

ALMA MATER STUDIORUM · UNIVERSITÀ DI BOLOGNA

SCUOLA DI SCIENZE

Corso di Laurea in Ingegneria e Scienze Informatiche

Implementazione in $Q\#$
dell'algoritmo di Shor per il
problema dei logaritmi discreti.

Relatore:
Chiar.mo Prof.
Moreno Marzolla

Presentata da:
Filippo Orazi

II Sessione di laurea
Anno Accademico: 2018/2019

Introduzione

Con computazione quantistica si intende l'utilizzo di determinati fenomeni fisici quantistici per l'esecuzione di calcoli e computazioni. L'approccio che viene utilizzato è completamente diverso da quello della computazione classica: mentre alla base di quest'ultima ci sono i bit, che possiedono solo due stati $0, 1$, alla base della computazione quantistica ci sono i Qubit, che possono presentarsi sia negli stati $0, 1$, sia in una sovrapposizione di questi due. Una spiegazione più approfondita verrà fornita nel capitolo 1.

Mentre l'hardware quantistico e lo sviluppo quantistico sono nati in tempi recenti (la prima esecuzione dimostrativa dell'algoritmo di Shor [12] per la fattorizzazione di numeri primi risale al 2001 da parte di IBM), la teoria riguardante questo campo nasce all'inizio degli anni ottanta con Paul Benioff [2] che definisce la macchina di Turing reversibile e Richard P. Feynman [8] che in una conferenza constatò la necessità di utilizzare per lo studio di fenomeni quantistici non un computer tradizionale che li simulava, ma un vero e proprio computer quantistico.

L'intero campo guadagnò notorietà solo nel 1994 quando Peter W. Shor pubblicò "Polynomial-Time Algorithms for Prime Factorization and Discrete Logarithms on a Quantum Computer" [12], un articolo dove venivano descritti due algoritmi che dimostravano come un problema classico di classe NP poteva essere risolto in tempo polinomiale se affrontato attraverso un approccio quantistico. L'importanza di questo articolo risiede nel fatto che entrambi gli algoritmi analizzati da Shor vengono utilizzati nella costruzione di chiavi crittografiche in virtù della loro intrinseca difficoltà di calcolo. Un approccio quantistico rende inutile una chiave di questo tipo minacciando tutti i sistemi di sicurezza basati su RSA.

In questa tesi verrà implementato, attraverso i linguaggi di programmazione Q# e Python, l'algoritmo che risolve il problema dei logaritmi discreti come esposto da Shor nel suo trattato.

I capitoli sono suddivisi in questo modo:

- Capitolo 1: Introduzione alla computazione quantistica e spiegazione di Qubit, circuiti e porte logiche
- Capitolo 2: Algoritmo di Shor, definizione problema, esposizione degli strumenti matematici (modulo dell'esponenziale e trasformazione quantistica di Fourier) e algoritmo di risoluzione.
- Capitolo 3: Introduzione al Q#, struttura e aspetti particolari del linguaggio.
- Capitolo 4: Implementazione e codice dell'algoritmo.

Indice

Introduzione	i
1 Fondamenti di computazione quantistica	1
1.1 Qubit	1
1.1.1 Rappresentazione Matematica	2
1.1.2 Rappresentazione Geometrica	2
1.1.3 Registri quantistici, misurazione ed entanglement	4
1.2 Porte logiche quantistiche e circuiti	6
1.2.1 Porte ad un Qubit	7
1.2.2 Porte a più Qubit	9
2 Il problema dei logaritmi discreti	13
2.1 Modulo dell'esponenziale	14
2.2 Trasformazione Quantistica di Fourier	16
2.3 Algoritmo di Shor	18
3 Il linguaggio Q#	22
3.1 Descrizione del linguaggio	23
3.1.1 Tipi primitivi	23
3.1.2 Funzioni e operazioni	24
3.2 Librerie	25
3.2.1 Standard libraries	26
3.2.2 Quantum numeric libraries	26

4	Implementazione dell'algoritmo di Shor in Q#	28
4.1	Shor	28
4.2	QuantumCopy	30
4.3	ModularExponentiation	31
4.4	ModularMultiplication	32
4.5	Risultati	33
	Conclusioni	34

Elenco delle figure

1.1	La sfera di Bloch, rappresentazione geometrica di un Qubit $ \psi\rangle$	4
1.2	Circuito quantistico d'esempio.	6
1.3	Qubit sulla sfera di Bloch quando viene utilizzata una porta	8
2.1	Controlled- M_x , porta fondamentale per il calcolo del modulo dell'esponenziale	15
2.2	Circuito di QFT applicata ad un registro di 4 bit.	17
2.3	Porta QTF schematizzata.	18

Capitolo 1

Fondamenti di computazione quantistica

In questo capitolo verranno esposti i principali elementi della computazione quantistica: Qubit, Porte logiche quantistiche e circuiti. Gli argomenti saranno trattati da un punto di vista matematico senza approfondire la natura fisica degli elementi ed in generale la fisica quantistica.

Il capitolo è pensato come una introduzione al concetto di computazione quantistica e risulta sufficiente per poter comprendere i capitoli successivi.

1.1 Qubit

I Qubit sono l'unità fondamentale della computazione quantistica come i bit lo sono per la computazione classica. Mentre il bit classico è un elemento che può trovarsi in due stati ben distinti $0, 1$, un Qubit è un elemento molto più complesso che possiede sia gli stati $0, 1$, sia le loro sovrapposizioni.

1.1.1 Rappresentazione Matematica

Per avere una descrizione migliore di un Qubit ne diamo una rappresentazione matematica attraverso la notazione di Dirac: un Qubit può essere visto come un vettore normalizzato in uno spazio di Hilbert \mathbb{C}^2 dove i vettori $|0\rangle$ e $|1\rangle$, corrispondenti a $(1,0)^T$ e $(0,1)^T$, formano una base ortogonale detta anche base computazionale standard. Un Vettore può essere visto come la combinazione lineare delle basi S di uno spazio a n dimensioni:

$$|\psi\rangle = \sum_{i=0}^n a_i |S_i\rangle$$

Nel caso dello spazio a due dimensioni \mathbb{C}^2 la formula precedente diventa semplicemente:

$$|\psi\rangle = \alpha |0\rangle + \beta |1\rangle$$

Il vettore $|\psi\rangle$ rappresenta un Qubit solo se le ampiezze α e β rispettano la condizione di normalizzazione: $|\alpha|^2 + |\beta|^2 = 1$ con $\alpha, \beta \in \mathbb{C}$, perché il quadrato del modulo dei coefficienti $|\alpha|^2$ e $|\beta|^2$ indica con quale probabilità il Qubit assumerà i valori $|0\rangle$ o $|1\rangle$ in seguito ad una misurazione.

