

Approfondimenti sull'algoritmo di Shor

Antonio Allocca, mat.0522501527¹

¹Università degli studi di Salerno, Fisciano, Corso di Quantum Computing

June 19, 2024

Abstract

L'algoritmo di Shor rappresenta un pilastro fondamentale nel campo della computazione quantistica, promettendo di risolvere il problema della fattorizzazione degli interi in tempo polinomiale, sfidando così la sicurezza di molti protocolli crittografici basati su RSA. Questo paper propone un'analisi dettagliata dell'algoritmo di Shor, con un focus particolare sui suoi principi teorici, la sua implementazione pratica e le sue implicazioni nella crittografia moderna. Ne verranno analizzati gli aspetti matematici e verrà messa in evidenza la sua efficienza.

1 Introduzione

Tutta la teoria riportata in questo paper è stata scritta seguendo la notazione di Dirac [1] e la teoria presente sul libro *An Introduction to Quantum Computing* [2] e quella spiegata a lezione dal Prof. Fabrizio Illuminati [3] a cui va un ringraziamento per la chiarezza delle lezioni e delle risorse.

Negli ultimi anni l'interesse per la computazione quantistica ed i suoi benefici è sempre in aumento ed è giusto domandarsene i motivi. I progressi ottenuti in questo settore ci fanno ben sperare per quanto riguarda la risoluzione di problemi complessi grazie a questo nuovo approccio che sfrutta la meccanica quantistica. Tra gli algoritmi quantistici di maggiore rilevanza, l'algoritmo di **Shor** merita particolare attenzione per le sue potenziali applicazioni nella fattorizzazione di numeri interi, un problema di fondamentale importanza in crittografia e parleremo anche delle sue possibili implementazioni con il linguaggio *Python*. La crittografia classica è ciò che fino ad oggi ha protetto i nostri dati e la nostra privacy

sfruttando problemi computazionalmente troppo onerosi da risolvere in caso di *brute-force*. Ciò che rende questi problemi computazionalmente onerosi non è l'algoritmo in sé, quanto la taglia dell'input. Infatti con una chiave di cifratura di n bits, avremo 2^n possibili chiavi. Per esempio, ad oggi, chiavi di 128 bit (2^{128} possibili chiavi) sono considerati **out-of-reach**, ovvero un algoritmo di *brute-force* ci impiegherebbe troppo tempo per poter risolvere il problema. Alcuni schemi di cifratura asimmetrici a chiave pubblica come **RSA** ad oggi utilizzano chiavi di 2048 bit rendendo la rottura della chiave pressoché impossibile. **RSA** infatti utilizza uno dei problemi più computazionalmente onerosi al crescere del numero input, ovvero il problema della fattorizzazione. La chiave pubblica include un modulo N che è il prodotto di due grandi numeri primi p e q . Sebbene N sia noto pubblicamente, i fattori p e q sono mantenuti segreti. La sicurezza dell'algoritmo dipende dal fatto che, con le attuali risorse computazionali, non esiste un metodo efficiente per fattorizzare N quando è sufficientemente grande. Alcune stime dicono che l'algoritmo di **Shor** sia in grado di rompere **RSA** con una chiave di 2040 bit in sole 8 ore [4]. Tale risultato è sorprendente, ma mette anche timore al punto che è nata una nuova branca della crittografia che prende il nome di **post-quantum cryptography**. Prima di parlare dell'algoritmo ci sarà una parte introduttiva in cui si evidenziano le basi teoriche che servono per comprendere meglio l'algoritmo.

2 Quantum Phase Estimation and the Quantum Fourier Transform

Alcuni gate come quello di Hadamard sono *self-inverse* e possono essere utilizzati per codificare le informazioni. Ad esempio, ecco una generalizzazione dell'Hadamard gate che opera su $x, x \in \{0, 1\}$

$$H|x\rangle = \frac{1}{\sqrt{2}} \sum_{y \in \{0,1\}} (-1)^{xy} |y\rangle$$

Più in generale, un Hadamrd gate che opera su più qbit sarà della forma:

$$H^{\otimes n}|\mathbf{x}\rangle = \frac{1}{\sqrt{2}} \sum_{\mathbf{y} \in \{0,1\}^n} (-1)^{\mathbf{x}\mathbf{y}} |\mathbf{y}\rangle$$

Ovviamente, siccome è un gate *self-inverse*, come può essere utilizzato per codificare la fase, può essere utilizzato anche per **decodificare** la fase.

L'Hadamard però codifica le informazioni in una forma veramente particolare, ovvero della forma $(-1)^{\mathbf{x}\mathbf{y}}$, ma questo è riduttivo in quanto dovrebbe essere in grado di codificare tutte le fasi.

La **fase** infatti è un numero complesso della forma:

$$e^{2\pi i \omega}, \omega \in \{0, 1\}$$

A questo punto quando faremo riferimento a $|y\rangle$ faremo riferimento ad un intero $0 < y < 2^n - 1$ e ovviamente alla sua rappresentazione binaria.

Phase Estimation Problem

Input: The state $\frac{1}{\sqrt{2}} \sum_{y=0}^{2^n-1} e^{2\pi i \omega} |y\rangle$

Problem: Obtain a good estimate of the phase parameter ω

Esiste un algoritmo quantistico per stimare il parametro ω . Tale algoritmo funziona per quando

$$\frac{1}{\sqrt{2}} \sum_{y=0}^{2^n-1} e^{2\pi i \omega} |y\rangle$$

ha una fase della forma del tipo:

$$\omega = 0.x_1, x_2, \dots, x_n$$

Più in particolare, l'algoritmo restituisce un intero x quando $\omega = \frac{x}{2^n}$. Per un ω arbitrario, l'algoritmo di stima della fase restituirà x t.c. $\frac{x}{2^n}$ è il più vicino ad ω con un'alta probabilità. Uno dei parametri più importanti a questo punto sarà n che indica il numero di qubit (più ne sono, più la stima è precisa).

A questo punto se siamo in gradi di determinare per un arbitrario ω la fase allora possiamo fare diverse cose.

$$\frac{1}{\sqrt{2}} \sum_{y=0}^{2^n-1} e^{2\pi i \omega} |y\rangle = \frac{1}{\sqrt{2}} \sum_{y=0}^{2^n-1} e^{2\pi i \frac{x}{2^n} y} |y\rangle \longrightarrow |x\rangle$$

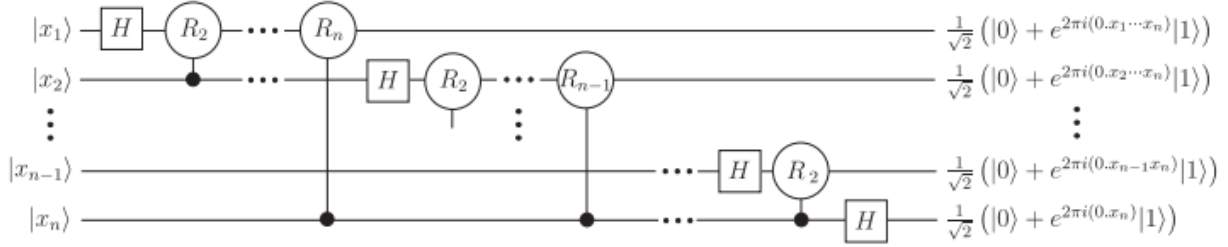
Ma a questo punto possiamo calcolarci anche l'inverso:

$$|x\rangle \longrightarrow \frac{1}{\sqrt{2}} \sum_{y=0}^{2^n-1} e^{2\pi i \frac{x}{2^n} y} |y\rangle$$

Quest'ultima equazione prende il nome di **Quantum Fourier Transform (QFT)** su n qubits. Può essere indicato anche come QFT_{2^n} . Esiste un circuito quantistico efficiente per determinare QFT

Generalizzando il problema, indichiamo con

$$QFT_m : |x\rangle \rightarrow \frac{1}{\sqrt{m}} \sum_{y=0}^{m-1} e^{2\pi i \frac{x}{m} y} |y\rangle$$

Fig. 1: Circuito quantistico efficiente per determinare QFT

La QFT definita sulle basi $|0\rangle, |1\rangle, \dots, |m\rangle$.

