}} \sum_{s=0}^{r-1} |u_s\rangle = |1\rangle \tag{18}$$

Se aplicarmos o algoritmo QPE em U usando como autoestado o estado $|1\rangle$, ao invés de termos no estado do contador uma estimativa para uma fase apenas, teremos uma superposição uniforme de estimativas de fases para os diferentes valores de s. Após a medição, podemos obter uma dessas fases:

$$\theta_s = \frac{s}{r} \tag{19}$$

Com essa estimativa em mãos, podemos então aplicar um algoritmo como o de frações contínuas sobre θ_s para calcular r. O circuito quântico para o algoritmo de Shor é basicamente o circuito do QPE com o autoestado e operador U adequados para o problema, conforme Figura 1.7.

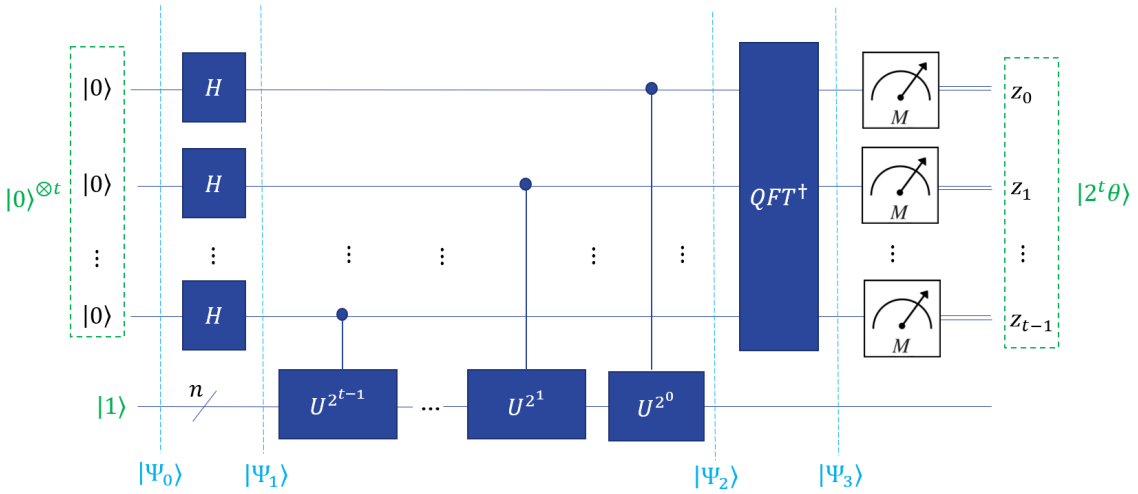

Figura 1.7: Circuito para implementar o algoritmo de Shor

Vejamos a execução desse circuito para o exemplo que temos tratado até aqui que é fatorar o número $N = 15$. Na primeira etapa, selecionamos $a = 13$ visto que $gdc(13,15) = 1$. Para representar o número 15 precisamos de 4 bits e consequentemente nosso registrador auxiliar precisará de 4 qubits, fazendo com que $n = 4$. Para o contador vamos utilizar $t = 2n = 8$ (geralmente, fazer o tamanho do contador ser o dobro do ancilla traz resultados satisfatórios [Beauregard 2003]). Nossa função será $f(x) = 13^x \ mod\ 15$ e o operador será de tal forma que $U|y\rangle = |13y \ mod\ 15\rangle$.

Há várias formas de implementar o operador U mas não vamos abordar esse ponto nesse material dada toda a complexidade adicional. Referências podem ser encontradas em [Beauregard 2003, Monz et al. 2016]. Vale notar que há formas didaticamente mais

simples, mas que detratam a performance do algoritmo, e outras mais complicadas porém mais eficientes. A parte mais custosa do circuito é justamente a sequência de aplicações controladas do operador U, então se conseguirmos construir os circuitos para essas operaçoes de maneira eficiente, as demais etapas também o serão e teremos implementado adequamente o circuito.

Ao executar o circuito por diversas vezes, obteremos uma distribuição de probabilidades em relação às estimativas que desejamos, conforme Figura 1.8. Esse histograma mostra as cadeias de bits correspondentes aos estados medidos e suas respectivas amostragens. A partir desses resultados, montamos a Tabela 1.2. Na tabela são apresentados as *strings* binárias medidas, sua representação decimal, o pós-processamento para obter a estimativa de θ_s para cada caso, as frações irredutíveis das estimativas, as frações irredutíveis teóricas (fins didáticos, na prática não saberiamos a priori) e o período r calculado.