1.1.2 Rappresentazione Geometrica

Un approccio geometrico alla visualizzazione di un Qubit consiste nell'associare $|\psi\rangle$ ad un punto sulla superficie della sfera di Bloch, una sfera di raggio unitario ai cui poli sono presenti gli stati base $|0\rangle$ e $|1\rangle$. Questa associazione renderà molto più chiaro l'effetto dei gate quantistici sui Qubit perché ne fornirà una simulazione visiva. Per poter procedere è necessario ricordare innanzitutto alcune proprietà dei numeri immaginari:

Un numero immaginario $z = a + ib$ può essere rappresentato attraverso coordinate cartesiane (a, b) nel piano formato dagli assi dei numeri reali e dei numeri immaginari. Da questa rappresentazione si può ricavare la rappresentazione polare considerando $r = \sqrt{a^2 + b^2}$ come modulo del vettore z nel piano sopracitato e ϕ come l'angolo

compreso tra z e l'asse dei numeri reali.

z è identificato da (r, ϕ) che possono sostituire (a, b) nella formula iniziale ottenendo:

$$z = r(\cos(\phi) + i \sin(\phi)) \quad (1.1)$$

che attraverso la legge di Eulero diventa:

$$z = r(\cos(\phi) + i \sin(\phi)) = r e^{i\phi} \quad (1.2)$$

Tornando ora al Qubit, questo può essere scritto come $|\psi\rangle = r_0 e^{i\phi_0} |0\rangle + r_1 e^{i\phi_1} |1\rangle$ utilizzando l'equazione (1.2) per trasformare a, b . È importante notare che rimane invariata la condizione $r_0^2 + r_1^2 = 1$, perché questa condizione ci consente di scrivere r_0 e r_1 come coordinate di un punto appartenente ad una circonferenza di raggio unitario: $r_0 = \cos(\rho)$ e $r_1 = \sin(\rho)$ dove $\rho = \theta/2$ rendendo entrambi dipendenti da un unico parametro θ e ottenendo:

$$|\psi\rangle = \cos(\theta/2) e^{i\phi_0} |0\rangle + \sin(\theta/2) e^{i\phi_1} |1\rangle \quad (1.3)$$

con $0 < \theta < \pi$. Raccogliendo il termine $e^{i\gamma}$, detto fase globale, otteniamo:

$$|\psi\rangle = e^{i\gamma} (\cos(\theta/2) e^{i\phi_0} |0\rangle + \sin(\theta/2) e^{i\phi_1} |1\rangle) \quad (1.4)$$

dove $\varphi = \phi_1 - \phi_0$ e $\gamma = \phi_0$, con $0 < \varphi < 2\pi$.

A livello fisico sappiamo che la fase globale $e^{i\gamma}$ non è rilevante, perché osservando due Qubit non si trovano differenze tra gli stati $|\psi\rangle$ e $e^{i\gamma} |\psi\rangle$, perciò si può tralasciare senza alterare il risultato. Per concludere l'associazione di partenza tra un Qubit $|\psi\rangle$ e un punto sulla sfera di Bloch notiamo che al parametro φ si può far corrispondere l'angolo della proiezione del punto sul piano x, y mentre al parametro θ si può far corrispondere l'angolo sferico del punto con l'asse z , la coppia di parametri (φ, θ) descrive quindi anche un punto su una sfera di raggio unitario. Risulta ora facile vedere che $|0\rangle$ e $|1\rangle$ corrispondono ai poli della sfera:

$$\theta = 0 \implies |\psi\rangle = |0\rangle$$

$$\theta = \pi \implies |\psi\rangle = |1\rangle$$

Come si può vedere nella Figura 1.1.

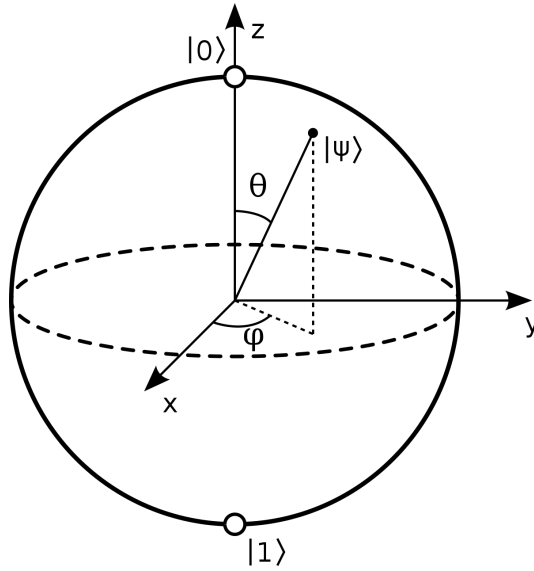


Figura 1.1: La sfera di Bloch, rappresentazione geometrica di un Qubit $|\psi\rangle$.

1.1.3 Registri quantistici, misurazione ed entanglement

Fin qui abbiamo focalizzato l'attenzione sul singolo Qubit, ora invece spiegheremo come i Qubit possono interagire tra loro e lavorare insieme.

Un registro quantistico è un insieme di Qubit che vengono utilizzati per svolgere un dato calcolo. La vera differenza tra questo e il suo equivalente classico sta nella potenza di calcolo di un gruppo di Qubit: mentre con un registro classico di n bit il numero di stati distinti che si può ottenere è pari a 2^n , con un registro quantistico di n Qubit viene generato uno spazio di Hilbert 2^n -dimensionale (\mathbb{C}^{2^n}), al cui interno ogni vettore normalizzato corrisponde ad uno stato possibile del sistema. In generale un registro quantistico può essere considerato come un prodotto tensore tra n vettori

$$\bigotimes_{j=1}^n |i_j\rangle$$

che per semplicità può essere scritto come $|i_1 i_2 \dots i_{n-1} i_n\rangle$.

Una parte fondamentale del processo di computazione quantistica è la fase di misurazione di un Qubit, perché l'atto di misurare riduce lo stato di sovrapposizione ad uno stato classico $(0, 1)$ e se questo viene fatto nel momento sbagliato rischia di rendere

nulli i passi precedenti e di influenzare quelli successivi.

Prendiamo ora come esempio un registro quantistico a 2 Qubit, questo può essere descritto dalla combinazione lineare dei vettori della base computazionale $|00\rangle$, $|01\rangle$, $|10\rangle$ e $|11\rangle$:

$$|\psi\rangle = a_{00}|00\rangle + a_{01}|01\rangle + a_{10}|10\rangle + a_{11}|11\rangle$$

con $|a_{00}|^2 + |a_{01}|^2 + |a_{10}|^2 + |a_{11}|^2 = 1$ ossia con $|\psi\rangle$ normalizzato. Come in precedenza i coefficienti $|a_{xy}|^2$ indicano la probabilità che a seguito di una misurazione il registro assuma il risultato $|xy\rangle$.

Quando viene effettuata una misurazione sul primo Qubit tutto il sistema collasserà modificando le probabilità e le possibili combinazioni in uscita. Se la misurazione registra il valore 0 (con probabilità $|a_{00}|^2 + |a_{01}|^2$) dallo stato:

$$|\psi\rangle = a_{00}|00\rangle + a_{01}|01\rangle + a_{10}|10\rangle + a_{11}|11\rangle$$

si passerà allo stato:

$$|\psi\rangle = \frac{a_{00}|00\rangle + a_{01}|01\rangle}{\sqrt{|a_{00}|^2 + |a_{01}|^2}}$$

perdendo le combinazioni $a_{10}|10\rangle$ e $a_{11}|11\rangle$ e dovendo normalizzare nuovamente i coefficienti a_{0x} .

In questo caso si può notare che il risultato della misurazione di uno dei Qubit del registro non influisce sulle probabilità dell'altro, ma ciò non è sempre vero: esistono stati particolari dei registri (ottenibili attraverso passaggi che non saranno analizzati in questa tesi) dove i Qubit al loro interno sono legati tra di loro. Questi stati sono detti entangled e l'esempio più famoso è la coppia EPR discussa da Albert Einstein, Boris Podolsky e Nathan Rosen [7] che attraverso un esperimento mentale ipotizzarono un registro di questa forma:

$$|\psi\rangle = \frac{|11\rangle + |00\rangle}{\sqrt{2}}$$

In questo esperimento i due Qubit vengono poi dati a due persone, Alice e Bob, che si allontanano l'uno dall'altro. Una volta percorsa una grande distanza Bob compie la misurazione sul suo Qubit, ma questa fa collassare tutto il registro definendo in

maniera certa il valore del Qubit di Alice. Questo fenomeno, che negli anni successivi è stato anche dimostrato sperimentalmente, è chiamato teletrasporto quantistico, la sua più grande applicazione al momento è la possibilità di trasportare attraverso bit classici lo stato di un Qubit, permettendo così di trasportare una quantità di informazione enorme attraverso pochi bit classici.