Possiamo anche definire l'inversa della QFT come QFT_m^{-1} che opera sulle basi $|0\rangle, |1\rangle, \dots, |m\rangle$ come:

$$QFT_m^{-1} : |x\rangle \rightarrow \frac{1}{\sqrt{m}} \sum_{y=0}^{m-1} e^{-2\pi i \frac{x}{m} y} |y\rangle$$

3 Eigenvalue Estimation

Sia U Un operatore unitario con apposito circuito quantistico che lo implementa, $|\psi\rangle$ un eigenvector e $e^{2\pi i \omega}$ il suo rispettivo eigenvalue. Possiamo rendere U un gate controllato $c-U$ e impostiamo come target l'eigenvector $|\psi\rangle$. Se il bit di controllo è in stato $|1\rangle$ applico U .

$$\begin{aligned} c-U|1\rangle|\psi\rangle &= |1U|\psi\rangle\rangle \\ &= |1\rangle e^{2\pi i \omega} |\psi\rangle \\ &= e^{2\pi i \omega} |1\rangle |\psi\rangle \end{aligned}$$

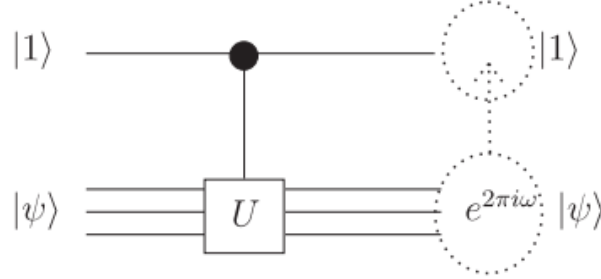
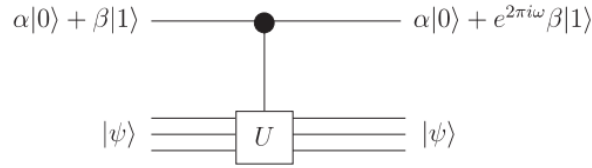
Circuito disegnato in 2.

Impostiamo il qubit di controllo di superposizione $\alpha|0\rangle + \beta|1\rangle$ e applichiamo $c-U$ per codificare l'eigenvalue di U nella relativa fase tra le basi $|0\rangle$ e $|1\rangle$ sul bit di controllo come illustrato in 3.

Eigenvalue Estimation Problem

Input: A quantum circuit implementing an operator U , and an eigenstate $|\psi\rangle$ with corresponding eigenvalue $e^{2\pi i \omega}$.

Problem: Obtain a good estimate for ω .


 Fig. 2: Circuito quantistico per $c - U$

 Fig. 3: Circuito quantistico per $c - U$ con codifica sul bit di controllo

Dato uno stato della forma $\frac{1}{\sqrt{2}} \sum_{y=0}^{2^n-1} e^{2\pi i \omega y} |y\rangle$ possiamo riscriverla come:

$$\left(\frac{|0\rangle + e^{2\pi i (2^{n-1} \omega)} |1\rangle}{\sqrt{2}} \right) \left(\frac{|0\rangle + e^{2\pi i (2^{n-2} \omega)} |1\rangle}{\sqrt{2}} \right) \cdots \left(\frac{|0\rangle + e^{2\pi i (\omega)} |1\rangle}{\sqrt{2}} \right)$$

possiamo approssimare il valore di ω attraverso QFT^{-1}

Se avessimo un circuito quantistico in grado di creare tale stato, possiamo utilizzare QFT^{-1} per stimare il rispettivo eigenvalue.

$|\psi\rangle$ è anche autovettore di U^2 con il corrispondente autovalore $(e^{2\pi i \omega})^2 = e^{2*2\pi i \omega}$. In generale possiamo dire che per un intero x , dal momento che $|\psi\rangle$ è autovettore di U^x allora l'autovalore sarà $e^{x*2\pi i \omega}$.

Se implementiamo $c - U^{2^j}$ e impostiamo il qubit di controllo a $\frac{|0\rangle + |1\rangle}{\sqrt{2}}$ e il qubit di target lasciamo $|\psi\rangle$ allora il risultato sarà:

$$c - U^{2^j} \left(\left(\frac{|0\rangle + |1\rangle}{\sqrt{2}} \right) |\psi\rangle \right) = \left(\frac{|0\rangle + e^{2\pi i (2^j \omega)} |1\rangle}{\sqrt{2}} \right) |\psi\rangle$$

Lo stato così ottenuto lo possiamo visualizzare come circuito in 4.

A questo punto applichiamo QFT^{-1} allo stato così ottenuto ed otteniamo lo stato $|\tilde{\omega}\rangle$ che è una buona approssimazione dell'autovalore ω . Il circuito che mostra come applichiamo QFT^{-1} allo stato precedentemente ottenuto è mostrato in 5.

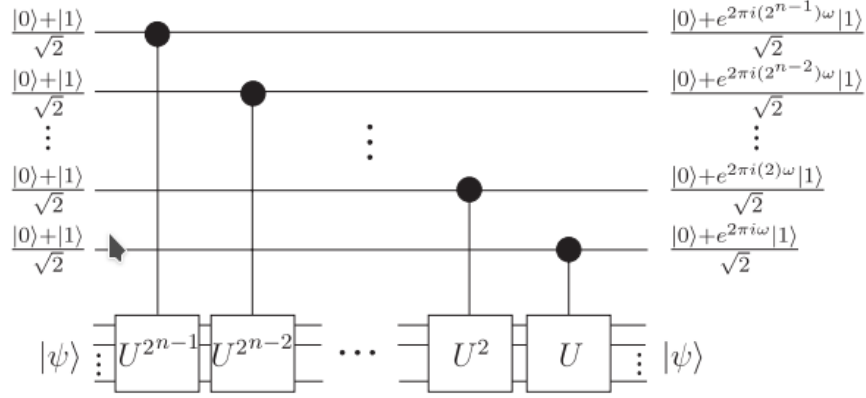


Fig. 4: Prima fase eigenvalue estimation

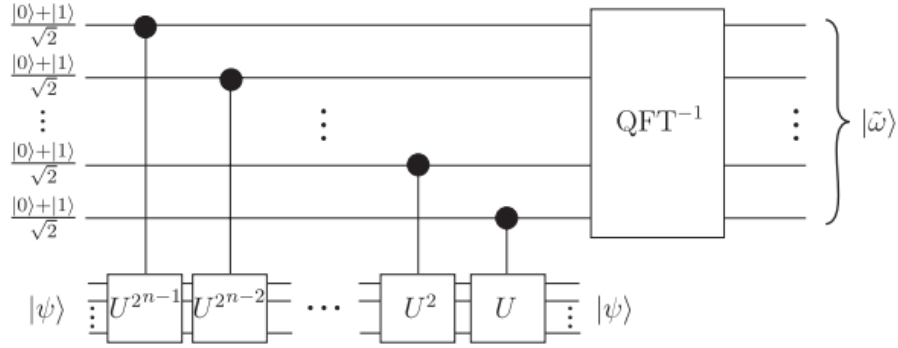


Fig. 5: Circuito eigenvalue estimation

I primi n qubit nel registro di controllo possono essere rappresentati con $|0\rangle^{\otimes n}$ in quanto basterà applicare l'Hadamard $H^{\otimes n}$ e trasormarli in

