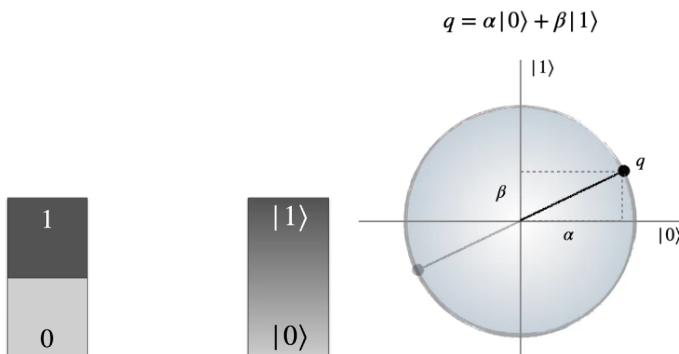


Appunti di Quantum Computer Programming

1. Introduzione

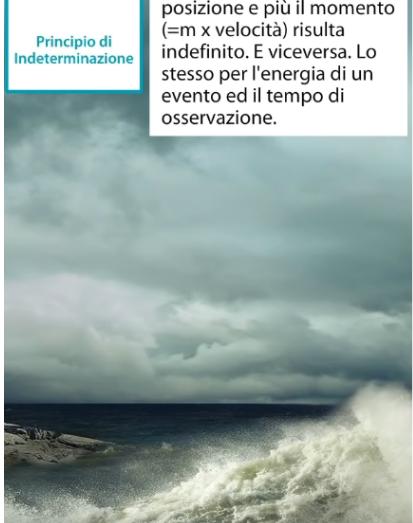
“Un computer quantistico è un dispositivo che prende dei dati in input ed esegue qualche tipo di operazione su quei dati con un processo che può essere descritto solo impiegando la fisica quantistica.” In cosa si differenzia la computazione classica da quella quantistica? La rappresentazione classica è rappresentata con i bit, che possono essere 0 o 1. La rappresentazione quantistica, invece, può assumere tutte le sfumature comprese tra 1 e 0, si ottiene quindi una sovrapposizione di 0 e 1. Un qubit rappresenta infiniti valori.



Ovviamente per sovrapposizione non si deve intendere necessariamente la metà tra 0 e 1, può essere più 1 o più zero, dipende da due costanti α, β . Il vettore individuato dal punto q ha norma 1.

Digressione sulla meccanica quantistica:

Il dualismo onda-particella. Il principio di indeterminazione, secondo cui l'osservazione (misurazione) disturba la particella in un modo impredicibile. Di seguito una slide riassuntiva:

Fatti empirici / Osservazioni	Interpretazioni
 Princípio di Indeterminazione 	L'osservazione: - conferisce significato alla posizione e alla velocità di una particella. - Crea il valore di posizione e velocità. - Disturba la particella in un modo impredicibile. L'indeterminismo limita: - l'esistenza di valori definiti - il senso delle proprietà - cosa possiamo conoscere
 Interpretazione di Copenaghen VS.  Princípio di Complementarità 	L'indeterminismo non limita l'esistenza di tali valori quando non si osserva. La misura non disturba l'oggetto osservato. Non è il cambiamento impredicibile alla base dell'indeterminismo, ma che a seconda dell'esperimento ha senso parlare di alcune grandezze oppure di altre.

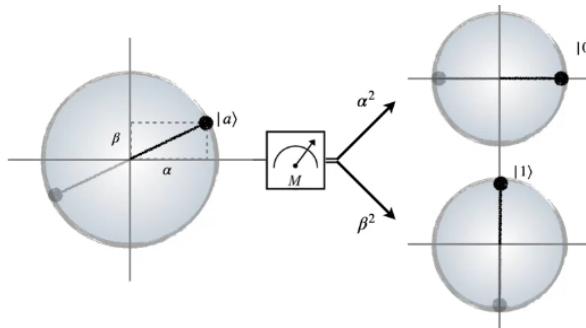
Princípio di sovrapposizione quantistica:

Finchè un oggetto non è misurato può trovarsi contemporaneamente in più stati, nel momento in cui l'oggetto è misurato collappa nello stato in cui è misurato e da quel momento non cambia più.

Quantum entanglement (correlazione tra particelle): una particella messa in relazione per tanto tempo con un'altra particella instaura una certa correlazione che persiste anche se le due particelle vengono allontanate.

Misurazione del Qubit:

La brutta notizia è che non possiamo osservare lo stato in cui si trova il qubit. Ciononostante lo stato può effettivamente essere misurato, solo che il processo di misurazione è un'operazione irreversibile che fa collassare il qubit in uno dei suoi stati base:



Alfa e beta in un certo senso rappresentano delle probabilità. Quindi se si volesse mantenere lo stato di sovrapposizione non bisogna misurare, rimanendo quindi in uno stato di incertezza fino alla fine.

"Fare un calcolo classico sui bit è come cucinare gli ingredienti di una torta individualmente prima di cuocerli insieme nella torta."

Superdense coding: un qubit può essere usato per trasmettere due bit in uno. Sono quindi molto utili per la rappresentazione di molte informazioni in bit:

	1	2	3	4	5	6	7	8	9	10
	2	4	8	16	32	64	128	256	512	1024

	10	100	1000	10000
	10^3	10^{30}	10^{300}	10^{3000}

Grazie al principio di sovrapposizione è possibile simulare più input contemporaneamente. Dando una serie di input ad un algoritmo quantistico, l'algoritmo trasformerà la sovrapposizione di tutti gli input nella sovrapposizione di tutti gli output (non tutti gli output!).

The Space and the States:

Lo stato dei sistemi quantistici sarà sempre descritto da un singolo vettore unitario a che giace in qualche spazio vettoriale fisso di dimensione $N=2^n$ per qualche n (numero di qubit). Quindi lo stato è sempre un vettore dove ogni componente del vettore a_k è un numero reale o complesso a seconda se lo spazio è reale o complesso. Ogni componente è chiamata **ampiezza**.

Come spazio si possono prendere solo vettori la cui norma è 1, ovvero vettori unitari. Rappresentabile geometricamente come una ipersfera.

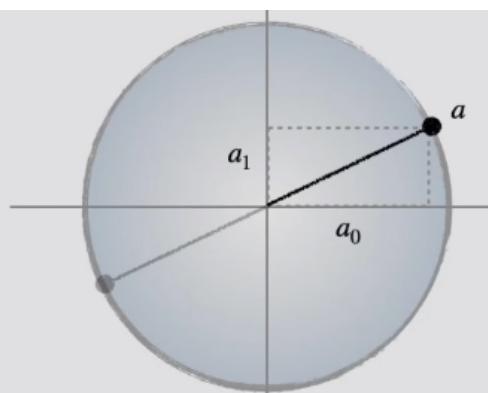
$$a = \begin{bmatrix} a_0 \\ a_1 \\ \vdots \\ a_{N-1} \end{bmatrix} \quad \|a\| = \sqrt{a_0^2 + a_1^2 + \dots + a_{N-1}^2} = 1$$

Es:

$$\text{A single Qubit } a = \begin{bmatrix} a_0 \\ a_1 \end{bmatrix}$$

can be represented as a point in the unit circle, where a_0 represents the coordinate in the x axis while a_1 represents the coordinate in the y axis.

$$\|a\| = \sqrt{a_0^2 + a_1^2} = 1$$



Gli stati si distinguono tra stati generali e stati base. Gli **stati base** sono denotati da un vettore e_k dove tutte le componenti sono 0 eccetto la k-esima che è 1:

$$e_0 = \begin{bmatrix} 1 \\ 0 \\ 0 \\ \vdots \\ 0 \end{bmatrix} \quad e_1 = \begin{bmatrix} 0 \\ 1 \\ 0 \\ \vdots \\ 0 \end{bmatrix} \quad e_2 = \begin{bmatrix} 0 \\ 0 \\ 1 \\ \vdots \\ 0 \end{bmatrix} \quad e_{N-1} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ \vdots \\ 1 \end{bmatrix}$$

$$|0\rangle = e_0 = \begin{bmatrix} 1 \\ 0 \end{bmatrix} \quad |1\rangle = e_1 = \begin{bmatrix} 0 \\ 1 \end{bmatrix}$$

Con questa operazione:

$$|k\rangle = e_k = \begin{bmatrix} 0 \\ 0 \\ \vdots \\ 1 \\ \vdots \\ 0 \end{bmatrix} \quad |a\rangle = \begin{bmatrix} a_0 \\ a_1 \\ a_2 \\ \vdots \\ a_{N-1} \end{bmatrix}$$

$$\langle k | a \rangle = a_k$$

si vuole selezionare il k-esimo elemento all'interno del vettore a che ne conterrà un certo n . Ogni vettore che non si trova nello stato base è nello stato di sovrapposizione.

Uno **stato generale** è uno stato base o una superimposizione di due o più stati base.

$$u = 4e_0 + 3e_1 = \begin{bmatrix} 4 \\ 3 \end{bmatrix} \quad \hat{u} = \frac{1}{5}(4e_0 + 3e_1) = \frac{1}{5} \begin{bmatrix} 4 \\ 3 \end{bmatrix}$$

Vettore unitario, a seguito della
normalizzazione (1/5)

$$|u\rangle = 4|0\rangle + 3|1\rangle \quad |\hat{u}\rangle = \frac{1}{5}(4|0\rangle + 3|1\rangle)$$

Definizione: uno stato generale u è dato dalla superimposizione di tutte le basi, dello spazio di riferimento:

$$u = a_0e_0 + a_1e_1 + \dots + a_{N-1}e_{N-1} = \begin{bmatrix} a_1 \\ a_2 \\ \vdots \\ a_{N-1} \end{bmatrix}$$

Prodotto tensoriale:

$$\begin{bmatrix} a_1 \\ a_2 \end{bmatrix} \otimes \begin{bmatrix} b_1 \\ b_2 \end{bmatrix} = \begin{bmatrix} a_1 & \begin{bmatrix} b_1 \\ b_2 \end{bmatrix} \\ a_2 & \begin{bmatrix} b_1 \\ b_2 \end{bmatrix} \end{bmatrix} = \begin{bmatrix} a_1b_1 \\ a_1b_2 \\ a_2b_1 \\ a_2b_2 \end{bmatrix}.$$

Esempi. Caso vettori:

$$\begin{bmatrix} 1 \\ 2 \end{bmatrix} \otimes \begin{bmatrix} 3 \\ 4 \\ 6 \\ 8 \end{bmatrix} = \begin{bmatrix} 3 \\ 4 \\ 6 \\ 8 \end{bmatrix}, \quad \begin{bmatrix} 1 \\ 0 \end{bmatrix} \otimes \begin{bmatrix} 0 \\ 1 \end{bmatrix} = \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix}.$$

$$a = \begin{bmatrix} 0 \\ 2 \\ 1 \\ 3 \\ 0 \end{bmatrix} \quad N=4 \quad b(00)=2 \quad b(01)=1 \dots$$

$$b = \begin{bmatrix} 3 \\ 1 \\ 1 \end{bmatrix} \quad N=2 \quad b(0)=3 \quad b(1)=1$$

$$e = \begin{bmatrix} x, x, x, x, x, x, x, x \\ 000, 001, 010, 011, 100, 101, 110, 111 \end{bmatrix} \quad N=8$$

$$e(101) = a(10) \times b(1) = 3 \times 1$$

Caso matrici:

$$U = \begin{bmatrix} 2 & 1 \\ 0 & 1 \end{bmatrix}$$

$$Q = \begin{bmatrix} -1 & 0 \\ 0 & -2 \end{bmatrix}$$

$$U \otimes Q = \begin{bmatrix} 2 \times Q & 1 \times Q \\ 0 \times Q & 1 \times Q \end{bmatrix} = \begin{bmatrix} -2 & 0 & -1 & 0 \\ 0 & -4 & 0 & -2 \\ 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & -2 \end{bmatrix}$$

Caso vettore-matrice:

$$U = \begin{bmatrix} 2 \\ 1 \end{bmatrix} \quad Q = \begin{bmatrix} \frac{1}{\sqrt{2}} & 0 \\ -1 & 1 \end{bmatrix}$$

$$U \otimes Q = \begin{bmatrix} \frac{2}{\sqrt{2}} & 0 \\ -2 & 2 \\ \frac{1}{\sqrt{2}} & 0 \\ -1 & 1 \end{bmatrix} \quad Q \otimes U = \begin{bmatrix} \frac{1}{\sqrt{2}} U & 0 U \\ -1 U & 1 U \end{bmatrix} = \begin{bmatrix} \frac{2}{\sqrt{2}} & 0 \\ \frac{1}{\sqrt{2}} & 0 \\ -2 & 2 \\ -1 & 1 \end{bmatrix}$$

Definizione: Se lo stato può essere rappresentato come prodotto tensoriale di due stati allora è separabile (separable) [1]; altrimenti è non separabile (entangled, correlato) [2].

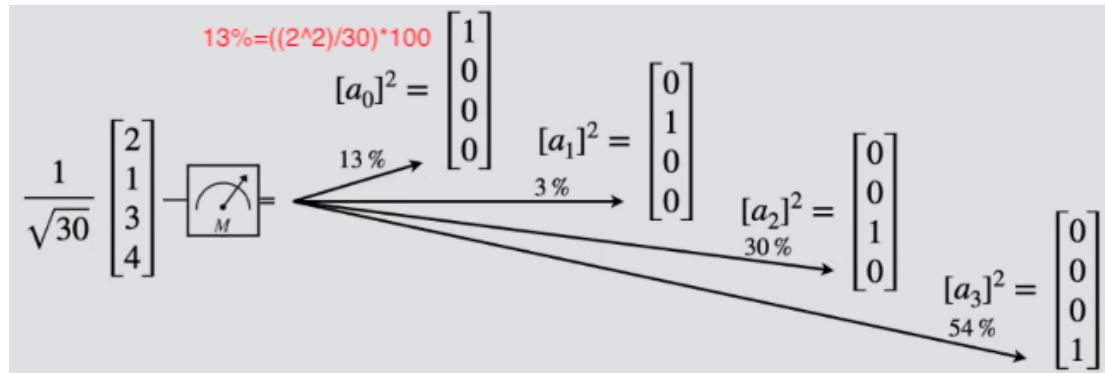
Uno stato entangled non è separabile e si vede dal fatto che nell'immagine [2] i qubit hanno dimensioni differenti.

$$[e_0]^2 = \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \end{bmatrix} = e_0 \otimes e_0 = \begin{bmatrix} 1 \\ 0 \end{bmatrix} \otimes \begin{bmatrix} 1 \\ 0 \end{bmatrix}$$

$$[e_1]^2 = \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \end{bmatrix} = e_0 \otimes e_1 = \begin{bmatrix} 1 \\ 0 \end{bmatrix} \otimes \begin{bmatrix} 0 \\ 1 \end{bmatrix} \quad [1]$$

$$[e_3]^3 = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} = [e_1]^2 \otimes e_1 = \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \end{bmatrix} \otimes \begin{bmatrix} 0 \\ 1 \end{bmatrix} \quad [2]$$

Correlazione con il concetto di probabilità:



Se dei Qubit sono separabili, in un sistema di dimensione n , allora i qubit possono essere misurati indipendentemente, collasserà solo il qubit misurato. Se invece i qubit sono correlati (entangled) la misurazione non solo farà collassare il qubit misurato ma anche il qubit con cui è correlato collasserà. Altro esempio:

64%=((4^2)/25)*100. Il collassare ad 10 al primo qubit non influenza il secondo

$$q = \frac{1}{5\sqrt{5}} \begin{bmatrix} 3 \\ 4 \\ 6 \\ 8 \end{bmatrix} = \frac{1}{\sqrt{5}} \begin{bmatrix} 1 \\ 2 \end{bmatrix} \otimes \frac{1}{5} \begin{bmatrix} 3 \\ 4 \end{bmatrix} \xrightarrow{\text{Measurement}} \begin{bmatrix} 1 \\ 0 \end{bmatrix} \otimes \begin{bmatrix} 0 \\ 1 \end{bmatrix} = \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \end{bmatrix}$$

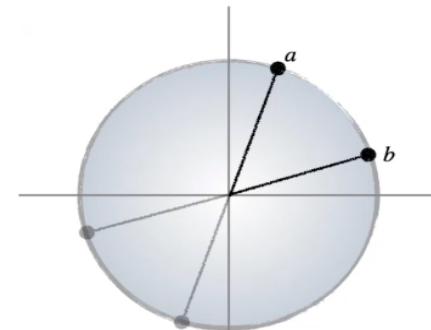
Quantum operations:

Le operazioni (che sono sostanzialmente matrici unitarie particolari) che si operano sono lineari ed invertibili. Per far ciò si opera esclusivamente con vettori unitari. Questo permette di passare da uno stato ad un altro potendo tornare anche indietro. Esempio:

Example

$$a = \frac{1}{\sqrt{10}} \begin{bmatrix} 1 \\ 3 \end{bmatrix} \quad U = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}$$

$$b = Ua = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \times \frac{1}{\sqrt{10}} \begin{bmatrix} 1 \\ 3 \end{bmatrix} = \frac{1}{\sqrt{10}} \begin{bmatrix} 3 \\ 1 \end{bmatrix}$$



Quindi un quantum programs è composto da:

$$e_0 = \begin{bmatrix} 1 \\ 0 \\ \vdots \\ 0 \end{bmatrix};$$

1. Start: lo stato iniziale è sempre composto da un vettore elementare
2. Move: Se il sistema si trova in qualche stato a , allora possiamo spostarlo applicando una trasformazione unitaria U . Quindi a si sposterà nel nuovo stato b , dove $b = Ua$;
3. End: Otteniamo le informazioni di out dalla computazione quantistica tramite un misuramento. Se lo stato finale è c , allora k è vista come una probabilità $|c|^2$.

*esempi vedi slide 1 pagina 44.

Prodotto interno booleano:

Date 2 stringhe Booleane x, y di lunghezza m , allora $x \bullet y$ è definito prodotto interno booleano:

$$x_1 y_1 \oplus x_2 y_2 \oplus \dots \oplus x_n y_n$$

dove \oplus è lo XOR.

Esempio:

$$\begin{aligned}
 100110 \bullet 110010 &= \\
 &= (1 \cdot 1) \oplus (0 \cdot 1) \oplus (0 \cdot 0) \oplus (1 \cdot 0) \oplus (1 \cdot 1) \oplus (0 \cdot 0) = \\
 &= 1 \oplus 0 \oplus 0 \oplus 0 \oplus 1 \oplus 0 = 0
 \end{aligned}$$

Inner product:

The **inner product** of two *real* vectors a and b is given by

$$\langle a, b \rangle = \sum_{k=0}^m a(k)b(k).$$

Complex spaces need a different definition, given in section 3.4. The vectors a and b are **orthogonal** if their inner product is 0.

Esempio:

$$\begin{aligned}
 a &= \begin{bmatrix} 4 \\ 0 \\ -1 \\ 2 \end{bmatrix} & b &= \begin{bmatrix} 1 \\ -2 \\ 3 \\ 1 \end{bmatrix} \\
 \langle a, b \rangle &= 4 \times 1 + 0 \times (-2) + (-1) \times 3 + 2 \times 1 = 4 - 3 + 2 = 3
 \end{aligned}$$

Due vettori a, b sono **ortogonali** se il loro inner product è 0, ovvero:

$$\langle a, b \rangle = \sum_{k=0}^m a(k)b(k) = 0$$

Esempio:

$$\begin{aligned}
 a &= \begin{bmatrix} 4 \\ 0 \end{bmatrix} & b &= \begin{bmatrix} 0 \\ 1 \end{bmatrix} \\
 \langle a, b \rangle &= 4 \times 0 + 0 \times 1 = 0 + 0 = 0
 \end{aligned}$$

2. Algebra lineare di base

Hilbert Spaces: Del tutto analogo allo spazio euclideo, si usa H_N per denotare lo spazio di Hilbert di dimensione N . Gli elementi sono dei vettori reali a di dimensione N .

La somma di due vettori (con la stessa dimensione) a di dimensione N è definito come:

$$\begin{bmatrix} \mathbf{a}(0) \\ \vdots \\ \mathbf{a}(N-1) \end{bmatrix} + \begin{bmatrix} \mathbf{b}(0) \\ \vdots \\ \mathbf{b}(N-1) \end{bmatrix} = \begin{bmatrix} \mathbf{a}(0)+\mathbf{b}(0) \\ \vdots \\ \mathbf{a}(N-1)+\mathbf{b}(N-1) \end{bmatrix}.$$

Se a è ancora un vettore di H_N e c un numero reale, allora il prodotto $b = ca$ è definito come:

$$c \begin{bmatrix} \mathbf{a}(0) \\ \vdots \\ \mathbf{a}(N-1) \end{bmatrix} = \begin{bmatrix} c\mathbf{a}(0) \\ \vdots \\ c\mathbf{a}(N-1) \end{bmatrix}.$$

L'essenza astratta dello spazio di Hilbert è che ogni vettore ha una **norma unitaria**.

$$\|\mathbf{a}\| = \left| \sum_k \mathbf{a}(k)^2 \right|^{1/2}.$$

Nel caso di due dimensioni la norma del vettore rappresentato da:

$$\mathbf{a} = \begin{bmatrix} r \\ s \end{bmatrix}$$

è data da $\sqrt{r^2 + s^2}$

Es:

$$\mathbf{u} = \begin{bmatrix} 2/3 \\ -1/3 \\ -2/3 \end{bmatrix} \quad \|\mathbf{u}\| = \sqrt{(2/3)^2 + (-1/3)^2 + (-2/3)^2} = \sqrt{4/9 + 1/9 + 4/9} = 1$$

Se abbiamo un vettore generico (non unitario) possiamo normalizzarlo in maniera tale che diventi unitario: $\hat{\mathbf{u}} = \frac{1}{\|\mathbf{u}\|} \mathbf{u}$. Esempio:

$$\begin{aligned} \mathbf{u} &= \begin{bmatrix} 2 \\ 2 \end{bmatrix} & \|\mathbf{u}\| &= \sqrt{4+4} = 2\sqrt{2} & \hat{\mathbf{u}} &= \frac{1}{2\sqrt{2}} \begin{bmatrix} 2 \\ 2 \end{bmatrix} = \begin{bmatrix} 1/\sqrt{2} \\ 1/\sqrt{2} \end{bmatrix} \\ \mathbf{u} &= \begin{bmatrix} 4/3 \\ 2/3 \\ 4/3 \end{bmatrix} & \|\mathbf{u}\| &= \sqrt{(4/3)^2 + (2/3)^2 + (4/3)^2} = \sqrt{36/9} = 2 & \hat{\mathbf{u}} &= \frac{1}{2} \begin{bmatrix} 4/3 \\ 2/3 \\ 4/3 \end{bmatrix} = \begin{bmatrix} 2/3 \\ 1/3 \\ 2/3 \end{bmatrix} \end{aligned}$$

```
def normalizza(v):
    v = v/np.linalg.norm(v)
    return v
# esempio
print("Normalizza ",normalizza([1,1]))
```

Prodotto tensoriale di due matrici: $c(ij) = a(i)b(j)$. Esempio:

Assume $a = [2,4,1, -3]$ so that $a(00) = 2$, $a(01) = 4$, $a(10) = 1$ and $a(11) = -3$

Assume $b = [-2,0,1,2]$ so that $b(00) = -2$, $b(01) = 0$, $b(10) = 1$ and $b(11) = 2$

Then $c = a \otimes b = [-4,0,2,4, -8,0,4,8, -2,0,1,2,6,0, -3, -6]$

$$c(0000) = a(00)b(00) = 2 \times (-2) = -4$$

$$c(0001) = a(00)b(01) = 2 \times 0 = 0$$

$$c(0010) = a(00)b(10) = 2 \times 1 = 2$$

...

Matrici:

Le matrici, come detto precedentemente, rappresentano degli operatori.

Una delle proprietà fondamentale delle matrice è la **linearità**:

$$U(a + b) = Ua + Ub$$

Se U è una matrice e utilizziamo U^k per indicare la k-esima potenza della matrice, allora:

$$U^k = \underbrace{UU\cdots U}_{k \text{ copies}}.$$

Esempio con la matrice di Hadamard:

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

H_1

$$H^2 = H \times H = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \times \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} =$$

$$= \frac{1}{2} \begin{bmatrix} 2 & 0 \\ 0 & 2 \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$$

da ciò ne segue che:

$$H^k = \begin{cases} H & \text{SE } k \text{ E' DISPARO} \\ I & \text{SE } k \text{ E' PARI} \end{cases}$$

Nota: Se moltiplichiamo una matrice quadrata per un vettore otteniamo un vettore.

Matrici unitarie: una matrice U si dice unitaria se $U^T U = I$. Esempi:

$$\begin{aligned} H &= \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ -1 & 1 \end{bmatrix} & H^T &= \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & -1 \\ 1 & 1 \end{bmatrix} & H^T H &= \frac{1}{2} \begin{bmatrix} 2 & 0 \\ 0 & 2 \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \\ X &= \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} & X^T &= \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} & X^T X &= \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \end{aligned}$$

Un altro modo per capire se la matrice è unitaria è dato dalla verifica che i singoli vettori (riga o colonna) che compongono la matrice presi a 2 a 2 sono ortogonali tra loro. Esempio:

imposte

$$\begin{aligned} U &= \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 0 & 1 \\ 0 & \sqrt{2} & 0 \\ -1 & 0 & 1 \end{bmatrix} \\ u_0 &= \frac{1}{\sqrt{2}} \begin{bmatrix} 1 \\ 0 \\ 1 \end{bmatrix} & \|u_0\| &= \sqrt{\left(\frac{1}{\sqrt{2}}\right)^2 + 0 + \left(\frac{1}{\sqrt{2}}\right)^2} = 1 \\ u_1 &= \frac{1}{\sqrt{2}} \begin{bmatrix} 0 \\ \sqrt{2} \\ 0 \end{bmatrix} & \|u_1\| &= \sqrt{0 + \left(\frac{1}{\sqrt{2}}\right)^2 + 0} = 1 \\ u_2 &= \frac{1}{\sqrt{2}} \begin{bmatrix} -1 \\ 0 \\ 1 \end{bmatrix} & \|u_2\| &= \sqrt{\left(-\frac{1}{\sqrt{2}}\right)^2 + 0 + \left(\frac{1}{\sqrt{2}}\right)^2} = 1 \end{aligned}$$

Nota: Siamo interessati alle matrici unitarie perché tali matrici mantengono la distanza euclidea. Quindi se U è una matrice unitaria ed a un vettore, allora $\|Ua\| = \|a\|$. Di conseguenza possiamo utilizzare come operatori queste matrici unitarie per spostarci da un vettore ad un altro.

Matrici di permutazione: Una matrice quadrata si dice di permutazione se per ogni riga e colonna ci sono solo zeri tranne una che ha 1. Tali matrici sono anche unitarie, e si utilizzano per apportare scambi tra righe e colonne. Esempi:

$$\begin{aligned} P &= \begin{bmatrix} 0 & 0 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix} & P^T &= \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 0 & 0 \end{bmatrix} & P^T P &= \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \\ U &= \begin{bmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix} & U^T &= \begin{bmatrix} 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix} & U^T U &= \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \end{aligned}$$

$$\begin{aligned} P &= \begin{bmatrix} 0 & 0 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix} & A &= \begin{bmatrix} 2 & 1 & 3 \\ 0 & 3 & 4 \\ 3 & 4 & 6 \end{bmatrix} \\ PA &= \begin{bmatrix} 3 & 4 & 6 \\ 2 & 1 & 3 \\ 0 & 3 & 4 \end{bmatrix} & & \text{(permuting rows of } A\text{)} \\ AP &= \begin{bmatrix} 1 & 3 & 2 \\ 3 & 4 & 0 \\ 4 & 6 & 3 \end{bmatrix} & & \text{(permuting columns of } A\text{)} \end{aligned}$$

3. Boolean functions, quantum bits, and feasibility

Una funzione booleana f è una mappatura da $\{0,1\}^n$ a $\{0,1\}^m$ per una dato numero n, m . Quando definiamo una funzione $f(x_1, \dots, x_n) = (y_1, \dots, y_m)$, dobbiamo pensare che x_i sono gli input e le y_j sono gli output. Esempio:

$f: \{0,1\}^2 \rightarrow \{0,1\}$	$f(x_1, x_2) = 1 \iff x_1 = x_2$
$f: \{0,1\}^2 \rightarrow \{0,1\}$	$f(x_1, x_2) = 1 \iff x_1 = 1 \text{ and } x_2 = 1$
$f: \{0,1\}^2 \rightarrow \{0,1\}^2$	$f(x_1, x_2) = (x_2, x_1)$
$f: \{0,1\}^2 \rightarrow \{0,1\}^2$	$f(x_1, x_2) = (x_1 \vee x_2, x_1 \wedge x_2)$
$f: \{0,1\}^3 \rightarrow \{0,1\}^2$	$f(x_1, x_2, x_3) = (x_1 \cdot x_3, x_2 \cdot x_3)$

Nel caso in cui per rappresentare l'output sia necessitano un numero di bit minore rispetto all'input si procede con un padding di zeri, ovvero si mantiene lo stesso numero di bit dell'input ma si inseriscono degli zeri nella parte più significativa.

Quando il numero di bit di output è contraddistinto da un solo bit si parla di **predicato**.

Funzioni basi:

- AND: Si tratta di una funzione $f(x_1, \dots, x_n)$ definita in 1 se e solo se ogni termine è 1. Ovvero:

$$f(1, 1, 1) = 1 \text{ and } f(1, 0, 1, 1) = 0$$

- OR: Si tratta di una funzione $f(x_1, \dots, x_n)$ definita in 1 se almeno una componente è 1. Ovvero:

$$f(0, 1, 1) = 1 \text{ and } f(0, 0, 0, 0) = 0$$

- XOR: Si tratta di una funzione $f(x_1, \dots, x_n)$ definita in 1 se il numero degli 1 è dispari. Ovvero:

$$f(0, 1, 1) = 0 \text{ and } f(1, 1, 1, 1, 1) = 1$$

Le operazioni binarie suseposte possono essere applicate a due stringhe separate bit a bit. Esempio:

$$x = 1011001 \quad y = 1101111$$

$$z = x \oplus y = (1 \oplus 1)(0 \oplus 1)(1 \oplus 0)(1 \oplus 1)(0 \oplus 1)(0 \oplus 1)(1 \oplus 1) = 0110110$$

Prodotto interno booleano fra stringhe: Il risultato (un unico bit) è ottenuto facendo AND bit a bit tra le due stringhe x, y del risultato ne viene fatto lo XOR: $x \bullet y = \text{XOR}(x_1 \wedge y_1, x_2 \wedge y_2, \dots, x_n \wedge y_n)$. Esempio:

$$x = 1011001 \quad y = 1101111$$

$$z = x \bullet y = \text{XOR}(1 \wedge 1)(0 \wedge 1)(1 \wedge 0)(1 \wedge 1)(0 \wedge 1)(0 \wedge 1)(1 \wedge 1) =$$

$$= \text{XOR}(1001001) = 1$$

Se il numero di 1 è dispari allora il risultato è 1, 0 altrimenti.

Fattibilità:

Non tutte le funzioni booleane sono fattibili. Sicuramente la funzione più semplice è la funzione OR: basta dare un'occhiata ai bit di input e controllare se uno è un 1. Se presente un solo 1, allora chiaramente la funzione OR è 1. AND è simile. Più difficile sembrerebbe la funzione XOR. La ragione intuitiva è che in questo caso bisogna contare il numero di bit, e questo conteggio deve essere esatto. Un esempio di funzione non elementare è la funzione PRIME (test di primalità): la funzione $f(x_1, \dots, x_n)$ assume 1 se la stringa che rappresenta la sua rappresentazione binaria $x = x_1, x_2, \dots, x_n$ è un numero primo (divisibile per 1 o per se stesso).

Un'altro esempio è la funzione FACTOR: data la funzione $f(x_1, \dots, x_n, w_1, w_2, \dots, w_n)$, ottenuta dalla concatenazione di 2 stringhe di lunghezza n , che rappresentano due numeri interi in binario, restituisce 1 se e solo se x (il primo) non ha divisori più grandi di w (il secondo).

"La risoluzione di FACTOR risulta intrattabile dal punto di vista della complessità. PRIME per quanto complicata invece è fattibile (tempo polinomiale)"

Dato un problema, qualsiasi funzione booleana può essere definita dalla sua **tavella di verità**. Esempio della tabella di verità per la funzione or esclusivo (XOR):

x	y	$x \oplus y$
0	0	0
0	1	1
1	0	1
1	1	0

In generale, una funzione booleana $f(x_1, \dots, x_n)$ è definita da una tabella di verità che ha 2^n righe, una per ogni possibile ingresso. Così, se $n = 3$, ci sono otto possibili righe:

000, 001, 010, 011, 100, 101, 110, 111

La difficoltà è che, al crescere del numero di ingressi, la tabella di verità aumenta esponenzialmente in dimensione. Quindi, rappresentare le funzioni booleane con le loro tabelle di verità è sempre possibile, ma non sempre fattibile.

Rappresentazione quantistica degli argomenti booleani:

Sia $N = 2^n$. Ogni coordinata nello spazio di Hilbert N-dimensionale corrisponde a una stringa binaria di lunghezza n . Lo schema di codifica standard assegna a ciascun indice $j \in [0, \dots, n - 1]$ la stringa binaria di n bit che denota j in notazione binaria, con gli 0 iniziali se necessario. Questo produce l'ordinamento lessicografico standard sulle stringhe. Ad esempio, con $n = 2$ e $N = 4$, mostriamo l'indicizzazione applicata a una matrice di permutazione:

	00	01	10	11
00	1	0	0	0
01	0	1	0	0
10	0	0	0	1
11	0	0	1	0

La mappatura è $f(0, 0) = 00$, $f(01) = 01$, $f(10) = 11$, $f(11) = 10$, e in generale $f(x_1, x_2) = (x_1, x_1 \oplus x_2)$. Pertanto, l'operatore scrive lo XOR nel secondo bit lasciando uguale il primo.

Si può anche dire che nega il secondo bit se e solo se il primo bit è 1. Questa negazione stessa è rappresentata su un bit da una matrice che abbiamo visto prima, ora con lo schema di indicizzazione, è:

$$X = \begin{array}{c|cc} & 0 & 1 \\ \hline 0 & 0 & 1 \\ 1 & 1 & 0 \end{array}$$

Così, la negazione è controllata dal primo bit, il che spiega il nome "controlled-NOT" (CNOT) per l'intera operazione 4×4 . Potremmo pensarla come una sorta di *if*.

Per ottenere una funzione booleana generale $y = f(x_1, \dots, x_n)$, abbiamo bisogno di $n + 1$ coordinate booleane, che implica $2N = 2^{n+1}$ coordinate di matrice. Quello che calcoliamo veramente è la funzione F che va da $n + 1$ a $n + 1$, n sono gli input che rimangono inalterati anche nell'output, mentre l'output è dato dagli stessi input ma in XOR con z :

$$F(x_1, \dots, x_n, z) = (x_1, \dots, x_n, z \oplus f(x_1, \dots, x_n))$$

Si tratta di una matrice quadrata e quindi le funzioni sono invertibili.

Esempio di NOT:

NOT function	Example
$f(x) = (\neg x)$	
$F(x, z) = (x, z \oplus (\neg x))$	
	$F(0,0) = (0,0 \oplus (\neg 0)) = (0,1)$
	$F(0,1) = (0,1 \oplus (\neg 0)) = (0,0)$
	$F(1,0) = (1,0 \oplus (\neg 1)) = (1,0)$
	$F(1,1) = (1,1 \oplus (\neg 1)) = (1,1)$

Esempio di AND:

AND function	
$f(x_1, x_2) = (x_1 \wedge x_2)$	
$F(x_1, x_2, z) = (x_1, x_2, z \oplus (x_1 \wedge x_2))$	
	$F(0,0,0) = (0,0,0 \oplus (0 \wedge 0)) = (0,0,0)$
	$F(0,0,1) = (0,0,1 \oplus (0 \wedge 0)) = (0,0,1)$
	$F(0,1,0) = (0,1,0 \oplus (0 \wedge 1)) = (0,1,0)$
	$F(0,1,1) = (0,1,1 \oplus (0 \wedge 1)) = (0,1,1)$
	$F(1,0,0) = (1,0,0 \oplus (1 \wedge 0)) = (1,0,0)$
	$F(1,0,1) = (1,0,1 \oplus (1 \wedge 0)) = (1,0,1)$
	$F(1,1,0) = (1,1,0 \oplus (1 \wedge 1)) = (1,1,1)$
	$F(1,1,1) = (1,1,1 \oplus (1 \wedge 1)) = (1,1,0)$

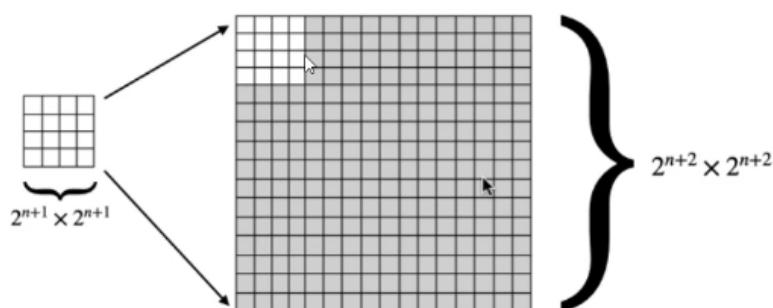
La sua prima virtù, necessaria alla fisica quantistica, è che la funzione F è **invertibile**, infatti F è il suo stesso inverso:

$$\begin{aligned} F(F(x_1, \dots, x_n, z)) &= F(x_1, \dots, x_n, z \oplus y) \\ &= (x_1, \dots, x_n, (z \oplus y) \oplus y) = (x_1, \dots, x_n, z). \end{aligned}$$

La sua seconda virtù è che possiamo rappresentare le funzioni booleane con **matrici di permutazione**:

P_{NOT}		P_{AND}								P_{OR}							
		000	001	010	011	100	101	110	111	000	001	010	011	100	101	110	111
00	00	0	1	0	0					000	1						
01	01	1	0	0	0					001		1					
10	10	0	0	1	0					010			1				
11	11	0	0	0	1					011				1			
										100					1		
										101				1			
										110					1		
										111						1	

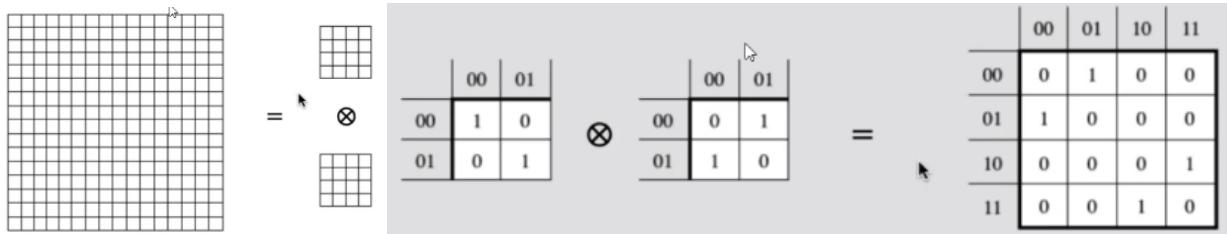
Il contro è che la dimensione della matrice di permutazione P_f **cresce esponenzialmente**:



Durante la computazione è probabile che servano alcune variabili di appoggio, tali variabili prendono il nome di **ancilla qubits**.

Quantum Feasibility:

È comunque possibile (non sempre) scomporre le dimensioni eccessive di una matrice in matrici più piccole, sapendo che il prodotto tensoriale delle matrici piccole restituiscono la matrice di partenza:



Quantum Circuits:

Esempio di scomposizione:

Target computation

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

64 ×

$f(0,1) = 1$
 $F(0,1,0) = (0,1,1)$
 $P_f \times e_{010} = e_{011}$

(1) Scomposizione del vettore, (2) scomposizione dell'operatore e (3) applicazione delle moltiplicazioni:

$$(1) \quad e_{010} = \begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} = e_0 \otimes e_1 \otimes e_0 = \begin{bmatrix} 1 \\ 0 \end{bmatrix} \otimes \begin{bmatrix} 0 \\ 1 \end{bmatrix} \otimes \begin{bmatrix} 1 \\ 0 \end{bmatrix}$$

$$(2) \quad P_f = \begin{bmatrix} 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \otimes \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \otimes \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}$$

IDENTITY

XX

$$(3) \quad P_f e_{010} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} 1 \\ 0 \end{bmatrix} \otimes \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} 0 \\ 1 \end{bmatrix} \otimes \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} 1 \\ 0 \end{bmatrix}$$

Rappresentando quanto detto otteniamo:

Quantum Circuit

	12 ×	12 ×
Qubit-line	$\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} 1 \\ 0 \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \end{bmatrix}$	$= \begin{bmatrix} 0 \\ 0 \end{bmatrix}$
Qubit-line	$\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} 0 \\ 1 \end{bmatrix} = \begin{bmatrix} 0 \\ 1 \end{bmatrix}$	$= \begin{bmatrix} 1 \\ 0 \end{bmatrix}$
Qubit-line	$\begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} 1 \\ 0 \end{bmatrix} = \begin{bmatrix} 0 \\ 1 \end{bmatrix}$	$= \begin{bmatrix} 0 \\ 0 \end{bmatrix}$

24 ×

La scomposizione ha prodotto un notevole decremento di operazioni. Tutte le operazioni fatte sono reversibili.

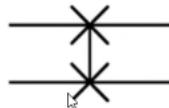
La computazione ha, quindi, due livelli:

- Livello orizzontale, le operazioni sono moltiplicazioni scalari tra matrici o vettori;
- Livello verticale, i vettori o matrici sono moltiplicati con operatori tensoriali.

4. Matrici Speciali

Matrice di SWAP. Lo swap gate S scambia due qubits ovvero $S|a, b\rangle = |b, a\rangle$:

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



le

$$e_{01} = \begin{bmatrix} 1 \\ 0 \end{bmatrix} \otimes \begin{bmatrix} 0 \\ 1 \end{bmatrix} = \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \end{bmatrix}$$

$$SWAP\ e_{01} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \end{bmatrix} = \begin{bmatrix} 0 \\ 1 \end{bmatrix} \otimes \begin{bmatrix} 1 \\ 0 \end{bmatrix} = e_{10}$$

Esempio:

Nota: rispetto alla computazione classica questa operazione non necessita di variabili di appoggio.