1.2 Porte logiche quantistiche e circuiti

Dopo aver analizzato il comportamento dei Qubit passiamo ora a descrivere come è possibile cambiare lo stato di questi attraverso porte logiche, come avveniva per i bit classici. Così come i Qubit sono equivalenti a dei vettori le porte possono essere considerate come trasformazioni o matrici bidimensionali che vengono applicate sui vettori. Le regole della meccanica quantistica indicano che le trasformazioni che sono permesse sui vettori devono essere unitarie (una matrice A è unitaria se il suo coniugato trasposto è uguale al suo inverso $A^\dagger A = I$), in questo modo viene sempre rispettata la regola secondo la quale la somma delle probabilità degli stati deve essere uguale ad 1:

$$\sum_{i=1}^n |a_i|^2 = 1$$

Un circuito logico quantistico è l'applicazione di più porte logiche quantistiche ad uno o più registri di Qubit ed ha una struttura come quella in figura 1.2: Ogni linea corrisponde al processo di un Qubit ed ogni rettangolo ad una porta che può essere a singolo Qubit o multi-Qubit.

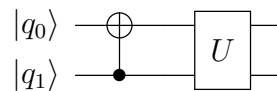


Figura 1.2: Circuito quantistico d'esempio.

1.2.1 Porte ad un Qubit

Le porte ad un Qubit sono quelle che, come il not nella computazione classica, hanno effetto su un solo Qubit modificandone lo stato o, visivamente, la posizione sulla sfera di Bloch. Di seguito vengono prima elencate e poi descritte più nel dettaglio:

- I, matrice identità.
- H, porta di Hadamard.
- X, not quantistico.
- Y, rotazione attorno all'asse y.
- Z, rotazione attorno all'asse z.

La porta di identità è esattamente quello che ci si aspetta, una matrice identità che non influisce sul Qubit.

$$\text{---}\boxed{I}\text{---} \quad I = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$$

La porta di Hadamard è la porta più utilizzata, che applica una rotazione di $\pi/2$ radianti attorno all'asse y seguita da una riflessione rispetto al piano (y, z) . È la porta che viene utilizzata per creare una sovrapposizione, infatti un vettore della base $|0\rangle$ o $|1\rangle$ viene ruotato in modo da essere ad ugual distanza dai poli.

$$\text{---}\boxed{H}\text{---} \quad H = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}$$

$$H|0\rangle = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 \\ 1 \end{pmatrix} = \frac{|0\rangle + |1\rangle}{\sqrt{2}} = |+\rangle$$

$$H|1\rangle = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 \\ -1 \end{pmatrix} = \frac{|0\rangle - |1\rangle}{\sqrt{2}} = |-\rangle$$

Gli stati ottenuti $|+\rangle$ e $|-\rangle$ sono stati particolari che possono essere utilizzati come base dello spazio vettoriale.

Le porte X, Y, Z sono dette anche matrici di Pauli e rappresentano le rotazioni del

punto intorno agli assi.

La porta X o σ_x viene definita not quantistico in quanto applica una rotazione di π radianti attorno all'asse x invertendo i coefficienti:

$$\text{---}\boxed{X}\text{---} \quad X = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}$$

$$X|0\rangle = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \begin{pmatrix} 1 \\ 0 \end{pmatrix} = \begin{pmatrix} 0 \\ 1 \end{pmatrix} = |1\rangle$$

$$X|1\rangle = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \begin{pmatrix} 0 \\ 1 \end{pmatrix} = \begin{pmatrix} 1 \\ 0 \end{pmatrix} = |0\rangle$$

La porta Y o σ_y è la porta che compie rotazioni di π intorno all'asse y :

$$\text{---}\boxed{Y}\text{---} \quad Y = \begin{bmatrix} 0 & -i \\ i & 0 \end{bmatrix}$$

La porta Z o σ_z è la porta che compie rotazioni attorno all'asse z , come per le porte X e Y anche Z compie rotazioni di π cambiando la fase del Qubit. Questo tipo di rotazione è particolarmente importante ed esistono altre porte che lo applicano con angoli diversi: le principali prendono il nome di S, T mentre una porta che compie una rotazione di un generico ϕ attorno a z è detta R_ϕ .

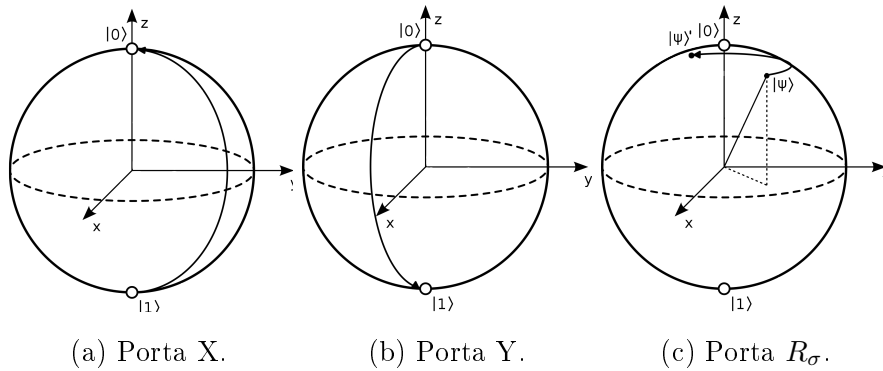


Figura 1.3: Qubit sulla sfera di Bloch quando viene utilizzata una porta

Qui è riportato uno schema riassuntivo delle porte che mostra il nome, l'asse e l'angolo di rotazione, definizione matriciale e simbolo.

Porta	Rotazione	Asse di rotazione	Matrice	Simbolo
I	0	-	$\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$	
H	$-\pi/2$	-	$\frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}$	
X	π	x	$\begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}$	
Y	π	y	$\begin{bmatrix} 0 & -i \\ i & 0 \end{bmatrix}$	
Z	π	z	$\begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix}$	
S	$\pi/2$	z	$\begin{bmatrix} 1 & 0 \\ 0 & i \end{bmatrix}$	
T	$\pi/4$	z	$\begin{bmatrix} 1 & 0 \\ 0 & e^{i\pi/4} \end{bmatrix}$	
R_ϕ	ϕ	z	$\begin{bmatrix} 1 & 0 \\ 0 & e^{i\phi} \end{bmatrix}$	

1.2.2 Porte a più Qubit

Oltre alle porte che agiscono su un solo Qubit ne esistono anche alcune che permettono di agire su un Qubit sulla base dello stato degli altri, ne verranno presentati tre: lo SWAP, il Controlled gate e porte di Toffoli.

Lo SWAP è il gate che permette di scambiare la posizione di due Qubit all'interno del circuito, viene rappresentato nei circuiti come in figura, mentre la sua

racpresentazione matematica è data da:

$$\begin{array}{c} \text{---} \times \text{---} \\ \text{---} \times \text{---} \end{array} \quad \text{SWAP} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Il controlled-U è una porta che come suggerisce il nome applica un controllo prima di essere eseguita. Come primo esempio prendiamo il CNOT o not condizionato: questo utilizza un Qubit di controllo e un Qubit target, la negazione del secondo deriva dallo stato del primo (tutto ciò che verrà mostrato considera come condizione che il Qubit di controllo sia uguale a $|1\rangle$ ma è possibile costruire anche porte che richiedono il controllo uguale a $|0\rangle$). La matrice di riferimento è la seguente:

$$\begin{array}{c} |controllo\rangle \text{---} \bullet \text{---} \\ |target\rangle \text{---} \oplus \text{---} \end{array} \quad \text{CNOT} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix}$$

Quando questa viene applicata agli stati $|00\rangle = (1 \ 0 \ 0 \ 0)^T$ e $|01\rangle = (0 \ 1 \ 0 \ 0)^T$ questi non cambiano perché il Qubit di controllo (il primo) è 0, mentre se il CNOT viene applicata agli stati $|10\rangle = (0 \ 0 \ 1 \ 0)^T$ e $|11\rangle = (0 \ 0 \ 0 \ 1)^T$ il secondo Qubit, ovvero il target, cambierà stato.