$\frac{|0\rangle + |1\rangle}{\sqrt{2}}$
Possiamo quindi riscrivere lo stato come:

$$H^{\otimes n} |0\rangle^{\otimes n} = \frac{1}{\sqrt{2^n}} \sum_{x=0}^{2^n-1} |x\rangle$$

Però abbiamo anche che:

$$QFT |0\rangle^{\otimes n} = H^{\otimes n} |0\rangle^{\otimes n}$$

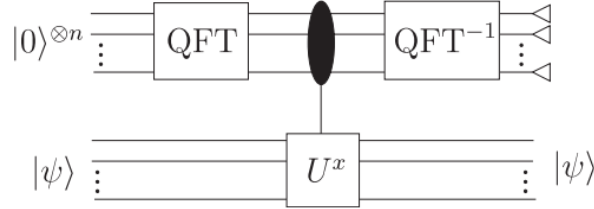


Fig. 6: Stima dell'autovalore $e^{2\pi i\omega}$ di $|\psi\rangle$ sull'operatore unitario U , versione compatta

Quindi l'intero circuito per la stima dell'eigenvalue può essere rappresentato della figura mostrata in 6.

4 Problema della fattorizzazione

L'algoritmo **RSA** trova la sua forza nell'utilizzo del problema della fattorizzazione di numeri semiprimi [5]. Molte aree della matematica hanno cercato di risolvere in modo efficiente questo problema utilizzando curve ellittiche, teoria dei numeri e appunto quantum computing.

Integer Factorization Problem

Input: An integer N .

Problem: Output positive integers $p_1, p_2, \dots, p_l, r_1, r_2, \dots, r_l$ where the p_i are distinct primes and $N = p_1^{r_1} p_2^{r_2} \dots p_l^{r_l}$

Assumendo che N sia dispari e non sia una composizione di una potenza di primi ma il prodotto di due primi distinti, il problema sarà non banale. Possiamo ridurre la taglia in input con $O(\log N)$ in quanto possiamo scartare tutti i numeri pari e le potenze di numeri primi per trovare le soluzioni ammissibili. Potremo provare a *splittare* N un numero logaritmico di volte e valutare in modo efficiente la primalità di un numero [6].

A questo punto non ci resta altro che valutare il problema dello splitting.

Splitting an Odd Non-Prime-Power Integer

Input: An odd integer N that has at least two distinct prime factors.

Problem: Output two integers $N_1, N_2, 1 < N_1 < N, 1 < N_2 < N$, such that $N = N_1 * N_2$.

Il problema dello splitting può essere ridotto al trovare l'ordine di un intero. Dato un intero a e N t.c. $\text{GCD}(a, N) = 1$, l'ordine di $a \pmod{N}$ è il più piccolo intero positivo r t.c. $a^r \equiv 1 \pmod{N}$. A questo punto non ci resta altro che definire in modo formale il problema:

4.1 Order-Finding Problem

Order-Finding Problem

Input: Integers a and N such that $GCD(a, N) = 1$ (i.e. a is relatively prime to N).

Problem: Find the order of a modulo N

Per trovare un intero a t.c. a è coprimo ad N dovremmo applicare EEA ad ogni intero compreso in $\{2, 3, \dots, N-2\}$. Se $GCD(a, N) = 1$ allora il rispettivo rank r è pari con una probabilità di $\frac{1}{2}$. Se r è pari allora:

$$b = a^{\frac{r}{2}} \mod N \rightarrow b^2 - 1 = 0 \mod N$$

Quindi questo porta all'importante risultato che N divide $(b+1)(b-1)$.

Se N ha almeno 2 fattori primi, allora per un intero a scelto uniformemente a caso con r pari, la probabilità che:

$$Pr[GCD(a^{\frac{r}{2}} - 1 \mod N, N) \in \text{Non-trivial factors of } N] \geq \frac{1}{2}$$

Il problema dell'*order-finding* può essere ridotto a trovare una frazione $\frac{x}{2^n}$ t.c. $\left| \frac{k}{r} - \frac{x}{2^n} \right| \leq \frac{1}{2r^2}$, o più formalmente:

Sampling Estimates to an Almost Uniformly Random Integer Multiple of $\frac{1}{r}$

Input: Integers a and N such that $GCD(a, N) = 1$. Let r denote the (*unknown*) order of a .

Problem: Output a number $x \in \{0, 1, 2, \dots, 2^{n-1}\}$ such that for each $k \in \{0, 1, \dots, r-1\}$ we have:

$$Pr\left(\left|\frac{x}{2^n} - \frac{k}{r}\right| \leq \frac{1}{2r^2}\right) \geq c \frac{1}{r}$$

È possibile definire una roadmap per le riduzioni in modo da capirle in modo grafico e più chiaramente in [7](#).

4.1.1 La stima degli autovalori per il problema dell'order findig

Sia U_a un operatore unitario e reversibile che associa:

$$U_a : |s\rangle \rightarrow |sa \mod N\rangle, 0 \leq s < N$$

Dove a è coprimo con N , quindi $\exists x | (ax \mod N) = 1$, U può essere implementato in modo efficiente e possiamo fare delle assunzioni del tipo:

$$U_a : |s\rangle \rightarrow |sa \mod N\rangle, 0 \leq s < N$$

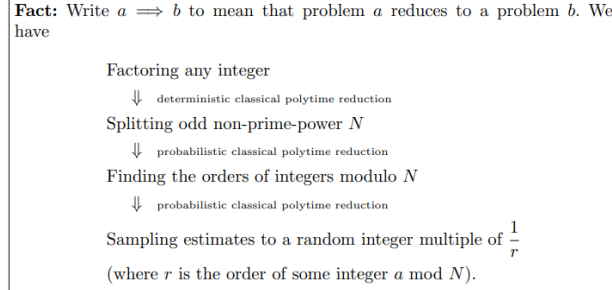


Fig. 7: Reduction roadmap

$$(a^r \bmod N) = 1$$

$$U_a^r : |s\rangle \rightarrow |sa^r \bmod N\rangle = |s\rangle$$

Sotto questo punto di vista U_a è la r -esima radice dell'operatore d'identità I .

$$U_a^r = I$$

$$U_a = \sqrt[r]{I}$$

Definito l'operatore U , consideriamo il seguente stato:

$$|u_k\rangle = \frac{1}{\sqrt{r}} \sum_{s=0}^{r-1} e^{-2\pi i \frac{k}{r} s} |a^s \bmod N\rangle$$