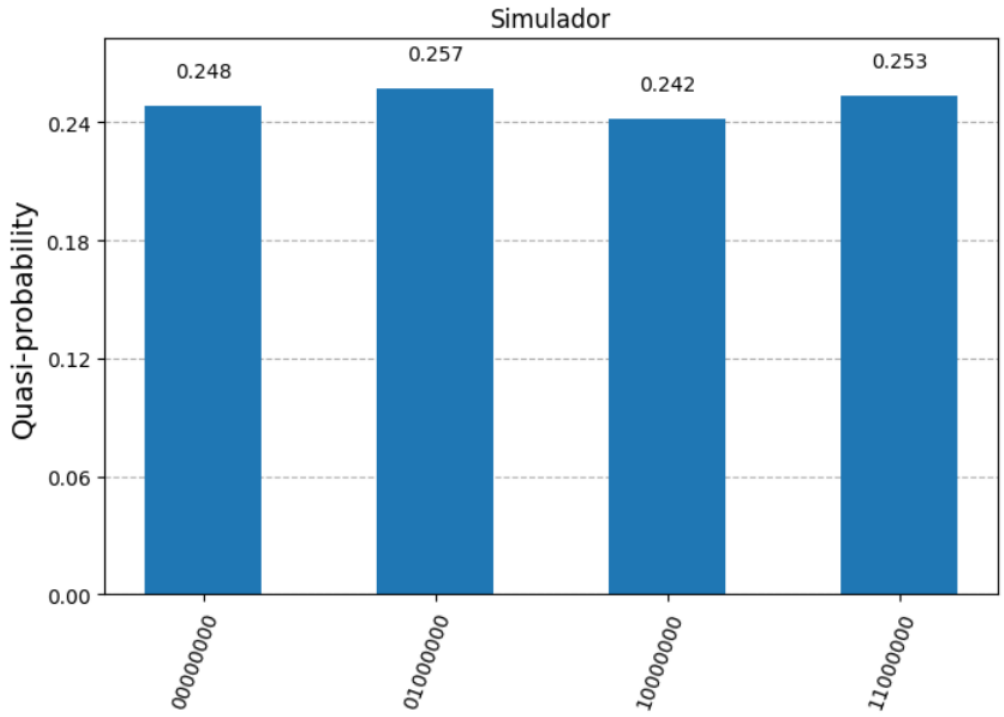

Figura 1.8: Histograma obtido após a execução do circuito

Tabela 1.2: Encontrando o período

Medição do Contador ($t = 8 \rightarrow 2^8 = 256$)					
Binário	Decimal	θ_s	s/r (calculado)	s/r (teórico)	r (calculado)
00000000	0	$0/256 = 0.00$	$0/1$	$0/4$	1
01000000	64	$64/256 = 0.25$	$1/4$	$1/4$	4
10000000	128	$128/256 = 0.50$	$1/2$	$2/4$	2
11000000	192	$192/256 = 0.75$	$3/4$	$3/4$	4

Supondo que realizamos a medição a partir da qual obtivemos o período correto $r = 4$ que é par, o prosseguimento é similar ao mostrado anteriormente:

$$p = (13^{4/2} + 1) \; mod \; 15 \equiv 5 \; (mod \; 15) \tag{20}$$

Verificamos que $p \neq 1$, então $q = N/p = 15/5 = 3$ e a saída do algoritmo será $\{p,q\} = \{3,5\}$. No caso de obtermos $r = 1$ teremos o caso trivial com fatores $\{1,N\}$ e precisaríamos repetir o processo novamente. No caso de $r = 2$ conseguiríamos encontrar um dos valores, bastando realizar a divisão final para obter o outro. Porém, vamos considerar na nossa análise apenas os resultados exatos para o período.

Selecionar um número a coprimo com um número N grande não é uma tarefa difícil. Como N já é produto de dois primos, se encontrarmos um número a que não seja coprimo com N, significa que ele é um fator não trivial de N e não precisamos executar o algoritmo de Shor, bastando calcular $\{p,q\} = \{gcd(a,N),\ N/gcd(a,N)\}$. Porém, a probabilidade disso acontecer é muito pequena.