Pauli Gates:

Si tratta di 3 matrici 2×2 di tipo Hermitiane e unitarie. Tali matrici operano su un solo qubit. Le matrici di Pauli X,Y,Z si riferiscono alla rotazione attorno agli assi x,y,z rispettivamente.

$$X = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \quad Y = \begin{bmatrix} 0 & -i \\ i & 0 \end{bmatrix} \quad Z = \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix}$$

1. La prima matrice X di Pauli corrisponde esattamente ad un NOT, oppure si potrebbe pensare ad uno swap tra due componenti di un solo qubit. Inoltre opera su una sola linea di qubit. Esempio:

$$Xe_1 = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} 0 \\ 1 \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \end{bmatrix} = e_0$$

$$X|\psi\rangle = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \frac{1}{5} \begin{bmatrix} 3 \\ 4 \end{bmatrix} = \frac{1}{5} \begin{bmatrix} 4 \\ 3 \end{bmatrix}$$

2. Il secondo operatore Y opera sui numeri complessi, ma non ci servirà in questo corso.
3. Il terzo operatore Z, applica una fase negativa alla componente 1 del qubit. Esempio:

$$Ze_1 = \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix} \begin{bmatrix} 0 \\ 1 \end{bmatrix} = \begin{bmatrix} 0 \\ -1 \end{bmatrix} = -e_1$$

$$Z|\psi\rangle = \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix} \frac{1}{5} \begin{bmatrix} 3 \\ 4 \end{bmatrix} = \frac{1}{5} \begin{bmatrix} 3 \\ -4 \end{bmatrix}$$

Da un punto di vista geometrico, all'interno della sfera di raggio 1 del qubit, questa operazione non fa altro che riflettere rispetto all'asse x il vettore.

Matrice di Hadamard:

La matrice di Hadamard è una matrice unitaria quadrata dove tutte le componenti sono +1 o -1 e dove ciascuna riga è mutuamente ortogonale.

In termini geometrici questo significa che ogni paio di righe della matrice di Hadamard rappresentano due vettori perpendicolari tra loro.

$$\frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}$$

La matrice sottostante fa riferimento ad un solo qubit ($N = 2$), la matrice di Hadamard H_N di ordine N può essere definita ricorsivamente per qualsiasi valore di N che sia una potenza di 2:

$$H_1 = [1] \quad \text{And for } N \geq 2 \text{ we have}$$

$$H_N = H_{N/2} \otimes H = \frac{1}{\sqrt{2}} \begin{bmatrix} H_{N/2} & H_{N/2} \\ H_{N/2} & -H_{N/2} \end{bmatrix}$$

Dove $H_2 = H$ e per $N \geq 4$. Se vogliamo usare n e non N come marcitore, allora scriviamo $H^{\otimes n}$ usando un apice invece di un pedice:

$$H_4 = H^{\otimes 2} \quad H_8 = H^{\otimes 3} \quad H_{16} = H^{\otimes 4}$$

We set $H = H^{\otimes 1} = H_2$

$$H^{\otimes 0} = H_1 = [1]$$

$$H^{\otimes 1} = H_2 = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}$$

$$H^{\otimes 2} = H_4 = \frac{1}{2} \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & -1 & 1 & -1 \\ 1 & 1 & -1 & -1 \\ 1 & -1 & -1 & 1 \end{bmatrix}$$

$$H^{\otimes 3} = H_8 = \frac{1}{2\sqrt{2}} \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & -1 & 1 & -1 & 1 & -1 & 1 & -1 \\ 1 & 1 & -1 & -1 & 1 & 1 & -1 & -1 \\ 1 & -1 & -1 & 1 & 1 & -1 & -1 & 1 \\ 1 & 1 & 1 & 1 & -1 & -1 & -1 & -1 \\ 1 & -1 & 1 & -1 & -1 & 1 & -1 & 1 \\ 1 & 1 & -1 & -1 & -1 & -1 & 1 & 1 \\ 1 & -1 & -1 & 1 & -1 & 1 & 1 & -1 \end{bmatrix}$$

Questa rappresentazione ricorsiva implica alcuni fatti importanti. Ad esempio implica facilmente che, in generale, H_N è uguale a $\frac{1}{\sqrt{N}} A$, dove A è una matrice di soli ± 1 . Tuttavia, spesso è più utile avere la seguente definizione diretta delle componenti di H_N .

LEMMA: Per ogni riga r e colonna c , $H_N[r, c] = (-1)^{r \bullet c}$ dove ricordiamo $r \bullet c$ è l'inner product (prodotto interno) tra r e c trattati come stringhe booleane. Esempi:

$$H_2 = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}$$

$$H_2[0,0] = (-1)^{0 \bullet 0} = (-1)^0 = 1$$

$$H_2[1,0] = (-1)^{1 \bullet 0} = (-1)^0 = 1$$

$$H_2[0,1] = (-1)^{0 \bullet 1} = (-1)^0 = 1$$

$$H_2[1,1] = (-1)^{1 \bullet 1} = (-1)^1 = -1$$

impi

$$H_4 = \frac{1}{2} \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & -1 & 1 & -1 \\ 1 & 1 & -1 & -1 \\ 1 & -1 & -1 & 1 \end{bmatrix}$$

$$H_4[00,00] = (-1)^{00 \bullet 00} = (-1)^0 = 1$$

$$H_4[11,01] = (-1)^{11 \bullet 01} = (-1)^1 = -1$$

$$H_4[10,10] = (-1)^{10 \bullet 10} = (-1)^1 = -1$$

$$H_4[11,11] = (-1)^{11 \bullet 11} = (-1)^2 = 1$$

$$H_4[10,11] = (-1)^{10 \bullet 11} = (-1)^1 = -1$$

Quindi, per ogni vettore a , il vettore $b = H_N a$ è definito da:

$$\mathbf{b}(x) = \frac{1}{\sqrt{N}} \sum_{t=0}^{N-1} (-1)^{x \bullet t} \mathbf{a}(t).$$

Esempi:

ple

$$H_4 = \frac{1}{2} \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & -1 & 1 & -1 \\ 1 & 1 & -1 & -1 \\ 1 & -1 & -1 & 1 \end{bmatrix}$$

$$a = \frac{1}{4} \begin{bmatrix} 2 \\ 0 \\ \sqrt{3} \\ 3 \end{bmatrix}$$

$$b = H_4 a = \frac{1}{8} \begin{bmatrix} 5 + \sqrt{3} \\ \sqrt{3} - 1 \\ -1 - \sqrt{3} \\ 5 - \sqrt{3} \end{bmatrix}$$

$$b(00) = \frac{1}{2} \left((-1)^{00 \bullet 00} 2 + (-1)^{00 \bullet 01} 0 + (-1)^{00 \bullet 10} \sqrt{3} + (-1)^{00 \bullet 11} 3 \right) = 5 + \sqrt{3}$$

$$b(01) = \frac{1}{2} \left((-1)^{01 \bullet 00} 2 + (-1)^{01 \bullet 01} 0 + (-1)^{01 \bullet 10} \sqrt{3} + (-1)^{01 \bullet 11} 3 \right) = \sqrt{3} - 1$$

$$b(10) = \frac{1}{2} \left((-1)^{10 \bullet 00} 2 + (-1)^{10 \bullet 01} 0 + (-1)^{10 \bullet 10} \sqrt{3} + (-1)^{10 \bullet 11} 3 \right) = -1 - \sqrt{3}$$

$$b(11) = \frac{1}{2} \left((-1)^{11 \bullet 00} 2 + (-1)^{11 \bullet 01} 0 + (-1)^{11 \bullet 10} \sqrt{3} + (-1)^{11 \bullet 11} 3 \right) = 5 - \sqrt{3}$$

La matrice di Hadamard permette di fare una superimposizione:

$$H_4 = \frac{1}{2} \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & -1 & 1 & -1 \\ 1 & 1 & -1 & -1 \\ 1 & -1 & -1 & 1 \end{bmatrix} \quad a = \frac{1}{2} \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \end{bmatrix} \quad b = H_4 a = \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

$$b(00) = \frac{1}{2}((-1)^{00*00}1 + (-1)^{00*01}1 + (-1)^{00*10}1 + (-1)^{00*11}1) = \frac{1}{2}2 = 1$$

$$b(01) = \frac{1}{2}((-1)^{01*00}1 + (-1)^{01*01}1 + (-1)^{01*10}1 + (-1)^{01*11}1) = 0$$

$$b(10) = \frac{1}{2}((-1)^{10*00}1 + (-1)^{10*01}1 + (-1)^{10*10}1 + (-1)^{10*11}1) = 0$$

$$b(11) = \frac{1}{2}((-1)^{11*00}1 + (-1)^{11*01}1 + (-1)^{11*10}1 + (-1)^{11*11}1) = 0$$

Ci sono due vettori molto importanti in computazione quantistica. Applicando a 0 la matrice di Hadamard otteniamo +, applicando ad 1 al matrice di Hadamard otteniamo -.

Un'applicazione della porta di Hadamard a 0 o 1 qubit produrrà uno stato quantistico che, se osservato, sarà uno 0 o 1 con uguale probabilità. Questo è esattamente come lanciare una moneta equa nel modello probabilistico standard di calcolo.

$\frac{ 0\rangle + 1\rangle}{\sqrt{2}} = \frac{1}{\sqrt{2}} \left(\begin{bmatrix} 1 \\ 0 \end{bmatrix} + \begin{bmatrix} 0 \\ 1 \end{bmatrix} \right) = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 \\ 1 \end{bmatrix} \quad +\rangle$ $\frac{ 0\rangle - 1\rangle}{\sqrt{2}} = \frac{1}{\sqrt{2}} \left(\begin{bmatrix} 1 \\ 0 \end{bmatrix} - \begin{bmatrix} 0 \\ 1 \end{bmatrix} \right) = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 \\ -1 \end{bmatrix} \quad -\rangle$		$H 0\rangle = He_0 = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \begin{bmatrix} 1 \\ 0 \end{bmatrix} = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 \\ 1 \end{bmatrix} = +\rangle$ $H 1\rangle = He_1 = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \begin{bmatrix} 0 \\ 1 \end{bmatrix} = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 \\ -1 \end{bmatrix} = -\rangle$ oppure: $ 0\rangle \xrightarrow{\boxed{H}} +\rangle$ $ 1\rangle \xrightarrow{\boxed{H}} -\rangle$
---	--	--

Tuttavia, se la porta di Hadamard viene applicata due volte di seguito, lo stato finale è sempre lo stesso dello stato iniziale:

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

$$|0\rangle \xrightarrow{\boxed{H}} \boxed{H} \xrightarrow{\boxed{H}} |0\rangle$$

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

$$|1\rangle \xrightarrow{\boxed{H}} \boxed{H} \xrightarrow{\boxed{H}} |1\rangle$$

Implementazione della Matrice di Hadamard in python:

```
H = 1/np.sqrt(2)*np.array([[1,1],[1,-1]])

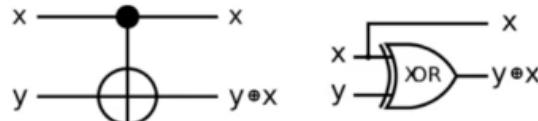
def Had(n):
    if(n==1):
        return H
    return np.kron(H,Had(n-1))
```

Toffoli Gate:

Un aspetto molto importante durante la computazione quantistica è il concetto di **reversibilità**, ovvero la possibilità, dopo aver applicato una funzione, di tornare indietro (undo).

Però non tutti i circuiti sono reversibili, come lo XOR, perché se otteniamo 0 non sappiamo se è stato ottenuto da input pari a 00 o 11. In questo caso diremo che gli input sono stati **consumati**.

La reversibilità sta nel fatto che nessun input venga consumato, stesso numero di variabili di input e output. Nell'esempio di prima per realizzare uno XOR reversibile si crea una linea che mantenga una linea di input. Realizzando il controlled NOT (CNOT):

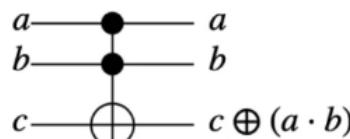


input	output	input	output
x	y	x	y+x
0>	0>	0>	0>
0>	1>	0>	1>
1>	0>	1>	1>
1>	1>	1>	0>

input	output	input	output
x	y	x	y+x
0	0	0	0
0	1	0	1
1	0	1	1
1	1	1	0

Gate di Toffoli come generalizzazione del CNOT:

La variante a 3 input del CNOT gate è chiamata Toffoli gate. Si mantiene inalterati gli input a, b che fungono da controlli e si sostituisce la terza linea (il target) c con $c \oplus (a \cdot b)$.



Se a, b sono entrambi ad 1 viene applicato il NOT a c .

Se uno tra a, b è 0 allora il controllo non va a buon fine e c passa invariato.

L'unico modo per applicare il NOT a c è che $(a \cdot b) = 1$ e che quindi $a, b = 1$, in tutti gli altri casi $(a \cdot b) = 0$ e c rimane invariato.

Sostanzialmente il Toffoli gate è un doppio *if*. Di seguito la matrice che la rappresenta:

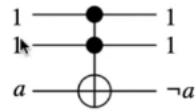
	000	001	010	011	100	101	110	111
000	1							
001		1						
010			1					
011				1				
100					1			
101						1		
110							1	
111								1

Gate Universale:

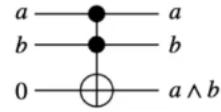
Sapendo come fare il gate NAND possiamo rappresentare ogni altro gate usando una combinazione di gate NAND.

The Toffoli gate is a universal reversible gate

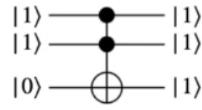
$$NOT(a) = T(1,1,a)$$



$$AND(a, b) = T(a, b, 0)$$



Esempio di applicazione del Toffoli gate:



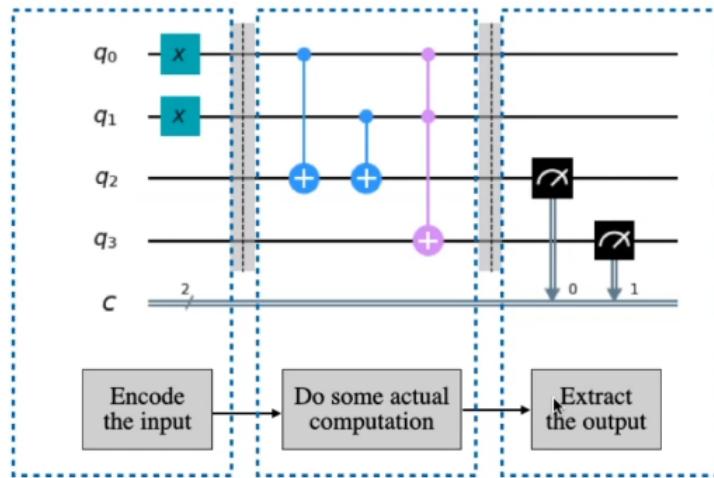
$$T(|1\rangle|1\rangle|0\rangle) = T|110\rangle = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 1 \\ 0 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 1 \end{bmatrix} = |1\rangle|1\rangle|1\rangle$$

5. Circuiti Quantistici

Un circuito quantistico è formato da una prima parte caratterizzata dalla codifica dell'input specifico che si vuole calcolare.

Successivamente c'è una parte centrale caratterizzata dalla computazione effettiva.

Infine, l'ultima parte, è adibita alla misurazione ovvero all'estrazione dell'output dal circuito quantistico (o algoritmo quantistico).



Un circuito viene generalmente eseguito più volte: La ragione per la quale viene eseguito molte volte è mostrato in un istogramma è perché i computer quantistici possono avere un po' di casualità nei loro risultati.

Quantum Simulator: Il risultato può essere ottenuto da un simulatore quantistico, ovvero su una macchina classica ma su un numero basso di qubit (circa 30).

Gate di base:

Not Gate X	Controlled Not Gate CNOT - CX	Controlled Not Gate Reversed	Controlled CNOT Gate CCNOT - CCX	Controlled CNOT Gate Reversed	Controlled CNOT Gate Mixed
$\begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}$	$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix}$	$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix}$	$\begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}$	$\begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}$	
$ 0\rangle \rightarrow 1\rangle$ $ 1\rangle \rightarrow 0\rangle$	$ 00\rangle \rightarrow 00\rangle$ $ 01\rangle \rightarrow 01\rangle$ $ 10\rangle \rightarrow 11\rangle$ $ 11\rangle \rightarrow 10\rangle$	$ 00\rangle \rightarrow 00\rangle$ $ 01\rangle \rightarrow 11\rangle$ $ 10\rangle \rightarrow 10\rangle$ $ 11\rangle \rightarrow 01\rangle$	$ 000\rangle \rightarrow 000\rangle$ $ 101\rangle \rightarrow 101\rangle$ $ 111\rangle \rightarrow 110\rangle$ $ 110\rangle \rightarrow 111\rangle$	$ 000\rangle \rightarrow 000\rangle$ $ 101\rangle \rightarrow 101\rangle$ $ 111\rangle \rightarrow 011\rangle$ $ 011\rangle \rightarrow 111\rangle$	$ 000\rangle \rightarrow 000\rangle$ $ 101\rangle \rightarrow 111\rangle$ $ 111\rangle \rightarrow 111\rangle$ $ 111\rangle \rightarrow 101\rangle$

Nel caso ci fosse una linea in mezzo?

CX3

	$ 000\rangle \rightarrow 000\rangle$ $ 001\rangle \rightarrow 001\rangle$ $ 010\rangle \rightarrow 010\rangle$ $ 011\rangle \rightarrow 011\rangle$ $ 100\rangle \rightarrow 101\rangle$ $ 101\rangle \rightarrow 100\rangle$ $ 110\rangle \rightarrow 111\rangle$ $ 111\rangle \rightarrow 110\rangle$
$\begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \end{bmatrix}$	

Multi-Controlled-CX:

```
def mcx(n):
    N = 2**n
    M = Idt(n)
    M[-2],M[-1] = M[-1].copy(),M[-2].copy()
    return M
```

General-CX e General-CCX:

Prende in input un qubit di controllo, un qubit target e il numero di qubit coinvolti. Mentre il G-CCX prende due controlli come input.

```
def gcx(c,t,n):
    mat = idt(n)
    formatstr = "{:0>"+str(n)+"b}"
    for q in range(2**n):
        v = formatstr.format(q)
        if(v[c]=="0" and v[t]=="0"):
            b = [c for c in v]
            b[c] = "1"
            control = int(''.join(b), 2)
            b[t] = "1"
            target = int(''.join(b), 2)
            trow = mat[target].copy()
            crow = mat[control].copy()
            mat[control] = trow
            mat[target] = crow
    return mat
```

```
def gccx(c1,c2,t,n):
    mat = idt(n)
    formatstr = "{:0>"+str(n)+"b}"
    for q in range(2**n):
        v = formatstr.format(q)
        if(v[c1]=="0" and v[c2]=="0" and v[t]=="0"):
            b = [c for c in v]
            b[c1] = "1"
            b[c2] = "1"
            control = int(''.join(b), 2)
            b[t] = "1"
            target = int(''.join(b), 2)
            trow = mat[target].copy()
            crow = mat[control].copy()
            mat[control] = trow
            mat[target] = crow
    return mat
```

Simulazione dello Z gate di pauli:

$$Z = \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix}$$

Possiamo simulare lo z gate con:



$$HXH = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} = \frac{1}{2} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \begin{bmatrix} 1 & -1 \\ 1 & 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix} = Z$$

Inoltre possiamo simulare il controlled z gate:



$$(I \otimes H)(CNOT)(I \otimes H) = \frac{1}{2} \begin{bmatrix} 1 & 1 & 0 & 0 \\ 1 & -1 & 0 & 0 \\ 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & -1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} 1 & 1 & 0 & 0 \\ 1 & -1 & 0 & 0 \\ 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & -1 \end{bmatrix}$$

$$= \frac{1}{2} \begin{bmatrix} 1 & 1 & 0 & 0 \\ 1 & -1 & 0 & 0 \\ 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & -1 \end{bmatrix} \begin{bmatrix} 1 & 1 & 0 & 0 \\ 1 & -1 & 0 & 0 \\ 0 & 0 & 1 & -1 \\ 0 & 0 & 1 & 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & -1 \end{bmatrix} = CZ$$

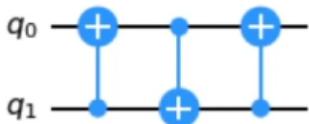
Si applica hadamard al qubit q_1 successivamente il CNOT e di nuovo hadamard. Se il CNOT non ha nessun effetto annulla l'effetto del gate H precedente, quindi se il CNOT ha effetto si ottiene lo Z altrimenti non succede nulla.

Quest'ultima opzione risulta più ottimizzata rispetto alla prima poiché si applica hadamard solo all'ultimo qubit e non a tutti.

Swapping di due qubit:

Lo SWAP può essere applicato utilizzando questa serie di operatori (tre controlled-not):

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

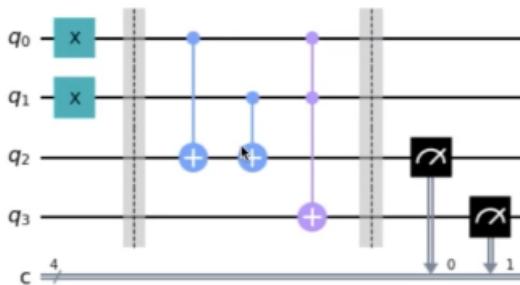


$|00\rangle \rightarrow |00\rangle \rightarrow |00\rangle \rightarrow |00\rangle$
 $|01\rangle \rightarrow |11\rangle \rightarrow |10\rangle \rightarrow |10\rangle$
 $|10\rangle \rightarrow |10\rangle \rightarrow |11\rangle \rightarrow |01\rangle$
 $|11\rangle \rightarrow |01\rangle \rightarrow |01\rangle \rightarrow |11\rangle$

```
def gSwap(r1,r2,n):
    CX1 = gcx(r2,r1,n)
    CX2 = gcx(r1,r2,n)
    CX3 = gcx(r2,r1,n)
    M = np.matmul(CX1,CX2)
    M = np.matmul(M,CX3)
    return M
```

Lo swap serve a spostare i qubit di posizione, nella computazione quantistica a volte serve fare operazioni tra due qubit che potrebbero trovarsi lontani, lo swap aiuta avvicinando i qubit per agevolare i calcoli tra essi.

Half Adder:



$$\begin{aligned} a_0 &= |0000\rangle \\ a_1 &= (X^{\otimes 2} \otimes I^{\otimes 2}) a_0 \\ a_2 &= (CX_3 \otimes I) a_1 \\ a_3 &= (I \otimes CX \otimes I) a_2 \\ a_4 &= CCX_4 a_3 \end{aligned}$$

I due input sono caratterizzati da q_0, q_1 , mentre le altre due linee di qubit q_2, q_3 rappresentano l'output.

Quest'ultimi necessari in quanto vogliamo mantenere la reversibilità. La misurazione viene fatta su l'output. L'ultima riga c sono delle linee di bit classici, dove viene riversato l'output della misurazione. Nell'immagine la X degli input significa che li stiamo negando, ciò perché vogliamo che gli input siano 11.

Analisi dei casi possibili:

1. caso 00: gli input q_1, q_2 poiché impostati a 0 non attivano in CNOT e quindi non alterano i valori di q_2, q_3 . Rimane tutto inalterato e la misurazione rileva 00;
2. caso 01: in questo caso q_0 è uguale ad 1 di conseguenza in CNOT collegato alla linea q_2 diventa 1. Il secondo CNOT non viene attivato poiché q_1 impostato a 0. Il CCNOT (o toffoli gate) viene attivato solo se entrambi gli input sono ad 1, quindi q_3 è 0. La somma tra 01 rimane 01;
3. caso 10: In maniera speculare si al passo precedente la somma tra 1 e 0 fa 10;
4. caso 11: In questo caso entrambi gli input sono 1 di conseguenza il toffoli gate viene attivato producendo nella linea del bit più significativo q_3 un 1. I due CNOT applicati alla linea q_2 producono una doppia variazione da 0 a 1 e subito dopo da 1 a 0. La somma tra 1+1 dà appunto 2 in binario 10.

Codice Half-Adder in algebra lineare:

```

def halfAdder(b0,b1):
    inp0 = get_state(str(b0))
    inp1 = get_state(str(b1))
    out0 = get_state("0")
    out1 = get_state("0")

    a1 = tp([inp0, inp1, out0, out1])
    a2 = np.matmul(gcx(0,2,4),a1)
    a3 = np.matmul(gcx(1,2,4),a2)
    a4 = np.matmul(gcx(0,1,3,4),a3)
    counts(a4)

# misurazione di un vettore
def counts(vec):
    print("COUNTS:")
    N = len(vec)
    n = int(np.log2(N))
    formatstr = "{:0>"+str(n)+"b}"
    for i in range(N):
        if(vec[i]>0):
            print(" -",
                  formatstr.format(i),
                  str(np.around((vec[i]**2)*100,decimals=2))+"%")

```

Implementazione in qiskit:

Creazione quantum circuit:

```
circuit = QuantumCircuit(4,2)
```

Dove il primo argomento identifica il numero di qubit nel sistema, mentre il secondo argomento identifica il numero di linee di bit standard.

```

simulator = QasmSimulator()
compiled_circuit = transpile(circuit, simulator)
job = simulator.run(compiled_circuit, shots=1000)
result = job.result()
qcl.draw_circuit(circuit)

```

Nella prima riga si crea un simulatore, successivamente nella riga 2 si compila il circuito, si esegue a questo punto, riga 3, il circuito nel simulatore creato prima indicando il numero di cicli di esecuzione tramite il parametro shots. Catturiamo il risultato nella riga 4 e disegniamo il circuito, riga 5.

Half e Full Adder in qiskit:

```

inp = QuantumRegister(2,"input")
out = QuantumRegister(2, "output")
cr = ClassicalRegister(2, "c")
circuit = QuantumCircuit(inp,out,cr)

circuit.x(inp[0])
circuit.x(inp[1])

circuit.barrier()

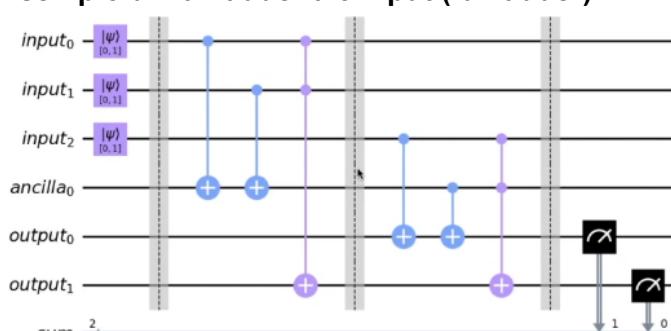
circuit.cx(inp[0],out[0])
circuit.cx(inp[1],out[0])
circuit.ccx(inp[0],inp[1],out[1])

circuit.barrier()

circuit.measure(out[0],0)
circuit.measure(out[1],1)

```

Esempio di half adder a 3 input (full adder):



```

halfAdder = QuantumCircuit(4)
halfAdder.cx(0,2)
halfAdder.cx(1,2)
halfAdder.ccx(0,1,3)

```

```
inp = QuantumRegister(3,"input")
out = QuantumRegister(2,"output")
anc = QuantumRegister(1, "ancilla")
cr = ClassicalRegister(2,"c")
circuit = QuantumCircuit(inp,anc,out,cr)

init0 = [1,0]
init1 = [0,1]
circuit.initialize(init0,0)
circuit.initialize(init0,1)
circuit.initialize(init1,2)
#circuit.x(0)
#circuit.x(1)
circuit.barrier()

circuit = circuit.compose(halfAdder,[0,1,anc[0],out[1]])
circuit.barrier()
circuit = circuit.compose(halfAdder,[inp[2],anc[0],out[0],out[1]])
circuit.barrier()

circuit.measure(4,0)
circuit.measure(5,1)
```

6. Algoritmi di base

Lancio della moneta (testa o croce):

Per ottenere un algoritmo quantistico che svolga questo compito è sufficiente inizializzare un qubit a cui si applica la matrice di Hadamard. Successivamente si misura.

$$1. a_0 = \begin{bmatrix} 1 \\ 0 \end{bmatrix}$$

$$2. a_1 = H a_0$$

$$3. \text{measure } a_1$$

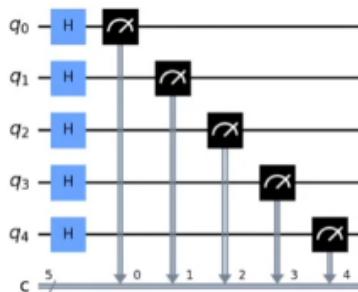
```
def coinflip():
    circuit = QuantumCircuit(1,1)
    circuit.h(0)
    circuit.measure(0,0)
    simulator = QasmSimulator()
    compiled_circuit = transpile(circuit, simulator)
    job = simulator.run(compiled_circuit, shots=1)
    result = job.result()
    qcl.draw_circuit(circuit)
    counts = result.get_counts(compiled_circuit)
    return list(counts.keys())[0]
```

Generazione di un numero random:

L'algoritmo prende un valore n in input e restituisce un numero random compreso tra 0 e $2^n - 1$.

Si tratta di una generalizzazione dell'esercizio precedente, n sono il numero di qubit a cui si applica la matrice di hadamard a ciascuno di essi.

Esempio $n = 5$:



```
def randint(n):
    circuit = QuantumCircuit(n,n)
    for i in range(n):
        circuit.h(i)
    for i in range(n):
        circuit.measure(i,i)
    simulator = QasmSimulator()
    compiled_circuit = transpile(circuit, simulator)
    job = simulator.run(compiled_circuit, shots=1)
    result = job.result()
    qcl.draw_circuit(circuit)
    counts = result.get_counts(compiled_circuit)
    return int(list(counts.keys())[0],2)
```

Nb: Importante l'ultima riga del codice in quanto una volta ottenuta la stringa binaria essa viene convertita in intero tramite il comando `int(*valore, *base_di_partenza)`.

Generazione una coppia di qubit correlati:

Si tratta di una coppia di qubit che quando vengono misurati, anche separatamente, producono lo stesso risultato, ovvero collassano entrambi nello stesso stato.

La correlazione sta nel fatto che non è possibile rappresentare il vettore in un tensor product tra più vettori, risultando inseparabile.

$$a_0 = e_{00} = \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}$$

1. Si parte dal vettore e_{00} , poiché si tratta di una coppia di qubit a_0 è composto da 4 elementi;
2. Il successivo vettore a_1 è il risultato dell'applicazione di Hadamard al solo primo qubit;
3. Il successivo vettore a_2 è il risultato dell'applicazione di un CNOT con controllo nel primo qubit e target nel secondo qubit. Quindi una misurazione darà come output 00 oppure 11. Grazie al CNOT si ottiene la correlazione vera e propria.

Esempio:

$$e_{00} \rightarrow (He_0) \otimes e_0 = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \begin{bmatrix} 1 \\ 0 \end{bmatrix} \otimes \begin{bmatrix} 1 \\ 0 \end{bmatrix} = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 \\ 1 \end{bmatrix} \otimes \begin{bmatrix} 1 \\ 0 \end{bmatrix} = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 \\ 0 \\ 1 \\ 0 \end{bmatrix} = \frac{1}{\sqrt{2}} (e_{00} + e_{10})$$

$$\frac{1}{\sqrt{2}} (e_{00} + e_{10}) \rightarrow \frac{1}{\sqrt{2}} \text{CNOT}(e_{00} + e_{10}) = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} 1 \\ 0 \\ 1 \\ 0 \end{bmatrix} = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 \\ 0 \\ 0 \\ 1 \end{bmatrix} = \frac{1}{\sqrt{2}} (e_{00} + e_{11})$$

Stati di Bell:

Dalla rappresentazione precedente, indipendentemente dall'input che si ha, si ottengono due qubit correlati. In particolare si ottengono le seguenti 4 configurazioni, chiamate stati di Bell.

Gli stati Bell sono quattro specifici stati quantistici di due qubit con la massima correlazione. Sono in una sovrapposizione di 0 e 1 cioè una combinazione lineare dei due stati. La loro correlazione significa quanto segue:

Il qubit posseduto da Alice (pedice "A") può essere 0 e 1. Se Alice misurasse il suo qubit in una base standard, il risultato sarebbe perfettamente casuale, 0 o 1 entrambi con probabilità 1/2. Ma se Bob (pedice "B") misurasse il suo qubit, il risultato sarebbe lo stesso di quello ottenuto da Alice. Quindi, se Bob misurasse il suo qubit, otterrebbe un risultato casuale a prima vista, ma se Alice e Bob comunicano, scoprirebbero che sebbene i loro risultati sembrino casuali, sono perfettamente correlati.

Grazie a questi stati è possibile realizzare il super dense coding e il quantum teleportation (vedi lezioni successive).

$$\begin{aligned} |\beta_{00}\rangle &= \frac{|00\rangle + |11\rangle}{\sqrt{2}}; \\ |\beta_{01}\rangle &= \frac{|01\rangle + |10\rangle}{\sqrt{2}}; \\ |\beta_{10}\rangle &= \frac{|00\rangle - |11\rangle}{\sqrt{2}}; \text{ and} \\ |\beta_{11}\rangle &= \frac{|01\rangle - |10\rangle}{\sqrt{2}}, \end{aligned}$$

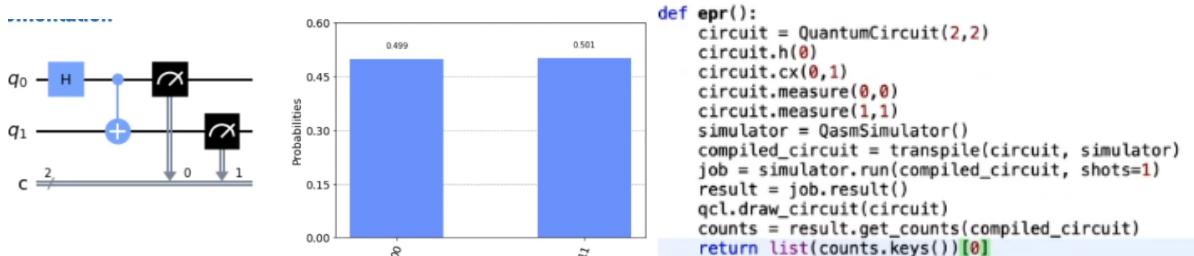
In	Out
$ 00\rangle$	$(00\rangle + 11\rangle)/\sqrt{2} \equiv \beta_{00}\rangle$
$ 01\rangle$	$(01\rangle + 10\rangle)/\sqrt{2} \equiv \beta_{01}\rangle$
$ 10\rangle$	$(00\rangle - 11\rangle)/\sqrt{2} \equiv \beta_{10}\rangle$
$ 11\rangle$	$(01\rangle - 10\rangle)/\sqrt{2} \equiv \beta_{11}\rangle$

Per i quattro ingressi di base a due qubit, $|00\rangle, |01\rangle, |10\rangle, |11\rangle$, il circuito emette uno stato Bell finale secondo l'equazione:

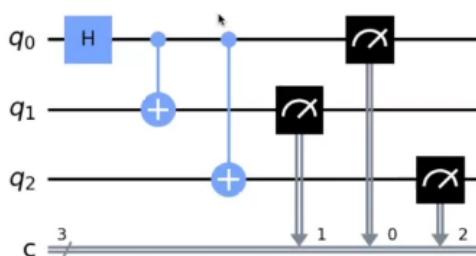
$$|\beta(x, y)\rangle = \left(\frac{|0, y\rangle + (-1)^x |1, Y\rangle}{\sqrt{2}} \right)$$

dove Y è la negazione di y .^[1]

Codice:



Applicato a 3 input si ha lo stato GHZ:



```
def ghz(b1,b2,b3):
    circuit = QuantumCircuit(3,3)
    if b1==1:
        circuit.x(0)
    if b2==1:
        circuit.x(1)
    if b3==1:
        circuit.x(2)
    circuit.barrier()
    circuit.h(0)
    circuit.cx(0,1)
    circuit.cx(0,2)
    circuit.measure(0,0)
    circuit.measure(1,1)
    circuit.measure(2,2)
    simulator = QasmSimulator()
    compiled_circuit = transpile(circuit, simulator)
    job = simulator.run(compiled_circuit, shots=1)
    result = job.result()
    qcl.draw_circuit(circuit)
    counts = result.get_counts(compiled_circuit)
    return list(counts.keys())[0]
```

Generalizzazione ad n input:

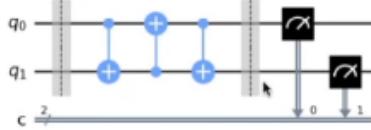
Prende in input due valori:

1. n identifica il numero di qubit che si vuole correlare;
2. una lista l che identifica le inizializzazioni dei qubit.

```
def gepr(n,l):
    circuit = QuantumCircuit(n,n)
    for i in range(n):
        if l[i]==1:
            circuit.x(i)
    circuit.barrier()
    circuit.h(0)
    for i in range(1,n):
        circuit.cx(0,i)
    circuit.barrier()
    for i in range(n):
        circuit.measure(i,i)
    simulator = QasmSimulator()
    compiled_circuit = transpile(circuit, simulator)
    job = simulator.run(compiled_circuit, shots=1)
    result = job.result()
    qcl.draw_circuit(circuit)
    counts = result.get_counts(compiled_circuit)
    return list(counts.keys())[0]
```

Swap di qubit:

```
def swap(a,b):
    circuit = QuantumCircuit(2,2)
    if a==1:
        circuit.x(0)
    if b==1:
        circuit.x(1)
    circuit.barrier()
    circuit.cx(0,1)
    circuit.cx(1,0)
    circuit.cx(0,1)
    circuit.barrier()
    circuit.measure(0,1)
    circuit.measure(1,0)
    simulator = QasmSimulator()
    compiled_circuit = transpile(circuit, simulator)
    job = simulator.run(compiled_circuit, shots=1)
    result = job.result()
    qcl.draw_circuit(circuit)
    counts = result.get_counts(compiled_circuit)
    return list(counts.keys())[0]
```



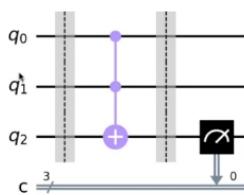
And, Or e Xor e generalizzazioni:

And:

Per simulare l'And si utilizza, come abbiamo visto nei capitoli precedenti, il toffoli gate. Il codice è rappresentato mediante un gate.

```
def andgate():
    circuit = QuantumCircuit(3)
    circuit.ccx(0,1,2)
    g = circuit.to_gate()
    g.name = "AND"
    return g

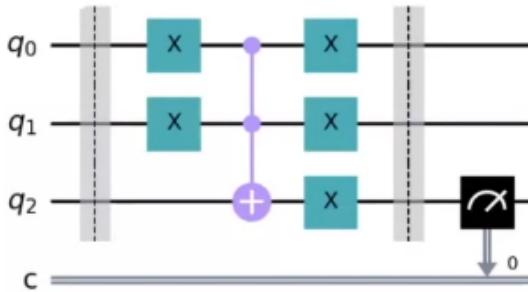
def andc(b1,b2):
    inp = QuantumRegister(2,"input")
    out = QuantumRegister(1, "output")
    cr = ClassicalRegister(1,"cr")
    circuit = QuantumCircuit(inp,out,cr)
    if b1==1:
        circuit.x(0)
    if b2==1:
        circuit.x(1)
    circuit.barrier()
    circuit.compose(andgate(),[0,1,2])
    circuit.barrier()
    circuit.measure(2,0)
    simulator = QasmSimulator()
    compiled_circuit = transpile(circuit, simulator)
    job = simulator.run(compiled_circuit, shots=1)
    result = job.result()
    qcl.draw_circuit(circuit)
    counts = result.get_counts(compiled_circuit)
    return list(counts.keys())[0]
```



Or: è il duale dell'AND; ecco il motivo dei gate X prima e dopo.

```
def orc(b1,b2):
    inp = QuantumRegister(2,"input")
    out = QuantumRegister(1, "output")
    cr = ClassicalRegister(1,"cr")
    circuit = QuantumCircuit(inp,out,cr)
    if b1==1:
        circuit.x(0)
    if b2==1:
        circuit.x(1)
    circuit.barrier()
    circuit.x(0)
    circuit.x(1)

    #circuit.cx(0,2)
    #circuit.cx(1,2)
    circuit.ccx(0,1,2)
    circuit.x(2)
    circuit.x(0)
    circuit.x(1)
    circuit.barrier()
    circuit.measure(2,0)
simulator = QasmSimulator()
compiled_circuit = transpile(circuit, simulator)
job = simulator.run(compiled_circuit, shots=1)
result = job.result()
qcl.draw_circuit(circuit)
counts = result.get_counts(compiled_circuit)
return list(counts.keys())[0]
```

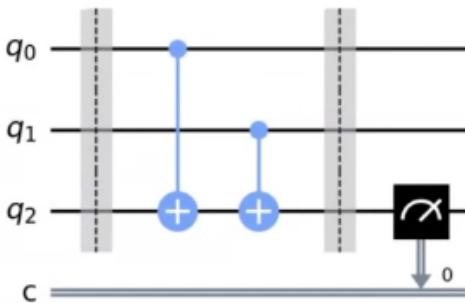


Xor:

```
def xor(b1,b2):
    inp = QuantumRegister(2,"input")
    out = QuantumRegister(1, "output")
    cr = ClassicalRegister(1,"cr")
    circuit = QuantumCircuit(inp,out,cr)
    if b1==1:
        circuit.x(0)
    if b2==1:
        circuit.x(1)
    circuit.barrier()

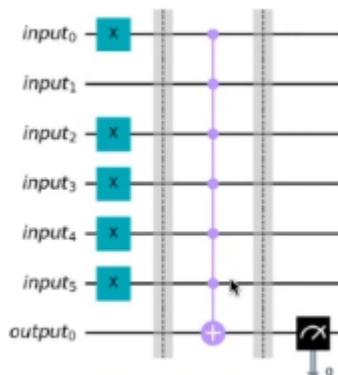
    circuit.cx(0,2)
    circuit.cx(1,2)

    circuit.barrier()
    circuit.measure(2,0)
simulator = QasmSimulator()
compiled_circuit = transpile(circuit, simulator)
job = simulator.run(compiled_circuit, shots=1)
result = job.result()
```



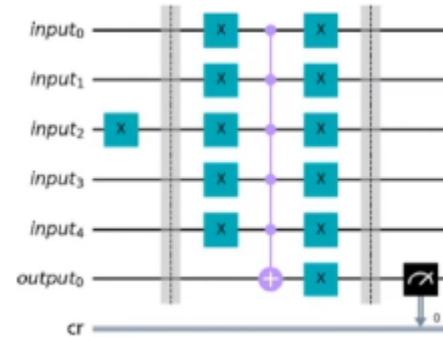
And generalizzato:

```
def gandc(blist):
    n = len(blist)
    inp = QuantumRegister(n,"input")
    out = QuantumRegister(1, "output")
    cr = ClassicalRegister(1,"cr")
    circuit = QuantumCircuit(inp,out,cr)
    for i in range(n):
        if blist[i]==1:
            circuit.x(i)
    circuit.barrier()
    circuit.mcx(list(range(n)),n)
    circuit.barrier()
    circuit.measure(n,0)
simulator = QasmSimulator()
compiled_circuit = transpile(circuit, simulator)
job = simulator.run(compiled_circuit, shots=1)
result = job.result()
qcl.draw_circuit(circuit)
counts = result.get_counts(compiled_circuit)
return list(counts.keys())[0]
```



Or generalizzato:

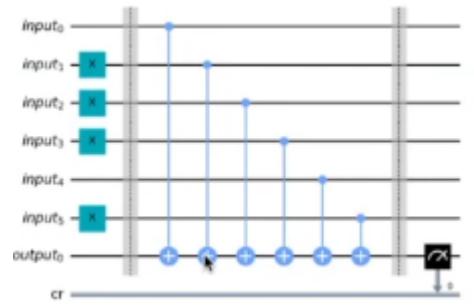
```
def gorc(blist):
    n = len(blist)
    inp = QuantumRegister(n, "input")
    out = QuantumRegister(1, "output")
    cr = ClassicalRegister(1, "cr")
    circuit = QuantumCircuit(inp,out,cr)
    for i in range(n):
        if blist[i]==1:
            circuit.x(i)
    circuit.barrier()
    for i in range(n):
        circuit.x(i)
    circuit.mcx(list(range(n)),n)
    for i in range(n+1):
        circuit.x(i)
    circuit.barrier()
    circuit.measure(n,0)
    simulator = QasmSimulator()
    compiled_circuit = transpile(circuit, simulator)
    job = simulator.run(compiled_circuit, shots=1)
    result = job.result()
    qcl.draw_circuit(circuit)
    counts = result.get_counts(compiled_circuit)
    return list(counts.keys())[0]
```



Xor generalizzato:

Le coppie di CNOT si annullano reciprocamente. Se il numero di NOT è dispari il totale di NOT equivale ad un solo NOT.

```
def gxorc(blist):
    n = len(blist)
    inp = QuantumRegister(n, "input")
    out = QuantumRegister(1, "output")
    cr = ClassicalRegister(1, "cr")
    circuit = QuantumCircuit(inp,out,cr)
    for i in range(n):
        if blist[i]==1:
            circuit.x(i)
    circuit.barrier()
    for i in range(n):
        circuit.cx(i,n)
    circuit.barrier()
    circuit.measure(n,0)
    simulator = QasmSimulator()
    compiled_circuit = transpile(circuit, simulator)
    job = simulator.run(compiled_circuit, shots=1)
    result = job.result()
    qcl.draw_circuit(circuit)
    counts = result.get_counts(compiled_circuit)
    return list(counts.keys())[0]
```



Prodotto interno booleano:

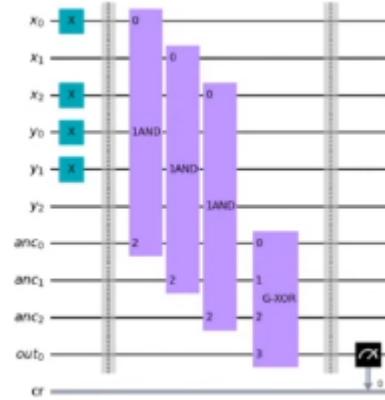
Ricordiamo come opera il prodotto interno booleano mediante il seguente esempio:

$$\begin{aligned} 100110 \cdot 110010 &= \\ &= (1 \cdot 1) \oplus (0 \cdot 1) \oplus (0 \cdot 0) \oplus (1 \cdot 0) \oplus (1 \cdot 1) \oplus (0 \cdot 0) = \\ &= 1 \oplus 0 \oplus 0 \oplus 0 \oplus 1 \oplus 0 = 0 \end{aligned}$$

```

def innerp(x,y):
    n = len(x)
    if(len(y)!=n):
        return "Errore nella lunghezza delle stringhe"
    xr = QuantumRegister(n,"x")
    yr = QuantumRegister(n,"y")
    anc = QuantumRegister(n,"anc")
    out = QuantumRegister(1,"out")
    cr = ClassicalRegister(1,"cr")
    circuit = QuantumCircuit(xr,yr,anc,out,cr)
    for i in range(n):
        if(x[i]=="1"):
            circuit.x(xr[i])
    for i in range(n):
        if(y[i]=="1"):
            circuit.x(yr[i])
    circuit.barrier()
    for i in range(n):
#        circuit.ccx(i,i+n,i+2*n)
        circuit = circuit.compose(andgate(),[i,i+n,i+2*n])
    circuit = circuit.compose(gxorgate(n),list(range(2*n,3*n+1)))
    circuit.barrier()
    circuit.measure(3*n,0)
    simulator = QasmSimulator()
    compiled_circuit = transpile(circuit, simulator)
    job = simulator.run(compiled_circuit, shots=1)
    result = job.result()
    qcl.draw_circuit(circuit)
    counts = result.get_counts(compiled_circuit)

```



7. Quantum oracles

Rappresenta l'implementazione quantistica di una funzione. Viene definita oracle poiché è simile ad una black box. Ci danno inoltre informazioni sull'output della funzione senza necessariamente dover calcolare l'informazione per tutti gli input.