Abbiamo che

$$U_a |u_k\rangle = \frac{1}{\sqrt{r}} \sum_{s=0}^{r-1} e^{-2\pi i \frac{k}{r} s} |a^s \bmod N\rangle$$

Applicando U_a a $|a^s \bmod N\rangle$ avremo $|a * a^s \bmod N\rangle$, quindi $|a^{s+1} \bmod N\rangle$. Ne segue

$$= \frac{1}{\sqrt{r}} \sum_{s=0}^{r-1} e^{-2\pi i \frac{k}{r} s} |a^{s+1} \bmod N\rangle$$

Portando fuori $e^{2\pi i \frac{k}{r}}$ otteniamo

$$= e^{-2\pi i \frac{k}{r}} \frac{1}{\sqrt{r}} \sum_{s=0}^{r-1} e^{-2\pi i \frac{k}{r} (s+1)} |a^{s+1} \bmod N\rangle$$

$$\begin{aligned} \text{Ma siccome } e^{2\pi i \frac{k}{r} r} |a^r \bmod N\rangle &= e^{-2\pi i \frac{k}{r} 0} |a^0 \bmod N\rangle \\ &= e^{2\pi i \frac{k}{r}} |u_k\rangle \end{aligned}$$

Ci ritroveremo che l'autovalore per $|u_k\rangle$ nell'operatore U_a sarà proprio $e^{-2\pi i \frac{k}{r}}$.

Sia k un intero t.c. $0 \leq k \leq r-1$, misurando lo stato $|u_k\rangle$ possiamo applicare l'algoritmo di eigenvalue estimation per risolvere il sampling problem.

$$|0\rangle |u_k\rangle \rightarrow \left| \widetilde{k/r} \right\rangle |u_k\rangle$$

Per stimare il valore r Possiamo mettere tutti gli stati $|u_k\rangle$ con $k \in \{0, 1, \dots, r-1\}$ in superposizione. L'algoritmo per la stima degli eigenvalue produrrà una superposizione di stati in entanglement con la stima degli autovalori. Quello che andremo a misurare sarà una stima degli autovalori. Per fare ciò non è necessario conoscere r .

$$\frac{1}{\sqrt{r}} \sum_{k=0}^{r-1} |u_k\rangle = \frac{1}{\sqrt{r}} \sum_{k=0}^{r-1} \frac{1}{\sqrt{r}} \sum_{s=0}^{r-1} e^{-2\pi i \frac{k}{r} s} |a^s \bmod N\rangle$$

Se $(s \bmod r) = 0$ allora $|a^s \bmod N\rangle = |1\rangle$ Calcoliamo l'amplitude di $|1\rangle$ è la somma di tutti i termini con $s = 0$.

$$\frac{1}{\sqrt{r}} \frac{1}{\sqrt{r}} \sum_{k=0}^{r-1} e^{-2\pi i \frac{k}{r} 0} = \frac{1}{r} \sum_{k=0}^{r-1} (1) = 1$$

Quindi l'amplitude di tutti gli altri stati deve essere per forza 0. (DEve uscire per forza $|1\rangle$)

$$\frac{1}{\sqrt{r}} \sum_{k=0}^{r-1} |u_k\rangle = |1\rangle$$

Quindi l'algoritmo per la stima degli autovalori associa lo stato di **input**:

$$\begin{aligned} |0\rangle |1\rangle &= |0\rangle \left(\frac{1}{\sqrt{r}} \sum_{k=0}^{r-1} |u_k\rangle \right) \\ &= \frac{1}{\sqrt{r}} \sum_{k=0}^{r-1} |0\rangle |u_k\rangle \end{aligned}$$

Allo stato di **output**:

$$\frac{1}{\sqrt{r}} \sum_{k=0}^{r-1} \left| \widetilde{k/r} \right\rangle |u_k\rangle$$

La lettura del primo registro ora fornirà un intero x t.c. $\frac{x}{2^n}$ è una buona stima per $\frac{k}{r}$ per qualche k scelto uniformemente a caso.

L'intero algoritmo e il circuito può essere riassunto e visualizzato in 8.

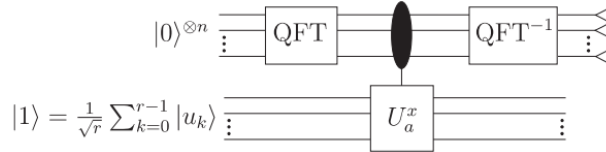
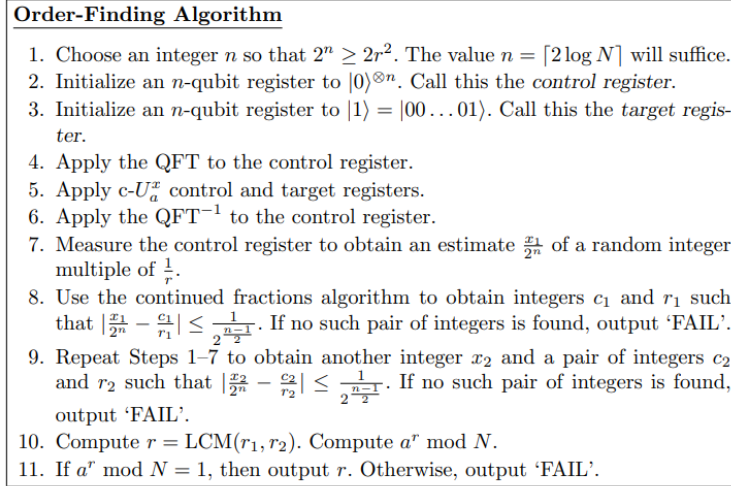


Fig. 8: Order-finding algoritmo e circuito

L'algoritmo di order-finding produrrà l'output corretto con una probabilità almeno $\frac{384}{\pi^6} > 0.399$, altrimenti produrrà un multiplo di r o fallirà.

Il rallentamento lo troviamo quando dobbiamo computare $c - U_a^{2^j}$ che richiede 2^j applicazioni dell'operatore $c - U_a$ e questo lo rende esponenziale. Per arginare questo problema dobbiamo notare che $c - U_a^{2^j} = c - U_{a^{2^j}}$, quindi moltiplicare $(a \bmod N)$ per 2^j volte è uguale a calcolare $a^{2^j} \bmod N$ una volta sola. Le tecniche aritmetiche di base come le potenze possono essere implementate con $O((\log n) \log \log(N) \log \log \log(N))$ gates, mentre la QFT richiede $O((\log n)^2)$ gates. Questa osservazione ci porta a poter utilizzare solamente $O((\log n)^2 \log \log(N) \log \log \log(N))$ gate logici per implementare il circuito. L'algoritmo euristico classico migliore ha una complessità di $e^{O((\log N)^{\frac{1}{3}} (\log \log N)^{\frac{2}{3}})}$, mentre il migliore a livello teorico ha una complessità di $e^{O((\log N)^{\frac{1}{2}} (\log \log N)^{\frac{1}{2}})}$. È possibile dunque stimare e confrontare i due approcci: classico e quantistico 9.