A parte mais complexa e custosa do algoritmo de Shor é encontrar r, ou seja, o período da função $f(x) = a^x \; (mod \; N)$. Esta é a etapa a ser realizada em um computador quântico. As demais etapas podem ser realizadas diretamente em computadores clássicos (pois os algoritmos conhecidos para resolvê-las são eficientes), executando o algoritmo completo em sistemas híbridos. Vale relembrar que a principal ideia do algoritmo de Shor é converter o problema da fatoração em um problema de encontrar o período de uma função e a partir disso retornar ao problema da fatoração. O circuito quântico é utilizado justamente para encontrar o período r desejado, constituindo basicamente uma execução de um QPE.

No caso do algoritmo de Shor, temos uma complexidade polinomial. Para notar esse fato, basta lembrar que a parte mais custosa do algoritmo é a etapa que nos retorna o período da função periódica. Como cada qubit dos n qubits do contador (a partir daqui vamos usar n ao invés de t) controla um bloco de rotações, teremos n blocos que precisarão ser executados. Cada bloco equivale à repetição parametrizada do operador U e, se apenas usarmos repetição simples, acabamos com um crescimento exponencial no custo computacional dessa parte.

Porém, através de outros métodos como a potenciação por quadrados, conseguimos construir cada bloco ao custo de uma multiplicação. Se considerarmos o algoritmo Schönhage-Strassen [Shor 1971] para multiplicação de números inteiros, sua complexidade temporal é dada por $O(n \cdot log(n) \cdot loglog(n))$. Ao ser repetido n vezes o custo total é $O(n^2 \cdot log(n) \cdot loglog(n))$, uma complexidade polinomial. É possível considerar um algoritmo teoricamente mais eficiente ($O(n \cdot log(n))$ para multiplicação de inteiros chamado Harvey-Hoeven [Harvey and van der Hoeven 2021], porém pela diferença ser pequena, sua superioridade só se mostraria em tamanhos de problemas muito grandes. Como esse algoritmo é recente e seu benefício pode ocorrer apenas em escalas que podem estar além do interesse prático, é mais usual encontrar a primeira formulação para a complexidade temporal do algoritmo de Shor. Também é comum encontrar a complexidade temporal $O(n^3)$ considerando a multiplicação de inteiros pelo algoritmo que aprendemos na escola, também chamado como algoritmo de multiplicação do livro-texto ($O(n^2)$). Para aprofundamentos teóricos e detalhes adicionais sobre os algoritmos apresentados, consultar [Watrous 2025, Nielsen and Chuang 2010].

Referências

[Beauregard 2003] Beauregard, S. (2003). Circuit for shor's algorithm using 2n+3 qubits. *Quantum Info. Comput.*, 3(2):175–185.

[Coppersmith 2002] Coppersmith, D. (2002). An approximate fourier transform useful in quantum factoring.

[Harvey and van der Hoeven 2021] Harvey, D. and van der Hoeven, J. (2021). Integer multiplication in time O(n log n). *Annals of Mathematics*.

[Kitaev 1995] Kitaev, A. Y. (1995). Quantum measurements and the abelian stabilizer problem.

[Monz et al. 2016] Monz, T., Nigg, D., Martinez, E. A., Brandl, M. F., Schindler, P., Rines, R., Wang, S. X., Chuang, I. L., and Blatt, R. (2016). Realization of a scalable shor algorithm. *Science*, 351(6277):1068–1070.

[Nielsen and Chuang 2010] Nielsen, M. A. and Chuang, I. L. (2010). *Quantum computation and quantum information*. Cambridge university press.

[O'Loan 2009] O'Loan, C. J. (2009). Iterative phase estimation. *Journal of Physics A: Mathematical and Theoretical*, 43(1):015301.

[Shor 1994] Shor, P. (1994). Algorithms for quantum computation: discrete logarithms and factoring. In *Proceedings 35th Annual Symposium on Foundations of Computer Science*, pages 124–134.

[Shor 1971] Shor, P. W. (1971). Schnelle multiplikation großer zahlen. *Computing 7*, 26(5):281–292.

[Watrous 2025] Watrous, J. (2025). Understanding quantum information and computation.