Boolean oracles:

Sono degli operatori (matrici) in grado di computare U_f .

Si applicano questi operatori ad un vettore, in cui la prima parte del vettore è x , gli input, e la seconda parte del vettore viene inizializzata a 0.

$$U_f |x, \bar{0}\rangle = |x, f(x)\rangle$$

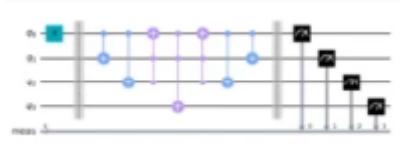
La barra sopra lo zero indica che la dimensione del numero dei qubit per la rappresentazione è abbastanza grande da contenere l'output.

Il risultato di questo operatore è il vettore di input x inalterato (questo per mantenere il concetto di reversibilità [2]) ed una $f(x)$. Esempio [1]:

$$\begin{aligned}
 &f: \{0,1\} \leftarrow \{0,1\} & U_f |10\rangle = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \end{bmatrix} = |10\rangle \\
 &f(x) = \neg x & \\
 &U_f = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} & U_f |00\rangle = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \end{bmatrix} = \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \end{bmatrix} = |01\rangle \\
 &[1] & U_f |x, \bar{0}\rangle = |x, f(x)\rangle \\
 & & U_f^T |x, f(x)\rangle = |x, \bar{0}\rangle
 \end{aligned}$$

$$\begin{aligned}
 &f: \{0,1\} \leftarrow \{0,1\} & U_f^T |10\rangle = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \end{bmatrix} = |10\rangle \\
 &f(x) = \neg x & \\
 &U_f^T = U_f = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} & U_f^T |01\rangle = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \end{bmatrix} = |00\rangle \\
 &[2] &
 \end{aligned}$$

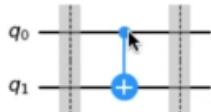
Per poter invertire la computazione e quindi tornare indietro (ancompute) bisogna ripetere le operazioni fatte invertendo l'ordine degli operatori:



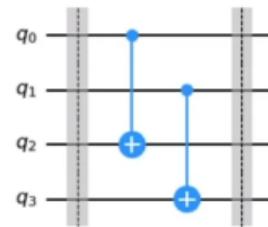
Creazione dei Quantum Oracle:

Un esempio è la **funzione identità** per copiare l'input nell'output. Se si esegue un CNOT per ogni linea di input nella corrispondente linea dell'output ottengo appunto la copia.

$$f(x) = x, \forall x \in \{0,1\}$$



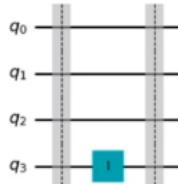
$$f(x) = x, \forall x \in \{0,1\}^2$$



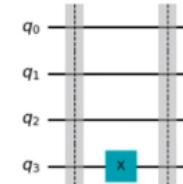
Un altro esempio sono le **funzioni costanti**, dove il risultato è sempre lo stesso a prescindere dall'input. Gli operatori si applicano solo sulle linee di output.

- if $f(x) = 0$ then apply the I gate at qbit-line n
- if $f(x) = 1$ then apply the X gate at qbit-line n

$$f(x) = 0, \forall x \in \{0,1\}^3$$

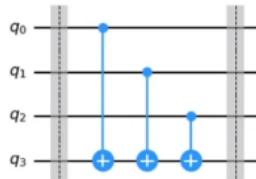


$$f(x) = 1, \forall x \in \{0,1\}^3$$



Ancora un esempio sono le **funzioni bilanciate**, sono funzioni in cui il numero di input che danno risultato 0 è uguale al numero di input che danno come risultato 1. La funzione quindi si bilancia dal numero di volte che restituisce 0 e dal numero di volte che restituisce 1.

$$f: \{0,1\}^3 \rightarrow \{0,1\}$$



$$f(x) = 0 \quad | \quad f(x) = 1$$

$f(x) = 0$	$f(x) = 1$
.	.
$x = 000$	$x = 001$
$x = 011$	$x = 010$
$x = 101$	$x = 100$
$x = 110$	$x = 111$

La funzione restituirà 0 se il numero di 1 è pari, 1 altrimenti.

Si possono mettere qua e là degli X gate per ottenere versioni diverse di funzioni bilanciate, da ricordare comunque di negare sia prima che dopo per mantenere la reversibilità del circuito.

Phase oracles:

In questo caso l'output $f(x)$ è tipicamente un singolo bit con valore 0 o 1. Se la funzione ha valore 1 il vettore di input assumerà una fase negativa.

$$P_f|x\rangle = (-1)^{f(x)}|x\rangle$$

$$P_f|x\rangle = \begin{cases} |x\rangle & \text{if } f(x) = 0 \\ -|x\rangle & \text{if } f(x) = 1 \end{cases}$$

"In generale per ottenere un phase oracle è sufficiente creare una matrice identità e mettere -1 in corrispondenza degli input che per quella funzione danno un valore 1".

Esempio di phase oracle per il not(x):

$$f : \{0,1\} \leftarrow \{0,1\}$$

$$f(x) = \neg x$$

$$P_f |1\rangle = \begin{bmatrix} -1 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} 0 \\ 1 \end{bmatrix} = \begin{bmatrix} 0 \\ 1 \end{bmatrix} = |1\rangle = (-1)^0 |1\rangle$$

$$P_f = \begin{bmatrix} -1 & 0 \\ 0 & 1 \end{bmatrix}$$

$$P_f |0\rangle = \begin{bmatrix} -1 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} 1 \\ 0 \end{bmatrix} = \begin{bmatrix} -1 \\ 0 \end{bmatrix} = -|0\rangle = (-1)^1 |0\rangle$$

Esempio di phase oracle per l'and:

$$f : \{0,1\}^2 \leftarrow \{0,1\}$$

$$f(x_1 x_2) = x_1 \wedge x_2$$

$$P_f |01\rangle = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & -1 \end{bmatrix} \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \end{bmatrix} = \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \end{bmatrix} = |01\rangle = (-1)^0 |01\rangle$$

$$P_f = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & -1 \end{bmatrix}$$

$$P_f |11\rangle = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & -1 \end{bmatrix} \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ -1 \end{bmatrix} = -|11\rangle = (-1)^1 |11\rangle$$

Esempio di phase oracle per lo xor:

$$f : \{0,1\}^3 \leftarrow \{0,1\}$$

$$f(x_1 x_2 x_3) = x_1 \oplus x_2 \oplus x_3$$

$$P_f |011\rangle = |011\rangle = (-1)^0 |011\rangle$$

$$P_f |010\rangle = -|010\rangle = (-1)^1 |010\rangle$$

$$P_f |110\rangle = |110\rangle = (-1)^0 |110\rangle$$

$$P_f |111\rangle = -|111\rangle = (-1)^1 |111\rangle$$

$$P_f = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & -1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & -1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 \end{bmatrix}$$

Da come si può vedere dall'immagine di sopra in prossimità delle righe il cui numero di 1 è dispari si applica una fase negativa (-1).

I phase oracle sono utili in vari task computazionali, "Ci sono alcuni problemi in cui è difficile trovare una soluzione ma è relativamente facile verificare se un input è una soluzione oppure no. In questi contesti gli oracle si applicano bene. **Quindi gli oracle non trovano una soluzione ma verificano se una soluzione va bene oppure no.** Applicando questo oracle su tutti gli input contemporaneamente è come se riuscissimo a trovare una soluzione o comunque fare un passo verso la soluzione finale".

Esempi: database search, string matching.

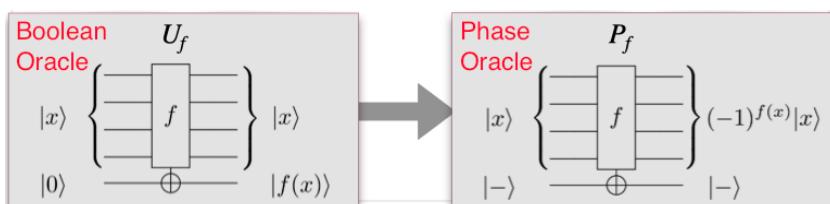
Creazione dei Phase Oracle:

Esiste una correlazione tra phase oracle e boolean oracle, ovvero partendo da un boolean oracle possiamo ottenere un phase oracle, cioè:

$$U_f |x, \bar{0}\rangle = |x, f(x)\rangle$$

Da un boolean oracle possiamo ottenere un phase oracle inizializzando il vettore di output, precedentemente uguale $\bar{0}$, a $|-\rangle$ (meno):

$$U_f |x, -\rangle = P_f |x\rangle = (-1)^{f(x)} |x\rangle |-\rangle$$



Esempio 1:

$$\begin{aligned}
 f: \{0,1\} &\leftarrow \{0,1\} & U_f |1,-\rangle &= \begin{bmatrix} 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 0 \\ 0 \\ 1/\sqrt{2} \\ -1/\sqrt{2} \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 1/\sqrt{2} \\ -1/\sqrt{2} \end{bmatrix} = |1,-\rangle \\
 f(x) &= \neg x & U_f |0,-\rangle &= \begin{bmatrix} 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1/\sqrt{2} \\ -1/\sqrt{2} \\ 0 \\ 0 \end{bmatrix} = \begin{bmatrix} -1/\sqrt{2} \\ 1/\sqrt{2} \\ 0 \\ 0 \end{bmatrix} = -|0,-\rangle
 \end{aligned}$$

Esempio 2:

$$\begin{aligned}
 f: \{0,1\}^2 &\leftarrow \{0,1\} \\
 f(x_1 x_2) &= x_1 \wedge x_2 \\
 U_f &= \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} & U_f |11,-\rangle &= \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 1/\sqrt{2} \\ -1/\sqrt{2} \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ -1/\sqrt{2} \\ 1/\sqrt{2} \end{bmatrix} = -|11,-\rangle
 \end{aligned}$$

Garbage:

L'allocazione e il mantenimento delle variabili di appoggio (ancilla bit) potrebbe causare alcuni problemi, come quello della reversibilità.

Gli algoritmi classici in genere non solo calcolano l'output desiderato, ma creano anche ulteriori informazioni lungo il percorso.

Tali residui aggiuntivi di calcoli non sono un problema significativo negli algoritmi classici, e la memoria che richiedono può facilmente essere recuperata semplicemente cancellandoli.

In una visione quantistica, tuttavia, le cose non sono così facili.

Assumiamo che vogliamo implementare una funzione Booleana f associata a un operatore unitario U_f .

Considerare il caso che un algoritmo classico esegue il seguente processo:

$$\begin{array}{c} \text{Input} \quad \text{Ancilla} \\ V_f |x, \bar{0}, \bar{0}\rangle = |x, f(x), g(x)\rangle \\ \text{Output} \end{array}$$

E supponiamo di non sapere come applicare U_f , ma sappiamo come applicare V_f .

Durante la computazione applicando una superimposizione, oltre alla correlazione dell'input con l'output otteniamo una seconda correlazione con gli ancilla (garbage). Quindi la presenza di questa garbage collegata ad x influenza la computazione successiva portando dei risultati inattesi. Ciò significa che avere una funzione che contiene delle variabili temporanee non ci consente di ottenere l'operatore inverso.

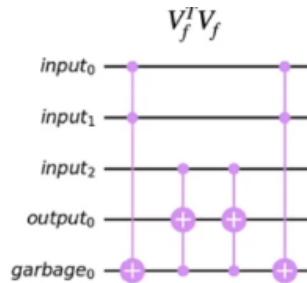
Il problema si risolve applicando lo stesso operatore ma a specchio, invertendo l'ordine. Esempio:

Example #3

$$f: \{0,1\}^3 \rightarrow \{0,1\}$$

$$f(x_1, x_2, x_3) = x_1 \wedge x_2 \wedge x_3$$

$$g(x_1, x_2, x_3) = x_1 \wedge x_2$$



$$V_f^T V_f |x,0,0\rangle = V_f^T |x, f(x), g(x)\rangle$$

$$= |x,0,0\rangle$$

It works!



Now we can look at the effect of first applying V_f and then applying V_f^T

Tuttavia non sempre tale operazione è possibile. In alternativa si possono tenere fuori le variabili temporanee (ancilla bit) dalla computazione, tecnica denominata *uncomputation*.

Questa tecnica consiste nell'aggiungere, oltre all'ancilla bit, un altro bit che serve a mantenere una copia dell'output.

$$|x,0,0,0\rangle \rightarrow |x, f(x), g(x), 0\rangle$$

OUT. ^{ANC.}
 ↓ ↓
 — —

Si procede a copiare, mediante CNOT, l'output nell'ultimo registro, ottenendo:

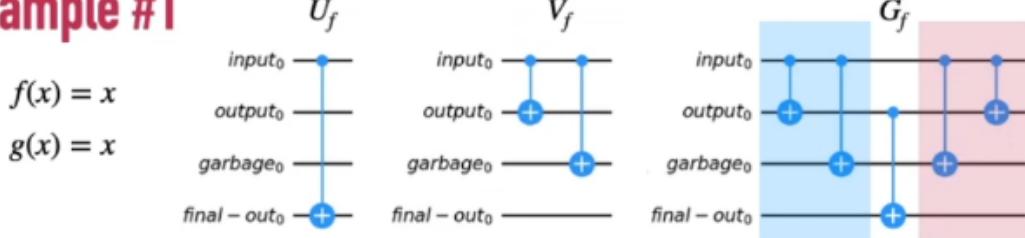
$$|x, f(x), g(x), 0\rangle \rightarrow |x, f(x), g(x), f(x)\rangle$$

Infine, applichiamo V_f^T per cancellare $f(x)$, $g(x)$. Questo lo si fa principalmente per cancellare gli ancilla bit. Quindi sfruttiamo la reversibilità del circuito per cancellare i valori temporanei.

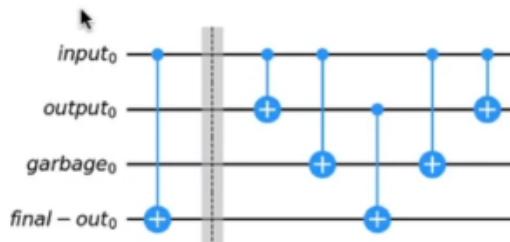
$$|x, f(x), g(x), f(x)\rangle \rightarrow |x, 0, 0, f(x)\rangle$$

Esempio:

Example #1



$$G_f U_f |x,0,0,0\rangle = G_f |x,0,0,f(x)\rangle = |x,0,0,0\rangle$$



It works!



8. Quantum teleportation & superdense coding

Quantum teleportation:

Il teletrasporto quantistico è una tecnica per trasferire informazioni quantistiche da un mittente, in una posizione, a un ricevitore, ad una certa distanza. Mentre il teletrasporto è comunemente rappresentato nella fantascienza come un mezzo per trasferire oggetti fisici da un luogo all'altro, il teletrasporto quantistico trasferisce solo informazioni quantistiche (questo per via del principio di non clonazione della meccanica quantistica che non permette di copiare qubit, nella destinazione deve esserci quindi un qubit già pronto per ospitare l'informazione del qubit di partenza). Il mittente non deve conoscere il particolare stato quantistico trasferito. Inoltre, la posizione del destinatario può essere sconosciuta, ma le informazioni classiche devono essere inviate da mittente a destinatario per completare il teletrasporto. Poiché le informazioni classiche devono essere inviate, il teletrasporto non può avvenire più velocemente della velocità della luce (per cui il teorema di non comunicazione non viene violato).

I componenti principali necessari per il teletrasporto includono un mittente, le informazioni (un qubit), un canale tradizionale, un canale quantistico e un ricevitore. Un fatto interessante è che il mittente non ha bisogno di conoscere il contenuto esatto delle informazioni che vengono inviate.

Alice vuole mandare una quantum information a Bob. In specifico Alice vuole mandare il seguente stato di qubit:

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

Alfa e beta sono due numeri reali potenzialmente di quantità infinita.

Il protocollo:

1. Alice e Bob hanno bisogno di una terza parte per inviarsi una coppia di qubit entangled. Questa figura (chiamata Telamon nelle slide) dà una qubit (nello stato entangled) ad Alice e uno a Bob. Per quanto detto prima data una coppia di qubit correlata quando uno dei due subisce una modifica anche l'altro verrà modificato.
2. Alice applica al suo qubit q_1 una matrice di Hadamard seguito da un CNOT controllato da $|\Psi\rangle$ (dal messaggio).
3. Alice misura i suoi qubits (distruggendo l'informazione che vuole inviare) ed invia a Bob i risultati, tramite un canale di comunicazione classico.
4. Bob riceve i bit inviati da Alice, applica delle trasformazioni appropriate ai suoi qubits per riottenere $|\Psi\rangle$.

Analisi del protocollo:

[passo 1] Supponiamo quindi che i qubit entangled siano una delle quattro combinazioni di Bell:

$$\begin{aligned} |\beta_{00}\rangle &= \frac{|00\rangle + |11\rangle}{\sqrt{2}}; \\ |\beta_{01}\rangle &= \frac{|01\rangle + |10\rangle}{\sqrt{2}}; \\ |\beta_{10}\rangle &= \frac{|00\rangle - |11\rangle}{\sqrt{2}}; \\ |\beta_{11}\rangle &= \frac{|01\rangle - |10\rangle}{\sqrt{2}}, \end{aligned}$$

Assumiamo che i qubit selezionati siano:

$$|e\rangle = \frac{1}{\sqrt{2}}(|00\rangle + |11\rangle)$$

Alice e Bob possiedono ciascuno un qubit entangled della coppia su scritta (denotati con A e B rispettivamente).

$$|e\rangle = \frac{1}{\sqrt{2}}(|0\rangle_A|0\rangle_B + |1\rangle_A|1\rangle_B)$$

Insieme con $|\Psi\rangle = \alpha|0\rangle + \beta|1\rangle$ formano un sistema a 3 qubit dove Alice possiede il primo e Bob il terzo.

$$\begin{aligned} |\psi\rangle \otimes |e\rangle &= \frac{1}{\sqrt{2}}(\alpha|0\rangle \otimes (|00\rangle + |11\rangle) + \beta|1\rangle \otimes (|00\rangle + |11\rangle)) \\ &= \frac{1}{\sqrt{2}}(\alpha|000\rangle + \alpha|011\rangle + \beta|100\rangle + \beta|111\rangle) \end{aligned}$$

, dove \otimes denota il prodotto tensoriale.

[passo 2] Alice prende i suoi due qubit e applica un CNOT gate con psi come controllo, seguito dall'applicazione della matrice di Hadamard sul primo qubit (psi) (ovviamente Alice lavora solo sui suoi qubit in quanto non ha a disposizione i qubit di Bob).

$$(H \otimes I \otimes I)(CNOT \otimes I)(|\psi\rangle \otimes |e\rangle)$$

Il CNOT siccome ha psi come controllo ha effetto soltanto sulla porzione in cui psi=1.
Se il primo qubit = 0 il qubit target non viene modificato, se invece il primo qubit = 1 il qubit target viene invertito.

$$\begin{aligned} &= (H \otimes I \otimes I)(CNOT \otimes I) \frac{1}{\sqrt{2}}(\alpha|000\rangle + \alpha|011\rangle + \beta|100\rangle + \beta|111\rangle) \\ &= (H \otimes I \otimes I) \frac{1}{\sqrt{2}}(\alpha|000\rangle + \alpha|011\rangle + \beta|110\rangle + \beta|101\rangle) \\ &= \frac{1}{2}(\alpha(|000\rangle + |011\rangle + |100\rangle + |111\rangle) + \beta(|010\rangle + |001\rangle - |110\rangle - |101\rangle)) \end{aligned}$$

Which can then be separated and written as

$$\begin{aligned} &= \frac{1}{2}(|00\rangle(\alpha|0\rangle + \beta|1\rangle) \quad \text{orange} \\ &\quad + |01\rangle(\alpha|1\rangle + \beta|0\rangle) \quad \text{blue} \\ &\quad + |10\rangle(\alpha|0\rangle - \beta|1\rangle) \quad \text{green} \quad \leftarrow \\ &\quad + |11\rangle(\alpha|1\rangle - \beta|0\rangle) \quad \text{pink} \quad) \end{aligned}$$

Quindi quello che hanno Alice e Bob è:

$$\begin{aligned} &= \frac{1}{2}(|00\rangle(\alpha|0\rangle + \beta|1\rangle) \\ &\quad + |01\rangle(\alpha|1\rangle + \beta|0\rangle) \quad \text{Sono degli stati che vanno in superimpostione, se Alice misura e ottiene 00 allora Bob ottiene quello in prima riga} \\ &\quad + |10\rangle(\alpha|0\rangle - \beta|1\rangle) \\ &\quad + |11\rangle(\alpha|1\rangle - \beta|0\rangle) \quad) \end{aligned}$$



[passo 3] Alice misura i suoi due qubit ed invia i risultati a Bob attraverso un canale di comunicazione classico. Il risultato che lei ottiene è sempre uno dei seguenti stati base standard, con la stessa probabilità:

$$|00\rangle, |01\rangle, |10\rangle, |11\rangle$$

nb: una volta fatta la misurazione, Alice non sa a priori in quale dei quattro stati colllasserà.

In base alle misurazioni di Alice, gli stati di Bob saranno proiettati in:

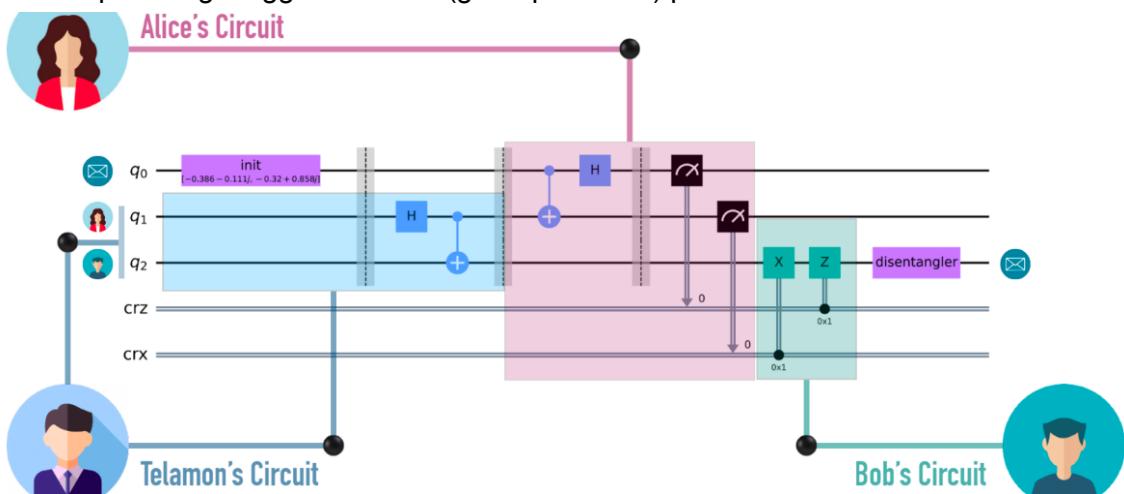
$$\begin{aligned}
 |00\rangle &\rightarrow (\alpha|0\rangle + \beta|1\rangle) \\
 |01\rangle &\rightarrow (\alpha|1\rangle + \beta|0\rangle) \\
 |10\rangle &\rightarrow (\alpha|0\rangle - \beta|1\rangle) \\
 |11\rangle &\rightarrow (\alpha|1\rangle - \beta|0\rangle)
 \end{aligned}$$



[passo 4] Bob riceve i bit da Alice ed applica una trasformazione appropriata ai suoi qubit per riottenere $|\Psi\rangle = \alpha|0\rangle + \beta|1\rangle$:

Bob's State	Bits Received	Gate Applied
$(\alpha 0\rangle + \beta 1\rangle)$	00	I
$(\alpha 1\rangle + \beta 0\rangle)$	01	X
$(\alpha 0\rangle - \beta 1\rangle)$	10	Z
$(\alpha 1\rangle - \beta 0\rangle)$	11	ZX

Ricapitolando: Alice dopo la misurazione avrà uno stato tra (00,01,10,11) ciascuno con il 25% di probabilità ed invia il suo stato a Bob. Lo stato correlato di Bob è cambiato per ottenere l'informazione di Ψ ma non abbastanza per essere preciso, bisogna aggiustarlo, questo perché la misurazione di Alice ha modificato qualcosa. A questo punto Bob prende la misurazione di Alice e capisce come è collassato lo stato di Alice e opera degli "aggiustamenti" (gate quantistici) per ottenere lo stato iniziale.



Implementazione in qiskit:

```
def random_state():
    psi = random_statevector(2)
    return psi      ↑ Funzione fornita da qiskit

# Returns a circuit for creating a Bell Pair
def create_bell_pair():
    qc = QuantumCircuit(2)
    qc.h(0)
    qc.cx(0,1)
    return qc    I

# Gate di Alice: prepara i due qubits, li
def alice_gate():
    psi = QuantumRegister(1,"message")
    ar = QuantumRegister(1,"alice qubit")
    br = QuantumRegister(1,"bob qubit")
    cr = ClassicalRegister(2,"code")
    qc = QuantumCircuit(psi,ar,br,cr)
    qc.cx(psi[0],ar[0])
    qc.h(psi[0])
    qc.barrier()
    # misura e invia due bit
    qc.measure(psi[0],0)
    qc.measure(ar[0],1)
    return qc

def bob_gate():
    qubit = QuantumRegister(1, "bob qubit")
    crz = ClassicalRegister(1, name="crz")
    crx = ClassicalRegister(1, name="crx")
    qc = QuantumCircuit(qubit,crz,crx)
    qc.x(qubit[0]).c_if(crx[0], 1)
    qc.z(qubit[0]).c_if(crz[0], 1)
    return qc

def Teleportation(psi):
    ar = QuantumRegister(1, name="alice qubit")
    br = QuantumRegister(1, name="bob qubit")
    qr = QuantumRegister(1, name="message")
    crz = ClassicalRegister(1, name="crz")
    crx = ClassicalRegister(1, name="crx")
    vr = ClassicalRegister(1,name="verify")
    qc = QuantumCircuit(qr,ar,br,crz,crx,vr)

    init_gate = Initialize(psi)
    init_gate.label = "init"
    inverse_init_gate = init_gate.gates_to_uncompute
        ↑ questa funzione crea l'inverso del gate a cui è applicato
    qc = qc.compose(init_gate, [qr[0]])
    qc.barrier()
    qc = qc.compose(create_bell_pair(),[ar[0],br[0]])
    qc.barrier()
    qc = qc.compose(alice_gate(), [qr[0],ar[0],br[0]])
    qc.barrier()
    qc = qc.compose(bob_gate(), [br[0]], [crz,crx])
    qc = qc.compose(inverse_init_gate, [br[0]])
    qc.measure(br,vr)

    qcl.draw_circuit(qc)
    simulator = QasmSimulator()
    compiled_circuit = transpile(qc, simulator)
    job = simulator.run(compiled_circuit, shots=1)
    result = job.result()
    counts = result.get_counts(qc)
    #qcl.counts(counts)
    if list(counts.keys())[0][0]=='0':
        print("Message Received")
```

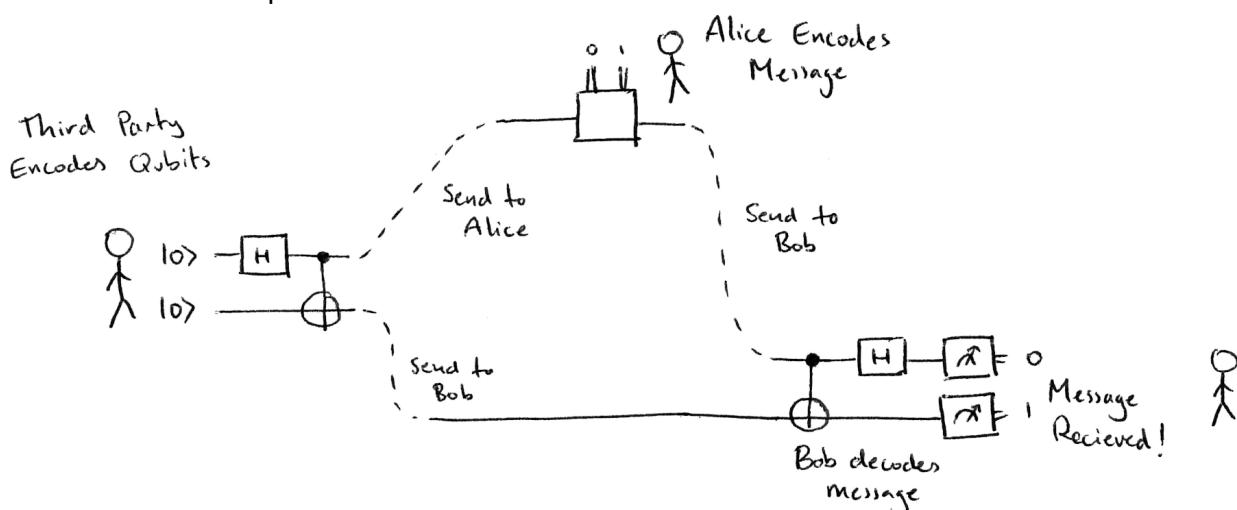
Superdense coding:

Si tratta di un protocollo di comunicazione quantistica per comunicare un numero di bit classici di informazioni trasmettendo un numero minore di qubit, presupponendo che mittente e destinatario precondividano uno stato con la massima correlazione (stati di Bell).

La codifica superdensa può essere considerata l'opposto del teletrasporto quantistico.

Panoramica generale:

Supponiamo che Alice voglia inviare due classici bit di informazioni (00, 01, 10, 11) a Bob usando i qubit (invece dei bit classici). A tale scopo, Charlie, una terza persona, prepara uno stato entangled (ad es. uno stato Bell) utilizzando un circuito o un gate Bell. Charlie quindi invia uno di questi qubit (nello stato Bell) ad Alice e l'altro a Bob. Una volta che Alice ottiene il suo qubit nello stato entangled, applica una certa porta quantistica al suo qubit a seconda del messaggio a due bit (00, 01, 10, 11) che vuole inviare a Bob. Il suo qubit entangled viene quindi inviato a Bob che, dopo aver applicato un'appropriata porta quantistica ed aver effettuato una misurazione, può recuperare il messaggio classico a due bit. Si osservi che Alice non ha bisogno di comunicare a Bob quale porta applicare per ottenere i bit classici corretti dalla sua misura proiettiva.



Analisi del protocollo:

Il protocollo può essere suddiviso in cinque diversi passaggi: preparazione, condivisione, codifica, invio e decodifica.

[1. Preparazione] Il protocollo inizia con la preparazione di uno stato entangled, che viene poi condiviso tra Alice e Bob. Supponiamo sia il seguente stato di Bell:

$$|\Phi^+\rangle = \frac{1}{\sqrt{2}}(|0\rangle_A \otimes |0\rangle_B + |1\rangle_A \otimes |1\rangle_B)$$

, dove \otimes denota il prodotto tensoriale.

[2. Condivisione] Dopo la preparazione dello stato Bell $|\Phi^+\rangle$, il qubit indicato dal pedice A viene inviato ad Alice e il qubit indicato dal pedice B viene inviato a Bob. A questo punto, Alice e Bob potrebbero trovarsi in luoghi completamente diversi, anche molto distanti l'uno dall'altro.

[3. Codifica] Applicando localmente una porta quantistica al suo qubit, Alice può trasformare lo stato entangled $|\Phi^+\rangle$ in uno qualsiasi dei quattro stati Bell (incluso, ovviamente, $|\Phi^+\rangle$ stesso). Da notare che questo processo non può "spezzare" l'entanglement tra i due qubit.

Descriviamo ora quali operazioni Alice deve eseguire sul suo qubit entangled, a seconda del messaggio classico a due bit che vuole inviare a Bob. Vedremo in seguito perché vengono eseguite queste operazioni specifiche. Ci sono quattro casi, che corrispondono alle quattro possibili stringhe a due bit che Alice potrebbe voler inviare.

- Se Alice vuole inviare la stringa classica a due bit 00 a Bob, applica la porta quantistica dell'identità, $\mathbb{I} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$, al suo qubit, in modo che rimanga invariato. Lo stato entangled risultante è quindi:

$$|B_{00}\rangle = \frac{1}{\sqrt{2}}(|0_A0_B\rangle + |1_A1_B\rangle)$$

In altre parole, lo stato entangled condiviso tra Alice e Bob non è cambiato, cioè è ancora $|\Phi^+\rangle$.

- Se Alice vuole inviare la classica stringa a due bit 01 a Bob, allora applica la porta quantistica NOT, $X = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}$, al suo qubit, in modo che lo stato quantico entangled risultante diventi:

$$|B_{01}\rangle = \frac{1}{\sqrt{2}}(|1_A0_B\rangle + |0_A1_B\rangle)$$

- Se Alice vuole inviare la classica stringa a due bit 10 a Bob, allora applica la porta quantistica di inversione di fase, $Z = \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix}$, al suo qubit, così diventa lo stato entangled risultante:

$$|B_{10}\rangle = \frac{1}{\sqrt{2}}(|0_A0_B\rangle - |1_A1_B\rangle)$$

- Se, invece, Alice vuole inviare la classica stringa a due bit 11 a Bob, allora applica la porta quantistica $Z * X$ al suo qubit, in modo che lo stato entangled risultante diventi:

$$|B_{11}\rangle = \frac{1}{\sqrt{2}}(|0_A1_B\rangle - |1_A0_B\rangle)$$

, dove X, Z sono due delle matrici di Pauli.

[4. Invio] Dopo aver eseguito una delle operazioni sopra descritte, Alice può inviare il suo qubit entangled a Bob attraverso un mezzo fisico convenzionale (fibra ottica, wireless, ecc..).

[5. Decodifica] Affinché Bob possa scoprire quali bit classici Alice le ha inviato, eseguirà l'operazione unitaria CNOT, con A come qubit di controllo e B come qubit target. Successivamente, Bob applica l'operatore Hadamard solo sul qubit entangled A , $H \otimes I$.

- Se lo stato entangled risultante fosse $|B_{00}\rangle$, allora dopo l'applicazione delle suddette operazioni unitarie lo stato entangled diverrà $|00\rangle$.
- Se lo stato entangled risultante fosse $|B_{01}\rangle$, allora dopo l'applicazione delle suddette operazioni unitarie lo stato entangled diverrà $|01\rangle$.
- Se lo stato entangled risultante fosse $|B_{10}\rangle$, allora dopo l'applicazione delle suddette operazioni unitarie lo stato entangled diverrà $|10\rangle$.
- Se lo stato entangled risultante fosse $|B_{11}\rangle$, allora dopo l'applicazione delle suddette operazioni unitarie lo stato entangled diverrà $|11\rangle$.

Queste operazioni eseguite da Bob possono essere viste come una misura che proietta lo stato entangled su uno dei quattro vettori base a due qubit $|00\rangle, |01\rangle, |10\rangle$ o $|11\rangle$.

Esempio:

Ad esempio, se lo stato entangled risultante (dopo le operazioni eseguite da Alice) era:

$$|B_{01}\rangle = \frac{1}{\sqrt{2}}(|1_A0_B\rangle + |0_A1_B\rangle)$$

allora applicando il CNOT con A come bit di controllo e B come bit target $|B'_{01}\rangle$ diventa:

$$|B'_{01}\rangle = \frac{1}{\sqrt{2}}(|1_A1_B\rangle + |0_A1_B\rangle)$$

Ora, la porta di Hadamard si applica solo ad A , ottenendo:

$$B''_{01} = \frac{1}{\sqrt{2}} \left(\left(\frac{1}{\sqrt{2}}(|0\rangle - |1\rangle) \right)_A \otimes |1_B\rangle + \left(\frac{1}{\sqrt{2}}(|0\rangle + |1\rangle) \right)_A \otimes |0_B\rangle \right)$$

Per semplicità, sbarazziamoci dei pedici, quindi abbiamo:

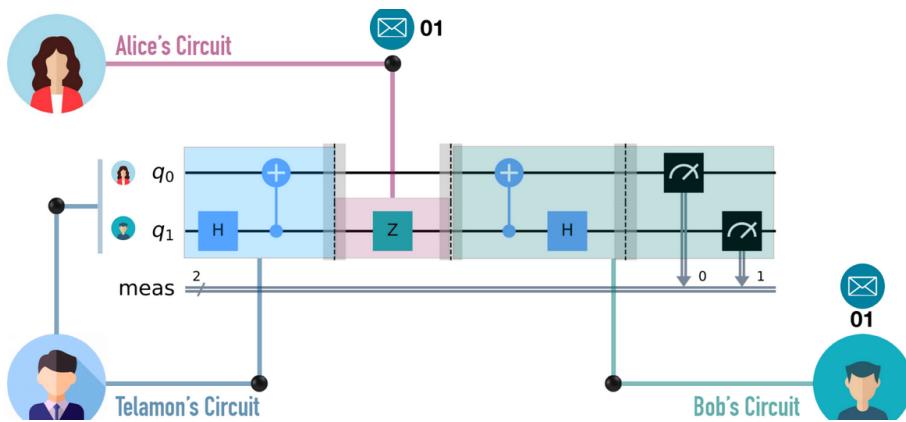
$$B''_{01} = \frac{1}{\sqrt{2}} \left(\frac{1}{\sqrt{2}}(|0\rangle - |1\rangle) \otimes |1\rangle + \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle) \otimes |1\rangle \right) = \frac{1}{\sqrt{2}} \left(\frac{1}{\sqrt{2}}(|01\rangle - |11\rangle) + \frac{1}{\sqrt{2}}(|01\rangle + |11\rangle) \right) = \frac{1}{2}|01\rangle - \frac{1}{2}|11\rangle + \frac{1}{2}|01\rangle + \frac{1}{2}|11\rangle = |01\rangle$$

Ora, Bob ha lo stato base $|01\rangle$, quindi sa che Alice voleva inviare la stringa a due bit 01.