Complexities for Order Finding	
• Finding the order of a random element in Z_N^*	
- Quantum complexity is in $O((\log N)^2 \log \log(N) \log \log \log(N))$.	
- Best-known rigorous probabilistic classical algorithm has complexity in $e^{O(\sqrt{\log N} \log \log N)}$.	
- Best-known heuristic ⁶ probabilistic classical algorithm has complexity in $e^{O((\log N)^{\frac{1}{3}} (\log \log N)^{\frac{2}{3}})}$.	
• Order finding in a black-box group	
- Quantum black-box complexity (for groups with unique encodings of group elements) is $O(\log r)$ black-box multiplications and $O(n + \log^2 r)$ other elementary operations.	
- Classical black-box complexity ⁷ is in $\Theta(\sqrt{r})$ black-box multiplications.	

Fig. 9: Complessità e differenze Order-finding

4.1.2 Approccio di Shor per l'order-finding

Per comprendere l'approccio di Shor, bisogna capire che il problema dell'order-finding non è altro che un adattamento per la stima della fase.

In generale quindi i passaggi saranno simili e mappabili uno con l'altro.

Procederemo dunque a spiegare quelli che sono i 4 passaggi principali (4 fasi individuabili pure ad occhio nel circuito quantistico) che andranno a fornire la stima $\frac{x}{2^n}$ che ci porterà ad individuare il rango dell'intero a fornito in input.

1. Crea lo stato

$$\begin{aligned}
 |\psi_0\rangle &= \sum_{x=0}^{2^n-1} \frac{1}{\sqrt{2^n}} |x\rangle |a^x \bmod N\rangle \\
 &= \sum_{b=0}^{r-1} \left(\frac{1}{\sqrt{2^n}} \sum_{z=0}^{m_b-1} |zr + b\rangle \right) |a^b \bmod N\rangle
 \end{aligned}$$

$$m_b \in \mathbb{N} \text{ t.c. } (m_b - 1)r + b \leq 2^n - 1$$

2. A questo punto il secondo registri, come visto nel punto 1, sarà del tipo:

$$|a^b \bmod N\rangle$$

Mentre il primo registro (quello con i qubit di controllo) lo lasceremo in superposizione.

$$\sum_{z=0}^{m_b-1} \frac{1}{\sqrt{m_b}} |zr + b\rangle$$

Ma siccome $\frac{1}{\sqrt{m_b}}$ non varia nella sommatoria possiamo portarlo fuori:

$$\frac{1}{\sqrt{m_b}} \sum_{z=0}^{m_b-1} |zr + b\rangle$$

Ora possiamo applicare $QFT_{m_b r}^{-1}$ al primo registro e produrre la seguente superposizione:

$$\sum_{j=0}^{r-1} e^{-2\pi i \frac{b}{r} j} |m_b j\rangle$$

A questo punto $\frac{x}{rm_b} = \frac{j}{r}$ ma c'è solo un problema: r ed m_b sono sconosciuti ma proprio per questo usiamo $QFT_{m_b r}^{-1}$, per avere una buona approssimazione di tale valore, ovviamente più saranno i bit di precisione, più il risultato sarà accurato.

3. Applichiamo come detto prima $QFT_{m_b r}^{-1}$ al primo registro e otteniamo il valore x .
4. Restituisci $\frac{x}{2^n}$

Tale algoritmo restituisce un multiplo di $\frac{1}{r}$ con una buona probabilità. In particolare:

$$\forall j \in \{0, 1, \dots, r-1\}, \Pr \left[\left| \frac{x}{2^n} - \frac{j}{r} \right| \leq \frac{1}{2^{n+1}} \right] \geq \frac{4}{r\pi^2}$$

Lo stato nello step 1 possiamo impostarlo come:

$$|\psi_0\rangle = \sum_{x=0}^{2^n-1} \frac{1}{\sqrt{2^n}} |x\rangle |1\rangle$$

Da notare che ora la differenza principale tra il problema dell'order finding e l'implementazione di Shor risiede principalmente (oltre che alla scelta dei qubit in input), soprattutto nell'operatore che si usa. Nell'implementazione generica infatti abbiamo $c - U_a^x$ che non ha una vera e propria implementazione. Nell'algoritmo di Shor invece ci basterà che tale operatore $c - U_a^x$ esegua un semplice mapping:

$$c - U_a^x : |x\rangle |y\rangle \rightarrow |x\rangle |ya^x \bmod N\rangle$$

Questo mapping può essere ridefinito ulteriormente in quanto abbiamo bisogno solo dello stato:

$$\sum_x |x\rangle |a^x\rangle$$

E può essere implementato con un semplice circuito V_a definito come segue:

$$V_a : |x\rangle |y\rangle \rightarrow |x\rangle |y \oplus a^x\rangle$$

Le differenze tra i due approcci in definitiva può essere individuato in 10.

	Shor's Analysis	Eigenvalue Estimation Analysis
Initial state	$ 0\rangle 1\rangle$	$\sum_k 0\rangle u_k\rangle$
$\xrightarrow{\text{QFT}}$	$\sum_x x\rangle 1\rangle$	$\sum_k \sum_x x\rangle u_k\rangle$
$\xrightarrow{c-U_f^x}$	$\sum_b (\sum_z zr+b\rangle) a^b\rangle$	$\sum_k \left(\sum_x e^{2\pi i \frac{kx}{r}} x\rangle \right) u_k\rangle$
$\xrightarrow{\text{QFT}^{-1}}$	$\sum_b \left \bigwedge_{r=1}^{\frac{1}{r}} \bigwedge_{r=1}^{\frac{1}{r}} \dots \bigwedge_{r=1}^{\frac{1}{r}} \right\rangle a^b\rangle$	$\sum_s \left \text{---} \bigwedge_{r=1}^{\frac{1}{r}} \text{---} \right\rangle u_k\rangle$
$\xrightarrow{\text{Tr}_2}$	$\sum_b \left \bigwedge_{r=1}^{\frac{1}{r}} \bigwedge_{r=1}^{\frac{1}{r}} \dots \bigwedge_{r=1}^{\frac{1}{r}} \right\rangle \langle \bigwedge_{r=1}^{\frac{1}{r}} \bigwedge_{r=1}^{\frac{1}{r}} \dots \bigwedge_{r=1}^{\frac{1}{r}} $	$\sum_k \left \text{---} \bigwedge_{r=1}^{\frac{1}{r}} \text{---} \right\rangle \langle \text{---} \bigwedge_{r=1}^{\frac{1}{r}} \text{---} $

Fig. 10: Shor vs Order Finding con Eigenvalue Estimation

5 Implementazioni

Dato il problema, per capire effettivamente il vantaggio quantistico rispetto alla normale implementazione dobbiamo quantomeno analizzarla.

5.1 Implementazione classica

L'implementazione classica porta con sè varie soluzioni ed esplorarle tutte sarebbe pressoché impossibile, per questo faremo riferimento all'algoritmo più efficiente conosciuto fin ora il letteratura, ovvero General number field sieve [7]. Tale algoritmo è più efficiente conosciuto fin ora con input più grandi di 10^{100} La sua complessità è esponenziale infatti è stimata essere:

$$\exp\left(\left(\frac{64}{9}\right)^{\frac{1}{3}}(\log n)^{\frac{1}{3}}(\log \log n)^{\frac{2}{3}}\right)$$

La complessità di tale algoritmo è **super-polinomiale** ma è **sub-esponenziale** al crescere della taglia in input.