La generalizzazione del CNOT è il controlled-U, una porta generica che applicata una trasformazione qualunque U solo se il Qubit target soddisfa la condizione richiesta.

La matrice generica è la seguente:

$$\begin{array}{c} \text{---} \bullet \text{---} \\ \text{---} \boxed{U} \text{---} \end{array} \quad \text{controlled-U} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & U_{00} & U_{01} \\ 0 & 0 & U_{10} & U_{11} \end{bmatrix}$$

dove la trasformazione base U è:

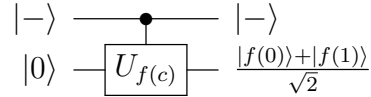
$$U = \begin{bmatrix} U_{00} & U_{01} \\ U_{10} & U_{11} \end{bmatrix}$$

La potenza di queste porte risiede nel fatto che non viene misurato e valutato il Qubit di controllo ma viene creata una sovrapposizione degli stati dei due: analizzando il controlled- U la trasformazione che applica su due Qubit ζ e ψ è:

$$U_{f(c)} |\zeta, \psi\rangle = |\zeta, \psi \oplus f(\zeta)\rangle$$

$f(c)$ viene applicata a tutti i possibili valori del Qubit controllo sfruttando l'effetto del parallelismo quantistico, una forma di parallelismo che a differenza di quello classico computa ogni possibilità contemporaneamente sullo stesso Qubit; ad esempio data la generica trasformazione $U_{f(c)}$ e due Qubit in input $|\zeta\rangle = |-\rangle$ e $|\psi\rangle = |0\rangle$ se applico $U_{f(c)} |\zeta\psi\rangle$ otterrò:

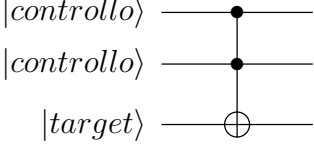
$$|\zeta\psi\rangle = \frac{|0, f(0)\rangle + |1, f(1)\rangle}{\sqrt{2}}$$



Questa porta logica avrà una grande importanza successivamente quando sarà necessario compiere le trasformazioni di Fourier quantistiche.

L'ultima porta multi-Qubit è il cosiddetto Toffoli gate, una porta che effettua un controllo su due Qubit anziché su uno e applica una trasformazione X (not) a seconda del risultato dei Qubit di controllo. Nota anche come CCNOT è una porta logica reversibile universale, cioè utilizzando unicamente porte di Toffoli è possibile costruire un equivalente di tutte le porte classiche.

Come per la porta CNOT anche il Toffoli gate crea uno stato di parallelismo quantistico che dipende da entrambi i Qubit di controllo.



$\text{Toffoli} =$

$$\begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \end{bmatrix}$$

Capitolo 2

Il problema dei logaritmi discreti

Il problema dei logaritmi discreti è da sempre considerato come intrattabile, è perciò alla base di diversi algoritmi di crittografia a chiave pubblica come l'algoritmo di Diffie-Hellman del 1976 [6] e tutti i suoi derivati: DSA, ElGamal Encryption, MQV Key Agreement, etc.. Non esistono algoritmi classici che risolvano in tempi soddisfacenti tutte le possibili forme (esistono casi particolari per cui alcuni algoritmi classici possono essere efficienti), ma nel 1996 Peter Shor propone un algoritmo di computazione quantistica che risolve il problema utilizzando semplicemente due trasformazioni quantistiche di Fourier e due moduli di esponenziali [12].

La definizione di logaritmo discreto è la seguente: dato un gruppo moltiplicativo G e un suo generatore g , per definizione ogni elemento $a \in G$ può essere scritto come $a = g^r$; in questa situazione r è definito come il logaritmo discreto di a nel gruppo G di g .

Il problema dei logaritmi discreti è definito come: dato un generatore g di un gruppo moltiplicativo ciclico $G \pmod{n}$ e un elemento $a \in G$ si deve trovare il logaritmo discreto dell'elemento a nel gruppo G di base g . Il problema, come detto in precedenza, presenta dei casi in cui può essere risolto in tempo polinomiale a seconda del gruppo moltiplicativo scelto: nella crittografia moderna vengono utilizzati gruppi Z_p^* dove p indica un numero primo sicuro da cui $G = \{g^k \pmod{n}\}$ con $0 < k < p - 2$. Un numero primo sicuro è un numero primo p tale che $p - 1$ non è il risultato del prodotto di numeri primi piccoli (nel qual caso si utilizza l'algoritmo di Pohlig-Hellman [10])

e perciò normalmente scelto come $p = 2u + 1$, dove u è un numero primo di grandi dimensioni, garantendo perciò la sicurezza di p , ma aumentandone anche la dimensione (normalmente almeno 1024 bit). In questo contesto si trova l'utilità e la potenza dell'algoritmo presentato da Shor capace in tempo polinomiale di trovare la chiave. Di seguito verranno esposti i principali strumenti matematici per la risoluzione del problema e verrà poi descritto l'algoritmo con i passaggi matematici.

2.1 Modulo dell'esponenziale

L'algoritmo che verrà implementato trova nell'operazione del modulo esponenziale il principale collo di bottiglia; questa parte di codice consuma una grande quantità di tempo e di spazio.

Il problema del modulo esponenziale è il seguente: dati x , n numeri classici e a numero contenuto in un registro di Qubit, si deve calcolare $x^a \pmod{n}$. Questo problema è classicamente risolvibile in maniera efficiente scomponendo a nella sua rappresentazione binaria e utilizzando le proprietà delle potenze per scomporre l'esponenziale in altri più piccoli più facilmente calcolabili. Chiamiamo quindi a_i la i -esima cifra binaria di a e definiamo Y_i come:

$$Y_i = \begin{cases} 1, & \text{se } a_i = 0 \\ x^{2^i} \pmod{n}, & \text{altrimenti} \end{cases}$$

In maniera classica quindi:

$$x^a \pmod{n} = \prod_{i=0}^{\log_2 a} Y_i \pmod{n}$$

Per poter derivare dall'algoritmo classico un algoritmo quantistico e successivamente costruire una porta quantistica capace di calcolare queste funzioni è necessario analizzare lo pseudo codice classico per individuare possibili criticità:

```

1:  $pow = 1$ 
2: for  $i = 0$  , lunghezza -1 do
3:   if  $a_i == 1$  then
4:      $pow = pow * x^{2^i} \pmod n$ 
5:   end if
6: end for

```

L'unica difficoltà che si può trovare risiede nella riga numero 4 dove un algoritmo quantistico avrebbe necessità di una porta che applichi la funzione reversibile

$$f(b) = bc \pmod n$$

(con c, n fissi). Per facilitare maggiormente l'operazione è possibile modificare ed espandere la riga in maniera analoga a quanto fatto in precedenza, trasformando la moltiplicazione in una serie di somme:

```

1:  $res = 0$ 
2: for  $i = 0$  , l -1 do
3:   if  $b_i == 1$  then
4:      $res = res + 2^i c \pmod n$ 
5:   end if
6: end for

```

La somma a riga 4 può essere trasformata in una porta reversibile detta *Modular adder gate* [1] che permette di eseguire l'operazione di somma modulare e di costruire la porta che permette la moltiplicazione $|x\rangle |0\rangle \rightarrow |ax \pmod n\rangle |0\rangle$, che definiamo come controlled- M_x (Figura 2.1).

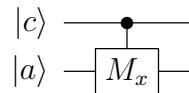


Figura 2.1: Controlled- M_x , porta fondamentale per il calcolo del modulo dell'esponenziale

A sua volta $C-M_x$ ci consente di calcolare $|x\rangle |0\rangle \rightarrow |x\rangle |x^a \pmod n\rangle$ come spiegato nello pseudo codice precedente.