Aspetto sicurezza:

La codifica superdensa è una forma di comunicazione quantistica sicura. Se un intercettatore, comunemente chiamato Eva, intercetta il qubit di Alice in rotta verso Bob, tutto ciò che ottiene Eva è solo una parte di uno stato entangled. Senza l'accesso al qubit di Bob, Eva non è in grado di ottenere alcuna informazione dal qubit di Alice. Una terza parte non è in grado di intercettare le informazioni comunicate attraverso la codifica superdensa e un tentativo di misurare uno dei qubit fa collassare lo stato di quel qubit e Bob e Alice se ne accorgerebbero.

Implementazione in qiskit:



Parte di Telamon, applicazione del CNOT e H:

```
def create_bell_pair():
    qc = QuantumCircuit(2)
    qc.h(0)
    qc.cx(0, 1)
    qc = qc.to_gate()
    qc.name = "BellPair"
    return qc
```

Parte di Alice, codifica del messaggio:

```
def encode_message(msg):
    qc = QuantumCircuit(1)
    if msg[1] == "1":
        qc.x(0)
    if msg[0] == "1":
        qc.z(0)
    qc = qc.to_gate()
    qc.name = "Encode"
    return qc
```

Parte di Bob, decodifica del messaggio. Bob applica il CNOT con controllo il bit ricevuto da Alice e target il suo bit, infine applica al bit di Alice il gate di Hadamard:

```
def decode_message():
    qc = QuantumCircuit(2)
    qc.cx(0, 1)
    qc.h(0)
    qc = qc.to_gate()
    qc.name = "Decode"
    return qc
```

Si rimette tutto insieme richiamando i gate scritti prima e procedendo a misurare l'output:

```
def DenseCoding(msg):
    if(len(msg)!=2 or msg[1] not in "01" or msg[0] not in "01"):
        print("Messaggio non valido")
    qc = QuantumCircuit(2,2)
    qc = qc.compose(create_bell_pair(),[0,1])
    qc.barrier()
    qc = qc.compose(encode_message(msg),[0,1])
    qc.barrier()
    qc = qc.compose(decode_message(),[0,1])
    qc.measure(0,0)
    qc.measure(1,1)
    qc.draw_circuit(qc)
    # simulation
    simulator = QasmSimulator()
    compiled_circuit = transpile(qc, simulator)
    job = simulator.run(compiled_circuit, shots=1000)
    result = job.result()
    counts = result.get_counts(qc)
    #qc1.counts(counts)
    print("Received: "+str(list(counts.keys())[0]))
```

9. Quantum key distribution, protocollo BB84:

Lo scopo è scambiarsi una chiave da utilizzare successivamente in una crittografia simmetrica, avendo a disposizione uno canale quantistico.

Alice e Bob per comunicare utilizzano un canale di comunicazione messo a disposizione da una terza parte (Eva).

Se qualora ci fosse un utente malevole che intercetta il messaggio, gli interlocutori se ne accorgerebbero ignorando il messaggio e provvedendo a scambiarsi di nuovo le chiavi.

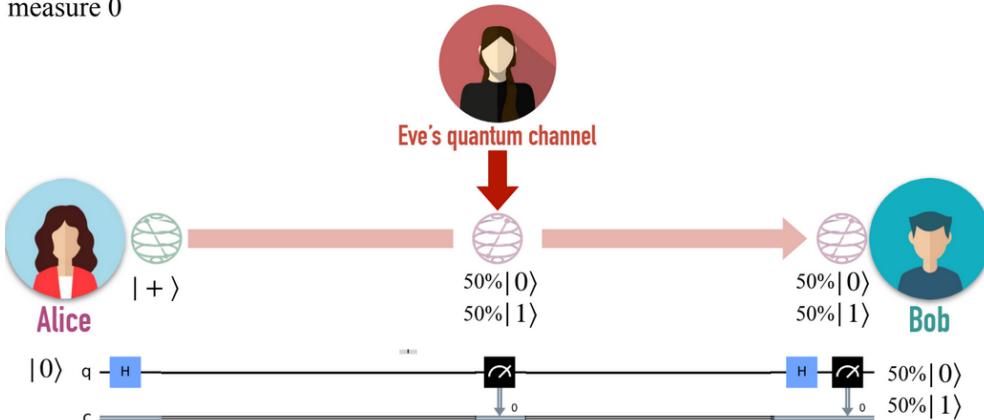
Il protocollo:

Abbiamo un canale quantistico e Alice vuole mandare un messaggio a Bob, per semplicità supponiamo che sia un solo qubit. L'idea di base è che se Eva vuole leggere il qubit dovrà misurarlo e di conseguenza cambierà lo stato. Questo cambiamento di stato viene percepito come un intercettamento del messaggio.

Esempio:

For Example, if Alice prepares a qubit in the state $|+\rangle$ (0 in the X-basis), and Bob measures it in the X-basis, Bob is sure to measure 0.

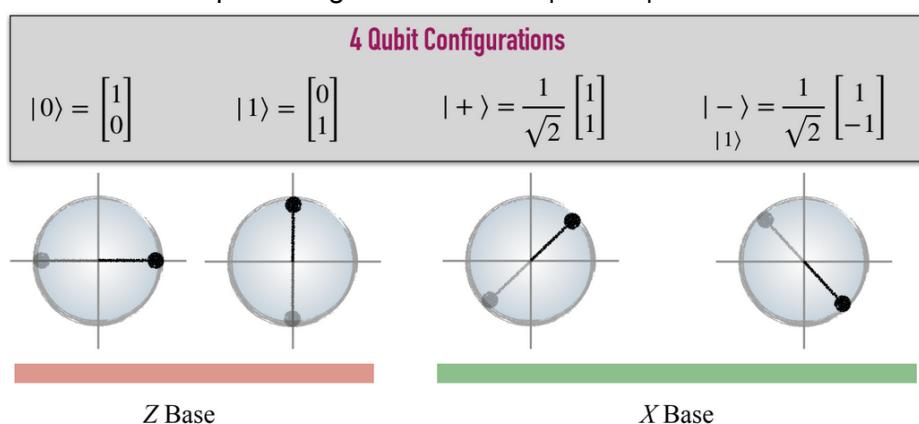
But if Eve tries to measure this qubit in the Z-basis before it reaches Bob, she will change the qubit's state from $|+\rangle$ to either $|0\rangle$ or $|1\rangle$, and Bob is no longer certain to measure 0



Qubit basis states:

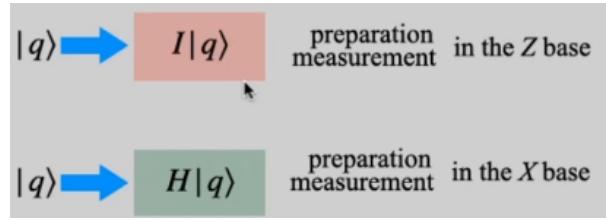
Utilizziamo principalmente due tipi di base, per indicare coppie di qubit che sono ortogonali tra di loro.

- Z-basis states sono due qubit ortogonali definiti in: $|0\rangle$ e $|1\rangle$
- X-basis states sono due qubit ortogonali definiti in: $|+\rangle$ e $|-\rangle$

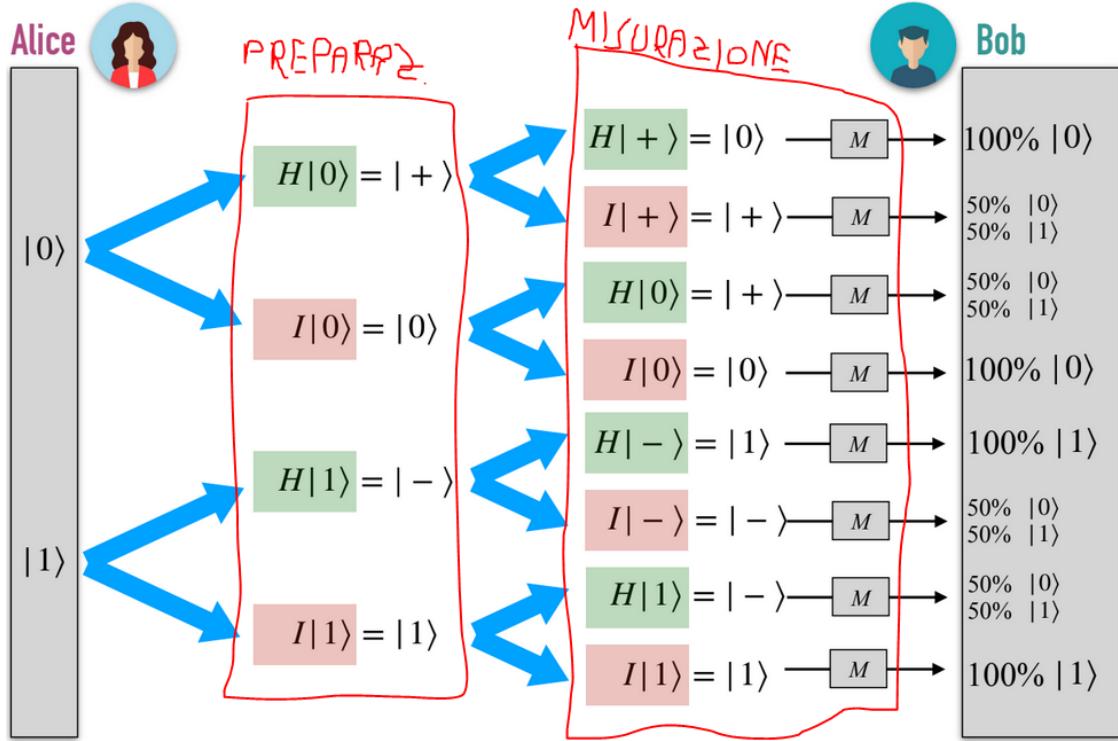


Per quanto riguarda la preparazione/misurazione del qubit:

- Per preparare/misurare un qubit sulla base Z dobbiamo lasciarlo sostanzialmente inalterato;
- Mentre per preparare/misurare un qubit nella base X dobbiamo applicare una matrice H, ovvero una rotazione in maniera tale che lo 0 diventi + l'1 diventi -

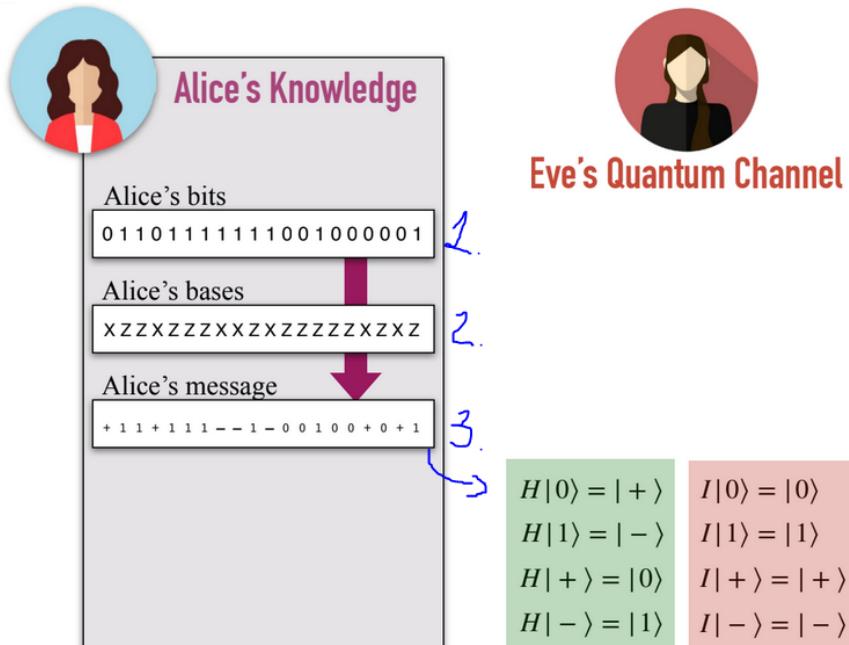


Ricapitolando:

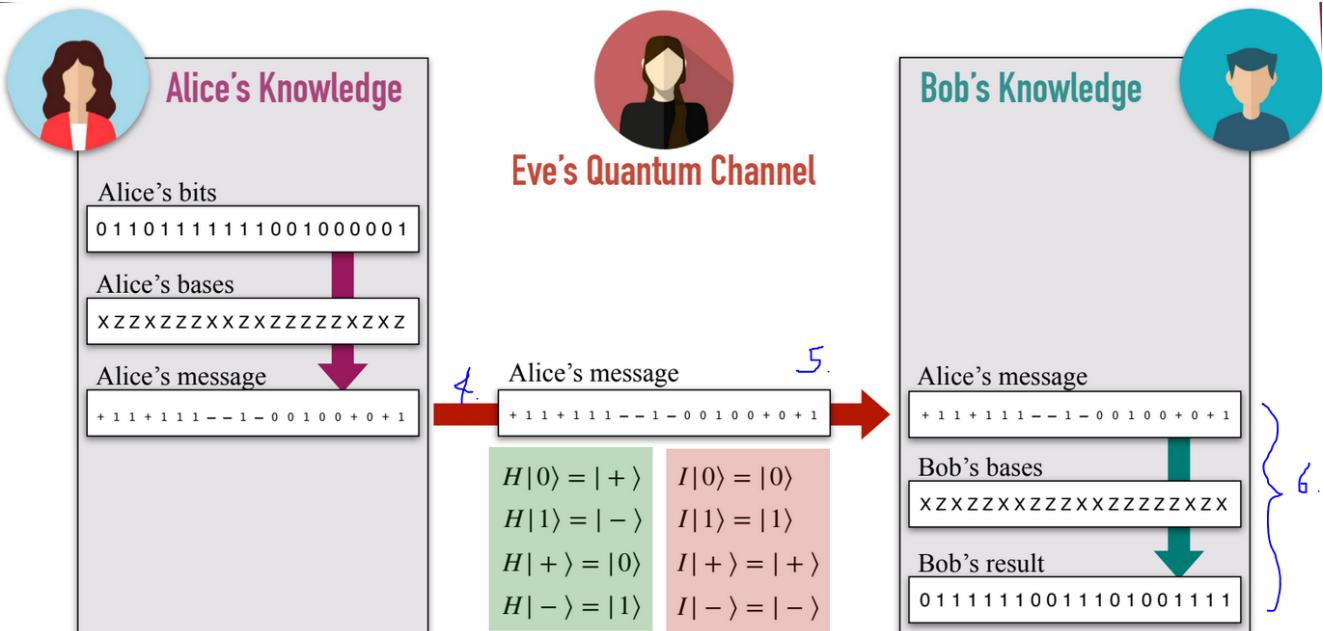


Esempio senza intercettazione:

1. Alice genera un insieme random di bit, tali bit rimangono privati ovvero non li condivide con nessuno;
2. Successivamente Alice crea, sempre in maniera random, una serie di basi X e Z, anche questo rimane privato;
3. Alice prepara codificando ciascun bit, creati nel punto 1, in un qubit per mezzo delle basi create al punto 2. Di seguito il riassunto dei primi 3 passi:

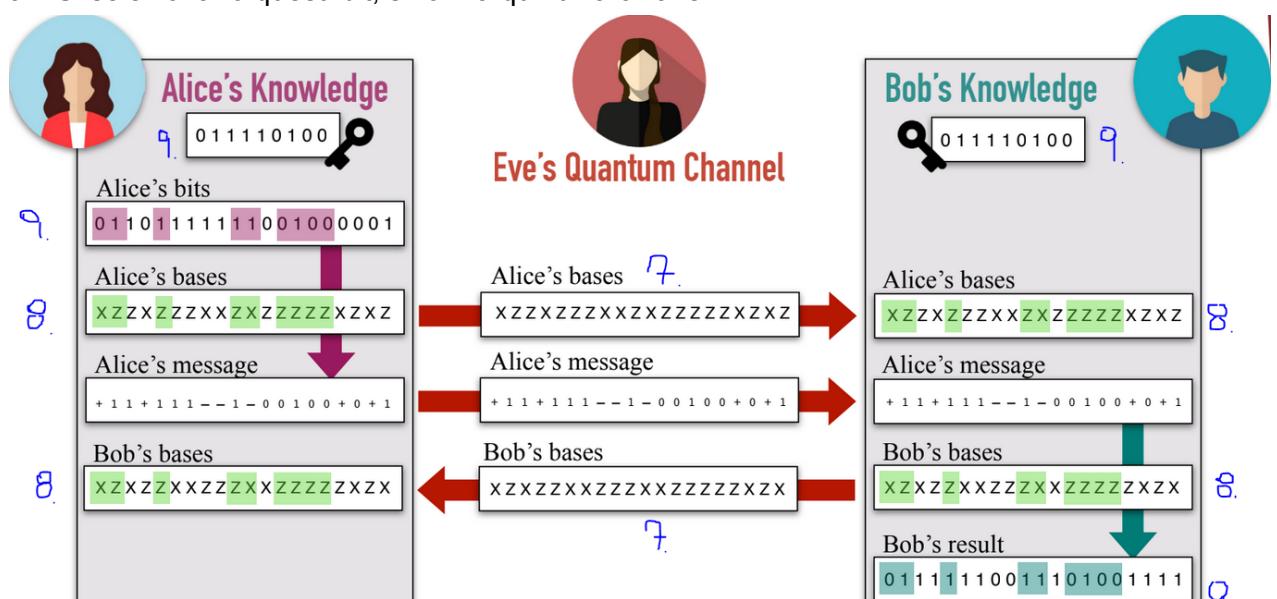


- Dopo la preparazione Alice manda a Bob il messaggio attraverso il canale quantistico messo a disposizione da Eva. "Questo è il punto in cui bisogna fidarsi di Eva".
- Supponiamo che Eva faccia passare il messaggio senza intercettare.
- Bob a questo punto riceve il messaggio di Alice e genera anche lui in maniera random una sequenza di basi (ricordiamo che Bob non conosce le basi generate da Alice). Successivamente Bob misura il messaggio ricevuto da Alice con le sue basi. Di seguito il riassunto dei passi 4-5-6:

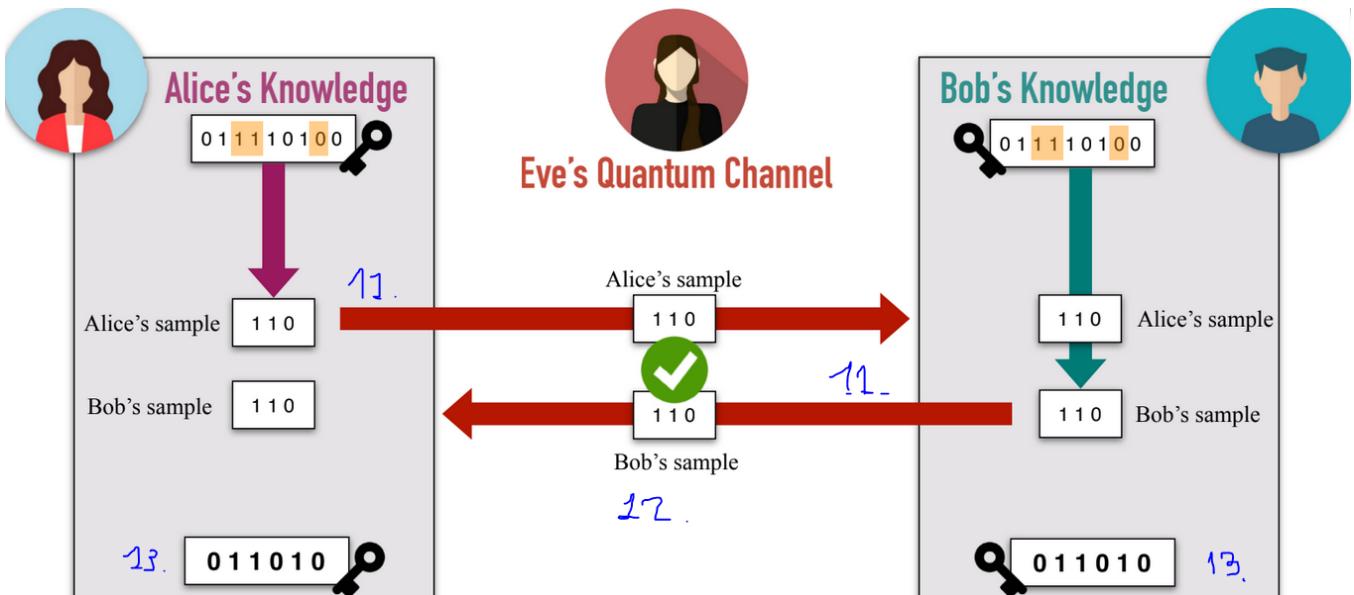


nb: ci sono dei casi in cui è possibile che i valori ottenuti dall'applicazione delle basi di Bob al messaggio porti a dei valori non corretti, ciò dovuto a valori dipendenti dalla probabilità di osservare un valore piuttosto che un'altro. Ricordiamo che Bob non può applicare le stesse basi di Alice in quanto non le conosce. Esempio immagine pagina precedente.

- A questo punto Alice condivide con Bob le basi che ha scelto per la preparazione. Bob fa lo stesso (misurazione).
Eva da ora conosce entrambe le basi con cui si è fatta preparazione e misurazione. Non ha però conoscenza dei bit.
- A questo punto Alice e Bob fanno il matching delle basi verificando per quali valori coincidono (evidenziati in verde sotto). Questi sono quindi i punti che permettono loro di ottenere lo stesso bit, se un bit è stato preparato e misurato con la stessa base siamo certi che il bit durante la transizione non abbia creato ambiguità.
- Si selezionano questi bit, si forma quindi la chiave.

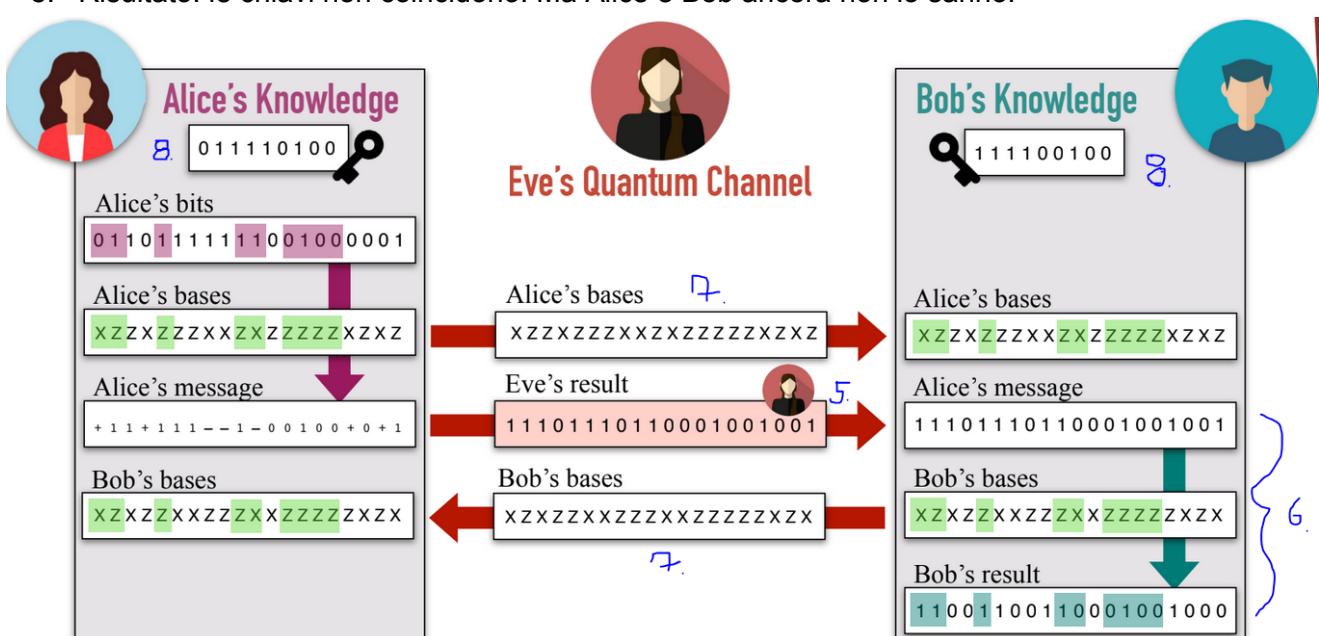


10. Alice e Bob a questo punto non sanno se la loro chiave coincide. Se Eva avesse intercettato il messaggio la chiave sarebbe compromessa.
nb: se la chiave è composta da pochi bit è probabile che, nonostante Eva abbia intercettato il messaggio, le chiavi siano uguali.
11. A questo punto Alice e Bob prendono un sample della chiave (solitamente la metà) se lo passano e verificano se coincidono.
12. Se coincidono allora non c'è stata intercettazione.
13. La chiave a questo punto sarà composta dalla chiave di prima meno i bit utilizzati per i sample (in quanto compromessi per via degli scambi).

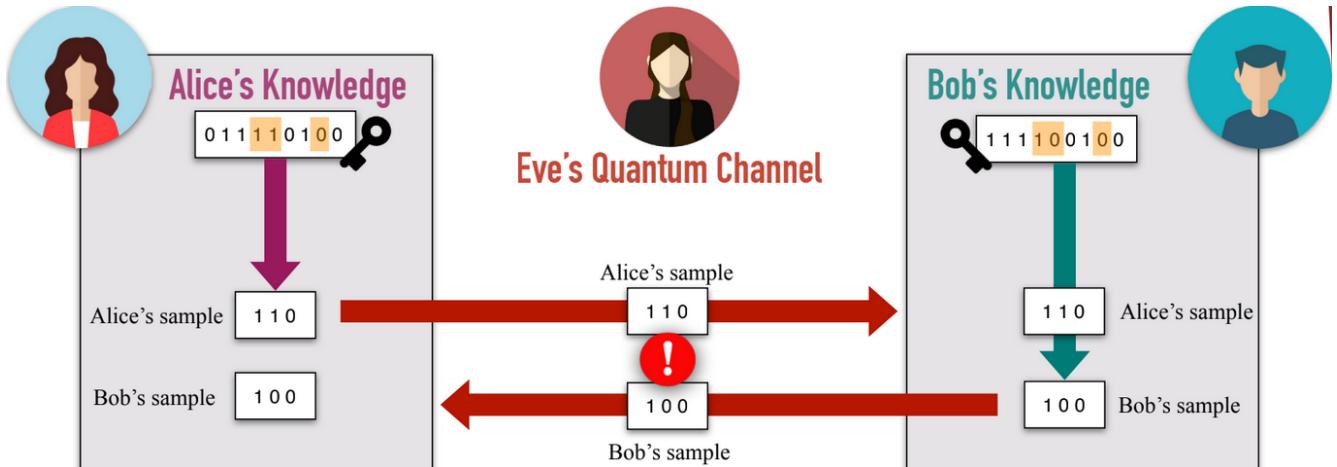


Esempio con intercettazione:

1. I passi da 1 a 4 sono gli stessi di prima.
5. Supponiamo che Eva intercetti il messaggio, tramite misurazioni. Per far ciò genera in maniera random delle basi.
6. A questo punto Bob riceve il messaggio da Alice, non sapendo che Eva ha intercettato tale messaggio. Bob genera delle basi random e le applica a i bit ricevuti.
7. Alice e Bob si scambiano le rispettive basi, ne fanno il matching ed estrapolano la chiave.
8. Risultato: le chiavi non coincidono! Ma Alice e Bob ancora non lo sanno.



9. Alice e Bob verificano se la loro chiave coincide. Estrapolano un sample dalla rispettiva chiave e verificano se tali sample coincidono. Se non c'è un matching c'è stata una intercettazione. Si scartano allora i bit e si ritenta la procedura.
 nb: naturalmente la certezza non c'è, ma per un numero elevato di bit la probabilità di registrare ed individuare un'intercettazione è molto alta.



Implementazione in qiskit:

Definiamo una funzione che codifichi la stringa di bit con la stringa delle basi [1]. Definiamo una funzione che, data la stringa del messaggio e delle basi, applica le misurazioni [2].

```
[1] def encode_message(bits, bases):
    message = []
    for i in range(n):
        qc = QuantumCircuit(1,1)
        if bases[i] == 0:
            if bits[i] == 0:
                pass
            else:
                qc.x(0)
        else:
            if bits[i] == 0:
                qc.h(0)
            else:
                qc.x(0)
                qc.h(0)
        qc.barrier()
        message.append(qc)
    return message
```



```
[2] def measure_message(message, bases):
    backend = Aer.get_backend('aer_simulator')
    measurements = []
    for q in range(n):
        if bases[q] == 0: # measuring in Z-basis
            message[q].measure(0,0)
        if bases[q] == 1: # measuring in X-basis
            message[q].h(0)
            message[q].measure(0,0)
    #Simulatore
    aer_sim = Aer.get_backend('aer_simulator')
    qobj = assemble(message[q], shots=1, memory=True)
    result = aer_sim.run(qobj).result()
    measured_bit = int(result.get_memory()[0])
    measurements.append(measured_bit)
    return measurements
```

La funzione *remove_garbage* [1] prende le basi di Alice e Bob e ne fa il matching prendendo i valori che combaciano. Questi valori formeranno la chiave. La funzione *sample_bits* [2] crea un sample per Alice e Bob con lo scopo di verificare se c'è stata una intercettazione.

```
[1] def remove_garbage(a_bases, b_bases, bits):
    good_bits = []
    for q in range(n):
        if a_bases[q] == b_bases[q]:
            good_bits.append(bits[q])
    return good_bits
```



```
[2] def sample_bits(bits, selection):
    sample = []
    for i in selection:
        i = np.mod(i, len(bits))
        sample.append(bits.pop(i))
    return sample
```

Infine richiamiamo il tutto:

```
np.random.seed(seed=0)
n = 100
alice_bits = randint(2, size=n)
alice_bases = randint(2, size=n)
message = encode_message(alice_bits, alice_bases)
...
# Interception!
eve_bases = randint(2, size=n)
intercepted_message = measure_message(message, eve_bases)
...
bob_bases = randint(2, size=n)
bob_results = measure_message(message, bob_bases)
alice_key = remove_garbage(alice_bases, bob_bases, alice_bits)
bob_key = remove_garbage(alice_bases, bob_bases, bob_results)
sample_size = 15
bit_selection = randint(n, size=sample_size)
bob_sample = sample_bits(bob_key, bit_selection)
print("  bob_sample = " + str(bob_sample))
alice_sample = sample_bits(alice_key, bit_selection)
print("alice_sample = "+ str(alice_sample))
if bob_sample != alice_sample:
    print("Eva ha intercettato.")
else: print("Eve non ha intercettato")
```

10. Quantum key distribution, protocollo E91:

A differenza del precedente protocollo questo sfrutta il principio di correlazione dei qubit.

Alice e Bob possiedono una parte di una coppia di qubit correlati e comunicano tramite un canale quantistico messo a disposizione da Eva. Non è importante chi dei 3 crea e distribuisce tali qubit.

nb: Eva potrebbe dire ad Alice e Bob di avergli dato dei qubit correlati ma ciò potrebbe non essere così.

Si hanno i seguenti vantaggi nell'avere queste coppie di qubit correlati:

1. In virtù della correlazione dei qubit misurando un qubit si ottiene la misurazione anche dell'altro, anche se sono a distanza;
2. La misurazione è comunque random, ovvero se i qubit sono in una sovrapposizione di 0 e 1 con la stessa probabilità, Alice quando misura il suo qubit ha la probabilità del 50% di ottenere 0 o 1, e lei non sa quindi cosa otterrà. Però di contro se lei ottiene 0 si ha la certezza che anche Bob otterrà 0;
3. Se Eva intercettasse il messaggio distruggerebbe la correlazione dei qubit, e Alice e Bob se ne accorgerebbero.

Il protocollo:

I protocolli utilizzati sono 3: Z, X, H, R

 $A_1 = Z = \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix}$ Z	 $B_1 = Z = \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix}$ Z
$A_2 = X = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}$ X	$B_2 = \frac{1}{\sqrt{2}}(Z - X) = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & -1 \\ -1 & -1 \end{bmatrix} = -XHX$ R
$A_3 = \frac{1}{\sqrt{2}}(Z + X) = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} = H$ H	$B_3 = \frac{1}{\sqrt{2}}(Z + X) = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} = H$ H

Misurazione proiettiva:

Una misurazione proiettiva di uno stato $|\Psi\rangle = (\alpha|0\rangle + \beta|1\rangle)$, dato un operatore di misurazione M è il risultato dato dalla seguente formula:

$$\langle \Psi | M | \Psi \rangle$$

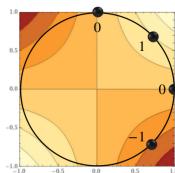
Il risultato della misurazione di $|\Psi\rangle$ è sempre un numero (scalare) compreso tra + 1 e - 1. Questo perche' dalla moltiplicazione di M a destra di un vettore colonna, mentre prima dell'operatore M, a sinistra, abbiamo un operatore riga. Se moltiplichiamo un vettore riga per un vettore colonna otteniamo uno scalare.

Otteniamo quindi la seguente disequazione:

$$| \langle \Psi | M | \Psi \rangle | \leq 1$$

Misurazione con X:

$$|\psi\rangle = |0\rangle \quad \langle 0 | X | 0 \rangle = [1 \ 0] \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} 1 \\ 0 \end{bmatrix} = [1 \ 0] \begin{bmatrix} 0 \\ 1 \end{bmatrix} = 0$$



$$|\psi\rangle = |1\rangle \quad \langle 1 | X | 1 \rangle = [0 \ 1] \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} 0 \\ 1 \end{bmatrix} = [0 \ 1] \begin{bmatrix} 1 \\ 0 \end{bmatrix} = 0$$

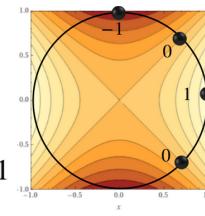
$$|\psi\rangle = |+\rangle \quad \langle + | X | + \rangle = \frac{1}{2} [1 \ 1] \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} 1 \\ 1 \end{bmatrix} = \frac{1}{2} [1 \ 1] \begin{bmatrix} 1 \\ 1 \end{bmatrix} = 1$$

$$|\psi\rangle = |-\rangle \quad \langle - | X | - \rangle = \frac{1}{2} [1 \ -1] \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} 1 \\ -1 \end{bmatrix} = \frac{1}{2} [1 \ -1] \begin{bmatrix} -1 \\ 1 \end{bmatrix} = -1$$

Dal grafico si nota che vale la proprietà di misurazione proiettiva, ovvero qualunque vettore normalizzato passato all'operatore restituirà un valore (scalare) compreso tra -1 e 1.

Misurazione con Z:

$$|\psi\rangle = |0\rangle \quad \langle 0|Z|0\rangle = [1 \ 0] \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix} \begin{bmatrix} 1 \\ 0 \end{bmatrix} = [1 \ 0] \begin{bmatrix} 1 \\ 0 \end{bmatrix} = 1$$



$$|\psi\rangle = |1\rangle \quad \langle 1|Z|1\rangle = [0 \ 1] \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix} \begin{bmatrix} 0 \\ 1 \end{bmatrix} = [0 \ 1] \begin{bmatrix} 0 \\ -1 \end{bmatrix} = -1$$

$$|\psi\rangle = |+\rangle \quad \langle +|Z|+\rangle = \frac{1}{2} [1 \ 1] \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix} \begin{bmatrix} 1 \\ 1 \end{bmatrix} = \frac{1}{2} [1 \ 1] \begin{bmatrix} 1 \\ -1 \end{bmatrix} = 0$$

$$|\psi\rangle = |-\rangle \quad \langle -|Z|- \rangle = \frac{1}{2} [1 \ -1] \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix} \begin{bmatrix} 1 \\ -1 \end{bmatrix} = \frac{1}{2} [1 \ -1] \begin{bmatrix} 1 \\ 1 \end{bmatrix} = 0$$

Dal grafico si nota che vale la proprietà di misurazione proiettiva, ovvero qualunque vettore normalizzato passato all'operatore restituirà un valore (scalare) compreso tra -1 e 1.

Misurazione con H:

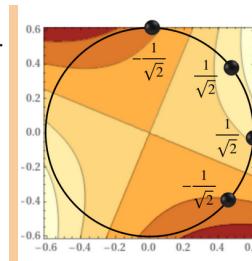
$$|\psi\rangle = |0\rangle \quad \langle 0|H|0\rangle = [1 \ 0] \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \begin{bmatrix} 1 \\ 0 \end{bmatrix} = [1 \ 0] \frac{1}{\sqrt{2}} \begin{bmatrix} 1 \\ 1 \end{bmatrix} = \frac{1}{\sqrt{2}}$$

$$|\psi\rangle = |1\rangle \quad \langle 1|H|1\rangle = [0 \ 1] \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \begin{bmatrix} 0 \\ 1 \end{bmatrix} = [0 \ 1] \frac{1}{\sqrt{2}} \begin{bmatrix} 1 \\ -1 \end{bmatrix} = -\frac{1}{\sqrt{2}}$$

$$|\psi\rangle = |+\rangle \quad \langle +|H|+\rangle = \frac{1}{2\sqrt{2}} [1 \ 1] \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \begin{bmatrix} 1 \\ 1 \end{bmatrix} = \frac{1}{2\sqrt{2}} [1 \ 1] \begin{bmatrix} 2 \\ 0 \end{bmatrix} = \frac{1}{\sqrt{2}}$$

$$|\psi\rangle = |-\rangle \quad \langle -|H|- \rangle = \frac{1}{2\sqrt{2}} [1 \ -1] \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \begin{bmatrix} 1 \\ -1 \end{bmatrix}$$

$$= \frac{1}{2\sqrt{2}} [1 \ -1] \begin{bmatrix} 0 \\ 2 \end{bmatrix} = -\frac{1}{\sqrt{2}}$$



Dal grafico si nota che vale la proprietà di misurazione proiettiva, ovvero qualunque vettore normalizzato passato all'operatore restituirà un valore (scalare) compreso tra -1 e 1.

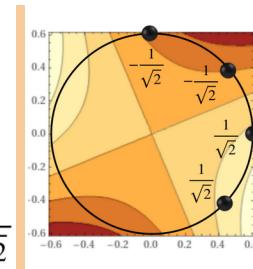
Misurazione con R:

$$|\psi\rangle = |0\rangle \quad \langle 0|R|0\rangle = [1 \ 0] \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & -1 \\ -1 & -1 \end{bmatrix} \begin{bmatrix} 1 \\ 0 \end{bmatrix} = [1 \ 0] \frac{1}{\sqrt{2}} \begin{bmatrix} 1 \\ -1 \end{bmatrix} = \frac{1}{\sqrt{2}}$$

$$|\psi\rangle = |1\rangle \quad \langle 1|R|1\rangle = [0 \ 1] \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & -1 \\ -1 & -1 \end{bmatrix} \begin{bmatrix} 0 \\ 1 \end{bmatrix} = [0 \ 1] \frac{1}{\sqrt{2}} \begin{bmatrix} -1 \\ -1 \end{bmatrix} = -\frac{1}{\sqrt{2}}$$

$$|\psi\rangle = |+\rangle \quad \langle +|R|+\rangle = \frac{1}{2\sqrt{2}} [1 \ 1] \begin{bmatrix} 1 & -1 \\ -1 & -1 \end{bmatrix} \begin{bmatrix} 1 \\ 1 \end{bmatrix} = \frac{1}{2\sqrt{2}} [1 \ 1] \begin{bmatrix} 0 \\ -2 \end{bmatrix} = -\frac{1}{\sqrt{2}}$$

$$|\psi\rangle = |-\rangle \quad \langle -|R|- \rangle = \frac{1}{2\sqrt{2}} [1 \ -1] \begin{bmatrix} 1 & -1 \\ -1 & -1 \end{bmatrix} \begin{bmatrix} 1 \\ -1 \end{bmatrix} = \frac{1}{2\sqrt{2}} [1 \ -1] \begin{bmatrix} 2 \\ 0 \end{bmatrix} = \frac{1}{\sqrt{2}}$$



Dal grafico si nota che vale la proprietà di misurazione proiettiva, ovvero qualunque vettore normalizzato passato all'operatore restituirà un valore (scalare) compreso tra -1 e 1.

Proprietà:

Dati due stati quantistici $|\Psi_1\rangle = (\alpha_1|0\rangle + \beta_1|1\rangle)$ e $|\Psi_2\rangle = (\alpha_2|0\rangle + \beta_2|1\rangle)$, e dati due operatori di misura M_1 e M_2 si ha:

$$\begin{aligned} & \langle \Psi_1 | M_1 | \Psi_1 \rangle \times \langle \Psi_2 | M_2 | \Psi_2 \rangle = \langle \Psi_1, \Psi_2 | M_1 \otimes M_2 | \Psi_1, \Psi_2 \rangle \\ & |\langle \Psi_1, \Psi_2 | M_1 \otimes M_2 | \Psi_1, \Psi_2 \rangle| \leq 1 \end{aligned}$$

Disuguaglianza CHSH:

Supponiamo di avere 4 variabili classiche (non quantistiche) denominate A_1, A_2, B_2, B_3 , e supponiamo che ciascuna di loro possa assumere due valori, +1 o -1. Allora:

$$A_1(B_3 + B_2) + A_2(B_3 - B_2) = \pm 2$$

Supponiamo ora che ciascuna di esse possa assumere un valore compreso tra -1 e +1 (quindi anche con la virgola). Prendendo il valore atteso (media) di queste quantità su N assegnazioni delle variabili casuali, otteniamo la disuguaglianza CHSH:

$$\frac{1}{N} \sum_N |(A_1 B_3) + (A_1 B_2) + (A_2 B_3) - (A_2 B_2)| \leq 2$$

*meglio se le 4 quantità abbiano la stessa numerosità.

Applichiamo ora la stessa formulazione nel settore quantistico, con bit non correlati:

Supponiamo di avere 4 operatori di misurazione A_1, A_2, B_2, B_3 . Supponiamo che $|\Psi_1 \Psi_2\rangle$ sia una coppia quantistica casuale di uno stato separabile, sappiamo che la sua misura proiettiva:

$$\langle M_1 M_2 \rangle = \langle \Psi_1 \Psi_2 | M_1 M_2 | \Psi_1 \Psi_2 \rangle \text{ sia un valore tra +1 e -1, per ogni } M_1 \in \{A_1, A_2\}, M_2 \in \{B_2, B_3\}.$$

Valgono le stesse considerazioni fatte prima, ovvero:

$$\frac{1}{N} \sum_N |\langle A_1 B_3 \rangle + \langle A_1 B_2 \rangle + \langle A_2 B_3 \rangle - \langle A_2 B_2 \rangle| \leq 2$$

*esempio vedi slide pagina 28.