Ma questo non basta, infatti per rompere **RSA-768** sono stimati 200 anni di computazione [8], ovvero impraticabile.

```
import math
def get_factors_of(num):
    poss_p = math.floor(math.sqrt(num))
```

```
if poss_p % 2 == 0:
    poss_p += 1
while poss_p < num:
    if num % poss_p == 0:
        return poss_p
    poss_p += 2
```

Tale implementazione ovviamente è molto inefficiente e non dovrebbe essere utilizzata con chiavi robuste.

5.2 Implementazione quantistica

Per l'implementazione classica ad oggi sono stati sviluppati vari framework, in particolare possiamo trovare due modi principali per implementare della computazione quantistica:

1. via **Internet**
2. in **Locale**

I vantaggi di eseguire il codice quantistico è la possibilità di interagire con veri circuiti quantistici dove andrà eseguito il nostro programma, mentre in locale i circuiti quantistici verranno **simulati** su circuiti quantistici **perfetti** quindi senza errore. Anche se abbiamo la possibilità di eseguire il nostro algoritmo su un computer quantistico simulato, senza errori, perdiamo il vantaggio principale di sviluppare su un computer quantistico, ovvero lo speed-up ottenuto.

5.2.1 Implementazione online

Ci sono vari siti online che permettono di eseguire codice quantistico, ma uno di questi è di proprietà di O'Reilly [9].

Al sito <https://oreilly-qc.github.io/#> possiamo trovare una semplice interfaccia grafica che ci permette di fare molte cose interessanti come la scrittura di codice, la scelta del linguaggio di programmazione e anche alcuni algoritmi preimpostati per far provare l'applicativo.

Quando l'algoritmo che abbiamo selezionato verrà eseguito, abbiamo tre cose utilissime stampate a schermo:

1. Il circuito quantistico stampato

2. Lo stato di ogni qubit per ogni fase
3. L'output del programma

La schermata iniziale è visualizzabile in [11](#).

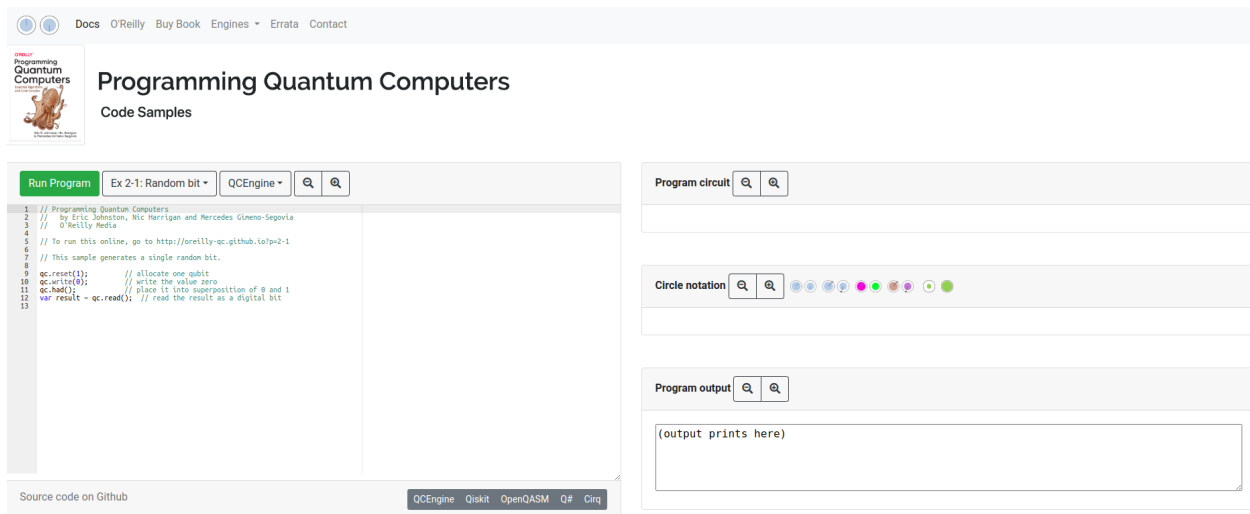


Fig. 11: Schermata iniziale

A questo punto possiamo provare ad eseguire l'algoritmo di Shor su alcuni input. Quindi selezioniamo l'algoritmo di Shor ed impostiamo i nostri parametri. Di seguito una dimostrazione step-by-step.