Dall'analisi della complessità fatta in [1], riguardante il gate progettato, sappiamo che per effettuare l'operazione appena descritta su n Qubit necessitiamo di $2n + 3$ Qubit effettivi e $O(n^2 d_{max})$ porte logiche dove d_{max} è la soglia oltre la quale vengono ignorate le trasformazioni quantistiche di Fourier (approfondite nella prossima sezione) che vengono utilizzate per costruire $C-U_x$.

2.2 Trasformazione Quantistica di Fourier

La trasformazione quantistica di Fourier o QFT in breve, è una operazione lineare unitaria che viene applicata su un registro di n Qubit modificandone l'ampiezza secondo la formula:

$$\sum_j a_j |j\rangle = \sum_k \tilde{a}_k |k\rangle$$

dove

$$\tilde{a}_k = \frac{1}{\sqrt{N}} \sum_{j=0}^{N-1} e^{2\pi i j k / N} a_j$$

Per rendere più chiara la trasformazione ed adattarla alle esigenze del nostro calcolo la analizziamo punto per punto:

Dato un registro quantistico composto di stati puri a tale che $0 \leq a \leq q$ dove $q = 2^l$ con $l \in \mathbb{N}$ a questo vengono applicate le porte H ed una porta controlled-U, detta in questo caso $S_{j,k}$, dalla forma:

$$S_{j,k} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & e^{i\pi/2^{j-k}} \end{bmatrix}$$

Con k si intende l'indice del Qubit target mentre con j si intende l'indice del Qubit di controllo. Riconosciamo ora la porta $S_{j,k}$ come una controlled- R_ϕ con $\phi = \pi/2^{j-k}$. Le porte vengono applicate ai singoli Qubit del registro a (ricordiamo che un registro non

è altro che l'insieme di più Qubit e quindi può rappresentare un numero in sistema binario) in maniera ordinata partendo dal Qubit meno significativo (Little-endian) secondo lo schema d'esempio in Figura 2.2.

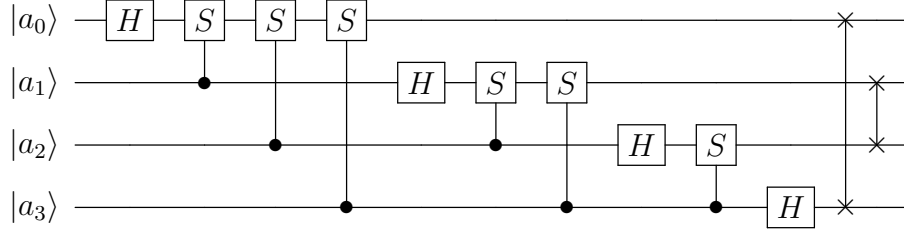


Figura 2.2: Circuito di QFT applicata ad un registro di 4 bit.

La QFT implementata in questo modo crea la seguente sovrapposizione:

$$\mathcal{F}(|a\rangle) = \frac{1}{\sqrt{q}} \sum_{b=0}^{q-1} e^{2\pi i ac/q} |b\rangle$$

Dove $|b\rangle$ è l'inverso del registro $|c\rangle$ desiderato, perciò vengono aggiunti dei gate di swap tra i Qubit in fondo al circuito in modo da ottenere il risultato esatto.

$$\frac{1}{\sqrt{q}} \sum_{b=0}^{q-1} |c\rangle e^{2\pi i ac/q}$$

Quando $d = j - k$ è molto grande la trasformazione $S_{j,k}$ modifica la fase del Qubit in maniera minima. Questo potrebbe essere un ostacolo sia a livello computazionale sia a livello fisico in quanto l'hardware quantistico potrebbe non essere adatto ad applicare queste modifiche così precise. Fortunatamente sappiamo che una QFT può essere approssimata prendendo una soglia d_{max} oltre la quale le porte $S_{j,k}$ non vengono considerate [5]. In [5] viene dimostrato infatti che l'errore introdotto da questa soglia è proporzionale a $n2^{-d_{max}}$ ed è perciò sufficiente scegliere un $d_{max} \in O(\lg \frac{n}{\epsilon})$, che ci permette di semplificare il circuito eliminando molte delle porte $S_{j,k}$ senza avere perdite di precisione significative.

Una QFT approssimata necessita di $O(n \log n)$ porte ma per brevità nei prossimi circuiti sarà indicata come in Figura 2.3:

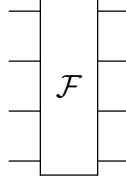


Figura 2.3: Porta QTF schematizzata.

2.3 Algoritmo di Shor

In questa sezione verrà illustrato il processo matematico svolto da Shor: ricordiamo che il problema dei logaritmi discreti in questione chiede di trovare un numero $0 < r < p - 1$ tale che, dato un gruppo moltiplicativo ciclico G , il suo generatore g , e la sua cardinalità, $p = |G|$, sia $r = \log_g(x \pmod{p})$ con $x \in G$. Detto ciò la prima cosa da fare è determinare un numero $q = 2^l$ tale che q sia vicino a p ossia: $p < q < 2p$. Successivamente individuiamo due numeri interi a e b compresi tra 0 e $p - 1$, li inseriamo in due registri di dimensione q e li poniamo in una sovrapposizione uniforme attraverso le porte di Hadamar, per poi calcolare $g^a x^{-b} \pmod{p}$ in un terzo registro ottenendo lo stato del sistema

$$\frac{1}{p-1} \sum_{a=0}^{p-2} \sum_{b=0}^{p-2} |a, b, g^a x^{-b} \pmod{p}\rangle$$

utilizziamo ora due QFT sui registi $|a\rangle$ e $|b\rangle$ per mandarli nella forma $|c\rangle$ e $|d\rangle$ ottenendo:

$$\mathcal{F} |a, b\rangle = \frac{1}{q} \sum_{c,d=0}^{q-1} \exp\left\{ \left(\frac{2\pi i}{q} (ac + bd) \right) \right\} |c, d\rangle$$

e complessivamente:

$$\frac{1}{(p-1)q} \sum_{a,b=0}^{p-2} \sum_{c,d=0}^{q-1} \exp\left\{\left(\frac{2\pi i}{q}(ac+bd)\right)\right\} |c, d, g^a x^{-b} \pmod{p}\rangle$$

Osserviamo ora lo stato del sistema attraverso una misurazione: la probabilità di ottenere lo stato $|c, d, y\rangle$ dove $y \equiv g^k \pmod{p}$ è come già spiegato, il quadrato del modulo delle ampiezze:

$$\left| \frac{1}{(p-1)q} \sum_{\substack{a,b \\ a-rb \equiv k}} \exp\left\{\left(\frac{2\pi i}{q}(ac+bd)\right)\right\} \right|^2$$

La somma viene fatta su (a, b) tali che $a - rb \equiv k \pmod{p-1}$ derivata dalla definizione di $r = \log_g x$ e dalla definizione del terzo registro $g^a x^{-b}$. Da questa condizione ricaviamo a :

$$a = br + k - (p-1) \left\lfloor \frac{br+k}{p-1} \right\rfloor$$

e lo sostituiamo nella formula precedente che diventa l'ampiezza di probabilità di trovare lo stato $|c, d, g^k \pmod{p}\rangle$:

$$\left| \frac{1}{(p-1)q} \sum_{b=0}^{p-2} \exp\left\{\frac{2\pi i}{q}(brc + kc + bd - c(p-1) \left\lfloor \frac{br+k}{p-1} \right\rfloor)\right\} \right|^2$$

Analizzando ora questa formula andiamo a definire in che modo è possibile, a seguito di una misurazione dei primi due registri indicare la validità del risultato ottenuto. Per prima cosa eliminiamo il fattore esponenziale $2\pi i kc/q$ che non influisce sulla probabilità, poi definiamo due variabili T e V come:

$$T = xc + d - \frac{x}{p-1} \{c(p-1)\}_q,$$

$$V = \left(\frac{bx}{p-1} - \left\lfloor \frac{bx+k}{p-1} \right\rfloor \right) \{c(p-1)\}_q,$$

dove il simbolo $\{z\}_q$ indica il residuo di $z \pmod{q}$ con $-q/2 < \{z\}_q < q/2$.