..con qubit correlati:

Supponiamo di avere 4 operatori di misurazione A_1, A_2, B_2, B_3 . Denominiamo con $|\Phi\rangle$ una coppia quantistica casuale di stati entangled (correlati):

$$|\Phi\rangle = |\Phi^-\rangle = \frac{1}{\sqrt{2}}(|01\rangle - |10\rangle)$$

computando il CHSH su $|\Phi\rangle$ otteniamo:

$$\begin{aligned} S &= |\langle A_1 B_3 \rangle + \langle A_1 B_2 \rangle + \langle A_2 B_3 \rangle - \langle A_2 B_2 \rangle| = \\ &= |\langle \Phi | ZH | \Phi \rangle + \langle \Phi | ZR | \Phi \rangle + \langle \Phi | XH | \Phi \rangle - \langle \Phi | XR | \Phi \rangle| \\ &= \left| -\frac{1}{\sqrt{2}} - \frac{1}{\sqrt{2}} - \frac{1}{\sqrt{2}} - \frac{1}{\sqrt{2}} \right| = 2\sqrt{2} \end{aligned}$$

CHSH inequality violation

*per i passaggi vedi lezione 01:18

Si nota che il risultato rappresenta una chiara violazione della disuguaglianza CHSH. Da ciò segue che finché si rimane nel campo delle variabili classiche e nel campo quantistico ma senza qubit entangled è tutto apposto la diseguaglianza CHSH non è violata. Se invece utilizziamo variabili entangled vi è una violazione della disuguaglianza CHSH.

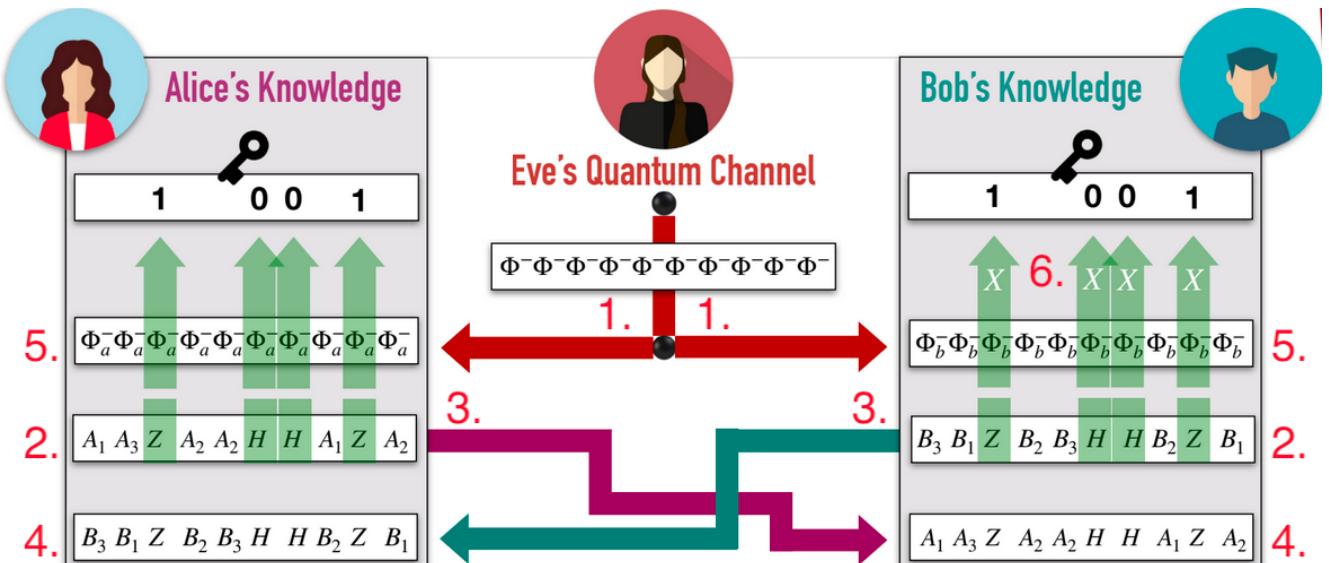
Risultato: **possiamo utilizzare il test di disuguaglianza CHSH per verificare se stiamo utilizzando dei qubit entangled (correlati).** Quindi se dalla misurazione proiettiva otteniamo un valore maggiore di 2 allora sono correlati, se invece otteniamo un valore minore di 2 i qubit non sono correlati.

Esempio senza intercettazione:

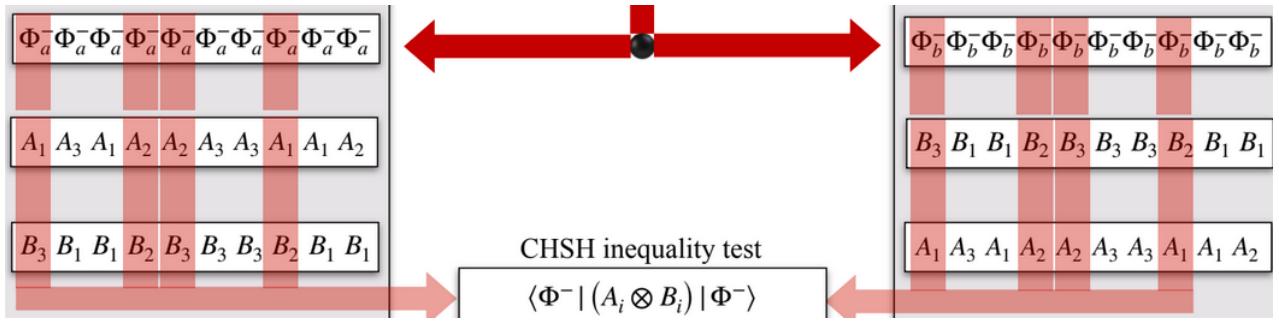
1. Eva (ma potrebbe essere Alice o Bob non è importante) prepara n coppie di qubit correlati e ne distribuisce metà ad Alice e metà a Bob.
2. A questo punto, Alice e Bob generano in maniera casuale una stringa di operatori di misurazione, tenendoli per il momento privati. In particolare Alice le genera dal set $\{A_1, A_2, A_3\}$, mentre Bob dal set $\{B_1, B_2, B_3\}$.
3. Adesso Alice e Bob si scambiano le stringhe contenenti gli operatori di misurazione. Il passaggio di queste stringhe non necessita obbligatoriamente di un canale quantistico.
4. Alice e Bob verificano per quali valori le loro basi combaciano nel caso $(A_1 B_1)$ e $(A_3 B_3)$, questo perché hanno gli stessi valori. Ovvero $A_1 = B_1 = Z$ e $A_3 = B_3 = H$:

 $A_1 = Z = \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix}$ $A_2 = X = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}$ $A_3 = \frac{1}{\sqrt{2}}(Z + X) = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} = H$	Z X H
 $B_1 = Z = \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix}$ $B_2 = \frac{1}{\sqrt{2}}(Z - X) = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & -1 \\ -1 & -1 \end{bmatrix} = -XH$ $B_3 = \frac{1}{\sqrt{2}}(Z + X) = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} = H$	Z R H

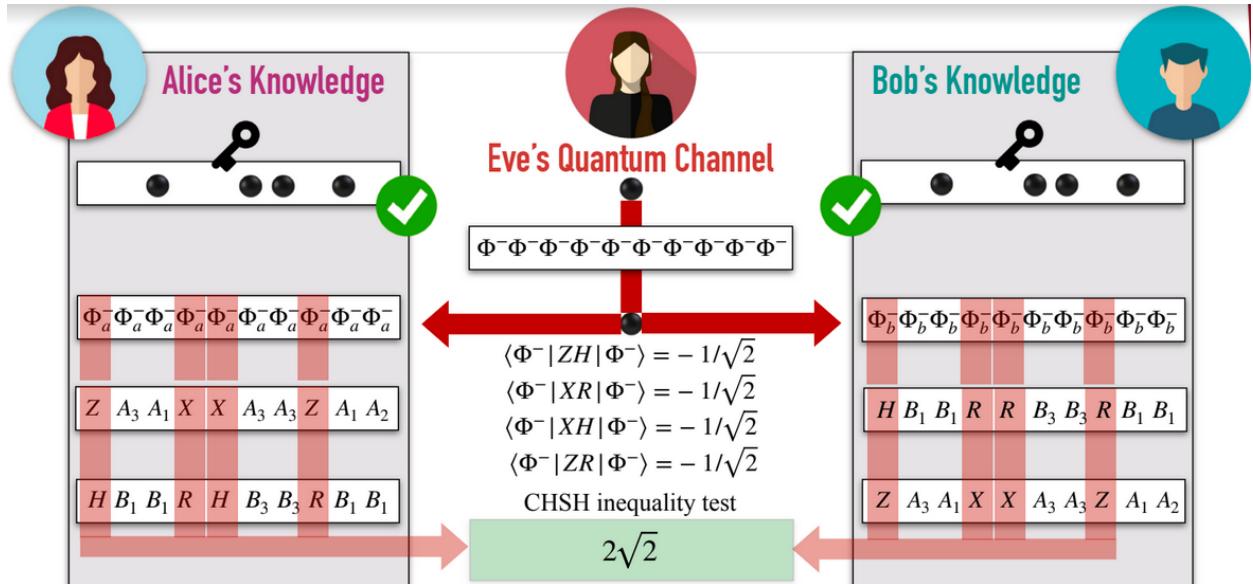
5. Alice e Bob utilizzano queste basi per fare la misurazione quantistica dei qubit selezionati.
6. Poiché il qubit su cui si lavora è: $|\Phi\rangle = |\Phi^-\rangle = \frac{1}{\sqrt{2}}(|01\rangle - |10\rangle)$ quando Alice ottiene 0 Bob ottiene 1 e viceversa, Bob deve quindi applicare l'operatore X per negare il risultato e ottenere così lo stesso valore di Alice.



7. A questo punto devono capire se questa chiave è ottenuta da qubit entangled. Per far ciò prendono tutte le coppie del tipo: (A_1, B_3) , (A_1, B_2) , (A_2, B_3) , (A_2, B_2) per verificare la disuguaglianza CHSH, da notare che non si prendono quelle utilizzate per la chiave. Ne fanno quindi la misurazione proiettiva.



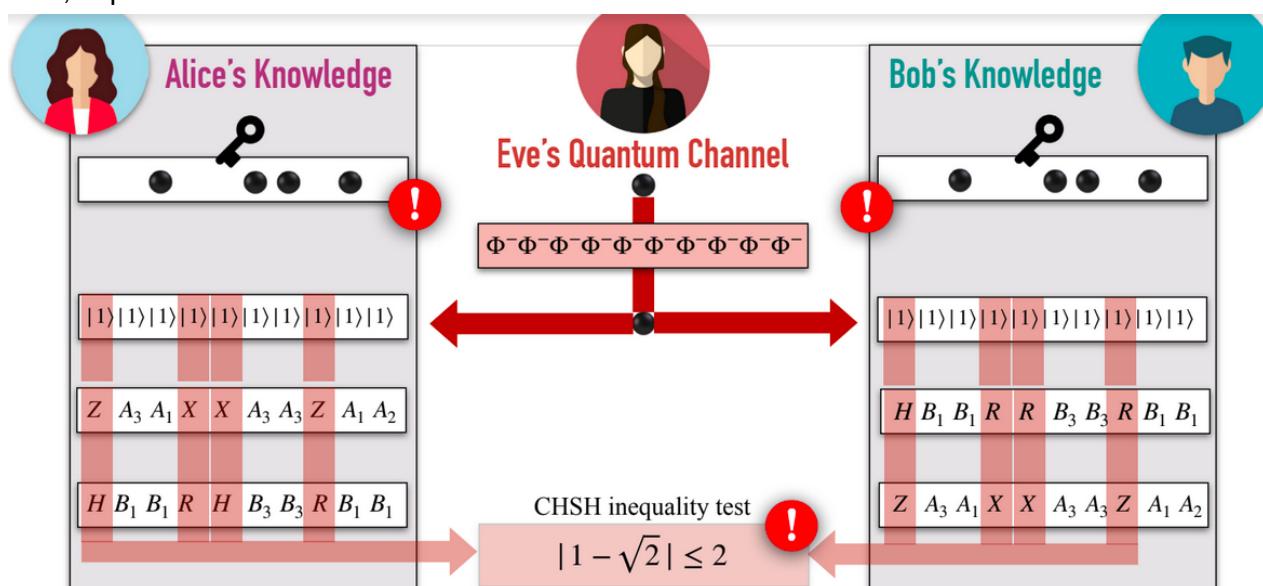
8. In questo caso i qubit sono correlati quindi il risultato viola la diseguaglianza CHSH, garantendo il corretto funzionamento del protocollo.



Esempio con intercettazione:

I passaggi sono analoghi al caso senza intercettazione. In questo caso, poiché siamo in presenza di qubit non correlati, il test CHSH non verrà violato e quindi Alice e Bob si accorgono che gli è stato fornito loro dei qubit non correlati.

attenzione: per un numero molto piccolo di qubit è probabile che, nonostante si utilizzino qubit non correlati, la procedura violi il test CHSH.



Implementazione:

Definiamo una funzione che restituisce una coppia di qubit. Prende due parametri in input:

1. *entangled* (di default a False) che indica se la coppia da creare risulta correlata o meno.
2. *pair* (di default a "11") che indica il valore di inizializzazione della coppia di qubit.

Utilizzando la funzione *get_state* (scritta nelle lezioni precedenti e riportata qui) inizializziamo i qubit.

```
def normalizza(v):
    v = v/np.linalg.norm(v)
    return v

def get_state(bstr):
    if type(bstr)==str:
        n = len(bstr)
        N = 2**n
        state = [0]*N
        pos = int(bstr,2)
        state[pos] = 1
        return np.array(state)
    if type(bstr)==list:
        state = list(bstr)
        return normalizza(np.array(state))
    if type(bstr)==int:
        state = np.random.rand(bstr)
        return normalizza(state)

def getPair(entangled=False, pair="11"):
    q = qc.get_state(pair)
    if(entangled):
        op1 = np.kron(qc.H,qc.I)
        op = np.matmul(qc.CX,op1)
        q = np.matmul(op,q)
    return q
```

Definiamo i due vettori contenenti le basi $A_1, A_2, A_3, B_1, B_2, B_3$. Definiamo inoltre dei vettori di supporto per l'applicazione del metodo CHSH.

```
A = [None, qc.Z, qc.X, qc.H]
B = [None, qc.Z, (qc.Z-qc.X)/np.sqrt(2), qc.H]

CHSHlist = [[1,2],[1,3],[2,3],[2,2]]
CHSHsign = [1,1,1,-1]
CHSHok = [[1,1],[3,3]]
```

Definiamo una funzione che esegue il misuramento proiettivo. Prende come parametri di input un vettore q e un operatore M .

```
def projective_measurement(q, M):
    return np.matmul(q,np.matmul(M,q))
```

Definiamo, infine, la funzione del protocollo E91. Prende come parametri di input:

1. Il numero N di coppie di qubit che Eva distribuisce ad Alice e Bob.
2. Una variabile booleana *interception* (di default a False) che simula la presenza di intercettazione da parte di Eva o meno.

```
def E91(N, interception=False):
    random.seed()
    # Telamon distributes N couples of entangled qubits
    qubits = []
    for i in range(N):
        qubits.append(getPair(True, "11"))
    print("Entangled qubits distributed ...")
    # if requested, simulate an interception
    if(interception):
        for i in range(N):
            qubits[i] = measure(qubits[i])
    # Alice generates a sequence of N measurement in the set {A1, A2, A3}
    a_meas = []
    for i in range(N):
        a_meas.append(randint(1,3))
    print("\nAlice measurements:")
    print(a_meas)
    # Bob generates a sequence of N measurement in the set {B1, B2, B3}
    b_meas = []
    for i in range(N):
        b_meas.append(randint(1,3))
    print("\nBob measurements:")
    print(b_meas)
    # Alice and Bob share the measurement and select the matching ones
    k_meas = []
    check = []
    for i in range(N):
        if([a_meas[i],b_meas[i]] in CHSHok):
            k_meas.append(i)
        if([a_meas[i],b_meas[i]] in CHSHlist):
            check.append(i)
    print("\nPosition list for generating the key:")
    print(k_meas)
    print("\nPosition list for checking CHSH inequality:")
    print(check)

    # check CHSH inequality
    CHSHval = 0
    for idx in range(len(check)):
        i = a_meas[check[idx]]
        j = b_meas[check[idx]]
        op = qc.tl([A[i],B[j]])
        qubit = qubits[check[idx]]
        s = np.matmul(op,qubit)
        if( [i,j] == [2,2] ):
            CHSHval -= np.matmul(qubit,s)
        else:
            CHSHval += np.matmul(qubit,s)
    CHSHval = np.abs(CHSHval*4/len(check))
    if CHSHval > 2:
        print("CHSH inequality test passed ("+str(CHSHval)+")")
    # Alice genera la chiave segreta
    AliceKey = ""
    BobKey = ""
    for i in range(len(k_meas)):
        s = [x for x in list(measure(qubits[k_meas[i]]))]
        if(s[1]==1):
            AliceKey = AliceKey+"0"
            BobKey = BobKey+"1"
        else:
            AliceKey = AliceKey+"1"
            BobKey = BobKey+"0"
    print("\nAlice's Secret Key: "+AliceKey)
    print("\nBob's Secret Key: "+BobKey)
else:
    print("CHSH inequality test non superato ("+str(CHSHval)+")")
```

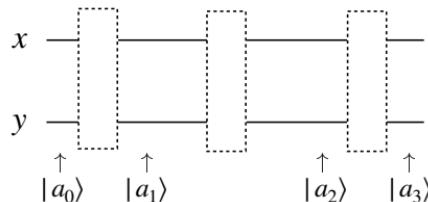
11. Algoritmo di Deutsch

L'algoritmo di Deutsch opera su una funzione booleana in input restituendo sempre una funzione booleana:

$$f: \{0, 1\} \rightarrow \{0, 1\}$$

L'obiettivo è capire se la funzione è costante oppure no. Ovviamente nella computazione classica rispondere a questa domanda prevede di interrogare la funzione due volte: una per 0 e una per 1, non c'è altro modo. Deutsch dimostra invece che nel caso quantistico basta una sola interrogazione.

L'algoritmo computa una serie di vettori a_0, a_1, a_2, a_3 ciascuno dei quali appartenente allo spazio di Hilbert $H_1 \times H_2$ dove H_1 e H_2 sono spazi bidimensionali. I vettori sono indicizzati in questo spazio attraverso xy , dove x e y sono due qubit.



Per qualunque funzione booleana f sia specificata, lavoriamo con la sua estensione invertibile, che qui simboleggiano come:

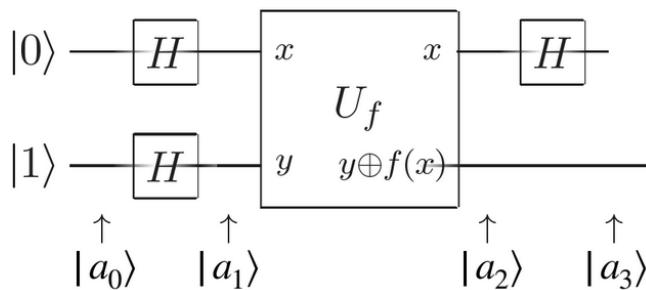
$$f^{-1}(xy) = x(f(x) \oplus y)$$

Quindi, l'input per l'algoritmo è in realtà la scelta di f come parametro.

1 $f(x) = \begin{cases} 0 & \text{if } x = 0 \\ 1 & \text{if } x = 1 \end{cases}$	$U_I = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix}$	2 $f(x) = \begin{cases} 1 & \text{if } x = 0 \\ 0 & \text{if } x = 1 \end{cases}$	$U_X = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$
3 $f(x) = \begin{cases} 1 & \text{if } x = 0 \\ 1 & \text{if } x = 1 \end{cases}$	$U_T = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix}$	4 $f(x) = \begin{cases} 0 & \text{if } x = 0 \\ 0 & \text{if } x = 1 \end{cases}$	$U_F = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$

La procedura:

1. Si parte dal vettore $a_0 = |01\rangle$
2. Otteniamo a_1 che è il risultato dell'applicazione di un operatore di Hadamard ad entrambe le linee di qubit.
3. In a_2 si applica l'oracle $U_{f^{-1}}$ dove $f^{-1}(xy) = x(f(x) \oplus y)$
4. Infine in a_3 applichiamo Hadamard solo alla prima linea di qubit. Scartiamo la seconda linea di qubit è prendiamo solo la prima.



Analisi:

- Il primo step è indipendente dalla funzione di input.

$$a_0 = |01\rangle = \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \end{bmatrix} \quad a_1 = H^{\otimes 2}|01\rangle = |+\rangle \otimes |- \rangle = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 \\ 1 \end{bmatrix} \otimes \frac{1}{\sqrt{2}} \begin{bmatrix} 1 \\ -1 \end{bmatrix} = \frac{1}{2} \begin{bmatrix} 1 \\ -1 \\ 1 \\ -1 \end{bmatrix}$$

Da notare che poiché i qubit non sono correlati possiamo applicare Hadamard in maniera indipendente. 0 applicando Hadamard diventa + e 1 applicando Hadamard diventa -.

Possiamo dimostrare che:

$$\text{For all } xy \longrightarrow a_1(xy) = \frac{1}{2}(-1)^y$$

Dimostrazione:

$$\begin{aligned} a_1(xy) &= \frac{1}{2} \sum_{t,u} (-1)^{x \cdot t} (-1)^{y \cdot u} a_0(ty) \\ &= \frac{1}{2} (-1)^{x \cdot 0} (-1)^{y \cdot 1} \end{aligned}$$

$a_0 = \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \end{bmatrix} \quad \begin{array}{l} a_0(00) = 0 \\ a_0(01) = 1 \\ a_0(10) = 0 \\ a_0(11) = 0 \end{array}$

$a_0(ty)$ è il vettore a cui si applica la matrice di Hadamard mentre il resto deriva dalla definizione di matrice di Hadamard. Si nota che tutte le componenti della sommatoria si annullano tranne una che è la componente 01. Quindi basta sostituire 0 a t e 1 a y ottenendo i valori riportati in riga 2.

Il fattore $(-1)^{x \cdot 0}$ si annulla ottenendo appunto:

$$= \frac{1}{2} (-1)^y$$

Quindi, ogni elemento del vettore:

$$a_1 = \frac{1}{2} \begin{bmatrix} 1 \\ -1 \\ 1 \\ -1 \end{bmatrix}$$

sarà 1 o -1 sulla base della posizione. Per $y = 0$ e $y = 2$ otteniamo 1, -1 negli altri casi.

- Possiamo dimostrare che:

$$\text{For all } xy \longrightarrow a_2(xy) = \frac{1}{2}(-1)^{f(x) \oplus y}$$

Dimostrazione:

$a_2(xy) = U_f a_1$ ovvero è uguale all'applicazione dell'oracle U_f al vettore a_1 .

Poiché U_f è una matrice di permutazione:

$$\begin{aligned} a_2(xy) &= U_f a_1 \\ &= a_1(x(f(x) \oplus y)) \\ &= \frac{1}{2} (-1)^{f(x) \oplus y} \end{aligned}$$

U_f is a permutation matrix
thus $U_f q = q(x(f(x) \oplus y))$

- Possiamo dimostrare che:

For all xy

$$|a_3(xy)|^2 = \frac{1}{8} |(-1)^{f(0)} + (-1)^{f(1) \oplus x}|^2$$

Dimostrazione:

$$a_3(xy) = \frac{1}{\sqrt{2}} \sum_t (-1)^{x \cdot t} a_2(ty)$$

We apply the H operator only to the first qubit

esplicitando $a_2(ty)$ si ottiene:

$$= \frac{1}{2\sqrt{2}} \sum_t (-1)^{x \cdot t} (-1)^{f(t) \oplus y}$$

i valori di t in realtà sono solo due possibili 0 o 1, possiamo riscrivere quindi in forma estesa:

$$\begin{aligned} a_3(xy) &= \frac{1}{\sqrt{2}} \sum_t (-1)^{x \cdot t} a_2(ty) & |a_3(xy)|^2 &= \frac{1}{8} |(-1)^{f(0)} + (-1)^{f(1) \oplus x}|^2 \\ &= \frac{1}{2\sqrt{2}} \sum_t (-1)^{x \cdot t} (-1)^{f(t) \oplus y} \\ &= \frac{1}{2\sqrt{2}} \left((-1)^{f(0) \oplus y} + (-1)^{x \oplus f(1) \oplus y} \right) \end{aligned}$$

Nell'ultimo passaggio: il primo o il secondo addendo cambia di segno ed il valore assoluto lo rende ininfluente.

Cosa succede se inizializziamo x a 0?

$$|a_3(xy)|^2 = \frac{1}{8} |(-1)^{f(0)} + (-1)^{f(1) \oplus x}|^2$$

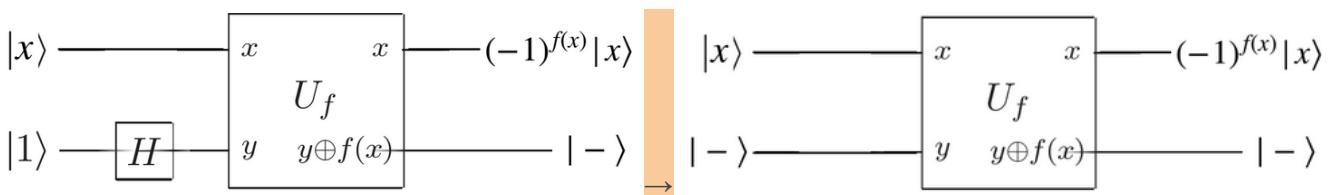
$$|a_3(0y)|^2 = \frac{1}{8} |(-1)^{f(0)} + (-1)^{f(1)}|^2$$

Allora:

- Se $f(0)$ e $f(1)$ sono entrambi 0 o entrambi 1 allora i valori del (-1) sono concordanti e si ottiene 2, ne segue che se l'espressione è uguale a $\frac{1}{8}2^2 = \frac{1}{2}$ la f è costante;
- Viceversa se $f(0)$ e $f(1)$ hanno lo stesso valore allora i valori (-1) saranno discordanti il risultato è 0. Ne segue che gli elementi all'interno del valore assoluto si annullano e di conseguenza la f non è costante.

Perché tutto ciò funziona?

Boolean Oracle e Phase Oracle sono collegati tra loro, vale a dire per implementare un Phase Oracle si può utilizzare un Boolean Oracle ed inizializzare l'ultima riga di qubit (output) a $|-\rangle$, quest'ultimo ottenuto applicando ad 1 l'operatore Hadamard.

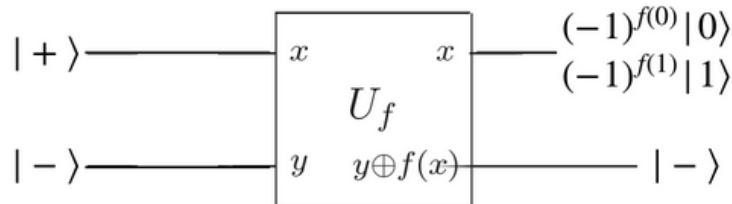


Il risultato di x sarà:

- Se $f(x) = 1$, la fase è negativa.

- Se $f(x) = 0$, x viene lasciato invariato.

Applicando Hadamard anche alla prima linea di qubit (in accordo a Deutsch), avremo la superimposizione degli input e output:



Sostanzialmente l'algoritmo di Deutsch è un Phase Oracle a cui gli applichiamo la matrice di Hadamard anche al primo qubit.

$$(-1)^{f(0)}|0\rangle + (-1)^{f(1)}|1\rangle = \begin{cases} 1/\sqrt{2} (|0\rangle + |1\rangle) \\ 1/\sqrt{2} (-|0\rangle - |1\rangle) \\ 1/\sqrt{2} (|0\rangle - |1\rangle) \\ 1/\sqrt{2} (-|0\rangle + |1\rangle) \end{cases} \begin{cases} |+\rangle & f(x) = 0, \forall x \in \{0,1\} \\ |-\rangle & f(x) = 1, \forall x \in \{0,1\} \\ |-\rangle & f(x) = x, \forall x \in \{0,1\} \\ |+\rangle & f(x) = \neg x, \forall x \in \{0,1\} \end{cases}$$

Se a $|+\rangle$ si applica la matrice di Hadamard torna a $|0\rangle$; Se la si applica a $|-\rangle$ il qubit torna a $|1\rangle$.

L'algoritmo di Deutsch è deterministico quantistico, nel senso che siamo sicuri che restituisce la risposta corretta con certezza anche con una sola esecuzione.

*esempi e visualizzazione grafica vedi slide da pagina 17.

Implementazione:

Applichiamo l'algoritmo di Deutsch usando l'algebra lineare:

```
Ui = qc.CX
Uf = qc.Idt(2)
Ut = np.kron(qc.I,qc.X)
Ux = np.array([[0,1,0,0],
               [1,0,0,0],
               [0,0,1,0],
               [0,0,0,1]])

def Deutsch(U):
    q = qc.get_state("01")
    q = np.matmul(qc.Had(2),q)
    q = np.matmul(U,q)
    q = np.matmul(np.kron(qc.H,qc.I),q)
    #print(q)
    qc.counts(q)
```

Adesso implementiamo l'algoritmo di Deutsch usando qiskit. Definiamo prima gli oracle:

```
# Boolean Oracle on {0,1} and output in {0,1}
# unique solution {0}
# Boolean Oracle on {0,1} and output in {0,1} def q1_0():
# unique solution {}
def q1_0():
    qc = QuantumCircuit(2)
    qc.x(0)
    qc.cx(0,1)
    qc.x(0)
    return qc
...
qc = QuantumCircuit(2)
qc.x(0)
qc.cx(0,1)
qc.x(0)
return qc
```

```

#####
# Two Solutions Oracles
#####

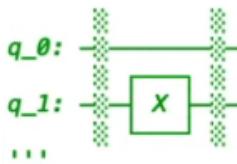
# Boolean Oracle on {0,1} and output in {0,1}
# Two solutions {0,1}
def q1_0_1():
    qc = QuantumCircuit(2)
    qc.x(1)
    return qc
...
q_θ: ┌─────────┐
      |          |
q_1: ┌──X──┐
      |          |
      +-----+
...
q_θ: ┌─────────┐
      |          |
q_1: ┌──X──┐
      |          |
      +-----+
...
```

La funzione di Deutsch:

```

def QDeutsch(U):
    circuit = QuantumCircuit(2,1)
    circuit.x(1)
    circuit.barrier()
    circuit.h(0)
    circuit.h(1)
    circuit.barrier()

    circuit = circuit.compose(U,[0,1])
    circuit.barrier()
    circuit.h(0)
    circuit.measure(0,0)
    qcl.draw_circuit(circuit)
```



12. Algoritmo di Deutsch-Jozsa

L'algoritmo opera su una funzione booleana che prende come input n valori e restituisce un solo output.

$$f: \{0, 1\}^n \rightarrow \{0, 1\}$$

Distingue se la funzione è costante o bilanciata (una funzione è bilanciata quando il numero di input che restituisce il valore 0 è uguale al numero di input che restituisce 1), (le applicazioni sono poche o nulle, l'utilizzo è prettamente di scopo didattico).

L'algoritmo presuppone che la funzione sia bilanciata o costante, dobbiamo quindi "fidarci" che la funzione data faccia parte di una di queste due categorie.

Il primo algoritmo che riesce a dare uno speed-up esponenziale rispetto alla computazione classica.

Poiché una soluzione classica al problema prevede un numero di passi pari a $O(N)$ (input: 2^n), mentre per l'Algoritmo di Deutsch-Jozsa gli basta un'unica valutazione della funzione.

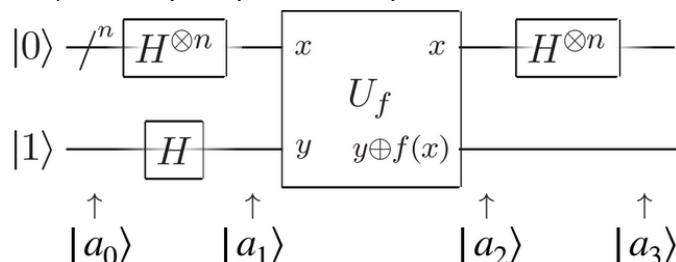
L'algoritmo di Deutsch-Jozsa è una generalizzazione dell'algoritmo di Deutsch.

Esempi di funzioni costanti/bilanciate:

	$f: \{0, 1\}^2 \rightarrow \{0, 1\}$		
	$f(x_1 x_2) = \begin{cases} 1 & \text{if } (x_1 \oplus x_2) = 0 \\ 0 & \text{if } (x_1 \oplus x_2) = 1 \end{cases}$	$f(00) = 0$	$f(10) = 1$
bilanciata \rightarrow		$f(01) = 1$	$f(11) = 0$
	$f: \{0, 1\}^2 \rightarrow \{0, 1\}$		
	$f(x_1 x_2) = 1, \forall x_1 x_2 \in \{0, 1\}^2$	$f(00) = 1$	$f(10) = 1$
costante \rightarrow		$f(01) = 1$	$f(11) = 1$
	$f: \{0, 1\}^3 \rightarrow \{0, 1\}$		
	$f(x_1 x_2 x_3) = \begin{cases} 1 & \text{if } (x_1 \oplus x_2 \oplus x_3) = 0 \\ 0 & \text{if } (x_1 \oplus x_2 \oplus x_3) = 1 \end{cases}$	$f(000) = 1$	$f(010) = 0$
bilanciata \rightarrow		$f(001) = 0$	$f(011) = 1$
		$f(100) = 0$	$f(110) = 1$
		$f(101) = 1$	$f(111) = 0$

La procedura:

- Il vettore iniziale di input a_0 dove $a_0(0^n 1) = 1$ ovvero tutti gli input sono 0 tranne l'ultimo qubit che è inizializzato ad 1. Ricordiamo che l'ultima linea di qubit è inizializzata a 1 perché (come per Deutsch) ci permette di utilizzare il boolean oracle come phase oracle.
- Il vettore successivo a_1 è il risultato dell'applicazione di Hadamard ad ogni qubit separatamente. Ricordiamo che applicando l'operatore H all'ultimo qubit, impostato ad 1, otteniamo una fase negativa.
- Il vettore a_2 è il risultato dell'applicazione del boolean oracle $U_{f^{-1}}$ dove $f^{-1}(xy) = x(f(x) \oplus y)$, ma siccome l'ultimo qubit ha una fase negativa si comporterà come un phase oracle.
- Il vettore a_3 è il risultato dell'applicazione dell'operatore H a tutti gli $a_0(0^n)$ qubit, ma non all'ultimo (in quanto lo scartiamo). Il tutto per riportare l'output nello stato iniziale (non sovrapposizione).



Analisi:

- Il vettore a_0 al primo passo è il vettore avente n copie di 0 tranne un 1 nella posizione 1:

$$a_0 = |0^n 1\rangle = \begin{bmatrix} 0 \\ 1 \\ 0 \\ \vdots \\ 0 \end{bmatrix}$$

- Adesso si applica la matrice di Hadamard a tutte le linee di qubit (cioè si applica H^{n+1} volte) in maniera indipendente. Ricordiamo che applicando H a 0 si ottiene $|+\rangle$, applicando H a 1 si ottiene $|-\rangle$.

Indispensabile poiche' grazie a cio' e' possibile utilizzare un boolean oracle come phase oracle

$$a_1 = H^{\otimes(n+1)} |0^n 1\rangle = |+\rangle^{\otimes n} \otimes |-\rangle$$

$$= \left(\frac{1}{\sqrt{2}} \begin{bmatrix} 1 \\ 1 \end{bmatrix} \right)^{\otimes n} \otimes \frac{1}{\sqrt{2}} \begin{bmatrix} 1 \\ -1 \end{bmatrix} = \frac{1}{2\sqrt{2^n}} \begin{bmatrix} 1 \\ -1 \\ 1 \\ -1 \\ \vdots \\ 1 \\ -1 \end{bmatrix}$$

- Possiamo dimostrare che:

$$\text{For all } xy \longrightarrow a_1(xy) = \frac{1}{\sqrt{2N}} (-1)^y$$

Dimostrazione:

$$\begin{aligned} a_1(xy) &= \frac{1}{\sqrt{2N}} \sum_{t,u} (-1)^{x \cdot t} (-1)^{y \cdot u} a_0(tu) \\ &= \frac{1}{\sqrt{2N}} (-1)^{x \cdot 0} (-1)^{y \cdot 1} \\ &= \frac{1}{\sqrt{2N}} (-1)^y \end{aligned}$$

$$a_0 = |0^n 1\rangle = \begin{bmatrix} 0 \\ 1 \\ 0 \\ \vdots \\ 0 \end{bmatrix}$$

- Possiamo dimostrare che, applicando il phase oracle otteniamo:

$$\text{For all } xy \longrightarrow a_2(xy) = \frac{1}{\sqrt{2N}} (-1)^{f(x) \oplus y}$$

Dimostrazione:

$$\begin{aligned} a_2(xy) &= a_1(x(f(x) \oplus y)) \\ &= \frac{1}{\sqrt{2N}} (-1)^{f(x) \oplus y} \end{aligned}$$

- Possiamo dimostrare che:

For all xy

$$|a_3(xy)|^2 = \frac{1}{2N^2} \left| \sum_t (-1)^{x \cdot t} (-1)^{f(t)} \right|^2$$

Dimostrazione:

$$a_3(xy) = \frac{1}{\sqrt{N}} \sum_t (-1)^{x \cdot t} a_2(ty)$$

$$= \frac{1}{\sqrt{2N}} \sum_t (-1)^{x \cdot t} (-1)^{f(t) \oplus y}$$

6. Possiamo dimostrare che:

Il misuramento del vettore a_3 ritorna $0^n y$, per qualche y , se e soltanto se f è una funzione costante. Quindi l'algoritmo di Deutsch-Jozsa distingue se la funzione f è costante o bilanciata usando una sola valutazione di $U_{f^{-1}}$.

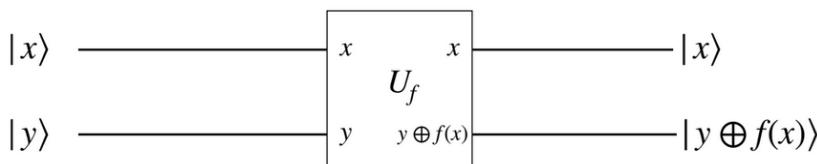
Dimostrazione:

$$\left| a_3(xy) \right|^2 = \frac{1}{2N} \left| \sum_t (-1)^{x \cdot t} (-1)^{f(t)} \right|^2$$

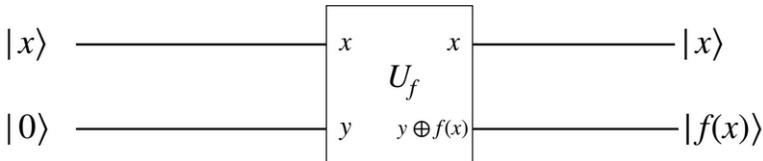
$$\left| a_3(0^n y) \right|^2 = \frac{1}{2N} \left| \sum_t (-1)^{f(t)} \right|^2$$

Perché tutto ciò funziona?

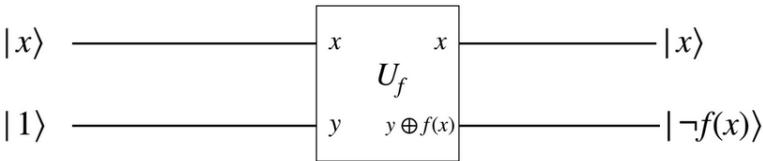
Noi sappiamo che quando applichiamo il boolean oracle ad un input $|x\rangle$ e un output $|y\rangle$, $|x\rangle$ viene lasciato invariato e la linea di output $|y\rangle$ otteniamo $|y \oplus f(x)\rangle$



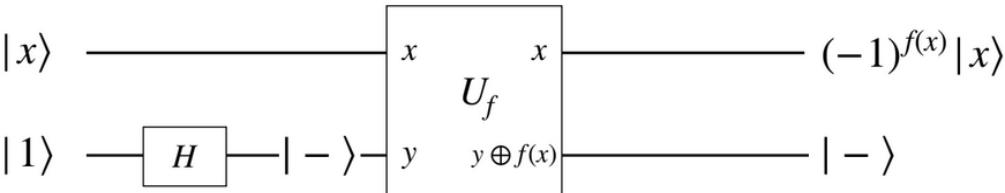
Se $y=0$ allora l'output dell'ultima riga lascierà $f(x)$ invariato:



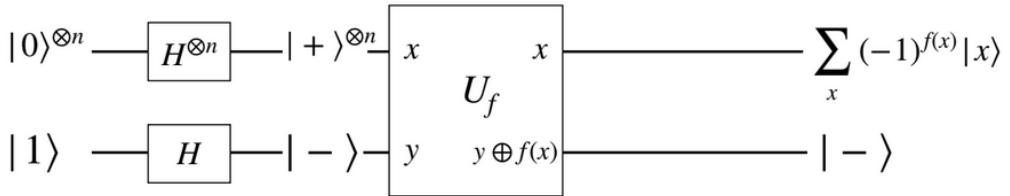
Se, invece, $y=1$ otteniamo la negazione a $f(x)$:



Se applichiamo la matrice di Hadamard ad 1 sappiamo che il boolean oracle si comporta come un phase oracle. Il risultato è sempre x invariato ma a cui abbiamo aggiunto una fase negativa quanto $f(x)=1$. Quindi quando $f(x)=1$ diventa $-x$, quando $f(x)=0$ la x rimane invariata:



Adesso si inizializzano tutti i qubit a 0 e si applica Hadamard, da ciò otteniamo una superimposizione di 0 e 1 (e come se tutti i possibili input, che vanno da 0 a 2^n , fossero messi in superimposizione) di conseguenza l'output del phase oracle sarà una superimposizione di tutti quanti i possibili input x .



Indichiamo con X_0 gli input che NON danno una soluzione per la funzione f . Mentre con X_1 gli input che danno una soluzione, ovvero i valori x tali che $f(x) = 1$:

$$X_0 = \{x: f(x) = 0\} \quad X_1 = \{x: f(x) = 1\}$$

Da ciò possiamo dire che il valore di output della prima riga è dato da:

$$\sum_x (-1)^{f(x)} |x\rangle = \sum_{x \in X_0} |x\rangle - \sum_{x \in X_1} |x\rangle \left\{ \begin{array}{ll} \sum_x |x\rangle = |+\rangle^{\otimes n} & \text{se } X_1 = \emptyset \\ -\sum_x |x\rangle = -|+\rangle^{\otimes n} & \text{se } X_0 = \emptyset \\ |\psi\rangle \in \pm \{|+\rangle, |-\rangle\}^n \setminus \{|+\rangle^n\} & \text{se } X_0 \neq \emptyset, X_1 \neq \emptyset \end{array} \right. |+\rangle$$

Se la funzione è costante si ottiene sempre una sequenza di n zeri. Possiamo definirla una **superimposizione distruttiva**, in quanto i qubit si annullano tra di loro e il risultato è 0. Ecco perché nei primi due casi otteniamo un valore 0 ripetuto n volte.