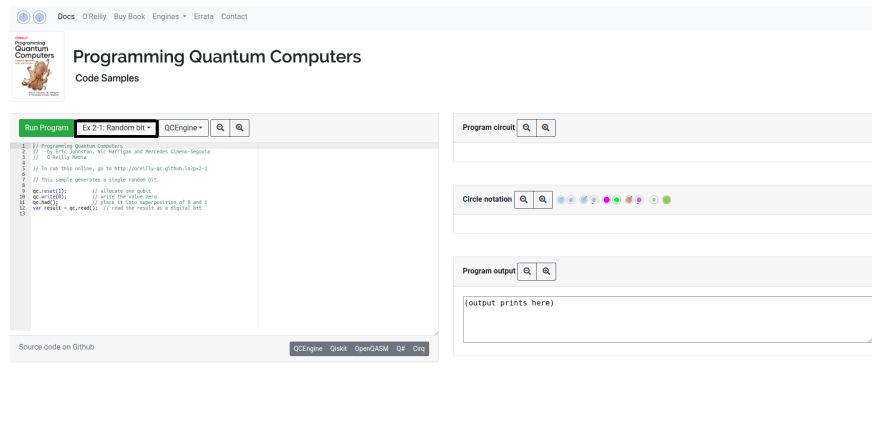


Fig. 12: Selezione Algoritmo di esempio

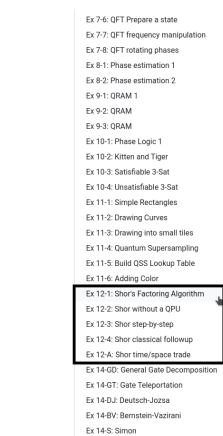


Fig. 13: Abbiamo più possibilità, scegliamo la prima come esempio

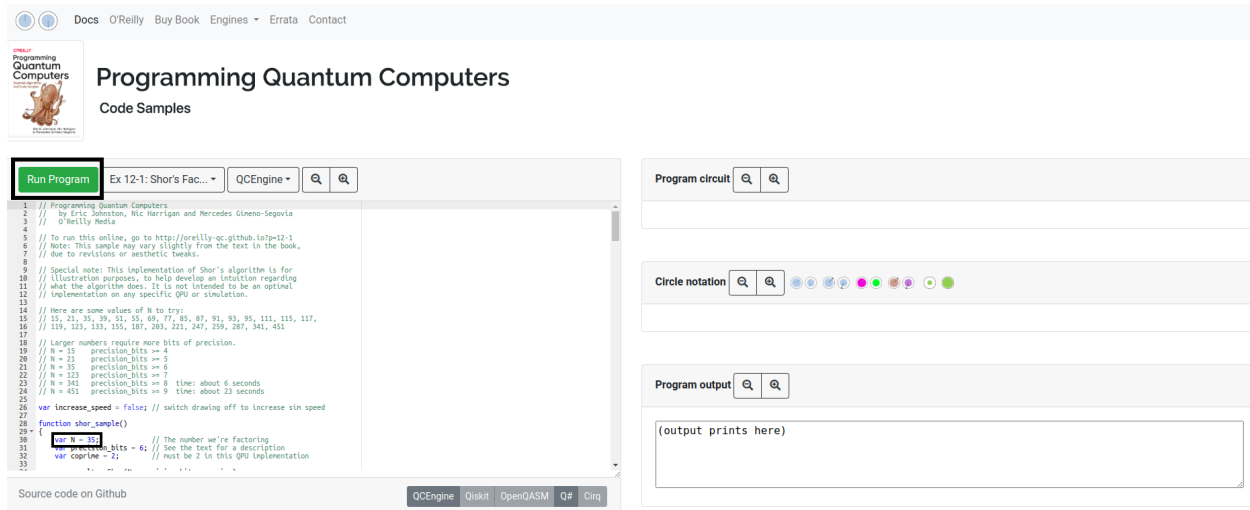


Fig. 14: Selezioniamo la N che vogliamo (Ad esempio 35) ed eseguiamo

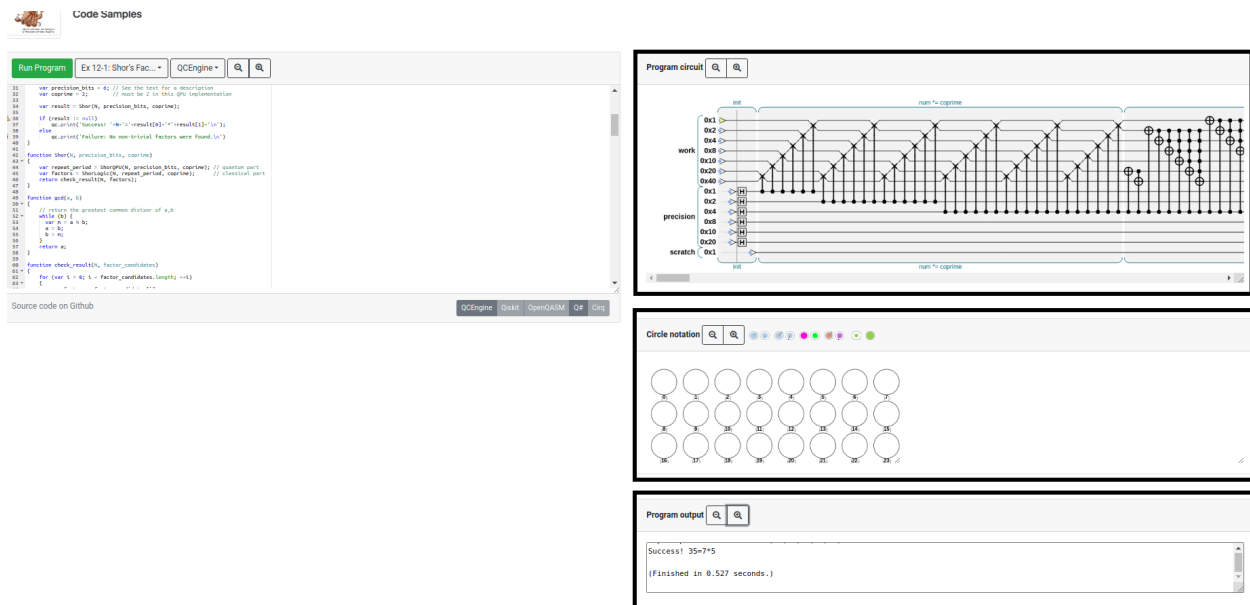


Fig. 15: Otteniamo il circuito quantistico (Nell'immagine ne è riportata solo una piccola parte), i qubit e il risultato in basso a destra.

A questo punto non ci resta altro che analizzare l'output dell'algoritmo. Bisogna precisare che comunque questo algoritmo è stato eseguito in locale, quindi su un computer quantistico simulato e con assenza di errori.

```
(output prints here)
QPU read result: 54
Repeat period candidates: 6,13,19,32
Failure: No non-trivial factors were found.
```

```
(Finished in 0.729 seconds.)
QPU read result: 44
Repeat period candidates: 3,6,10,13,16,32
Failure: No non-trivial factors were found.
```

```
(Finished in 0.567 seconds.)
QPU read result: 9
Repeat period candidates: 7,14,21,28,36,43,50
Success! 35=7*5
```

```
(Finished in 0.527 seconds.)
```

Come possiamo vedere, le prime due esecuzioni non hanno portato a buon fine in quanto né 7 e né 5 erano presenti in *Repeat period candidates*.

```
QPU read result: 54
Repeat period candidates: 6,13,19,32
Failure: No non-trivial factors were found.
```

```
(Finished in 0.729 seconds.)
QPU read result: 44
Repeat period candidates: 3,6,10,13,16,32
Failure: No non-trivial factors were found.
```

Non appena uno dei due fattori è stato candidato, il risultato è stato trovato e i fattori sono stati scritti come output.

```
(Finished in 0.567 seconds.)
QPU read result: 9
Repeat period candidates: 7,14,21,28,36,43,50
Success! 35=7*5
```

(Finished in 0.527 seconds.)

Ovviamente con numeri così piccoli non si riesce ad apprezzare a pieno lo speedup che si ottiene però comunque visualizzare praticamente queste cose rende più chiara la comprensione grazie alla sua rappresentazione sia grafica che visuale.

5.2.2 Implementazione locale

L'implementazione locale ad esempio può essere fatta con il framework Qiskit, famosissimo e pietra miliare in questo ambito per la sua semplicità d'uso e la velocità con la quale si può sviluppare.

Qiskit (Quantum Information Science Kit) è un framework open-source per lo sviluppo di software di calcolo quantistico creato da IBM. È progettato per facilitare lo sviluppo e l'esecuzione di algoritmi quantistici su vari tipi di hardware quantistico.

IBM fornisce la possibilità di implementare gli algoritmi quantistici sui loro computer al costo però di una chiave che non viene fornita a tutti, quindi per me al momento è impossibile recuperarla.

Qiskit si compone di alcuni componenti come:

1. Quskit Terra, per eseguire gli algoritmi su circuiti quantistici **simulati**
2. Quskit Aer, per eseguire gli algoritmi su circuiti quantistici **reali**
3. Quskit Ignis, per la gestione degli errori
4. Quskit Aqua

È possibile installarlo con il comando: *pip install qiskit*.

La documentazione è ampia ma soprattutto le community oggi possono fornire supporto in quanto ancora oggi è in continuo sviluppo. È possibile implementare Shor in modo molto semplice, basta importare la libreria che lo contiene ed eseguirlo. La documentazione ufficiale è fornita al seguente link: <https://docs.quantum.ibm.com/api/qiskit/0.29/qiskit.algorithms.Shor>.

Un esempio in codice Python può essere il seguente [10]:

```
from qiskit import IBMQ
from qiskit.utils import QuantumInstance
from qiskit.algorithms import Shor
```

```
IBMQ.enable_account('ENTER-API-KEY-HERE') # Enter your API token here
provider = IBMQ.get_provider(hub='ibm-q')

backend = provider.get_backend('ibmq_qasm_simulator')
# Specifies the quantum device

print('\nShors-Algorithm')
print('_____')
print('\nExecuting...\n')

factors = Shor(QuantumInstance(backend, shots=100,
skip_qobj_validation=False))

result_dict = factors.factor(N=21, a=2)
# Where N is the integer to be factored
result = result_dict.factors

print(result)
print('\nPress any-key-to-close')
input()
```

Come possiamo vedere è molto snello come codice, chiaro e pulito.