Otteniamo quindi:

$$\frac{1}{(p-1)q} \sum_{b=0}^{p-2} \exp\left\{\frac{2\pi i}{q}bT\right\} \exp\left\{\frac{2\pi i}{q}V\right\}$$

Da questa definiamo due condizioni che, se rispettate, indicheranno la correttezza del risultato ottenuto:

$$|\{T\}_q| = \left| rc + d - \frac{r}{p-q} \{c(p-1)\}_q - jq \right| \leq \frac{1}{2} \quad (2.1)$$

$$|\{c(p-1)\}_q| \leq q/12 \quad (2.2)$$

dove j rappresenta l'intero più vicino a T/q . Nella sua articolo Shor [12] dimostra che la probabilità di ottenere uno stato $|c, d, y\rangle$ che soddisfa le precedenti disequazioni (2.1), (2.2) è di almeno $1/(20q^2)$. Andiamo ora ad ottenere r da una coppia (c, d) attraverso la condizione (2.1) che riscriviamo dividendo per q :

$$-\frac{1}{2q} \leq \frac{d}{q} + r \left(\frac{c(p-1) - \{c(p-1)\}_q}{(p-1)q} \right) \leq \frac{1}{2q}$$

Notiamo che $c' = (c(p-1) - \{c(p-1)\}_q)/q$ è un numero intero e quindi non ci resta altro che approssimare d/q al multiplo di $1/(p-1)$ più vicino e dividere tutto per c' . Da qui è banale calcolare r .

Riportiamo di seguito un metodo equivalente ma più intuitivo rispetto a quello esposto da Shor, proposto da Proos e Zalka [11]: Dato lo stato iniziale:

$$\frac{1}{p-1} \sum_{a=0}^{p-2} \sum_{b=0}^{p-2} |a, b\rangle$$

calcoliamo il terzo registro come $g^a x^b \pmod{p}$ poi riprendiamo la considerazione fatta in precedenza secondo cui $g^a x^b$ può essere scritto come g^k con $k = a - b \log_g x = a - br$ da questa ricaviamo $a = (k - br) \pmod{p}$ e lo sostituiamo nel registro ottenendo uno stato:

$$\frac{1}{p-1} \sum_{b=0}^{p-2} |(k - br), b, g^a x^b \pmod{p}\rangle$$

Ora effettuiamo la QTF sui primi due registri ottenendo:

$$\frac{1}{(p-1)q} \sum_{b=0}^{p-2} \sum_{c,d=0}^{q-1} \exp \left\{ \left(\frac{2\pi i}{q} ((k - br)c + bd) \right) \right\} |c, d, g^k \pmod{p}\rangle$$

Risolvendo la sommatoria in b è facile vedere che se $d \equiv rc$ otterremo un'ampiezza $q \exp\{kc\}$ mentre sparisce nel caso questa condizione non sia vera, perciò otteniamo:

$$\frac{1}{(p-1)} \sum_{c,d=0}^{q-1} \exp\left\{\frac{2\pi i}{q}kc\right\} |c, rc \pmod{p}\rangle$$

Ora in seguito alla misurazione dei primi due registri possiamo ricavare $r = d(c)^{-1} \pmod{p}$. È importante sottolineare che un valore r valido deve essere calcolato in seguito all'individuazione di una coppia (c, d) valida, e ciò avviene con una probabilità minima di $1/(20q^2)$ come calcolato da Shor [12].

Capitolo 3

Il linguaggio Q#

Il linguaggio Q# è parte di quello che viene chiamato Quantum Development Kit (QDK), un insieme di servizi che Microsoft mette a disposizione degli sviluppatori. Dalla documentazione ufficiale [9] i prodotti presenti nel QDK sono:

- Q#: un linguaggio di programmazione pensato e sviluppato per la creazione di algoritmi quantistici.
- Q# library: le librerie del linguaggio che permettono di semplificare il processo di creazione degli algoritmi, dai più elementari come il teletrasporto quantistico ai più complicati come la simulazione di molecole.
- Local quantum machine simulator: un simulatore locale per i programmi sviluppati.
- Resource Estimator: permette di stimare le risorse dei programmi. Utile nel caso di algoritmi troppo complessi per poter essere simulati.
- Estensioni a Visual studio e Visual Studio Code: permettono il debug e il controllo sintattico del codice.
- qsharp: libreria che permette di utilizzare Python come linguaggio di appoggio per la computazione classica in alternativa al C#.

- **IQ#**: estensione utilizzata da Python e da Jupyter per la simulazione.

Analizziamo di seguito le principali caratteristiche del linguaggio e le librerie utilizzate in modo da rendere comprensibile il codice presente nel capitolo successivo.

3.1 Descrizione del linguaggio

Il linguaggio **Q#** è un linguaggio specifico per lo sviluppo di algoritmi quantistici a basso livello, appena sopra il linguaggio macchina, ed è pensato per poter essere utilizzato sia su processori quantistici, sia su simulatori.

3.1.1 Tipi primitivi

Q# è un linguaggio *strongly-typed* che non permette conversioni implicite tra tipi di variabili ed essendo pensato per eseguire istruzioni quantistiche presenta un set di tipi primitivi diverso da quelli tradizionali. Oltre alla presenza dei classici **Int**, **BigInt**, **Double**, **String** e **Bool**, esistono anche i seguenti tipi:

- **Qubit**: rappresenta un Qubit in maniera opaca. Il programmatore non può agire direttamente su questo tipo di dato ma deve necessariamente ricorrere a operazioni che lasciano al compilatore la libertà di implementarle a seconda dell'hardware sottostante.
- **Pauli**: rappresenta gli operatori di Pauli. Questi sono elementi che devono essere usati in caso di rotazioni o di misurazioni. Può assumere quattro valori costanti **PauliI**, **PauliX**, **PauliY** e **PauliZ**.
- **Result**: rappresenta il risultato di un operatore di misura **M** su un Qubit. può assumere due valori: **One** che corrisponde al autovalore -1 e **Zero** che corrisponde all'autovalore $+1$.
- **Unit**: indica una tupla vuota **()**. Viene utilizzato esclusivamente per indicare quando una operazione o una funzione non hanno un tipo di ritorno.

- **Range**: rappresenta un insieme di valori interi in sequenza caratterizzati da `start..step..stop` dove sia `start` sia `stop` sono compresi nel range e `step` è opzionale (il default è 1).

3.1.2 Funzioni e operazioni

Il `Q#` supporta la chiamata a funzioni come gli altri linguaggi ma per rendere più espressivo il codice le divide in due diverse categorie: **function** e **operation**. Le **function** sono utilizzate per indicare una parte di codice che esegue calcoli classici, non richiamano **operation**, non agisce su Qubit e perciò il risultato dipende solo dal valore che viene dato in input; una **function** viene definita pura in quanto ogni volta che le si fornisce uno stesso input restituisce sempre lo stesso output. La sintassi della loro dichiarazione è la seguente:

```
function NomeFunzione ( nomeVariabile : tipoVariabile) : (tipoRitorno) {
    ....
    codice
    ...
    return variabile;
}
```

Le **operation** sono il centro del linguaggio in quanto sono libere di modificare ed agire su Qubit (sempre in maniera indiretta, indicando la trasformazione che deve essere applicata, ma lasciando ai livelli sottostanti la libertà di implementarla). È possibile specializzare le **operation** dichiarate in modo che sia possibile richiamarne l'inversa o aggiungere un registro di controllo che ne determini l'applicazione. Abbiamo visto in precedenza che le trasformazioni applicate ad un Qubit in quanto unitarie devono essere reversibili, il costrutto **adjoint** permette di specificare il comportamento delle **operation** quando è necessario applicarne il processo inverso.