Mentre altrimenti è una funzione bilanciata, possiamo ottenere quindi una sequenza di qubit che sono o nello stato + o nello stato -. L'unico caso che non si può verificare è una sequenza di n +. L'importante quindi è che ci sia almeno un - da qualche parte (**superimposizione costruttiva**), ottenendo qualcosa di diverso da zero.

L'algoritmo così facendo è **deterministico**, ovvero quando si misura se si ottiene tutti zero vuol dire che la funzione è costante, se si ottiene qualcosa di diverso da zero allora è bilanciata.

Ricordiamo che abbiamo assunto all'inizio che la funzione di input o è bilanciata o è costante.

Graficamente si capisce se una funzione è bilanciata quando c'è almeno una fase negativa applicato o solo ad α o solo a β ; la presenza di una "croce", ovvero una fase negativa applicata contemporaneamente ad α e β , non cambia la distribuzione delle soluzioni ma solo il segno, e ciò nella misurazione non viene considerato.

*esempi e visualizzazione vedi slide da pagina 22.

Implementazione:

Implementazione dell'oracle:

```

def dj_oracle(case, n):
    oracle_qc = QuantumCircuit(n+1)
    if case == "balanced":
        b = np.random.randint(1,2**n)
        b_str = format(b, '0'+str(n)+'b')
        for qubit in range(len(b_str)):
            if b_str[qubit] == '1':
                oracle_qc.x(qubit)
        for qubit in range(n):
            oracle_qc.cx(qubit, n)
        for qubit in range(len(b_str)):
            if b_str[qubit] == '1':
                oracle_qc.x(qubit)
    if case == "constant":
        output = np.random.randint(2)
        if output == 1:
            oracle_qc.x(n)
    print("Oracle: ",oracle_qc.draw())
    oracle_gate = oracle_qc.to_gate()
    oracle_gate.name = "Oracle"
    return oracle_gate

```

Implementazione dell'algoritmo:

```

def dj_algorithm(oracle, n):
    dj_circuit = QuantumCircuit(n+1, n)
    dj_circuit.x(n)
    dj_circuit.h(n)
    for qubit in range(n):
        dj_circuit.h(qubit)
    dj_circuit.append(oracle, range(n+1))
    for qubit in range(n):
        dj_circuit.h(qubit)
    for i in range(n):
        dj_circuit.measure(i, i)
    return dj_circuit

n = 4
oracle_gate = dj_oracle('balanced', n)
dj_circuit = dj_algorithm(oracle_gate, n)
print(dj_circuit.draw())
#Simulazione
aer_sim = Aer.get_backend('aer_simulator')
transpiled_dj_circuit = transpile(dj_circuit, aer_sim)
qobj = assemble(transpiled_dj_circuit)
results = aer_sim.run(qobj).result()
answer = results.get_counts()
plot_histogram(answer)

```

13. Algoritmo di Bernstein-Vazirani

L'algoritmo deterministico di Bernstein-Vazirani può essere visto come un'estensione dell'algoritmo di Deutsch-Jozsa.

L'algoritmo opera su una funzione booleana che prende come input n valori e restituisce un solo output.

$$f: \{0, 1\}^n \rightarrow \{0, 1\}$$

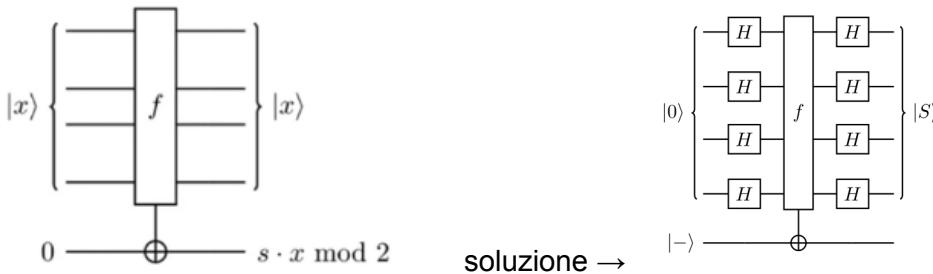
La funzione f garantisce che si ritorna il prodotto (prodotto interno) bit a bit tra la stringa di input x con qualche stringa s , dove entrambe le stringhe hanno lunghezza n . In altre parole, dato un input x :

$$f(x) = x \cdot s \pmod{2}$$

Ci si aspetta di trovare s .

Semplificando si fa il prodotto bit a bit tra x e s successivamente si contano il numero di 1, se è pari il risultato è 0, se è dispari il risultato è 1. L'obiettivo è trovare s .

Ci viene dato un oracle che descrive un modo per trovare una soluzione alla funzione dato un input; per quanto riguarda la funzione abbiamo la certezza che il risultato è dato dal prodotto interno booleano dell'input con una stringa s , ma non siamo a conoscenza di tale stringa. L'obiettivo è appunto trovarla.



Esempi:

Example #1

$$f: \{0, 1\}^2 \rightarrow \{0, 1\}$$

$$f(x_1 x_2) = \begin{cases} 1 & \text{if } (x_1 \oplus x_2) = 0 \\ 0 & \text{if } (x_1 \oplus x_2) = 1 \end{cases}$$

$$f(00) = 0 \cdot 1 \oplus 0 \cdot 1 = 0 \oplus 0 = 0$$

$$f(01) = 0 \cdot 1 \oplus 1 \cdot 1 = 0 \oplus 1 = 1$$

$$f(10) = 1 \cdot 1 \oplus 0 \cdot 1 = 1 \oplus 0 = 1$$

$$f(11) = 1 \cdot 1 \oplus 1 \cdot 1 = 1 \oplus 1 = 0$$

$$s = 11$$

Example #2

$$f: \{0, 1\}^2 \rightarrow \{0, 1\}$$

$$f(x_1 x_2) = x_1, \forall x_1 x_2 \in \{0, 1\}^2$$

$$f(00) = 0 \cdot 1 \oplus 0 \cdot 0 = 0 \oplus 0 = 0$$

$$f(01) = 0 \cdot 1 \oplus 1 \cdot 0 = 0 \oplus 0 = 0$$

$$f(10) = 1 \cdot 1 \oplus 0 \cdot 0 = 1 \oplus 0 = 1$$

$$f(11) = 1 \cdot 1 \oplus 1 \cdot 0 = 1 \oplus 0 = 1$$

$$s = 10$$

Example #3

$$f: \{0, 1\}^3 \rightarrow \{0, 1\}$$

$$f(x_1 x_2 x_3) = x_1 \oplus x_3$$

$$f(000) = 0 \quad f(010) = 0$$

$$f(001) = 1 \quad f(011) = 1$$

$$f(100) = 1 \quad f(110) = 1$$

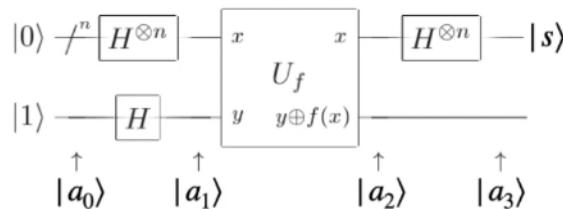
$$f(101) = 0 \quad f(111) = 0$$

$$s = 101$$

Per far ciò nel caso classico ci vogliono n interrogazioni. Con un sistema quantistico ne basta 1.

Procedura e analisi:

- Si inizializzano gli n qubits a 0 ($|0\rangle^{\otimes n}$) e il qubit di output ad 1 ($|1\rangle$).
- Il passo successivo è applicare Hadamard a tutte le linee di qubit. Le linee di input che sono 0 assumeranno una fase positiva ($|+\rangle$), anche la linea di output avrà una fase positiva solo che c'è un segno negativo prima ($-|+\rangle$).
- Si applica il Boolean Oracle $U_{f^{-1}}$
- Si applica Hadamard ad i primi n qubit di input, restituendo $|s\rangle$.



L'analisi:

Applicando Hadamard ad n linee di qubit otteniamo:

$$|a\rangle \xrightarrow{H^{\otimes n}} \frac{1}{\sqrt{2^n}} \sum_{x \in 0,1^n} (-1)^{x \cdot a} |x\rangle$$

$(-1)^{x \cdot a}$ è il prodotto interno tra l'input x ed a il vettore iniziale.

Per esempio se:

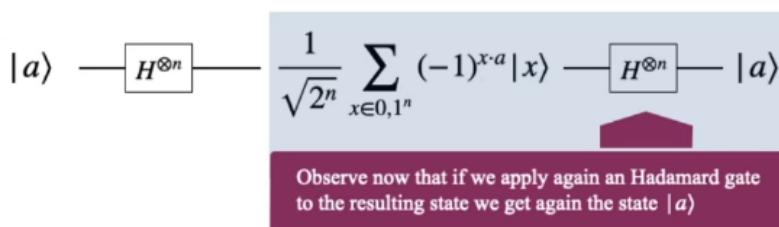
$$|a\rangle = |01\rangle = \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \end{bmatrix} \quad H^{\otimes 2}|a\rangle = \frac{1}{2} \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & -1 & 1 & -1 \\ 1 & 1 & -1 & -1 \\ 1 & -1 & -1 & 1 \end{bmatrix} \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \end{bmatrix} = \frac{1}{2} \begin{bmatrix} 1 \\ -1 \\ 1 \\ -1 \end{bmatrix} \quad \begin{array}{l} 00 \cdot 01 = 0 \\ 01 \cdot 01 = 1 \\ 10 \cdot 01 = 0 \\ 11 \cdot 01 = 1 \end{array}$$

oppure, ancora:

$$|a\rangle = |11\rangle = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix} \quad H^{\otimes 2}|a\rangle = \frac{1}{2} \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & -1 & 1 & -1 \\ 1 & 1 & -1 & -1 \\ 1 & -1 & -1 & 1 \end{bmatrix} \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix} = \frac{1}{2} \begin{bmatrix} 1 \\ -1 \\ -1 \\ 1 \end{bmatrix} \quad \begin{array}{l} 00 \cdot 11 = 0 \\ 01 \cdot 11 = 1 \\ 10 \cdot 11 = 1 \\ 11 \cdot 11 = 0 \end{array}$$

Da questi esempi deduciamo che l'applicazione del gate di Hadamard allo stato $|a\rangle$ crea una superimposizione di tutti i possibili stati aggiungendo una fase negativa a tutti quei input x per cui $x \cdot a = 1$, viceversa se $x \cdot a = 0$ la fase negativa non viene applicata.

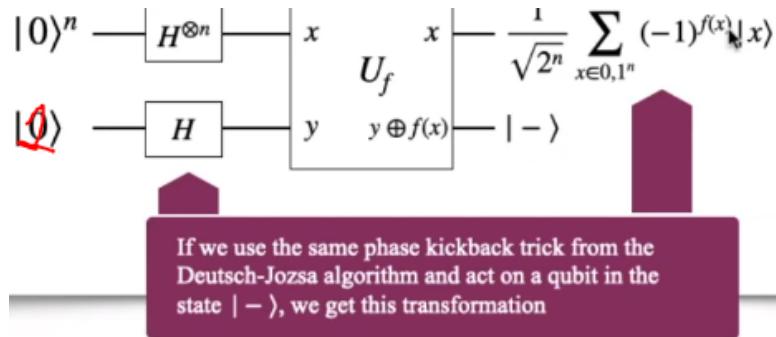
Osservazione:



Se, invece di partire da un vettore qualsiasi a , si parte da $|0\rangle^n$ la fase negativa si annullerebbe, perché il vettore in considerazione è $|00\rangle$ e facendo il prodotto booleano con qualsiasi input il risultato è sempre zero, in quanto la componente $(-1)^{x \cdot a}$ si annulla ottenendo solo $|x\rangle$ nella sommatoria.

$$|0\rangle^n \xrightarrow{H^{\otimes n}} \frac{1}{\sqrt{2^n}} \sum_{x \in 0,1^n} |x\rangle \quad |a\rangle = |00\rangle = \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \end{bmatrix} \quad H^{\otimes 2}|a\rangle = \frac{1}{2} \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & -1 & 1 & -1 \\ 1 & 1 & -1 & -1 \\ 1 & -1 & -1 & 1 \end{bmatrix} \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \end{bmatrix} = \frac{1}{2} \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \end{bmatrix} \quad \begin{array}{l} 00 \cdot 00 = 0 \\ 01 \cdot 00 = 0 \\ 10 \cdot 00 = 0 \\ 11 \cdot 00 = 0 \end{array}$$

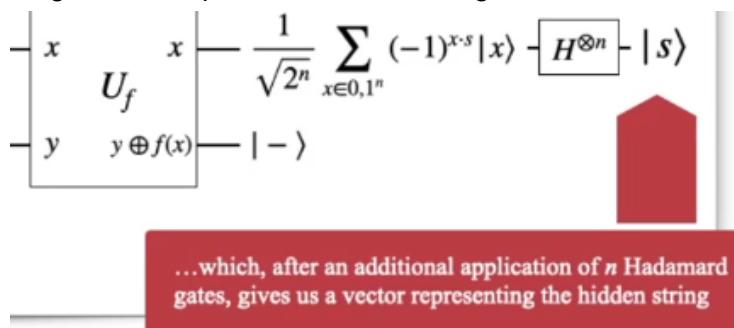
Si applica lo stesso stratagemma di Deutsch-Jozsa per utilizzare un boolean oracle come phase orale:



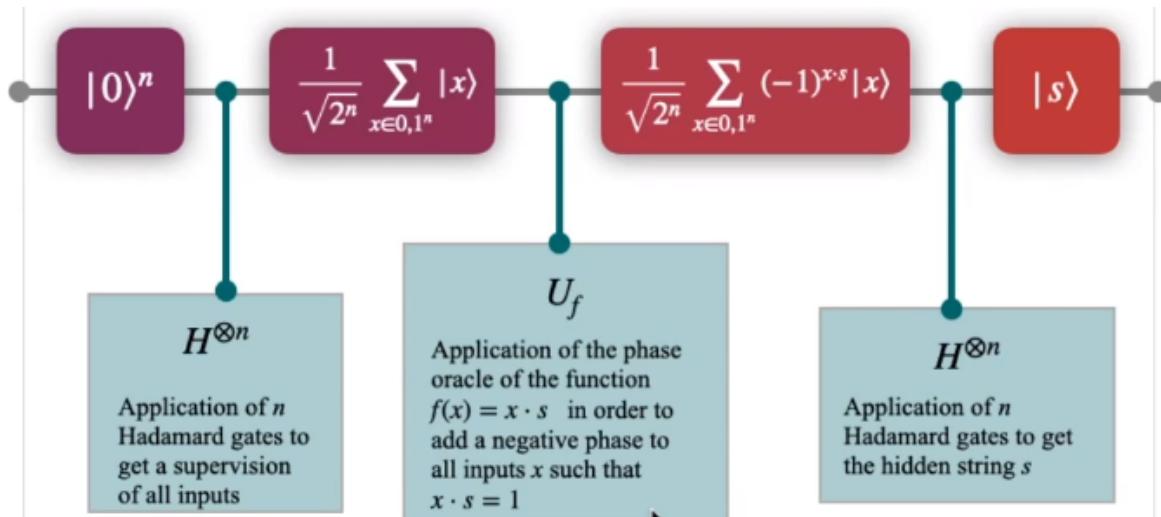
Arrivati a questo punto si sostituisce $f(x)$ con $x \cdot s$ e applicando il seguente principio:

$$\text{Now remember that } \frac{1}{\sqrt{2^n}} \sum_{x \in \{0,1\}^n} (-1)^{x \cdot a} |x\rangle \longrightarrow H^{\otimes n} \longrightarrow |a\rangle$$

Infine, applicando n volte il gate H all'input otteniamo la stringa nascosta s :



Ricapitolando:



"Sapendo che dalla superimposizione di tutti gli input a cui si aggiunge una fase negativa nei punti in cui il prodotto interno dell'input con una variabile a dà esattamente 1 e applicando il gate H si ottiene a , allora si fa in modo che la prima linea di qubit sia la superimposizione di tutti i possibili input aggiungendo una fase negativa negli input in cui il prodotto interno tra $x \cdot s = 1$ e applicando la matrice H si ottiene s ".

Esempio:

$$f: \{0,1\}^2 \longrightarrow \{0,1\}$$

$$f(x_1 x_2) = \begin{cases} 1 & \text{if } x_1 \oplus x_2 = 0 \\ 0 & \text{if } x_1 \oplus x_2 = 1 \end{cases}$$

$$f(00) = 0 \quad f(10) = 1$$

$$f(01) = 1 \quad f(11) = 0$$

Solution

$$s = 11$$

$$f(00) = 00 \cdot 11 = 0$$

$$f(01) = 01 \cdot 11 = 1$$

$$f(10) = 10 \cdot 11 = 1$$

$$f(11) = 11 \cdot 11 = 0$$

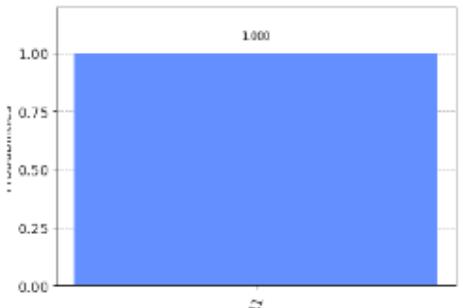
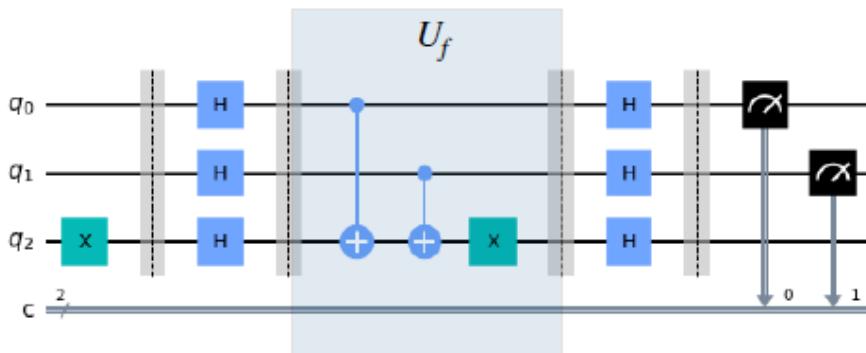
	000	001	010	011	100	101	110	111
000	1							
001		1						
010			1					
011				1				
100						1		
101							1	
110								1
111								1

$$a_0 = e_{001} = |001\rangle = \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \end{bmatrix} \otimes \begin{bmatrix} 0 \\ 1 \end{bmatrix} \rightarrow a_1 = H|00\rangle \otimes H|1\rangle = \frac{1}{2} \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & -1 & 1 & -1 \\ 1 & 1 & -1 & -1 \\ 1 & -1 & -1 & 1 \end{bmatrix} \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \end{bmatrix} \otimes \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \begin{bmatrix} 0 \\ 1 \end{bmatrix}$$

$$= \frac{1}{2} \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \end{bmatrix} \otimes \frac{1}{2\sqrt{2}} \begin{bmatrix} 1 \\ -1 \\ 1 \\ -1 \end{bmatrix} = \frac{1}{2\sqrt{2}} \begin{bmatrix} 1 \\ -1 \\ 1 \\ -1 \end{bmatrix} \leftarrow \text{qui un errorino, vedi sotto}$$

$$a_2 = U_f a_1 = \frac{1}{2\sqrt{2}} \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 \\ -1 \\ 1 \\ -1 \\ 1 \\ -1 \\ 1 \\ -1 \end{bmatrix} \rightarrow a_3 = \frac{1}{4} \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & -1 & 1 & -1 \\ 1 & 1 & -1 & -1 \\ 1 & -1 & -1 & 1 \end{bmatrix} \begin{bmatrix} 1 \\ -1 \\ -1 \\ 1 \end{bmatrix} \otimes \frac{1}{\sqrt{2}} \begin{bmatrix} 1 \\ -1 \end{bmatrix}$$

$$= \frac{1}{2\sqrt{2}} \begin{bmatrix} 1 \\ -1 \\ -1 \\ 1 \\ 1 \\ -1 \\ 1 \\ -1 \end{bmatrix} = \frac{1}{2} \begin{bmatrix} 1 \\ -1 \\ -1 \\ 1 \end{bmatrix} \otimes \frac{1}{\sqrt{2}} \begin{bmatrix} 1 \\ -1 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix} \otimes \frac{1}{\sqrt{2}} \begin{bmatrix} 1 \\ -1 \end{bmatrix} = |11\rangle \otimes \frac{|0\rangle - |1\rangle}{\sqrt{2}}$$



Implementazione:

```
#BERNSTEIN–VAZIRANI
s = '1011' # the hidden binary string
n = len(s) # number of qubits used to represent s

# We need a circuit with n qubits, plus one auxiliary qubit
# Also need n classical bits to write the output to
bv_circuit = QuantumCircuit(n+1, n)

# put auxiliary in state |-
bv_circuit.h(n)
bv_circuit.z(n)

# Apply Hadamard gates before querying the oracle
for i in range(n):
    bv_circuit.h(i)

# Apply barrier
bv_circuit.barrier()

# Apply the inner-product oracle
s = s[::-1] # reverse s to fit qiskit's qubit ordering
for q in range(n):
    if s[q] == '0':
        bv_circuit.i(q)
    else:
        bv_circuit.cx(q, n)

#Apply Hadamard gates after querying the oracle
for i in range(n):
    bv_circuit.h(i)

# Measurement
for i in range(n):
    bv_circuit.measure(i, i)

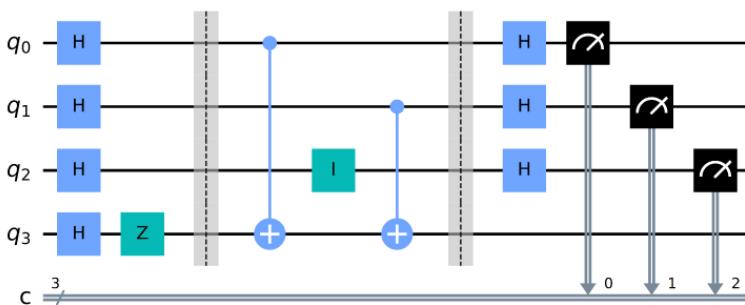
print(bv_circuit.draw())

# use local simulator
aer_sim = Aer.get_backend('aer_simulator')
shots = 1024
qobj = assemble(bv_circuit)
results = aer_sim.run(qobj).result()
answer = results.get_counts()

plot_histogram(answer)

sn=list(answer)[0][::-1]
if(sn==s):
    print("Risultato corretto")
else:
    print("Errore, le stringhe non combaciano: ",sn,"-",s)

plot_histogram(answer)
```



14. Algoritmo di Simon

Non si tratta di un algoritmo deterministico. Ha una probabilità molto alta di restituire gli output corretti, ma per una serie di rari e sfortunati casi è possibile che non lo siano.

Ci viene data una funzione nascosta f , che è garantita essere one-to-one (1:1) o two-to-one (2:1), dove tali funzioni possiedono le seguenti proprietà:

- one-to-one (biettiva): mappa esattamente un unico output per ciascun input. Un esempio di funzione che prende 4 input è:

$$f(1) \rightarrow 1, f(2) \rightarrow 2, f(3) \rightarrow 3, f(4) \rightarrow 4$$

- two-to-one (periodica): mappa esattamente 2 input su ogni output univoco. Un esempio di funzione che prende 4 input è:

$$f(1) \rightarrow 1, f(2) \rightarrow 2, f(3) \rightarrow 1, f(4) \rightarrow 2$$

Questa mappatura two-to-one è in accordo a una stringa di bit nascosta, s , dove:

dati x_1, x_2 : $f(x_1) = f(x_2)$

e' garantito: $x_1 \otimes x_2 = s$

In questo caso f si dice periodica con periodo s .

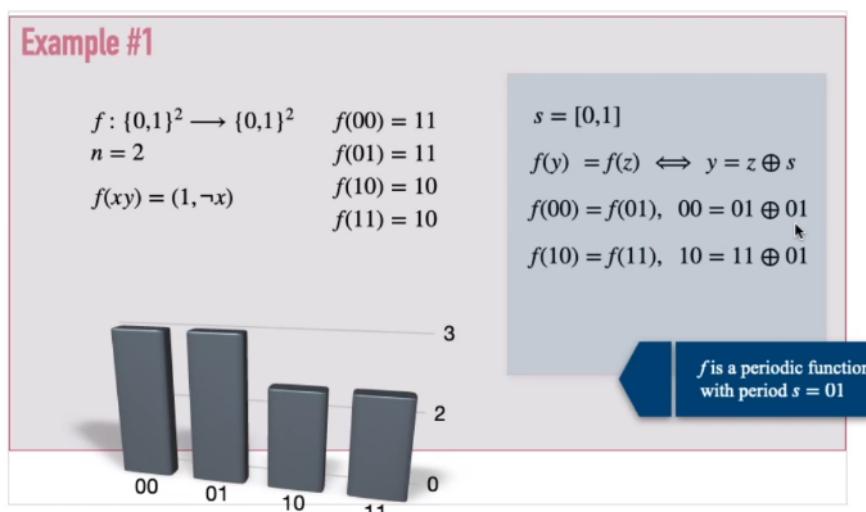
L'obiettivo è individuare questa stringa s che caratterizza la funzione. Se questa stringa s è fatta da tutti zero allora la funzione è biettiva, ovvero che ad ogni input è associato un output diverso.

Quindi, l'algoritmo di Simon riceve in input una funzione che abbiamo la promessa essere:

- uno-a-uno (biettiva), ogni input è mappato su un unico output.
- oppure, due-a-uno (periodica), dove ogni coppia di input è mappata su un unico output diverso.

L'algoritmo di Simon riesce a capire tra questi due casi. Qualora fosse periodica riesce a trovare la stringa s che definisce la caratterizzazione della periodicità.

Esempio 1:



Si tratta di una funzione periodica poiché ci sono coppie di input che hanno lo stesso output e $s = [0, 1]$ è la stringa nascosta che stiamo cercando.

Esempio 2:

Example #2

$$f: \{0,1\}^3 \longrightarrow \{0,1\}^3$$

$$n = 3$$

$$\begin{aligned} f(000) &= 111 & f(100) &= 101 \\ f(001) &= 101 & f(101) &= 111 \\ f(010) &= 000 & f(110) &= 100 \\ f(011) &= 100 & f(111) &= 000 \end{aligned}$$

$$s = [1,0,1]$$

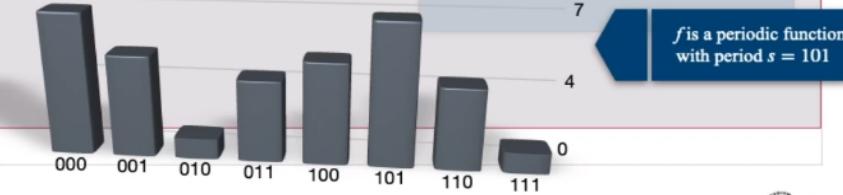
$$f(y) = f(z) \iff y = z \oplus s$$

$$f(000) = f(101), \quad 000 = 101 \oplus 101$$

$$f(001) = f(100), \quad 001 = 100 \oplus 101$$

$$f(010) = f(111), \quad 010 = 111 \oplus 101$$

$$f(011) = f(110), \quad 011 = 110 \oplus 101$$



Esempio 3:

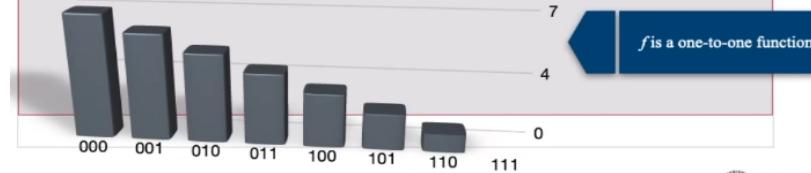
Example #3

$$f: \{0,1\}^3 \longrightarrow \{0,1\}^3$$

$$n = 3$$

$$\begin{aligned} f(000) &= 111 & f(100) &= 011 \\ f(001) &= 110 & f(101) &= 010 \\ f(010) &= 101 & f(110) &= 001 \\ f(011) &= 100 & f(111) &= 000 \end{aligned}$$

f is a periodic function with period s = 101



Si tratta di una funzione uno-a-uno (biettiva) in quanto tutti gli output sono collegati ad input tutti diversi tra loro. La stringa s è composta da tutti quanti zero.

Nel caso classico ci vorrebbero nel caso peggiore 2^{n-1} controlli. Mentre con un algoritmo quantistico bastano n .

L'idea di base dell'algoritmo di Simon è costruire un circuito quantistico che ci permette di trovare $n - 1$ stringhe di lunghezza n linearmente indipendenti, chiamati:

$$z_1, z_2, \dots, z_{n-1} \in \{0, 1\}^n$$

in maniera tale da costruire il sistema:

$$\left\{ \begin{array}{l} z_1 \cdot s = 0 \\ z_2 \cdot s = 0 \\ \vdots \\ z_{n-1} \cdot s = 0 \end{array} \right.$$

dove:

$$z_i \cdot s = (z_i)_1 s_1 \oplus (z_i)_2 s_2 \oplus \dots \oplus (z_i)_n s_n$$

Se otteniamo $n - 1$ equazioni di questo tipo allora possiamo trovare s . In altre parole creiamo un sistema in maniera tale da trovare tutti i quanti i valori s_n che compongono la stringa s .

Quello che fa il circuito quantistico è trovare le stringhe z_1, z_2, \dots, z_{n-1} che moltiplicate s danno zero in maniera tale da creare un sistema che se risolto ci da s .

Tale sistema si risolve in maniera classica, per esempio con il metodo di [eliminazione di Gauss](#). Il circuito quantistico serve per ottenere solo le stringhe z_1, z_2, \dots, z_{n-1} . È un mix di computazione classica e quantistica.

Example #2

$$n = 3 \quad f: \{0,1\}^3 \longrightarrow \{0,1\}^3 \quad s = 101$$

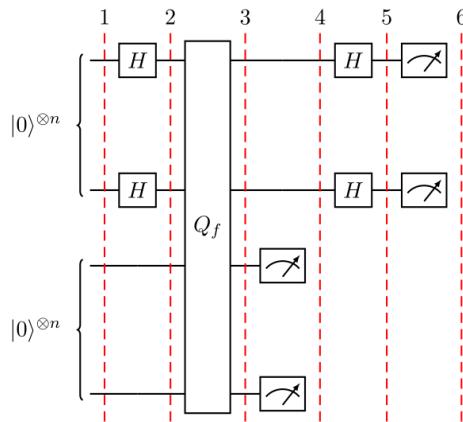
$$\begin{array}{ll} f(000) = 111 & f(100) = 101 \\ f(001) = 101 & f(101) = 111 \\ f(010) = 000 & f(110) = 100 \\ f(011) = 100 & f(111) = 000 \end{array}$$

$$\begin{array}{ll} z_1 = 011 & \left\{ \begin{array}{l} z_1 \cdot s = 0 \\ z_2 \cdot s = 0 \end{array} \right. \\ z_2 = 010 & \left\{ \begin{array}{l} (0 \cdot 1) \oplus (1 \cdot 0) \oplus (1 \cdot 1) = 1 \oplus 1 \oplus 0 = 0 \\ (0 \cdot 1) \oplus (1 \cdot 0) \oplus (0 \cdot 1) = 1 \oplus 1 \oplus 1 = 0 \end{array} \right. \end{array}$$

$$\left\{ \begin{array}{l} (0 \cdot s_1) \oplus (1 \cdot s_2) \oplus (1 \cdot s_3) = 0 \\ (0 \cdot s_1) \oplus (1 \cdot s_2) \oplus (0 \cdot s_3) = 0 \end{array} \right.$$



L'algoritmo si compone di 5 step:



Questa volta come input ci sono due gruppi di qubit ciascuno dei quali a dimensione n .

Procedura e analisi:

1. Inizializziamo un insieme E di equazioni a zero (insieme vuoto).
2. Ciclo while finché E non ha un'unica soluzione, ovvero finché il sistema non è completo cioè non contiene abbastanza equazioni da poter ottenere una soluzione:
 - a. Eseguo un circuito quantistico ottenendo una nuova stringa z .
 - b. Che moltiplicata per s restituisce zero, $z \cdot s = 0$

Nota che se il circuito quantistico restituisce $z = 0^n$ la soluzione non è considerabile accettabile quindi si scarta. Si scarta anche se il circuito quantistico restituisce una soluzione già inserita nella lista E .
3. Ciclo while termina, soluzione del sistema trovata. Si risolve il sistema di equazioni per ottenere un'unica soluzione s (parte classica), per esempio con il metodo di [eliminazione di Gauss](#).
4. Se $s = 0^n$ la funzione f è biettiva.

Lo step 2 va ripetuto $O(n)$ volte per ottenere n equazioni linearmente indipendenti.

La procedura quantistica:

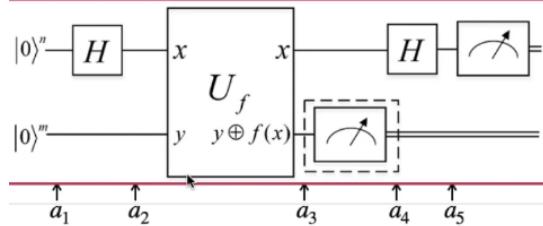
Concentriamoci nel circuito quantistico che ci permette di ottenere una stringa $z \cdot s = 0$ (dove \cdot è il prodotto booleano interno). I passi:

1. Come input si hanno due gruppi di qubit inizializzati a 0^n :
 $|0\rangle^{\otimes n} |0\rangle^{\otimes n}$
2. Si applica la matrice di Hadamard al primo gruppo di qubit.
3. Si applica l'oracle U_f .
4. Si misura solo il secondo gruppo di qubit.

"Come sappiamo quando l'oracle viene eseguito mette in correlazione l'input con l'output, nel momento in cui si misura la linea di output (secondo registro) si sta scegliendo una possibilita' tra le tante della

linea di output, quando si sceglie un possibile output automaticamente il primo registro (dato che sono registri correlati) si modifica."

5. Si applica al primo di nuovo una matrice di Hadamard.
6. Si misura il primo registro per ottenere la stringa booleana z



Ricapitolando: Inizializziamo a 0^n entrambi i registri (perché input e output hanno la stessa dimensione), si applica Hadamard solo al primo registro per metterlo in una superimposizione di tutti i possibili input, si applica l'oracle ottenendo sopra tutti i possibili input e sotto tutti i possibili output, pero' non appena si misura il secondo registro tra tutti i possibili output se ne sceglie solo uno, di conseguenza si modificano anche i registri dell'input, a questi registri si applica Hadamard e si misura ottenendo la stringa z che moltiplicata (inner product) per s dovrebbe essere uguale a 0.

Analisi:

1. Due registri di input vengono inizializzate a zero: $|a_1\rangle = |0\rangle^{\otimes n} |0\rangle^{\otimes n}$
2. Si applica il gate Hadamard al primo registro ottenendo:

$$|a_2\rangle = (H^{\otimes n}|0\rangle^{\otimes n})|0\rangle^{\otimes n} = |+\rangle^{\otimes n}|0\rangle^{\otimes n} = \frac{1}{\sqrt{2^n}} \sum_{x \in \{0,1\}^n} |x\rangle|0\rangle^{\otimes n}$$

Il primo registro si mette quindi in una superimposizione di tutti i possibili input.

3. Applicando l'oracle U_f (che non è un phase oracle, in quanto il secondo registro non è inizializzato a 1) a $|a_2\rangle$ si ottiene che nella linea di input passano invariati i valori quindi rimane la superimposizione di tutti quanti i possibili input pero' la linea di output da 0^n vengono trasformati nella superimposizione di tutti i possibili output ottenendo:

$$|a_3\rangle = U_f|a_2\rangle = \frac{1}{\sqrt{2^n}} \sum_{x \in \{0,1\}^n} |x\rangle|f(x)\rangle$$

*da notare che all'interno della sommatoria ci sarebbe dovuto essere $|x\rangle|y\rangle$ ma siccome $y=0$ rimane soltanto $f(x)$.

4. (questi ultimi sono dei qubit correlati) Si misura adesso il secondo registro. Tra tutti i possibili output ne viene scelto uno a caso $f(x)$. Il problema per come è imposto associa ad ogni output una coppia di input, quindi dal momento in cui si osserva un solo output automaticamente nel primo registro la superimposizione di tutti gli input si riduce alla superimposizione solo di due input, quelli che hanno dato l'output corrispondente nel secondo registro. Il secondo registro collassa e di conseguenza collassa anche il primo poiché sono correlati, in particolare collassa nei due unici input che mi danno quell'output.

Il primo registro diventa:

$$|a_4\rangle = \frac{1}{\sqrt{2^n}}(|x\rangle + |y\rangle)$$

dove x e y sono due input che danno lo stesso output che corrisponde a $f(x)$.

5. Applicando la matrice di Hadamard al primo registro, si ottiene:

$$|a_5\rangle = \frac{1}{\sqrt{2^{n+1}}} \sum_{z \in \{0,1\}^n} [(-1)^{x \cdot z} + (-1)^{y \cdot z}] |z\rangle$$

Se i valori $(-1)^{x \cdot z} (-1)^{y \cdot z}$ sono uguali allora non si avra' il valore 0^n e quindi la funzione è periodica.

6. Quindi misurando il primo registro avremo un output soltanto se $(-1)^{y \cdot z} = (-1)^{x \cdot z}$ che significa:

$$\begin{aligned}x \cdot z &= y \cdot z \\x \cdot z &= (x \oplus s) \cdot z \\x \cdot z &= x \cdot z \oplus s \cdot z \\s \cdot z &= 0 \pmod{2}\end{aligned}$$

Verrà misurata una stringa z , il cui inner product con $s = 0$.

7. Infine si risolve il sistema lineare mediante un metodo della computazione classica, per esempio il metodo di [eliminazione di Gauss](#).

[*un esempio](#)

Implementazione:

Alcune definizioni di funzioni utili:

1. procedura che compila ed esegue la simulazione del circuito e restituisce i *counts*;
2. realizza lo *XOR* tra due variabili a e b ;
3. restituisce un numero x in una stringa binaria di n bit.
4. calcola la rappresentazione binaria di x e ne fa' lo *XOR* con la stringa hs ;
5. calcola la rappresentazione binaria di x e restituisce le posizioni degli 1 contenenti in x ;
6. calcola la rappresentazione binaria di x e restituisce le posizioni degli 0 contenenti in x ;
7. calcola il prodotto interno booleano tra z e hs e ne fa il *mod 2*.

```
[1] def compile_and_run(circuit, shots):
    simulator = QasmSimulator()
    compiled_circuit = transpile(circuit, simulator)
    job = simulator.run(compiled_circuit, shots=shots)
    result = job.result()
    counts = result.get_counts(compiled_circuit)
    qcl.counts(counts)
    return counts

[2] def strxor(x, hs):
    # x è un numero, hs è una stringa
    r = ""
    n = len(hs)
    x = binary(x, n)
    for i in range(n):
        r = r + str(xor(x[i], hs[i]))
    return r

[3] def xor(a,b):
    if a==b:
        return 0
    return 1

[4] def binary(x,n):
    formatstr = "{:0>" + str(n) + "b}"
    x = formatstr.format(x)
    return x

[5]-[6] def bitset(x,n):
    x = binary(x,n)
    return [i for i in range(n) if x[i]=='1']

[7] def bitnotset(x,n):
    x = binary(x,n)
    return [i for i in range(n) if x[i]=='0']

[8] def cdot(z,hs):
    n = len(hs)
    r = 0
    for i in range(n):
        r = [r + (int(z[i])*int(hs[i]))] % 2
    return r
```

Definiamo l'algoritmo di Simon che prende in input la stringa nascosta hs . Crea una funzione che abbia la stringa s come caratterizzazione, e cerca una stringa che moltiplicata per s sia uguale a 0, rappresentato in figura [1] qui sotto. Definiamo, inoltre, la parte quantistica dell'algoritmo di Simon [2].

```

def SimonOracle(hs):
    n = len(hs)
    sol = [None]*(2**n)
    out = 1
    for x in range(2**n):
        if(sol[x]==None):
            sol[x] = out
            y = int(strxor(x,hs),2)
            sol[y] = out
            out = out+1
    #print(sol)

    qc = QuantumCircuit(2*n)
    for x in range(2**n):
        y = sol[x]
        for i in bitnotset(x,n):
            qc.x(i)
        for i in bitset(y,n):
            qc.mcx(list(range(n)),n+i)
        for i in bitnotset(x,n):
            qc.x(i)
    qc.barrier()
    qccl.draw_circuit(qc)
    return qc

[1]

def SimonQ(hs):
    n = len(hs)
    xr = QuantumRegister(n,"x")
    yr = QuantumRegister(n,"y")
    cr = ClassicalRegister(n, "cr")
    qc = QuantumCircuit(xr,yr,cr)

    qc.h(xr)

    oracle = SimonOracle(hs)
    qc = qc.compose(oracle, list(xr)+list(yr))

    qc.measure(yr,cr)

    qc.h(xr)
    qc.barrier()
    qc.measure(xr,cr)
    qccl.draw_circuit(qc)

    counts = compile_and_run(qc, 1)
    z = list(counts.keys())[0]
    if (cdot(z,hs)==0):
        print("Risultato corretto")
    else:
        print("Risultato non corretto")
    return z

[2]

```

Definiamo, infine, la funzione che restituisce il numero di iterazioni per la ricerca delle stringhe z:

```

def Simon(hs):
    n = len(hs)
    S = [None]*(2**n)
    nsol = 0
    iterazioni = 0
    while(nsol < n-1):
        z = SimonQ(hs)
        iterazioni = iterazioni + 1
        if(z!="0"*n and S[int(z,2)]==None):
            S[int(z,2)]=1
            nsol = nsol+1
    print("Soluzione trivata in ",iterazioni," iterazioni")

```

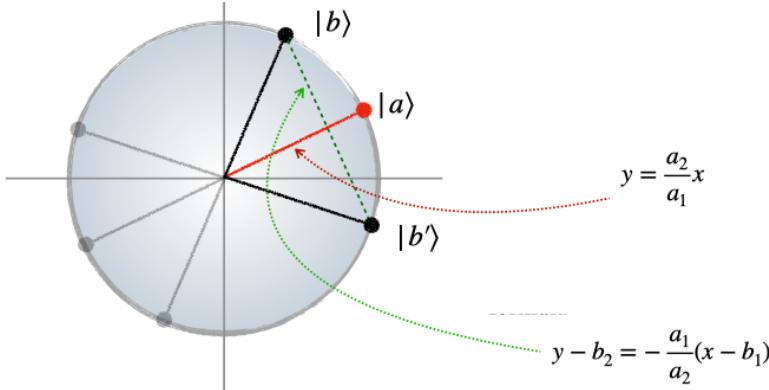
15. Riflessione

La riflessione è utile nell'algoritmo di ricerca di Grover.