In `IBMQ.enable_account('ENTER API KEY HERE')` *# Enter your API token here* possiamo selezionare un account per utilizzare un vero computer quantistico che esegue il nostro algoritmo, ma come precedentemente detto non mi è possibile farlo.

6 Crittografia Post-Quantistica

La crittografia post-quantistica è un campo della crittografia che sviluppa algoritmi progettati per essere sicuri contro attacchi da parte di computer quantistici. Questi algoritmi mirano a sostituire o integrare gli attuali algoritmi di crittografia che saranno vulnerabili con l'avvento di computer quantistici avanzati. Ecco una panoramica dettagliata:

6.1 Minacce del Quantum Computing alla Crittografia

Gli algoritmi quantistici pericolosi, ovvero quelli che minacciano la sicurezza della crittografia attuale, sfruttano le potenzialità dei computer quantistici per risolvere problemi matematici

complessi molto più velocemente rispetto ai computer tradizionali. Due degli algoritmi quantistici più noti in questo contesto sono l'algoritmo di Shor e l'algoritmo di Grover.

Algoritmo di Shor: Gli attuali sistemi di crittografia, come RSA, si basano sulla difficoltà di fattorizzare numeri grandi. Un numero enorme può essere creato facilmente moltiplicando due numeri primi, ma trovare questi numeri primi a partire dal prodotto è estremamente difficile per i computer classici, richiedendo un tempo computazionale molto lungo. L'algoritmo di Shor, però, sfrutta le proprietà quantistiche per fattorizzare questi numeri in modo efficiente, rendendo possibile rompere la crittografia RSA in tempi praticabili.

Algoritmo di Grover: Gli algoritmi di crittografia simmetrica, come AES, si basano sulla sicurezza delle chiavi segrete.

Decifrare un messaggio senza conoscere la chiave richiederebbe provare tutte le possibili combinazioni di chiavi, un compito che diventa impraticabile con chiavi sufficientemente lunghe. L'algoritmo di Grover può, però, ridurre significativamente il numero di tentativi necessari per trovare la chiave corretta, rendendo possibili attacchi efficaci contro crittografie simmetriche con chiavi più corte. Ad esempio, se un sistema utilizza una chiave di 128 bit, un computer quantistico che utilizza l'algoritmo di Grover potrebbe ridurre il problema di trovare la chiave a un livello equivalente a una ricerca esaustiva su una chiave di 64 bit, il che non è sufficientemente sicuro.

6.2 Obiettivi della Crittografia Post-Quantistica

Sicurezza a Lungo Termine: Uno degli obiettivi primari della crittografia post-quantistica è sviluppare algoritmi che non possano essere compromessi dai computer quantistici. Questi nuovi algoritmi devono essere in grado di resistere agli attacchi che sfruttano la potenza di calcolo quantistica, proteggendo così i dati sensibili nel lungo termine.

Efficienza: L'idea è garantire che possano essere implementati in modo pratico sui sistemi attuali senza necessitare di hardware specializzato e costoso.

Adattabilità ai Sistemi Esistenti: Ciò significa che i nuovi algoritmi devono poter sostituire o affiancare quelli attuali senza richiedere modifiche sostanziali alle infrastrutture di rete, ai protocolli di comunicazione e agli applicativi esistenti. Questa compatibilità riduce i costi e la complessità della transizione verso una sicurezza post-quantistica.

6.3 Classi di Algoritmi Post-Quantistici

Codici a Correzione di Errore

- **Codici di McEliece:** Utilizzano la difficoltà di decodificare messaggi criptati con errori aggiunti.

- Caratteristiche: Hanno chiavi pubbliche di grandi dimensioni ma sono molto veloci nella decodifica.

Crittografia Basata su Reticoli

- Problemi: LWE (Learning with Errors), SVP (Shortest Vector Problem).
- Algoritmi: NTRU, Kyber.
- Caratteristiche: Buona efficienza, sicurezza basata su problemi difficili da risolvere sia per computer classici che quantistici.

Crittografia Basata su Hash

- Merkle Trees: Utilizzano funzioni hash per creare firme digitali sicure.
- Caratteristiche: Sicurezza elevata, chiavi e firme di dimensioni considerevoli.

Crittografia Basata su Isogenie

- Principio: Usa isogenie tra curve ellittiche supersingolari.
- Algoritmi: SIKE (Supersingular Isogeny Key Encapsulation).
- Caratteristiche: Chiavi di piccole dimensioni ma processi computazionali più intensivi.

6.4 Standardizzazione e Adozione

NIST PQC Standardization Project Processo: NIST (National Institute of Standards and Technology) ha avviato un concorso per standardizzare algoritmi di crittografia post-quantistica.

Fasi: Il processo ha attraversato più fasi, riducendo il numero di candidati attraverso rigorose valutazioni di sicurezza ed efficienza.

Finalisti: Alcuni degli algoritmi finalisti includono CRYSTALS-Kyber, CRYSTALS-Dilithium, e NTRUEncrypt.

Conclusioni

La computazione quantistica in questi ultimi tempi è fonte di curiosità ed innovazioni. Le innovazioni in questo ambito possono essere talvolta positive per l'umanità, talvolta possono portare a preoccupazioni. Un esempio lampante di questi timori è proprio la crittografia post-quantistica narrata in alcuni libri come *Crypto* di *Dan Brown* in cui le persone dovevano affrontare alcune tematiche morali e filosofiche timorati dal fatto che niente era più segreto, tutto era decifrabile efficientemente. Approfondire questi temi è quindi fondamentale per le società del futuro che inevitabilmente dovranno affrontare questi temi. In generale, l'approccio di Shor per trovare l'ordine di un intero è stato dimostrato essere efficiente e i circuiti quantistici possono essere implementati velocemente grazie ai framework a nostra disposizione. Questo suggerisce il fatto che in futuro verranno ideati nuovi algoritmi per migliorare lo stile di vita delle persone in vari ambiti come quello medico o lo studio dei materiali.

Acknowledgments

References

- [1] P. A. M. Dirac, *The Principles of Quantum Mechanics*. International series of monographs on physics, Clarendon Press, 1981.
- [2] M. M. Phillip Kaye, Raymond Laflamme, *An Introduction to Quantum Computing*. Oxford University Press Inc., New York, 2007.
- [3] P. F. Illuminati., "Appunti e lezioni durante il corso di quantum computing," 2024.
- [4] "Breaking shor time estimation mit." <https://www.technologyreview.com/2019/05/30/65724/how-a-quantum-computer-could-break-2048-bit-rsa-encryption-in-8-hours/>. Accessed: 2024-05-14.
- [5] Wikipedia, "Numero semiprimo — wikipedia, l'enciclopedia libera," 2024. [Online; in data 14-maggio-2024].
- [6] Wikipedia contributors, "Primality test — Wikipedia, the free encyclopedia," 2024. [Online; accessed 14-May-2024].
- [7] Wikipedia contributors, "General number field sieve — Wikipedia, the free encyclopedia," 2024. [Online; accessed 14-May-2024].

- [8] “How long does the general number field sieve actually take?” <https://math.stackexchange.com/questions/1640930/how-long-does-the-general-number-field-sieve-actually-take>. Accessed: 2024-05-14.
- [9] “Oreilly- programming quantum computers.” <https://oreilly-qc.github.io/#>.
- [10] “Shor’s algorithm with code.” <https://quantumcomputinguk.org/tutorials/shors-algorithm-with-code>.