Il costrutto **controlled** permette l'utilizzo controllato dall'applicazione e l'eventuale creazione di stati entangled attraverso l'associazione ad un registro di controllo grazie al fenomeno descritto nella sezione 2.2 del capitolo 1.

La sintassi generica di una `operation` è la seguente:

```
operation NomeOperazione ( nomeVariabile : tipoVariabile) : (tipoRitorno)
is Ctl + Adj{
  body{
    \\ \\ definizione standard
    ...
    return variabile;
  }
  controlled(..){
    ...
  }
  adjoint(..){
    ...
  }
  controlled adjoint(..){
    ...
  }
}
```

i costrutti `ctl` e `adj` sono opzionali e all'occorrenza possono anche essere creati dal compilatore con la definizione `adjoint auto`.

3.2 Librerie

Il Quantum Development Kit mette a disposizione una grande varietà di librerie open source con licenza MIT e con licenza non commerciale Microsoft Research license. La loro documentazione è disponibile online presso il sito della Microsoft [9]. Le librerie si dividono in tre grandi macro-gruppi: Standard libraries, Quantum chemistry libraries e Quantum numeric libraries. In questo progetto non verranno utilizzate le librerie specifiche per la simulazione chimica in quanto non necessarie.

3.2.1 Standard libraries

Le librerie standard sono alla base del linguaggio e sono quelle che permettono le operazioni più elementari sui Qubit. Nel progetto verranno utilizzate le seguenti:

- `Microsoft.Quantum.Intrinsic`: contiene le operazioni che implementano le porte fondamentali della computazione quantistica come quelle spiegate nel capitolo 1 sezione 2, oltre che ad elementi più complessi. È presente inoltre la funzione `Message` che permette di stampare a video una stringa.
- `Microsoft.Quantum.Convert`: contiene le funzioni utilizzate per convertire il tipo di una variabile classica all'interno del codice, è necessaria perché il `Q#` non permette conversioni implicite.
- `Microsoft.Quantum.Arrays`: contiene funzioni per creare e manipolare array di dati. Alcune di queste funzioni possono essere utilizzate anche su array di Qubit in quanto non agiscono sul contenuto dell'array ma sulla sua forma.
- `Microsoft.Quantum.Canon`: Una delle prime librerie rilasciate, contiene molte funzioni ed operazioni di utilità anche se in buona parte deprecate.
- `Microsoft.Quantum.Math`: contiene funzioni matematiche classiche e alcuni tipi di dati non intrinseci nel linguaggio. Sono anche presenti operazioni che restituiscono numeri casuali che, nel caso queste fossero eseguite su una macchina quantistica, restituirebbero numeri realmente casuali al contrario di quello che succede su una macchina classica.

3.2.2 Quantum numeric libraries

Questo gruppo comprende una unica libreria, la `Microsoft.Quantum.Arithmetic`. Questa contiene al suo interno operazioni che implementano gli operatori di base della matematica, ma applicati a numeri interi rappresentati da sovrapposizioni di registri di Qubit. Ciò permette agli sviluppatori di operare con i Qubit senza dover creare da zero un circuito per l'esecuzione di tutte le operazioni base, garantendo oltretutto la

migliore implementazione possibile. La libreria si basa su tre tipi di rappresentazione di numeri interi nei registri di Qubit: Little Endian, Signed Little Endian e Fixed Point. Per ogni tipo sono disponibili un diverso range di operazioni. Nel programma verrà utilizzata principalmente la notazione Little endian per la quale sono presenti le operazioni di addizione, confronto, moltiplicazione, quadrato e divisione con resto, che sono sufficienti alla scrittura del codice.

Capitolo 4

Implementazione dell'algoritmo di Shor in Q#

In questo capitolo verranno illustrate le parti principali del codice sorgente che implementa l'algoritmo per la risoluzione dei logaritmi discreti. Il codice Q# viene compilato dalla libreria python `iqsharp` e successivamente richiamato da uno script attraverso un'apposita funzione. Di seguito verrà presentato prima la funzione principale chiamata `Shor` e successivamente le operazioni `QuantumCopy`, `ModularExponentiation`, `ModularMultiplication`, che implementano nello specifico parti essenziali dell'algoritmo.

4.1 Shor

L'operazione Shor è la parte centrale del programma che viene richiamata da codice python. Inizia con lo statement `using` che riserva lo spazio per 5 registri di Qubit, poi procede a trasferire i numeri a e b da registri classici a registri quantistici e li mette in sovrapposizione con l'utilizzo di porte di Hadamar (H) grazie alla funzione `multiHGate`. Viene poi richiamata la funzione `ModularExponentiation` per calcolare prima g^a e x^b , questi vengono poi moltiplicati per ottenere $g^a x^b$ come descritto nel procedimento in [11]. Si effettua poi una Trasformazione di Fourier approssimata sui

primi due registri grazie all'operazione di libreria `ApproximateQFT` e si misurano i registri. Come detto in precedenza il valore da noi cercato è dato dalla divisione c/d .

```
operation Shor(prime : Int, a : Int,
              b : Int,      l : Int,
              x : Int,      g : Int) : (Int, Int)
{
  using (target = Qubit[l*5]){

    let temp = Partitioned([1,1,1,1,1], target);
    QuantumCopy(temp[0], a, l);
    let quantum_a = temp[0];
    multiHGate(quantum_a, l);
    let quantum_ga = temp[1];
    ModularExponentiation(prime, g, quantum_a, l, quantum_ga);

    QuantumCopy(temp[2], b, l);
    let quantum_b = temp[2];
    multiHGate(quantum_b, l);
    let quantum_xb = temp[3];
    ModularExponentiation(prime, g, quantum_b, l, quantum_xb);

    let quantum_gaxb = temp[4];
    ModularMultiplication (quantum_ga,
                          quantum_xb,
                          quantum_gaxb,
                          l,
                          prime);

    ApproximateQFT(1, LittleEndianAsBigEndian(LittleEndian(quantum_a)));
    ApproximateQFT(1, LittleEndianAsBigEndian(LittleEndian(quantum_b)));
  }
}
```

```

    let c = MeasureRegister(quantum_a, 1);
    let d = MeasureRegister(quantum_b, 1);
    return (c,d);
  }
}

```

4.2 QuantumCopy

Operazione che esegue la copia di un numero inferiore a 2^{length} in un registro di Qubit in notazione little-endian attraverso una serie di porte X.

```

operation Quantum_copy(register: Qubit[], number : Int, lenght: Int)
: Unit {
  body {
    mutable exponent = number;
    for(i in 0 .. (lenght-1)){
      if(exponent%2!=0){
        X(register[i]);
      }
      set exponent = exponent/2;
    }
  }
}

```

ciò è possibile se il registro passato come parametro è un registro che possiede solo stati puri $|0\rangle$. Questi, attraverso un not quantistico (porta X), vengono portati al valore $|1\rangle$, quando necessario, emulando la notazione binaria di un numero classico. Una volta eseguita questa funzione si ottiene $|number\rangle$.

4.3 ModularExponentiation

Operazione che sfrutta l'algoritmo spiegato nel capitolo 2.1 per calcolare efficacemente $base^{\text{exponent}} \pmod{module}$ attraverso la funzione di libreria `MultiplyByModularInteger`, utilizzata in maniera controllata rispetto ai diversi Qubit del registro `exponent`.

```
operation ModularExponentiation
( module : Int,
  base: Int,
  exponent : Qubit[],
  length : Int,
  target : Qubit[]) : Unit{
body{
  X(target[0]);
  for( i in 0 .. (length-1)){
    let multiplier = Floor(PowD(IntAsDouble(base),
                                PowD(2.0,IntAsDouble(i))))%module;
    (Controlled MultiplyByModularInteger) ([exponent[i]],
                                           (multiplier,
                                            module,
                                            LittleEndian(target)));
  }
}
}
```

Risulta molto importante che il registro `target` sia un registro che possiede solo stati puri $|0\rangle$ in quanto necessita di assumere valore uno in notazione little-endian attraverso una porta X sul bit meno significativo; è altrettanto importante che `exponent` sia in notazione little-endian.