Aventi due vettori a, b bisogna fare una rotazione di b in maniera tale da ottenere lo speculare di b rispetto ad a chiamato b' .

Il risultato sarà un operatore di riflessione rispetto ad a in maniera tale che applicato a qualsiasi vettore b permetterà di ottenere il simmetrico di b rispetto ad a ottenendo appunto b' .

$$|a\rangle = \begin{bmatrix} a_1 \\ a_2 \end{bmatrix} \quad |b\rangle = \begin{bmatrix} b_1 \\ b_2 \end{bmatrix} \quad |b'\rangle = \begin{bmatrix} ? \\ ? \end{bmatrix}$$



La retta del vettore $|a\rangle$ è una retta che passa attraverso l'origine e il punto $a = (a_1, a_2)$, con coefficiente angolare:

$$y = \frac{a_2}{a_1}x$$

La retta che unisce $|b\rangle$ a $|b'\rangle$ è ortogonale alla retta su cui giace $|a\rangle$, quindi il suo coefficiente angolare è:

$$y = -\frac{a_1}{a_2}$$

Tale retta perpendicolare passante per il punto $b = (b_1, b_2)$ è ottenibile dalla formula:

$$y - b_2 = -\frac{a_1}{a_2}(x - b_1)$$

Ottenimento di a' :

Siamo interessati al punto di intersezione delle due rette, chiamato a' , in particolare alle coordinate di tale punto che si ottiene mettendo a sistema le due rette:

$$\left\{ \begin{array}{l} y = \frac{a_2}{a_1}x \\ y = -\frac{a_1}{a_2}(x - b_1) - b_2 \end{array} \right.$$

$$\left\{ \begin{array}{l} x = \frac{a_1(a_1b_1 + a_2b_2)}{a_2^2 + a_1^2} = a_1(a_1b_1 + a_2b_2) \\ y = \frac{a_2(a_1b_1 + a_2b_2)}{a_2^2 + a_1^2} = a_2(a_1b_1 + a_2b_2) \end{array} \right. \begin{matrix} \text{si omettono in quanto essendo} \\ \text{vettori unitari sono }=1 \end{matrix}$$

$$a' = \begin{bmatrix} a_1(a_1b_1 + a_2b_2) \\ a_2(a_1b_1 + a_2b_2) \end{bmatrix} = (a_1b_1 + a_2b_2) \begin{bmatrix} a_1 \\ a_2 \end{bmatrix} = \left(\begin{bmatrix} a_1 & a_2 \end{bmatrix} \begin{bmatrix} b_1 \\ b_2 \end{bmatrix} \right) \begin{bmatrix} a_1 \\ a_2 \end{bmatrix} = a \langle a, b \rangle$$

Possiamo definire l'operatore di proiezione $P_{a'}$, noi proiettiamo b su a e otteniamo a'_1 . Questo operatore

$P_{a'}$ non è altro che una matrice. L'operatore è realizzato in maniera tale che per ogni vettore b , $P_a b$ rappresenta la proiezione di b su a .

In altre parole si crea l'operatore di proiezione su a e dato il vettore b si ottiene, facendo $P_a b$, la proiezione di b su a , cioè a_1 . L'operatore di proiezione è data da:

We define the **Projection Operator** P_a
such that for all b we have $P_a b = a \langle a, b \rangle$.

It is given by $P_a = |a\rangle\langle a|$

$$P_a b = (|a\rangle\langle a|) |b\rangle = |a\rangle\langle a, b\rangle$$

Quindi l'operatore di proiezione P_a è la matrice $n \times m$ ottenuta dalla moltiplicazione del vettore colonna a per il vettore riga a . Cio' ci permette, una volta dato a , dato qualsiasi vettore b di trovare la proiezione di b su a .

Ottenimento di b' :

Per ottenere $|b'\rangle$ consideriamo che $|a'\rangle$ è il punto di mezzo del segmento $\overline{b - b'}$, da ciò otteniamo:

$$\begin{aligned} a'_1 &= \frac{b_1 + b'_1}{2} & b'_1 &= 2a'_1 - b_1 \\ a'_2 &= \frac{b_2 + b'_2}{2} & b'_2 &= 2a'_2 - b_2 \end{aligned} \rightarrow |b'\rangle = \begin{bmatrix} 2a'_1 - b_1 \\ 2a'_2 - b_2 \end{bmatrix} = 2 \begin{bmatrix} a'_1 \\ a'_2 \end{bmatrix} - \begin{bmatrix} b_1 \\ b_2 \end{bmatrix} = 2|a'\rangle - |b\rangle$$

A questo punto è possibile definire l'operatore di riflessione $Ref_{|a\rangle}$ tale che per tutti $|b\rangle$ abbiamo:

$$\begin{aligned} |b'\rangle &= Ref_{|a\rangle} |b\rangle \\ Ref_{|a\rangle} |b\rangle &= 2|a'\rangle - |b\rangle \\ &= 2P_a |b\rangle - |b\rangle \\ &= (2P_a - I) |b\rangle \end{aligned}$$

$$Ref_{|a\rangle} = 2|a\rangle\langle a| - I$$

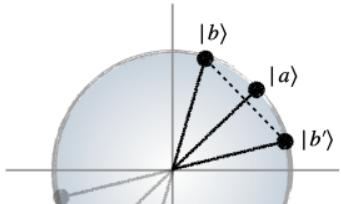
Questa formula rappresenta l'operatore di riflessione che permette di ottenere a partire da qualsiasi vettore a la riflessione di un altro vettore b .

Esempio:

For example, let a be the unit vector $a = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 \\ 1 \end{bmatrix}$

The projector operator is $P_a = |a\rangle\langle a| = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 \\ 1 \end{bmatrix} \times \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \end{bmatrix} = \frac{1}{2} \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix}$

The reflection operator is $Ref_a = 2P_a - I = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}$



$$|b\rangle = \frac{1}{5} \begin{bmatrix} 3 \\ 4 \end{bmatrix}$$

$$|b'\rangle = Ref_a |b\rangle = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \frac{1}{5} \begin{bmatrix} 3 \\ 4 \end{bmatrix} = \frac{1}{5} \begin{bmatrix} 4 \\ 3 \end{bmatrix}$$

*altri esempi vedi slide.

Riflessione su $|0\rangle$:

$$\begin{aligned} Ref_{|0\rangle} &= 2(|0\rangle\langle 0|) - I \\ &= 2 \begin{bmatrix} 1 & 0 \\ 0 & 0 \end{bmatrix} - \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \\ &= \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix} = Z \end{aligned}$$

che equivale a: q - H - X - H -

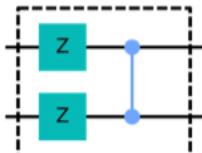
```
def ref():
    circuit = QuantumCircuit(1)
    circuit.z(0)
    gate = circuit.to_gate()
    gate.name = "Ref"
    return gate
```

Riflessione su $|00\rangle$:

$$Ref_{|00\rangle} = 2(|00\rangle\langle 00|) - I$$

$$\begin{aligned} &= 2 \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} - \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \\ &= \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & -1 \end{bmatrix} \end{aligned}$$

Io si può implementare mediante il seguente circuito:

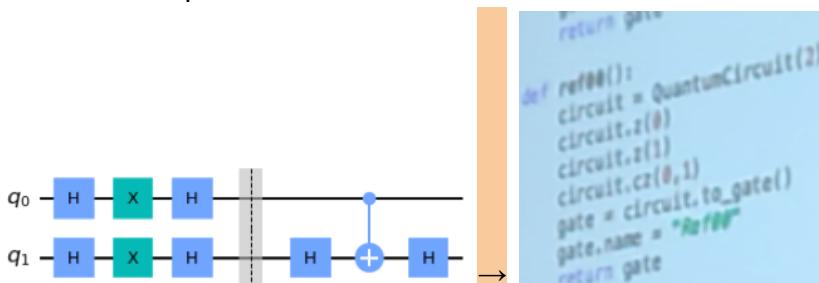


la linea blu è un control Z dove il primo qubit è il controllo e il secondo il target.

Implementazione algebrica ↗

$$\left(\begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix} \otimes \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix}\right) \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & -1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & -1 \end{bmatrix} \\ = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & -1 \end{bmatrix}$$

Il circuito e l'implementazione ↗



Riflessione su $|000\rangle$:

$$Ref_{|000\rangle} = 2(|000\rangle\langle 000|) - I$$

$$= \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & -1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & -1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & -1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & -1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & -1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 0 \end{bmatrix} \xrightarrow{\text{implementabile tramite}} \begin{array}{c} q_0 \xrightarrow{\text{H}} \text{X} \xrightarrow{\text{H}} \text{X} \\ q_1 \xrightarrow{\text{H}} \text{X} \xrightarrow{\text{H}} \text{X} \\ q_2 \xrightarrow{\text{H}} \text{X} \xrightarrow{\text{H}} \text{X} \end{array}$$

Implementazione algebrica ↗

$$\begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \otimes \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \otimes \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} = \begin{bmatrix} 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix} \otimes \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \xrightarrow{X^{\oplus n}}$$

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

$X^{\oplus n}$

CZ

$X^{\oplus n}$

moltiplichiamo le prime due matrici si ottiene \square

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

$X^{\oplus n} CZ$

$X^{\oplus n}$

=

$X^{\oplus n} CZ X^{\oplus n}$

```
def refpp():
    circuit = QuantumCircuit(3)
    for i in range(3):
        circuit.x(i)
    circuit.h(2)
    circuit.ccx(0,1,2)
    circuit.h(2)
    for i in range(3):
        circuit.x(i)
    gate = circuit.to_gate()
    gate.name = "RefPP"
    return gate
```

Riflessione su $| + \rangle$:

$$Ref_{|+}\rangle = 2(|+\rangle\langle +|) - I$$

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

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

$$\text{dove } | + \rangle \text{ è uguale a: } | + \rangle = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 \\ 1 \end{bmatrix} \text{ il tutto equivale a: } q - \boxed{x} -$$

Riflessione su $| ++ \rangle$:

$$Ref_{|++}\rangle = 2(|++\rangle\langle ++|) - I$$

$$= \frac{1}{2} \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \end{bmatrix} - \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$| ++ \rangle = \frac{1}{2} \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \end{bmatrix}$$

$$= \frac{1}{2} \begin{bmatrix} -1 & 1 & 1 & 1 \\ 1 & -1 & 1 & 1 \\ 1 & 1 & -1 & 1 \\ 1 & 1 & 1 & -1 \end{bmatrix}$$

Dato $| ++ \rangle = \frac{1}{2} \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \end{bmatrix}$ la riflessione è:

come possiamo ottenere la matrice?

Poiché questa è una riflessione su $| + \rangle$, vogliamo aggiungere una fase negativa a ogni stato ortogonale a $| + \rangle$. Un modo per farlo è usare l'operazione che trasforma lo stato $| + \rangle$ a $| 0 \rangle$, che già sappiamo essere la porta di Hadamard applicata a ciascun qubit: $H^{\otimes n}| + \rangle = | 0 \rangle$. Quindi applichiamo un circuito che aggiunge una fase negativa agli stati ortogonali a $| 0 \rangle$. Infine, trasformiamo lo stato $| 0 \rangle$ in $| + \rangle$ ancora: $H^{\otimes n}| 0 \rangle = | + \rangle$.

"Graficamente è come se facessimo una rotazione di 45 gradi spostandosi su 0, fare la riflessione su 0 e poi spostarsi nuovamente di 45 gradi".

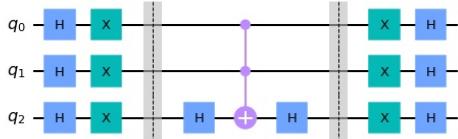
$$Ref_{|+}\rangle = H^{\otimes n} Ref_{|0}\rangle H^{\otimes n}$$

```
def refpp():
    circuit = QuantumCircuit(2)
    circuit.h(0)
    circuit.h(1)
    circuit.z(0)
    circuit.z(1)
    circuit.cz(0,1)
    circuit.h(0)
    circuit.h(1)
    gate = circuit.to_gate()
    gate.name = "RefPP"
    return gate
```

Riflessione su $|+++>$:

$$|+++> = \frac{1}{2\sqrt{2}} \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \end{bmatrix}$$

Dato circuito: la riflessione su 3 qubit $Ref_{|+++>} = H^{\otimes 3} Ref_{|000>} H^{\otimes 3}$ è rappresentato dal seguente



```
def refppp():
    circuit = QuantumCircuit(3)
    for i in range(3):
        circuit.h(i)
    for i in range(3):
        circuit.x(i)
    circuit.h(2)      I
    circuit.ccx(0,1,2)
    circuit.h(2)
    for i in range(3):
        circuit.x(i)
    for i in range(3):
        circuit.h(i)
    gate = circuit.to_gate()
    gate.name = "Ref+++"
    return gate
```

16. Algoritmo di Grover

Questo algoritmo può accelerare quadraticamente un problema di ricerca non strutturato, ma i suoi usi si estendono oltre una varietà di altri algoritmi (string matching,...). Per far ciò si avvale di una tecnica definita amplificazione dell'ampiezza.

Cosa si intende per ricerca non strutturata?

Supponiamo di avere un ampio elenco di N oggetti. Tra questi oggetti c'è un oggetto con una proprietà unica che desideriamo individuare; chiameremo questo vincitore w . Pensa a ogni elemento dell'elenco come a una scatola di un colore particolare. Supponiamo che tutti gli elementi nell'elenco siano grigi tranne il vincitore w , che è viola.



Utilizzando il calcolo classico per trovare la casella viola, si dovrebbe controllare in media $N/2$ di queste caselle e, nel peggior dei casi, tutte le N celle. Su un computer quantistico, tuttavia, possiamo trovare l'elemento contrassegnato in circa \sqrt{N} passaggi con la tecnica dell'amplificazione di ampiezza di Grover. Un'accelerazione quadratica è davvero un notevole risparmio di tempo per la ricerca di elementi contrassegnati in lunghi elenchi. Inoltre, l'algoritmo non utilizza la struttura interna dell'elenco, il che lo rende generico; questo è il motivo per cui fornisce immediatamente un'accelerazione quantistica quadratica per molti problemi classici.

Definiamo il cosiddetto **hit vector** w attraverso $w(x) = 1$ se x è soluzione, o se x appartiene alla possibile cerchia delle soluzioni S , e $w(x) = 0$ altrimenti. A condizione che il numero k di soluzioni sia diverso da zero, la divisione per \sqrt{k} fa w un vettore unitario, e quindi uno stato quantistico legale. Se potessimo costruire questo stato, la misurazione produrrebbe una soluzione con certezza. Quindi la soluzione potrebbe essere vista come uno stato.

Esempio:

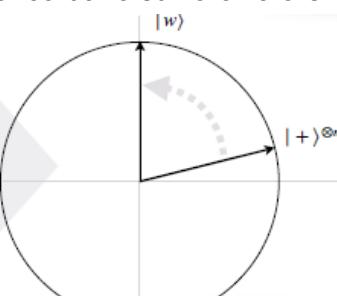
$$\begin{aligned} N &= 4 & \text{Database entries are all elements in the set } D = \{0,1,2,3\} \\ S &= \{1,3\} \\ \text{The hit vector is: } |w\rangle &= \frac{1}{\sqrt{2}} \begin{bmatrix} 0 \\ 1 \\ 0 \\ 1 \end{bmatrix} \end{aligned}$$

L'obiettivo è costruire uno stato w' abbastanza vicino a w in modo che la misurazione produca la soluzione con ragionevole probabilità. Il che si traduce in un algoritmo probabilistico. Esempio:

$$\begin{aligned} N &= 8 & \text{Database entries are all elements in the set } D = \{0,\dots,7\} \\ S &= \{5\} \\ |w\rangle &= \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 1 \\ 0 \\ 0 \end{bmatrix} & |w'\rangle &= \frac{1}{4\sqrt{2}} \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 5 \\ 1 \\ 1 \end{bmatrix} & \text{Thus we have a probability of } \left(\frac{5}{4\sqrt{2}}\right)^2 = 78\% \\ & & & & \text{of measuring the right solution} \end{aligned}$$

Proprietà solution-smoothness:

Tutti gli elementi di w che sono indicizzate dalle soluzioni hanno lo stesso valore, e anche gli elementi corrispondenti alle non-soluzioni concordano sul loro valore.



Every vector a in the two-dimensional plane spanned by w and $|+\rangle^{\otimes n}$ has the solution-smoothness property.

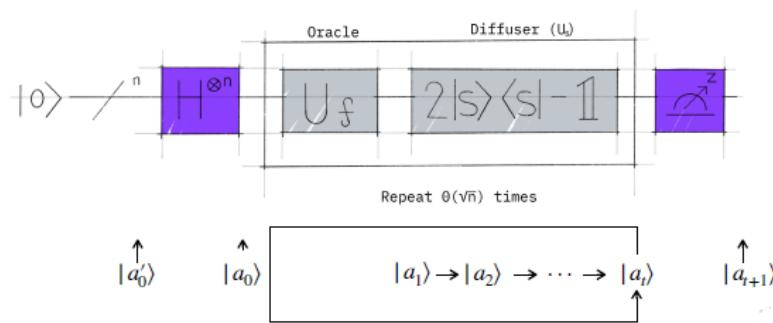
Tutti i vettori che giacciono su questa circonferenza godono della suddetta proprietà.

L'obiettivo dell'algoritmo di Grover è quindi di partire dallo stato $|+\rangle^{\otimes n}$ ovvero dalla superimposizione di tutti quanti i possibili valori e cercare in qualche modo di avvicinare questo stato a w in maniera tale che alla fine del processo il vettore $|+\rangle^{\otimes n}$ si sia avvicinato il più possibile a w .

Quindi l'algoritmo fa in modo che w attragga il vettore $+ a se fino a farlo coincidere o comunque avvicinarlo molto in modo di assicurare, che una volta misurato il vettore $+$, possa dare il vettore w .$

La procedura e l'analisi:

1. Si inizializza un registro di n-qubit nello stato zero, $|0\rangle^{\otimes n}$
2. Successivamente si applica il gate H a tutti gli n-qubit ottenendo uno stato di sovrapposizione, $|s\rangle = |+\rangle^{\otimes n}$
3. A questo punto si applica il phase oracle U_f
4. Ora si applica una riflessione sullo stato $|s\rangle$
I passi 3. e 4. vanno ripetuti t volte
5. Dopo t iterazioni si misurano tutti gli n-qubit del registro.



L'analisi:

1. Si inizializza un registro di n-qubit nello stato zero, $|a'_0\rangle = |0\rangle^{\otimes n}$

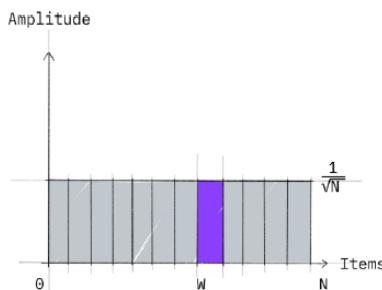
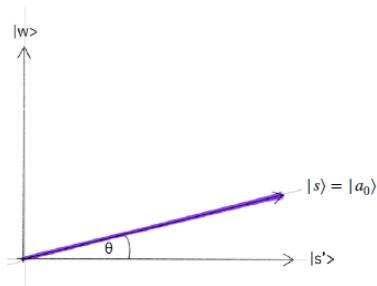
$$|a_0\rangle = \frac{1}{\sqrt{2^n}} \sum_{x \in \{0,1\}^n} |x\rangle$$

2. Si applica il gate H a tutti i qubit del registro,

Consideriamo il piano bidimensionale attraversato da $|w\rangle$ e $|s\rangle = |+\rangle^{\otimes n}$. Essi non sono del tutto perpendicolari perché non sono ortogonali tra solo (se si moltiplicano le loro componenti il risultato non è zero). Il problema è rappresentato dalla presenza di un 1 nel vettore w e di un 1 nella stessa posizione anche nel vettore s .

Introduciamo uno stato aggiuntivo $|s'\rangle$ che si trova nell'intervallo di questi due vettori, che è perpendicolare e si ottiene da $|s\rangle$ rimuovendo $|w\rangle$ e ridimensionando.

$$|w\rangle = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 1 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}, |s\rangle = \frac{1}{\sqrt{N}} \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \end{bmatrix} \rightarrow |s'\rangle = \frac{1}{\sqrt{N-1}} \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \\ 0 \\ 1 \\ 1 \\ 1 \\ 1 \end{bmatrix}$$

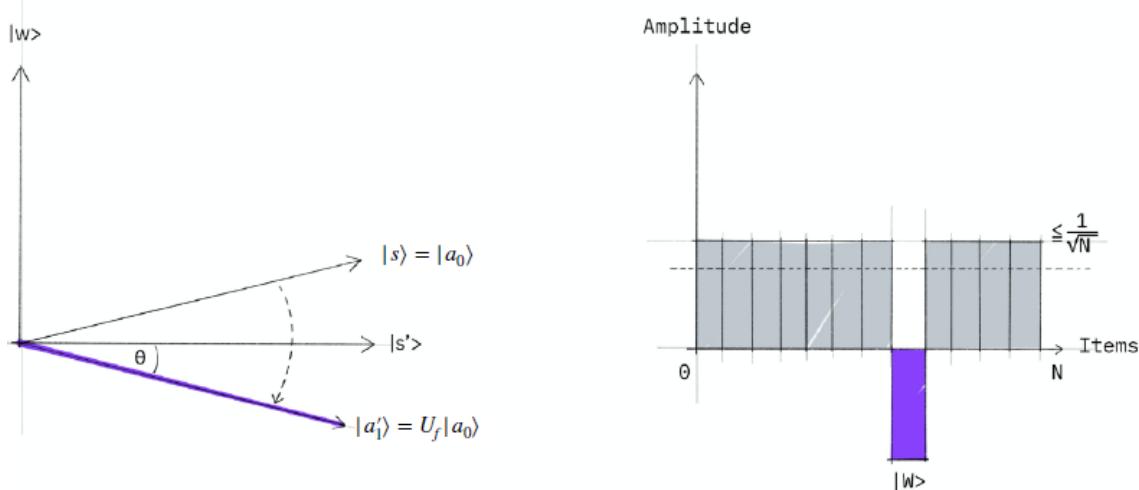


3. Applichiamo il phase oracle U_f :

$$|a'_1\rangle = \frac{1}{\sqrt{2^n}} \sum_{x \in \{0,1\}^n} (-1)^{f(x)} |x\rangle$$

Dall'applicazione del phase oracle otteniamo una fase negativa per tutti quei valori in cui $f(x) = 1$; mentre viene lasciato invariato se $f(x) = 0$. In questo caso la componente che attribuisce la fase negativa è una sola $|w\rangle$.

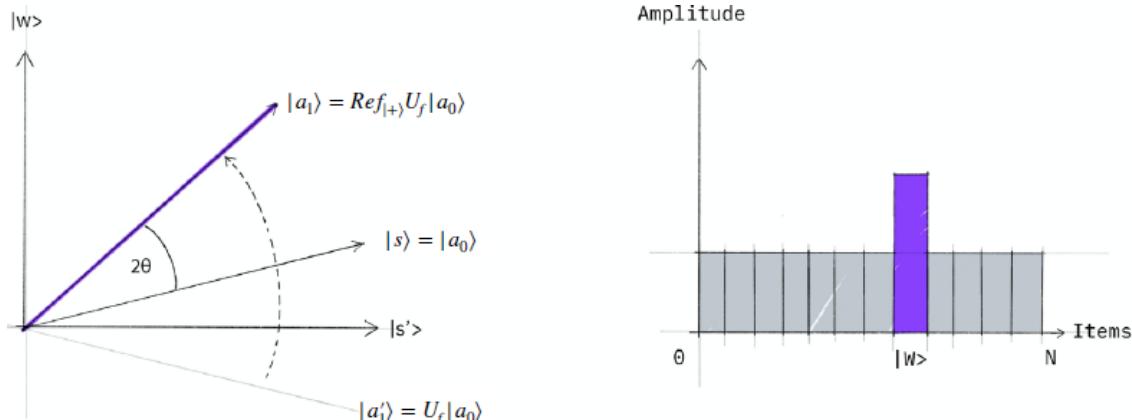
Questo comporta per $|w\rangle$ un cambio di segno dell'ampiezza:



Il risultato è una riflessione rispetto al vettore $|s'\rangle$ che è ortogonale a $|w\rangle$.

4. Si applica adesso una riflessione rispetto ad $|s\rangle$ del vettore $|a'_1\rangle$ ottenendo:

$$|a_1\rangle = [2(| + \rangle\langle + | - I)] |a'_1\rangle$$



Da notare che il vettore $|a_1\rangle$ risulta più vicino a $|w\rangle$ rispetto a quanto lo era al vettore $|s\rangle = |a_0\rangle$. Da notare come le ampiezze di tutte le componenti vengono riscalate; questo perché la norma deve essere =1. Questo ciclo di applicazione del phase oracle U_f e successiva riflessione viene iterata per t volte fino ad ottenere:

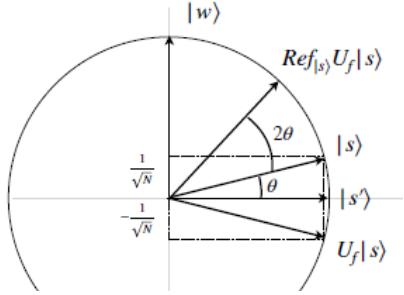
$$|a_t\rangle = (Ref_{|s\rangle} U_f)^t |a_0\rangle$$

Quante volte ripetere?:

Ogni applicazione del ciclo di Grover il vettore $|s\rangle$ ruota di circa $\frac{2}{\sqrt{N}}$ radianti dall'origine in direzione $|w\rangle$.

Dal momento che vogliamo ruotare $|s\rangle$ di circa $\frac{\pi}{2}$ radianti è sufficiente ripetere il ciclo:

$$\frac{\pi}{4}\sqrt{N} \text{ volte}$$



5. Infine misuriamo i qubit.

Esempio:

Consideriamo il caso N=4 realizzabile con 2 qubit, e assumiamo che lo stato obiettivo è dato da $|w\rangle = |11\rangle$. I due oracle U_f e P_f in questo caso operano nel seguente modo:

$$U_f = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \quad P_f = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & -1 \end{bmatrix}$$

Mentre, l'operatore di riflessione $Ref_{|s\rangle}$ opera nel seguente modo:

$$\begin{aligned} Ref_{|s\rangle} &= 2|s\rangle\langle s| - I = 2\left(\frac{1}{2}\begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \end{bmatrix}\frac{1}{2}\begin{bmatrix} 1 & 1 & 1 & 1 \end{bmatrix}\right) - \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \\ &= \frac{1}{2}\begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \end{bmatrix} - \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} = \frac{1}{2}\begin{bmatrix} -1 & 1 & 1 & 1 \\ 1 & -1 & 1 & 1 \\ 1 & 1 & -1 & 1 \\ 1 & 1 & 1 & -1 \end{bmatrix} \end{aligned}$$

Running the example with the Phase Oracle

$$\begin{aligned} P_f &= \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & -1 \end{bmatrix} \quad Ref_{|s\rangle} = \frac{1}{2}\begin{bmatrix} -1 & 1 & 1 & 1 \\ 1 & -1 & 1 & 1 \\ 1 & 1 & -1 & 1 \\ 1 & 1 & 1 & -1 \end{bmatrix} \\ |a_0\rangle &= H^{\otimes 2}|00\rangle = \frac{1}{2}\begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \end{bmatrix} \quad |a'_0\rangle = P_f|a_0\rangle = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & -1 \end{bmatrix}\frac{1}{2}\begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \end{bmatrix} = \frac{1}{2}\begin{bmatrix} 1 \\ 1 \\ 1 \\ -1 \end{bmatrix} \\ |a_1\rangle &= Ref_{|s\rangle}|a'_0\rangle = \frac{1}{2}\begin{bmatrix} -1 & 1 & 1 & 1 \\ 1 & -1 & 1 & 1 \\ 1 & 1 & -1 & 1 \\ 1 & 1 & 1 & -1 \end{bmatrix}\frac{1}{2}\begin{bmatrix} 1 \\ 1 \\ 1 \\ -1 \end{bmatrix} = \frac{1}{4}\begin{bmatrix} 0 \\ 0 \\ 0 \\ 4 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix} \end{aligned}$$

*altri esempi vedi slide

Implementazione:

```
def Refl(n):
    s = [1] * (2**n)
    s = s/np.linalg.norm(s)
    s = np.array([list(s)])
    m = np.matmul(np.transpose(s), s)
    m = 2*m - identity(2**n)
    return m

def PhaseO(w):
    n = len(w)
    P = identity(2**n)
    i = int(w, 2)
    P[i,i] = -1
    return P
```

```

def Grover(w):
    n = len(w)
    N = 2**n
    iter = int((np.pi/4)*np.sqrt(N))
    P = PhaseO(w)
    R = Refl(n)
    print("Iterazioni: ", iter)
    u = [1]*N
    s = u/np.linalg.norm(u)
    for i in range(iter):
        s = np.matmul(P,s)
        s = np.matmul(R,s)
    print(s)
    i = int(w,2)
    print((s[i]**2)*100, "%")

Grover("1110")

```

17. Quantum fourier transform

La trasformata di Fourier quantistica (QFT) è l'implementazione quantistica della trasformata di Fourier discreta sulle ampiezze di una funzione d'onda. Fa parte di molti algoritmi quantistici, in particolare l'algoritmo di fattorizzazione di Shor e la stima di fase quantistica.

La trasformata di Fourier classica agisce su un vettore $(x_0, x_1, \dots, x_{N-1}) \in \mathbb{C}^N$ e lo mappa nel vettore $(y_0, y_1, \dots, y_{N-1}) \in \mathbb{C}^N$ secondo la formula:

$$y_k = \frac{1}{\sqrt{N}} \sum_{j=0}^{N-1} x_j \omega_N^{jk}$$

dove $\omega_N^{jk} = e^{2\pi i \frac{jk}{N}}$.

Allo stesso modo, la trasformata quantistica di Fourier agisce su uno stato quantistico

$$|X\rangle = \sum_{j=0}^{N-1} x_j |j\rangle$$

e lo mappa nello stato quantistico $|Y\rangle = \sum_{k=0}^{N-1} y_k |k\rangle$ secondo la formula:

$$y_k = \frac{1}{\sqrt{N}} \sum_{j=0}^{N-1} x_j \omega_N^{jk}$$

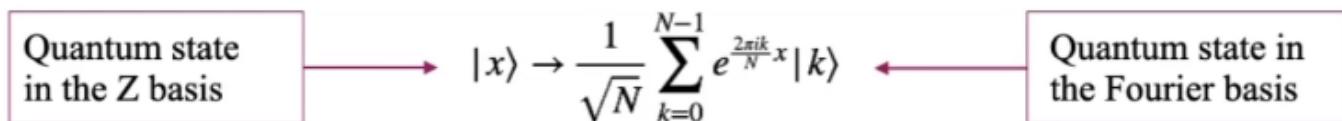
Intuizione: La QFT trasforma gli stati da una base ad un'altra. Prende uno stato che rappresenta un vettore nella base computazionale Z e lo trasforma nella base di Fourier.

Quando la QFT opera su un unico qubit allora la QFT coincide con il gate di Hadamard.

$$\begin{array}{c} |\text{State in the computational basis}\rangle \xrightarrow{\text{QFT}} |\text{State in the Fourier Basis}\rangle \\ \\ \text{QFT } |x\rangle = |\tilde{x}\rangle \end{array}$$

La QFT ha la sua inversa ovvero passare da $|\tilde{x}\rangle \rightarrow |x\rangle$.

Di seguito la trasformazione che opera la QFT, da notare che l'esponenziale con il suo esponente si chiama fase.



L'operatore che realizza questa trasformazione è definito nel seguente modo:

$$\begin{array}{c} \text{Quantum Fourier} \\ \text{Transform Operator} \end{array} \xrightarrow{} U_{QFT} = \frac{1}{\sqrt{N}} \sum_{j=0}^{N-1} \sum_{k=0}^{N-1} e^{\frac{2\pi i}{N} jk} |k\rangle \langle j|$$

\downarrow

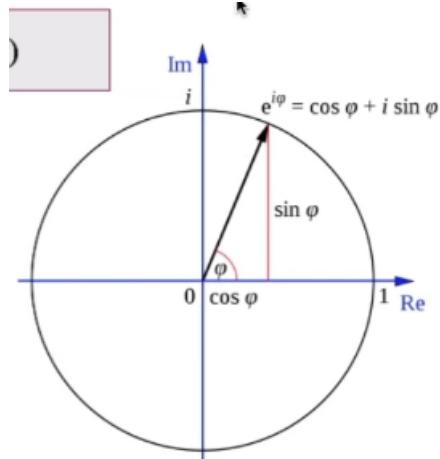
$$U_{QFT}|x\rangle = \frac{1}{\sqrt{N}} \left(|0\rangle + e^{\frac{2\pi i}{2} x} |1\rangle \right) \otimes \left(|0\rangle + e^{\frac{2\pi i}{2^2} x} |1\rangle \right) \otimes \dots \otimes \left(|0\rangle + e^{\frac{2\pi i}{2^{n-1}} x} |1\rangle \right) \otimes \left(|0\rangle + e^{\frac{2\pi i}{2^n} x} |1\rangle \right)$$

Formula di euler:

Abbandoniamo la rappresentazione di un qubit come cerchio e utilizziamo la sfera.

La formula di Euler afferma che per ogni numero reale x :

$$e^{ix} = \cos(x) + i \sin(x)$$



where e is the base of the natural logarithm, i is the imaginary unit, and \cos and \sin are the trigonometric functions cosine and sine respectively

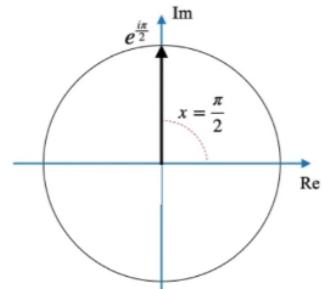
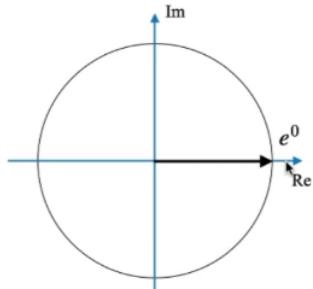
Esempi:

When $x = 0$, Euler's formula evaluates

$$e^0 = \cos(0) + i \sin(0) = 1 + 0 = 1$$

When $x = \pi/2$, Euler's formula evaluates:

$$e^{i\frac{\pi}{2}} = \cos\left(\frac{\pi}{2}\right) + i \sin\left(\frac{\pi}{2}\right) = 0 + i = i$$

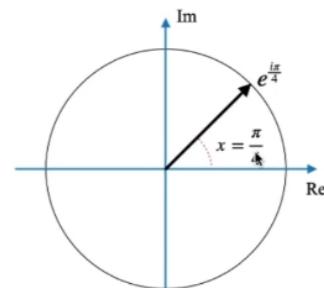
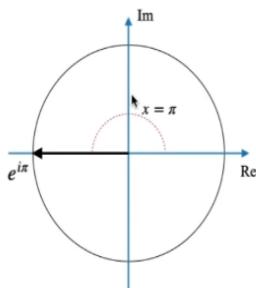


When $x = \pi$, Euler's formula evaluates to $e^{i\pi} + 1 = 0$, known as Euler's identity

$$e^{i\pi} = \cos(\pi) + i \sin(\pi) = -1 + 0 = -1$$

When $x = \pi/4$, Euler's formula evaluates

$$e^{i\frac{\pi}{4}} = \cos\left(\frac{\pi}{4}\right) + i \sin\left(\frac{\pi}{4}\right) = \frac{1}{\sqrt{2}} + i \frac{1}{\sqrt{2}} = \frac{1+i}{\sqrt{2}}$$



Sfera di Bloch e fase quantistica:

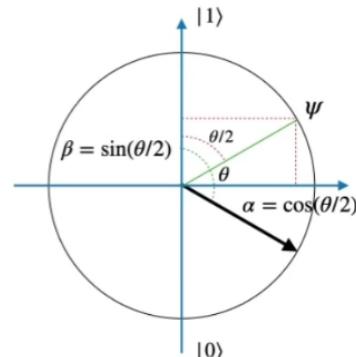
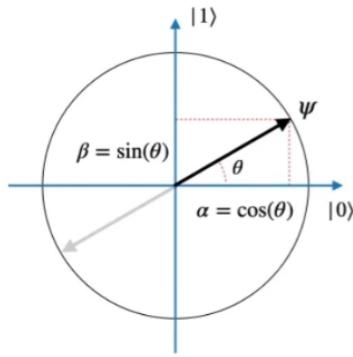
Nella rappresentazione di Bloch l'uno sta in alto e lo zero in basso, questo anche per una contrapposizione visiva.

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

$$|\psi\rangle = \cos(\theta)|0\rangle + \sin(\theta)|1\rangle$$

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

$$|\psi\rangle = \cos(\theta/2)|0\rangle + \sin(\theta/2)|1\rangle$$

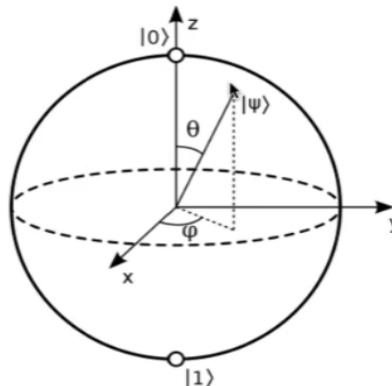


Questa rappresentazione la introduciamo per definire il concetto di fase quantistica. Utilizzando questa rappresentazione introduciamo un altro grado di libertà identificato da x , che nella misurazione non è percepibile, definita come **fase**. Dal punto di vista grafico il qubit ora è rappresentato da una sfera. La fase è definita da un certo angolo ϕ .

$$|\psi\rangle = \alpha|0\rangle + e^{i\phi}\beta|1\rangle$$

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

where ϕ is the quantum phase



Il motivo per la quale finora non lo abbiamo visto è che la fase ϕ è trascurabile poiché $|e^{i\phi}|^2 = 1$ il ché la rende ininfluente nel misuramento.

QFT applicata ad un qubit:

La QFT applicata ad un qubit rappresenta la matrice di Hadamard.