4.4 ModularMultiplication

Operazione che attraverso una serie di somme effettua una moltiplicazione modulare. Tra le librerie di Q# è presente un metodo per eseguire la moltiplicazione ma questo, oltre a richiedere un registro target di dimensioni doppie rispetto a quelle dei registri di partenza, non effettua l'operazione di modulo che è essenziale per la riuscita dell'algoritmo.

```
operation ModularMultiplication(a : Qubit[],
                                b : Qubit[],
                                result : Qubit[],
                                length : Int,
                                module :Int) : Unit
{
    body{
        for (i in 0 .. length - 1){
            let k = Floor(PowD(2.0,IntAsDouble(i)));
            for ( j in 0 .. k - 1){
                (Controlled MultiplyAndAddByModularInteger)
                    ([a[i]],
                     ( 1,
                       module,
                       LittleEndian(b),
                       LittleEndian(result))));
            }
        }
    }
}
```

Viene utilizzata l'operazione di libreria `MultiplyAndAddByModularInteger` che effettua la trasformazione $|b\rangle |result\rangle = |b\rangle |b + c * result \pmod{module}\rangle$ con un intero

$c = 1$ in modo da effettuare la somma con modulo di due registri. Questa somma è poi eseguita in maniera condizionata 2^i volte rispetto ad ogni termine a_i del registro a scomponendo la moltiplicazione $a * b \pmod{module}$ in una serie di a somme $b + b + \dots + b \pmod{module}$.

4.5 Risultati

Il programma realizzato è stato testato su vari hardware e software. È stata fatta una stima delle risorse attraverso la funzione `estimate_resources` della libreria `qsharp` su sistema operativo Windows e in ambiente Linux e, come prevedibile, entrambi hanno restituito lo stesso risultato:

```
{'CNOT': 224034, 'QubitClifford': 51964, 'R': 28674, 'Measure': 10,
  'T': 134327, 'Depth': 64222, 'Width': 34, 'BorrowedWidth': 0}
```

Di questo risultato la parte che più ci interessa è la *Width* che indica il numero di Qubit che vengono utilizzati. Mantenere 34 Qubit contemporaneamente è computazionalmente dispendioso e il programma non può essere eseguito su comuni elaboratori. È stato quindi scelto un server con 64 GB di RAM che non è comunque risultato sufficiente; successivamente non è stato possibile reperire risorse maggiori rispetto a quelle del server sopracitato, impedendo una completa esecuzione del programma.

Conclusioni

Come già detto in precedenza gli algoritmi noti per il calcolo del logaritmo discreto hanno una complessità computazionale esponenziale e non sono perciò utilizzabili per la risoluzione del problema in tempi brevi; questi algoritmi sono classificabili come algoritmi ‘brute force’ in quanto cercano di determinare il risultato provando ogni possibile combinazione. Ricordiamo che per calcolare nella sua interezza un gruppo ciclico \mathbb{Z}_p^* devono essere calcolati $g^1, g^2, g^3, \dots, g^{p-1}$ dove g è il generatore e p un numero primo sicuro che, come descritto nell’introduzione al capitolo 2, è un numero primo lungo normalmente 1024 bit, tale che $p - 1$ non è il risultato del prodotto di due numeri primi di piccole dimensioni. Questo calcolo cresce esponenzialmente in 2^q dove q è il numero di bit necessari per scrivere p . Nei moderni sistemi di sicurezza che si basano su questo algoritmo viene scelto $q = 1024$ bit ed è facile capire perché non sia possibile utilizzare un algoritmo ‘brute force’ per spezzare questa codifica.

Shor nel suo testo calcola per l’algoritmo quantistico una probabilità di successo del $1/20q^2$, questo significa che la riuscita dell’algoritmo ha un costo quadratico rispetto a q portando il problema da una classe NP ad una classe BQP (*bounded-error quantum polynomial time*), una classe di problemi che possono essere risolti in tempo polinomiale ammettendo una soglia d’errore. Il cambiamento di classe fa sì che i sistemi di sicurezza basati sul problema dei logaritmi discreti perdano di validità e diventino vulnerabili, ad esempio potrebbero essere violati la maggior parte di DSA, gli algoritmi di firma digitale, permettendo ad un intermediario di decodificare un messaggio, modificarlo e ricodificarlo senza lasciare tracce visibili.

Bisogna sottolineare che la decodificazione di un messaggio con $q = 1024$ bit, senza considerare i registri di supporto e i Qubit utilizzati dalle varie funzioni, necessita di

almeno 3 registri quantistici da 1024 Qubit e questa condizione non può essere soddisfatta con una simulazione su un computer tradizionale, per ogni Qubit utilizzato infatti, la memoria necessaria cresce in maniera esponenziale (nel 2018 la simulazione più avanzata permetteva al massimo 64 Qubit [4]); è necessario quindi utilizzare un computer quantistico, ma allo stato attuale i processori più evoluti sono costituiti da 53 (IBM), 72 (Google), 79 (IonQ) Qubit fisici che come tali non sono perfetti e presentano probabilità di errore. Con tali processori non è ovviamente possibile minacciare il sistema di criptaggio odierno che rimane quindi valido.

La costruzione di computer quantistici avanzati comporterà una minaccia alla sicurezza, ma contemporaneamente permetterà anche la costruzione di nuovi algoritmi che permetteranno di trasferire informazioni attendibili, un esempio è il protocollo BB84 già teorizzato nel 1984 [3] che utilizza misurazioni casuali dei Qubit lungo i vari assi della sfera di Bloch per creare chiavi criptografiche sicure.

Bibliografia

- [1] Stephane Beauregard. Circuit for Shor’s algorithm using $2n+3$ qubits.
- [2] Paul Benioff. Quantum Mechanical Models of Turing Machines That Dissipate No Energy. *Physical Review Letters*, 48(23):1581–1585, jun 1982.
- [3] Charles H. Bennett and Gilles Brassard. Quantum cryptography: Public key distribution and coin tossing. *Theoretical Computer Science*, 560:7–11, dec 2014.
- [4] Zhao-Yun Chen, Qi Zhou, Cheng Xue, Xia Yang, Guang-Can Guo, and Guo-Ping Guo. 64-qubit quantum circuit simulation. *Science Bulletin*, 63(15):964–971, 2018.
- [5] D. Coppersmith. An approximate fourier transform useful in quantum factoring. Technical report, IBM, 2002.
- [6] W. Diffie and M. Hellman. New directions in cryptography. *IEEE Transactions on Information Theory*, 22(6):644–654, nov 1976.
- [7] A. Einstein, B. Podolsky, and N. Rosen. Can quantum-mechanical description of physical reality be considered complete? *Physical Review*, 47(10):777–780, may 1935.
- [8] Richard P. Feynman. Simulating physics with computers. *International Journal of Theoretical Physics*, 21(6-7):467–488, jun 1982.
- [9] Microsoft. Q# documentation. <https://docs.microsoft.com/en-us/quantum/?view=qsharp-preview>.

- [10] S. Pohlig and M. Hellman. An improved algorithm for computing logarithms over $\text{GF}(p)$ and its cryptographic significance (Corresp.). *IEEE Transactions on Information Theory*, 24(1):106–110, jan 1978.
- [11] John Proos and Christof Zalka. Shor’s discrete logarithm quantum algorithm for elliptic curves. *Quantum Info. Comput.*, 3(4):317–344, July 2003.
- [12] Peter W. Shor. Polynomial-Time Algorithms for Prime Factorization and Discrete Logarithms on a Quantum Computer. *SIAM Journal on Computing*, 26(5):1484–1509, oct 1997.