Da $|\Psi\rangle = \alpha|0\rangle + \beta|1\rangle$ applichiamo la formula per ottenere la QFT:

$$y_k = \frac{1}{\sqrt{N}} \sum_{j=0}^{N-1} x_j e^{\frac{2\pi i}{N} jk}$$

k varia tra 0 e 1. $N = 2$. j saranno le componenti di $x_0 = \alpha$, $x_1 = \beta$. Allora:

$$y_0 = \frac{1}{\sqrt{2}} \left(\alpha e^{2\pi i \frac{0 \times 0}{2}} + \beta e^{2\pi i \frac{1 \times 0}{2}} \right) = \frac{1}{\sqrt{2}} (\alpha + \beta)$$

$$y_1 = \frac{1}{\sqrt{2}} \left(\alpha e^{2\pi i \frac{0 \times 1}{2}} + \beta e^{2\pi i \frac{1 \times 1}{2}} \right) = \frac{1}{\sqrt{2}} (\alpha - \beta)$$

Ricompattando si ottiene:

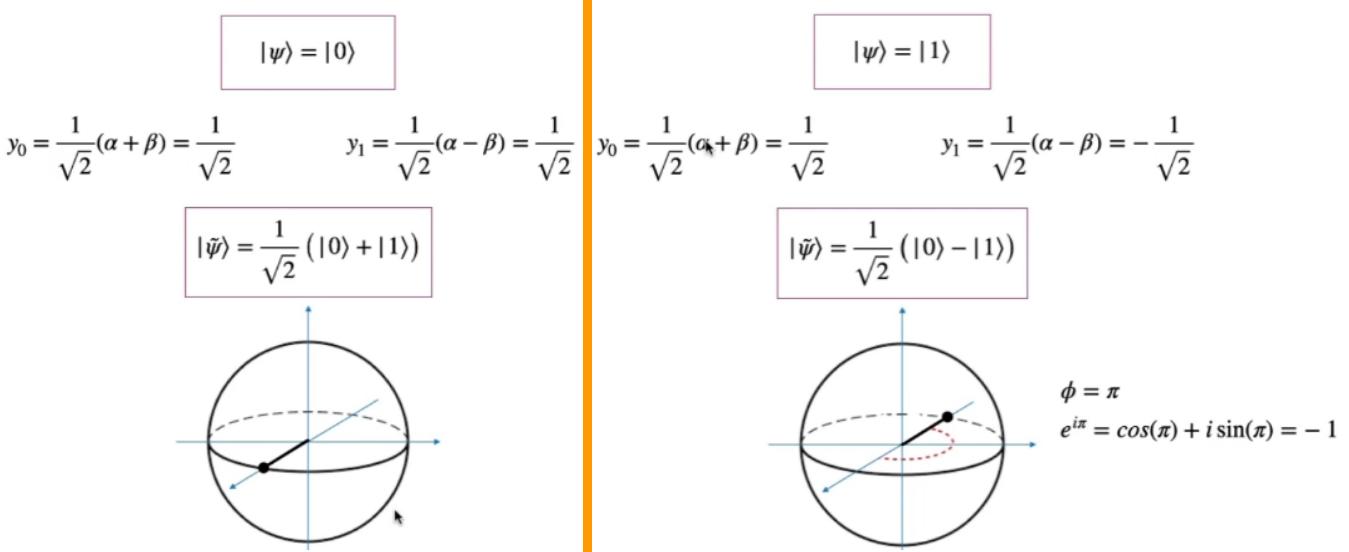
$$U_{QFT}|\psi\rangle = \frac{1}{\sqrt{2}}(\alpha + \beta)|0\rangle + \frac{1}{\sqrt{2}}(\alpha - \beta)|1\rangle$$

Possiamo ottenere lo stesso risultato anche se utilizziamo la formula di eulero.

$$y_0 = \frac{1}{\sqrt{2}} \left(\alpha (\cos(0) + i \sin(0)) + \beta (\cos(0) + i \sin(0)) \right) = \frac{1}{\sqrt{2}}(\alpha + \beta)$$

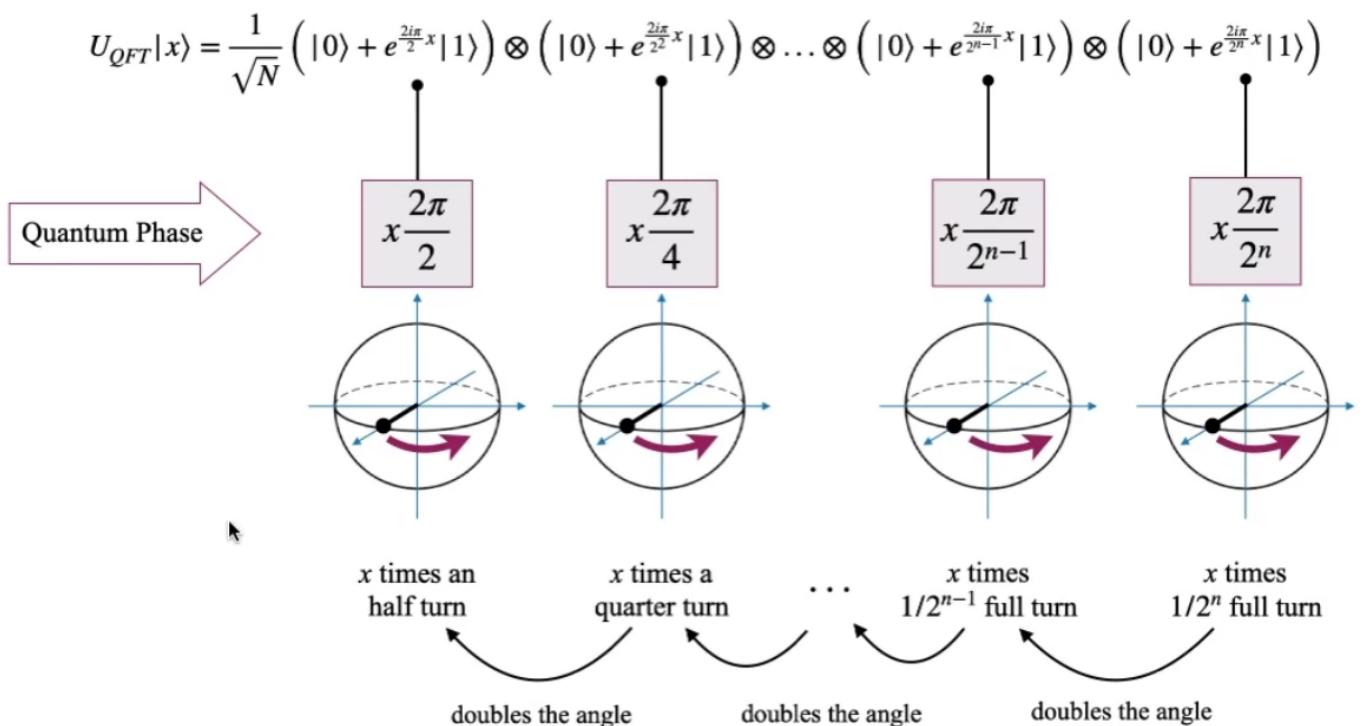
$$y_1 = \frac{1}{\sqrt{2}} \left(\alpha (\cos(0) + i \sin(0)) + \beta (\cos(\pi) + i \sin(\pi)) \right) = \frac{1}{\sqrt{2}}(\alpha - \beta)$$

Graficamente tutto ciò significa:



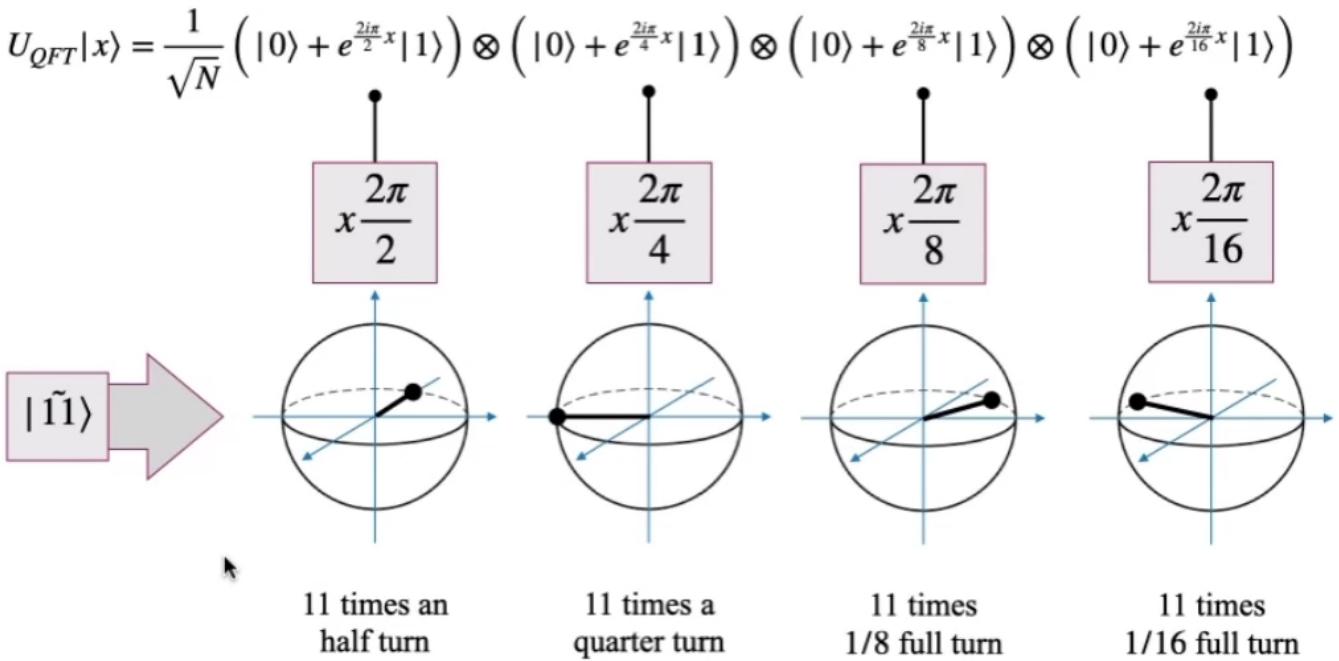
QFT applicata a n qubit:

Quello che fa la QFT alla fine è mettere i qubit in uno stato di superimpostione e poi applicare a ciascuno di essi una fase; tale fase non è stabilita a caso bensì dipende fortemente da quanti sono i qubit. Se i qubit sono n allora il qubit più a destra ruota di un fattore pari a $x \frac{2\pi}{2^n}$ successivamente gli altri del doppio fino ad arrivare a quello più a sinistra che ruoterà, in maniera più significativa, di $x \frac{2\pi}{2}$.



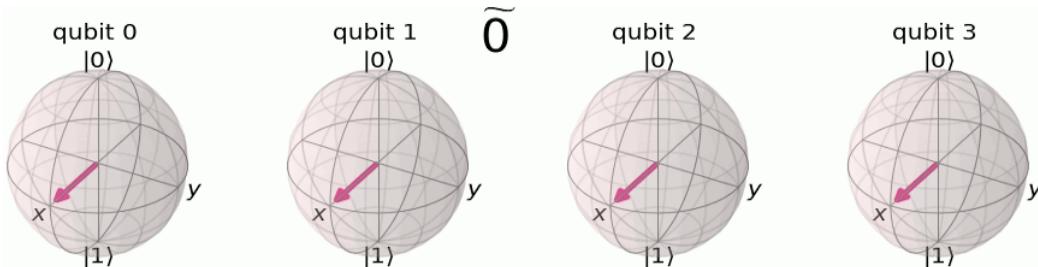
Esempio:

$$n = 4 \quad N = 16 \quad x = |1011\rangle = |11\rangle$$

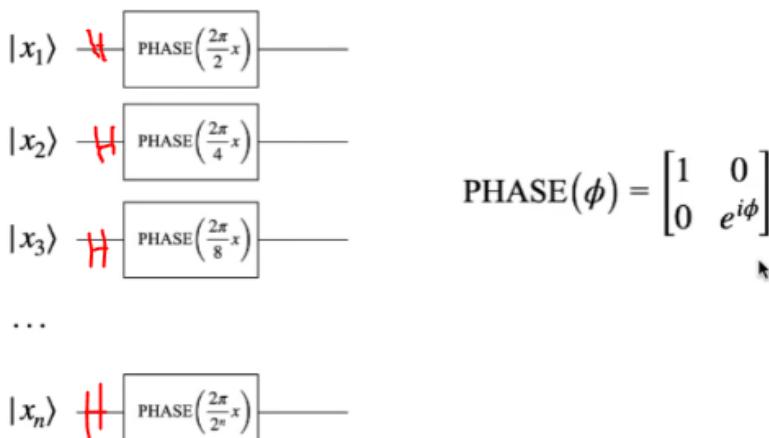


$11\tilde{}$ è la rappresentazione dell' 11 senza tilde con l'ausilio della QFT.

Da questa rappresentazione possiamo dedurre che **è possibile contare con la base di Fourier**. Così come codifichiamo un numero usando la base computazionale allo stesso modo, con la QFT, possiamo codificare un numero utilizzando la base di Fourier. La frequenza, che conta nella base computazionale, nella QFT si traduce nell'angolo di rotazione rispetto all'asse x .



Se volessimo generare un circuito per ottenere la rappresentazione di uno specifico numero nella base di fourier, lo si può fare nel seguente modo:



Il circuito con 2 qubits:

I qubit singolarmente vengono trasformati tramite:

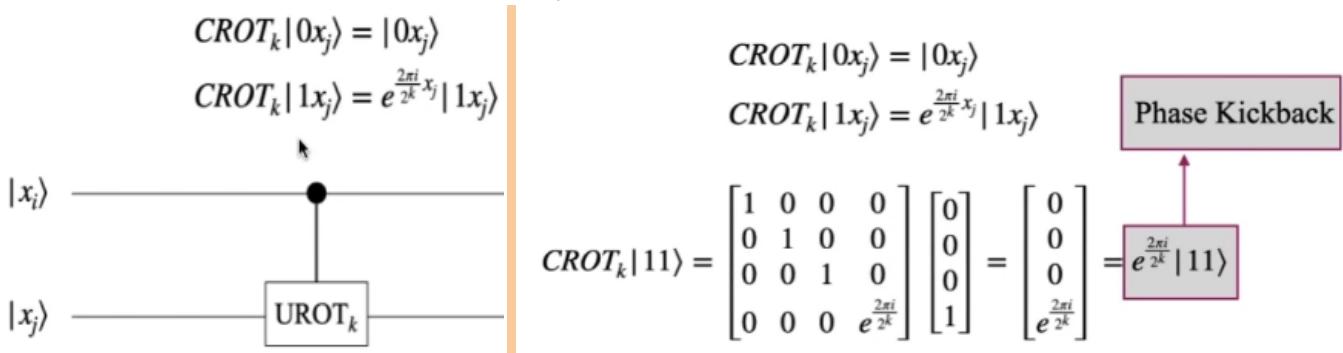
$$H|x_k\rangle = \frac{1}{\sqrt{2}} \left(|0\rangle + e^{\frac{2\pi i}{2}x_k} |1\rangle \right)$$

Successivamente si introduce un operatore chiamato $CROT_k$, ovvero una rotazione controllata di grado k . Consiste in:

$$CROT_k = \begin{bmatrix} I & 0 \\ 0 & UROT_k \end{bmatrix}$$

$$UROT_k = \begin{bmatrix} 1 & 0 \\ 0 & e^{\frac{2\pi i}{2^k}} \end{bmatrix}$$

Questo operatore lavora con due qubit $|x_i, x_j\rangle$ dove il primo qubit è il controllo ed il secondo è il target:



Da notare che la fase viene applicata ad entrambi i qubits, sia al primo (controllo) che al secondo (target). Questo effetto viene definito Phase Kickback.

Verifica con 2 qubits:

$$|\psi_0\rangle = |x_2\rangle \otimes |x_1\rangle$$

Applichiamo il gate H al qubit x_1 , ottenendo:

$$|\psi_1\rangle = |x_2\rangle \otimes \frac{1}{\sqrt{2}} \left[|0\rangle + e^{\frac{2\pi i}{2}x_1} |1\rangle \right]$$

Applichiamo il gate $CROT_2$, ovvero rotazione controllata:

$$|\psi_2\rangle = |x_2\rangle \otimes \frac{1}{\sqrt{2}} \left[|0\rangle + e^{\frac{2\pi i}{2^2}x_2 + \frac{2\pi i}{2}x_1} |1\rangle \right]$$

Applichiamo il gate H al secondo qubit x_2 , ottenendo:

$$|\psi_3\rangle = \frac{1}{\sqrt{2}} \left[|0\rangle + e^{\frac{2\pi i}{2}x_2} |1\rangle \right] \otimes \frac{1}{\sqrt{2}} \left[|0\rangle + e^{\frac{2\pi i}{2^2}x_2 + \frac{2\pi i}{2}x_1} |1\rangle \right]$$

Si scambiano i due qubits ottenendo:

$$|\psi_4\rangle = \frac{1}{\sqrt{2}} \left[|0\rangle + e^{\frac{2\pi i}{2^2}x_2 + \frac{2\pi i}{2}x_1} |1\rangle \right] \otimes \frac{1}{\sqrt{2}} \left[|0\rangle + e^{\frac{2\pi i}{2}x_2} |1\rangle \right]$$

Sviluppiamo i calcoli delle due fasi. Il primo qubit diventa:

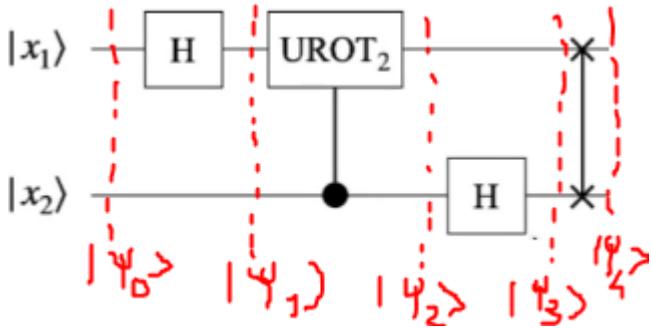
$$\frac{2\pi i}{2}x_1 + \frac{2\pi i}{2^2}x_2 = \frac{2\pi i}{2^2} (2^1x_1 + 2^0x_2) = \frac{2\pi i}{2^2}x$$

Mentre il secondo qubit diventa:

$$\frac{2\pi i}{2}x_2 = \frac{2\pi i}{2^2} (2x_2) = \frac{2\pi i}{2^2} (2x \bmod 2^2) = \frac{2\pi i}{2}x$$

"Riferendosi come esempio alla rappresentazione del numero x (su due qubit) e lo si moltiplica per 2 questo implica uno shift a sinistra senza riporto perdendo di conseguenza la cifra più significativa; così come la divisione ci fa perdere la cifra più a destra; ragion per cui x_1 non compare più nella moltiplicazione." Ottenendo in definitiva:

$$|\psi_4\rangle = \frac{1}{\sqrt{2}} \left[|0\rangle + e^{\frac{2\pi i}{2^2}x} |1\rangle \right] \otimes \frac{1}{\sqrt{2}} \left[|0\rangle + e^{\frac{2\pi i}{2}x} |1\rangle \right]$$



Verifica con 3 qubits:

$$|\psi_0\rangle = |x_3\rangle \otimes |x_2\rangle \otimes |x_1\rangle$$

Applichiamo il gate H al primo qubit x_1 :

$$|\psi_1\rangle = |x_3\rangle \otimes |x_2\rangle \otimes \frac{1}{\sqrt{2}} \left[|0\rangle + e^{\frac{2\pi i}{2}x_1} |1\rangle \right]$$

Applichiamo il gate $CROT_2$, ovvero rotazione controllata al primo qubit x_1 :

$$|\psi_2\rangle = |x_3\rangle \otimes |x_2\rangle \otimes \frac{1}{\sqrt{2}} \left[|0\rangle + e^{\frac{2\pi i}{2^2}x_2 + \frac{2\pi i}{2}x_1} |1\rangle \right]$$

Applichiamo un altro gate $CROT_3$, ovvero rotazione controllata sempre al primo qubit x_1 :

$$|\psi_3\rangle = |x_3\rangle \otimes |x_2\rangle \otimes \frac{1}{\sqrt{2}} \left[|0\rangle + e^{\frac{2\pi i}{2^3}x_3 + \frac{2\pi i}{2^2}x_2 + \frac{2\pi i}{2}x_1} |1\rangle \right]$$

Adesso lavoriamo sul secondo qubit x_2 applicando il gate H:

$$|\psi_4\rangle = |x_3\rangle \otimes \frac{1}{\sqrt{2}} \left[|0\rangle + e^{\frac{2\pi i}{2}x_2} |1\rangle \right] \otimes \frac{1}{\sqrt{2}} \left[|0\rangle + e^{\frac{2\pi i}{2^3}x_3 + \frac{2\pi i}{2^2}x_2 + \frac{2\pi i}{2}x_1} |1\rangle \right]$$

Applichiamo il gate $CROT_2$, ovvero rotazione controllata al secondo qubit x_2 :

$$|\psi_5\rangle = |x_3\rangle \otimes \frac{1}{\sqrt{2}} \left[|0\rangle + e^{\frac{2\pi i}{2^2}x_3 + \frac{2\pi i}{2}x_2} |1\rangle \right] \otimes \frac{1}{\sqrt{2}} \left[|0\rangle + e^{\frac{2\pi i}{2^3}x_3 + \frac{2\pi i}{2^2}x_2 + \frac{2\pi i}{2}x_1} |1\rangle \right]$$

Adesso lavoriamo sul terzo qubit x_3 applicando il gate H:

$$|\psi_6\rangle = \frac{1}{\sqrt{2}} \left[|0\rangle + e^{\frac{2\pi i}{2}x_3} |1\rangle \right] \otimes \frac{1}{\sqrt{2}} \left[|0\rangle + e^{\frac{2\pi i}{2^3}x_3 + \frac{2\pi i}{2^2}x_2} |1\rangle \right] \otimes \frac{1}{\sqrt{2}} \left[|0\rangle + e^{\frac{2\pi i}{2^3}x_3 + \frac{2\pi i}{2^2}x_2 + \frac{2\pi i}{2}x_1} |1\rangle \right]$$

Infine applichiamo gli scambi:

$$|\psi_7\rangle = \frac{1}{\sqrt{2}} \left[|0\rangle + e^{\frac{2\pi i}{2^3}x_3 + \frac{2\pi i}{2^2}x_2 + \frac{2\pi i}{2}x_1} |1\rangle \right] \otimes \frac{1}{\sqrt{2}} \left[|0\rangle + e^{\frac{2\pi i}{2^2}x_3 + \frac{2\pi i}{2}x_2} |1\rangle \right] \otimes \frac{1}{\sqrt{2}} \left[|0\rangle + e^{\frac{2\pi i}{2}x_3} |1\rangle \right]$$

Sviluppiamo i calcoli. Il primo qubit x_1 diventa:

$$\frac{2\pi i}{2}x_1 + \frac{2\pi i}{2^2}x_2 + \frac{2\pi i}{2^3}x_3 = \frac{2\pi i}{2^3} (2^2x_1 + 2^1x_2 + 2^0x_3) = \frac{2\pi i}{2^3}x$$

Il secondo qubit x_2 diventa:

$$\frac{2\pi i}{2}x_2 + \frac{2\pi i}{2^2}x_3 = \frac{2\pi i}{2^3} (2^2x_2 + 2^1x_3) = \frac{2\pi i}{2^3} (2x \bmod 2^3)$$

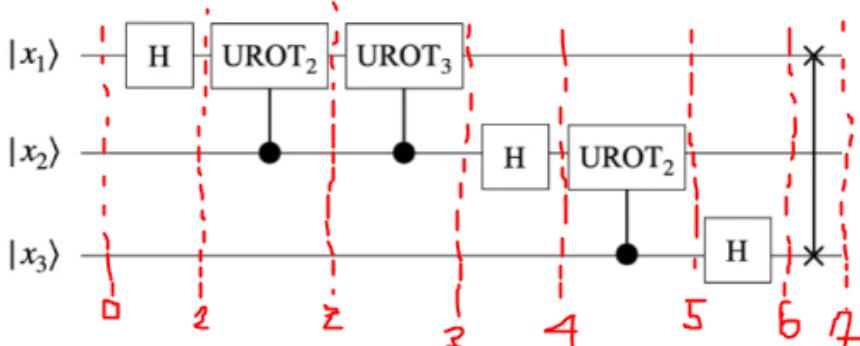
$$\frac{2\pi i}{2}x_1 + \frac{2\pi i}{2^2}x_2 + \frac{2\pi i}{2^3}x_3 = \frac{2\pi i}{2^3} (2^2x_1 + 2^1x_2 + 2^0x_3) = \frac{2\pi i}{2^3}x$$

Il terzo qubit x_3 diventa:

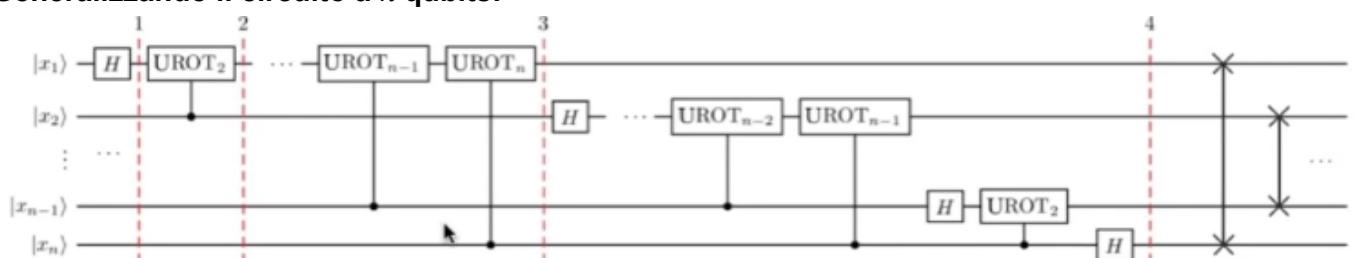
$$\frac{2\pi i}{2}x_3 = \frac{2\pi i}{2^3} (2^2x_3) = \frac{2\pi i}{2^3} (2^2x \bmod 2^3) = \frac{2\pi i}{2}x$$

Riassumendo:

$$|\psi_7\rangle = \frac{1}{\sqrt{2}} \left[|0\rangle + e^{\frac{2\pi i}{2^3}x} |1\rangle \right] \otimes \frac{1}{\sqrt{2}} \left[|0\rangle + e^{\frac{2\pi i}{2^2}x} |1\rangle \right] \otimes \frac{1}{\sqrt{2}} \left[|0\rangle + e^{\frac{2\pi i}{2}x} |1\rangle \right]$$

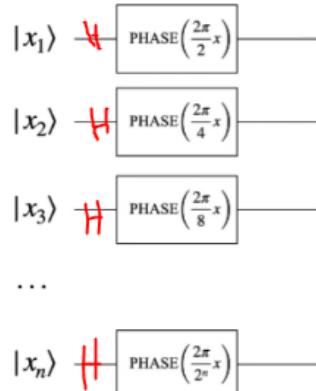


Generalizzando il circuito a n qubits:



Implementazione:

```
[1] def QFT(n):
    qc = QuantumCircuit(n)
    for t in range(n):
        qc.h(t)
        i = 1
        for c in range(t+1,n):
            phase = np.pi/(2**i)
            qc.cp(phase, c, t)
            i = i*2
    qc.barrier()
    #operiamo gli swap
    for q in range(int(n/2)):
        qc.swap(q, n-1-q)
    return qc
```



$$\text{PHASE}(\phi) = \begin{bmatrix} 1 & 0 \\ 0 & e^{i\phi} \end{bmatrix}$$

[2]

Per testare la funzione QFT [1] inizializziamo il circuito quantistico con un numero specifico utilizzando il circuito in [2], dove x è il valore da rappresentare.

Ottenendo un numero nella base di Fourier, applichiamo ad esso la trasformata inversa per ritrasformare il valore dalla sua rappresentazione nella base di Fourier nella rappresentazione nella base quantistica. Misurando dovremmo ottenere il valore indicato inizialmente.

Praticamente facciamo l'operazione inversa: partiamo dalla base di Fourier e utilizzando la trasformata inversa otteniamo il valore nella base computazionale standard, infine misuriamo.

Creiamo l'inizializzatore: una funzione che definisce un circuito che inizializza un valore nella base di Fourier. Questa funzione, chiamata FBI, prende in input un numero, $value$, e il numero, n , di qubit per la sua rappresentazione:

```
#inizializza uno specifico
#valore nella base di Fourier
def Init_QFT(val,n):
    qc = QuantumCircuit(n)
    for q in range(n):
        qc.h(q)
        phase = (np.pi/2**q)*val
        qc.p(phase,q)
    return qc

def Test_QFT(val, n):
    qc = QuantumCircuit(n,n)
    qc = qc.compose(Init_QFT(val,n))
    Inverse_QFT = QFT(n).inverse()
    qc = qc.compose(Inverse_QFT)
    qc.barrier()
    #procediamo con le misurazioni
    for q in range(n):
        qc.measure(q, n-1-q)
    print(qc.draw())
    #lanciamo il simulatore
    aer_sim = Aer.get_backend('aer_simulator')
    shots = 1024
    qobj = assemble(qc, shots=shots)
    results = aer_sim.run(qobj).result()
    counts = results.get_counts()
    plot_histogram(counts)
    return counts

plot_histogram(Test_QFT(29,5))
```

18. Quantum phase estimation

Si tratta di una stima della fase che un operatore applica ad un qubit. L'obiettivo è riuscire a capire, dato un operatore, qual è la fase che questo operatore applica ad uno stato quantistico.

Alcuni operatori vengono utilizzati senza conoscere la loro struttura interna, per esempio nell'algoritmo di Grover non sappiamo esattamente qual è l'angolo di rotazione che imporrà l'operatore di Grover;

Conoscendo la fase di rotazione dell'operatore di Grover è possibile capire quanto ci vuole per arrivare a $\frac{\pi}{2}$, quindi contare il numero di iterazioni, e dato che il numero di iterazioni è strettamente correlato al numero di soluzioni della funzione allora possiamo capire quante soluzioni ci sono per una specifica funzione.

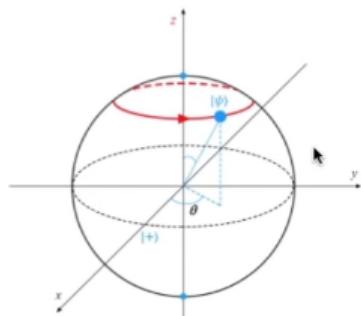
Quindi, **quello che ci serve per poter capire quante soluzioni ci sono nel problema di ricerca di Grover è proprio la Phase Estimation**.

Phase Shift Gate:

Il phase shift è una famiglia di gate a singoli qubit che mappano le basi $|0\rangle \rightarrow |0\rangle$ e $|1\rangle \rightarrow e^{i\theta}|1\rangle$. La probabilità di misurare $|0\rangle$ o $|1\rangle$ non cambia applicando questo gate, tuttavia modifica la fase dello stato quantistico. Questo è equivalente a tracciare un cerchio orizzontale sulla sfera di Bloch di θ radianti.

Per una phase shift θ con periodo 2π , il gate di phase shift è rappresentato dalla matrice:

$$P(\theta) = \begin{bmatrix} 1 & 0 \\ 0 & e^{i\theta} \end{bmatrix}$$



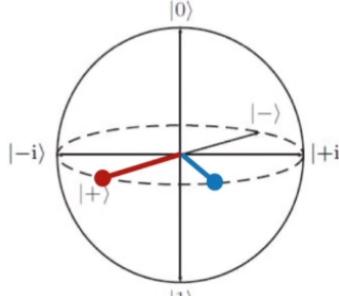
Operatore T:

Rappresenta una rotazione di 45 gradi ($\frac{\pi}{4}$ radianti) attorno all'asse z.

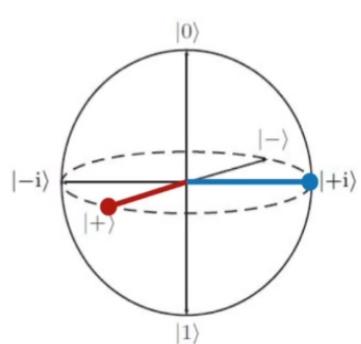
Operatore S:

Rappresenta una rotazione di 90 gradi ($\frac{\pi}{2}$ radianti) attorno all'asse z. Questi operatori sono in relazione, nel senso per ottenere l'operatore S è possibile moltiplicare per 2 l'operatore T.

$$\begin{aligned} T &= \begin{bmatrix} 1 & 0 \\ 0 & e^{i\frac{\pi}{4}} \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & \frac{1}{\sqrt{2}(1+i)} \end{bmatrix} \\ T(|0\rangle + |1\rangle) &= |0\rangle + e^{i\frac{\pi}{4}}|1\rangle \\ &= |0\rangle + (\cos \frac{\pi}{4} + i \sin \frac{\pi}{4})|1\rangle \\ &= |0\rangle + \left(\frac{1}{\sqrt{2}} + \frac{1}{\sqrt{2}}i \right)|1\rangle \end{aligned}$$



$$\begin{aligned} S &= \begin{bmatrix} 1 & 0 \\ 0 & e^{i\frac{\pi}{2}} \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & i \end{bmatrix} \\ S(|0\rangle + |1\rangle) &= |0\rangle + e^{i\frac{\pi}{2}}|1\rangle \\ &= |0\rangle + (\cos \frac{\pi}{2} + i \sin \frac{\pi}{2})|1\rangle \\ &= |0\rangle + i|1\rangle \end{aligned}$$

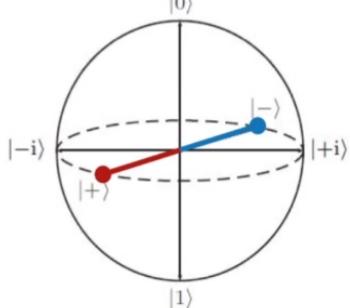


Operatore Z:

Si tratta di una gate unitario che opera in un solo qubit. In particolare mappa 1 in -1 lasciando 0 inalterato. Opera ruotando di 180 gradi (π radianti) attorno all'asse z.

$$Z = \begin{bmatrix} 1 & 0 \\ 0 & e^{i\pi} \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix}$$

$$\begin{aligned} Z(|0\rangle + |1\rangle) &= |0\rangle + e^{i\pi}|1\rangle \\ &= |0\rangle + (\cos \pi + i \sin \pi)|1\rangle \\ &= |0\rangle - |1\rangle \end{aligned}$$

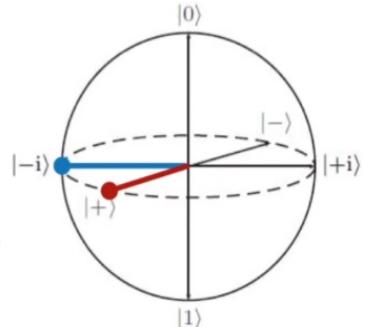


Operatore coniugato di S (S^\dagger):

Rappresenta una rotazione di 270 gradi ($\frac{3}{2}\pi$ radianti).

$$S^\dagger = \begin{bmatrix} 1 & 0 \\ 0 & e^{i\frac{3\pi}{2}} \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & -i \end{bmatrix}$$

$$\begin{aligned} S^\dagger(|0\rangle + |1\rangle) &= |0\rangle + e^{i\frac{3\pi}{2}}|1\rangle \\ &= |0\rangle + (\cos \frac{3\pi}{2} + i \sin \frac{3\pi}{2})|1\rangle \\ &= |0\rangle - i|1\rangle \end{aligned}$$



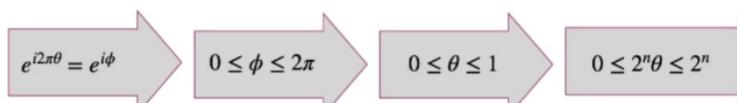
Quantum phase estimation:

La stima della fase quantistica è una delle subroutine più importanti nel calcolo quantistico. Serve come elemento centrale per molti algoritmi quantistici. L'obiettivo dell'algoritmo è il seguente:

Dato un operatore unitario U (di cui non conosciamo le caratteristiche), vogliamo capire qual è la fase θ che esso applica allo stato quantistico, in particolare si presuppone che:

$$U|\psi\rangle = e^{2\pi i \theta} |\psi\rangle$$

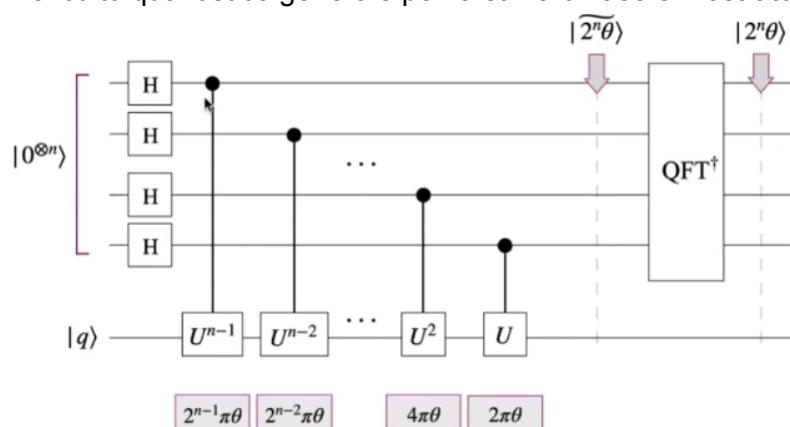
Dove $|\psi\rangle$ è un autovettore e $e^{2\pi i \theta}$ è il corrispondente autovalore. Poiché U è unitario, tutti i suoi autovalori hanno norma 1. Inoltre:



Invece di misurare θ , che è un numero frazionario, misuriamo $2^n\theta$ che è un numero compreso tra 0 e 2^n sperando che sia un numero intero; infatti se $2^n\theta$ è un numero intero la misurazione avviene con precisione e l'algoritmo non sbaglia, se così non fosse allora c'è una probabilità (bassissima) che l'algoritmo sbaglia restituendo l'intero più vicino.

Intuizione:

Il circuito quantistico generale per la stima di fase è mostrato di seguito.

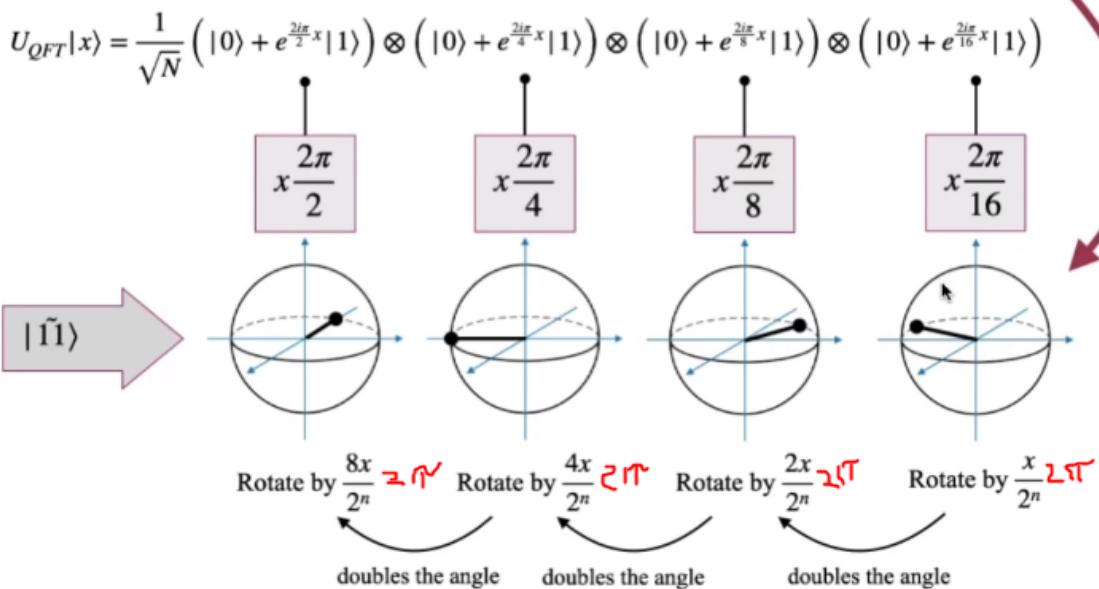


Il registro superiore contiene t qubit di "conteggio" e quello inferiore contiene il qubit nello stato $|\psi\rangle$.

L'algoritmo di stima della fase usa la phase kickback per scrivere la fase di U (espresso nella base di Fourier) nei t qubit nel registro di conteggio. Usiamo infine la QFT inversa per convertirlo dalla base di Fourier alla base computazionale, fatto ciò misuriamo i qubit.

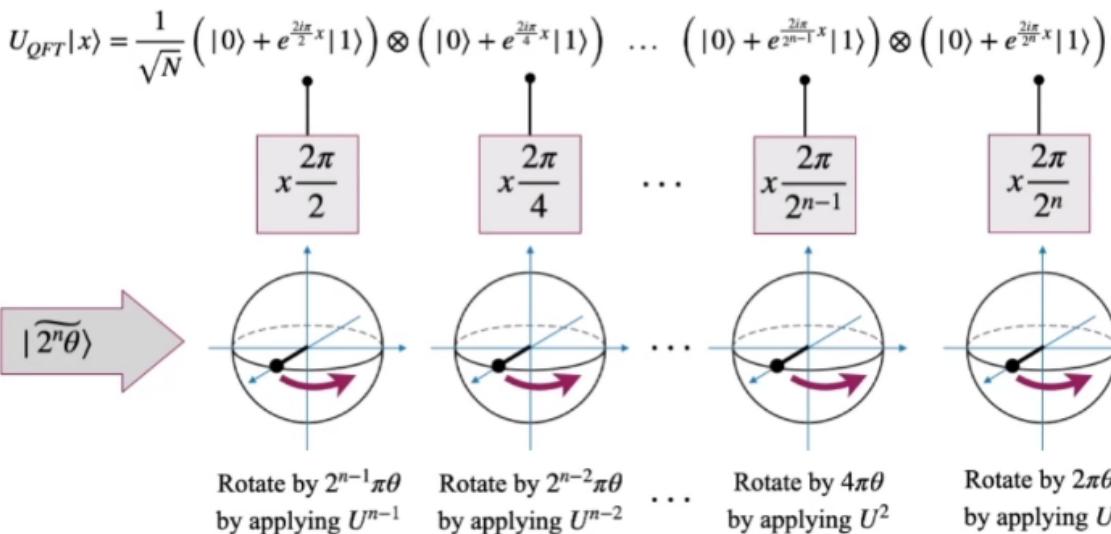
$$n = 4 \quad N = 16 \quad x = |1011\rangle = |11\rangle$$

Completes a full rotation counting between 0 and 2^n



Generalizzando:

$$n \quad N = 2^n \quad x = |2^n\theta\rangle$$



Quindi questo è un modo per stimare la fase che un certo operatore U applicherebbe ad uno stato q . L'algoritmo che fa la stima di questa fase applica la fase all'operatore q ma non ci interessa ciò che q rappresenta alla fine, ciò che ci interessa è la rappresentazione del vettore contatore. Ma q è indispensabile per applicare la phase kickback.

Esempio:

Prendiamo un gate che implementa una rotazione di $\frac{5}{4}\pi$ e utilizziamo la quantum phase estimation per stimare la sua fase.

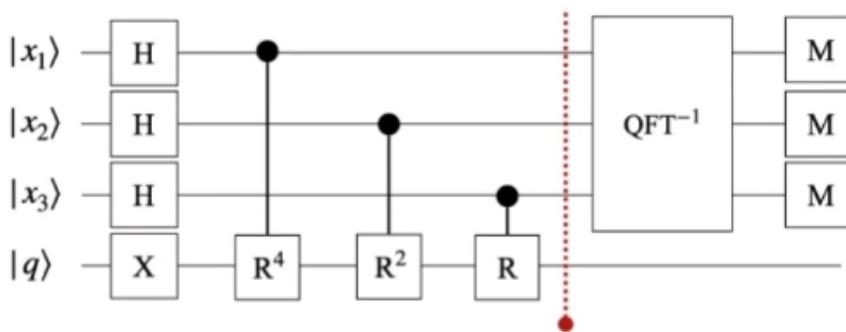
Il gate è ottenibile mediante:

$$R|1\rangle = TZ|1\rangle = \begin{bmatrix} 1 & 0 \\ 0 & e^{i\pi/4} \end{bmatrix} \begin{bmatrix} 1 & 0 \\ 0 & e^{i\pi} \end{bmatrix} \begin{bmatrix} 0 \\ 1 \end{bmatrix} = e^{i5\pi/4}|1\rangle$$

Since QPE will give us θ where:

We expect to measure $2^n\theta = 5$, so that

$$\theta = \frac{5}{2^n} = \frac{5}{8}$$



$$|\psi_0\rangle = |0\rangle \otimes |0\rangle \otimes |0\rangle \otimes |1\rangle$$

$$|\psi_1\rangle = \frac{1}{2\sqrt{2}} (|0\rangle + |1\rangle) \otimes (|0\rangle + |1\rangle) \otimes (|0\rangle + |1\rangle) \otimes |1\rangle$$

$$|\psi_2\rangle = \frac{1}{2\sqrt{2}} \left(|0\rangle + e^{\frac{20i\pi}{4}} |1\rangle \right) \otimes \left(|0\rangle + e^{\frac{10i\pi}{4}} |1\rangle \right) \otimes \left(|0\rangle + e^{\frac{5i\pi}{4}} |1\rangle \right) \otimes e^{\frac{35i\pi}{4}} |1\rangle$$

\downarrow \downarrow \downarrow

$e^{\frac{2\pi i}{2} \cdot 5}$ $e^{\frac{2\pi i}{4} \cdot 5}$ $e^{\frac{2\pi i}{8} \cdot 5}$

$$|\psi_3\rangle = |101\rangle \otimes e^{\frac{35i\pi}{4}} |1\rangle \longrightarrow \theta = \frac{5}{2^3} = \frac{5}{8}$$

19. Quantum counting

Si tratta di un algoritmo quantistico per contare in modo efficiente il numero di soluzioni per un determinato problema di ricerca. L'algoritmo si basa sull'algoritmo di stima della fase quantistica e sull'algoritmo di ricerca di Grover.

I problemi di conteggio sono comuni in diversi campi, per quanto riguarda il calcolo quantistico, è necessaria la capacità di eseguire il conteggio quantistico in modo efficiente per utilizzare l'algoritmo di ricerca di Grover (poiché l'esecuzione dell'algoritmo di ricerca di Grover richiede di sapere quante soluzioni esistono). Inoltre, questo algoritmo risolve il problema dell'esistenza quantistica (vale a dire, ci dice se esiste una soluzione al problema).

Il problema in breve:

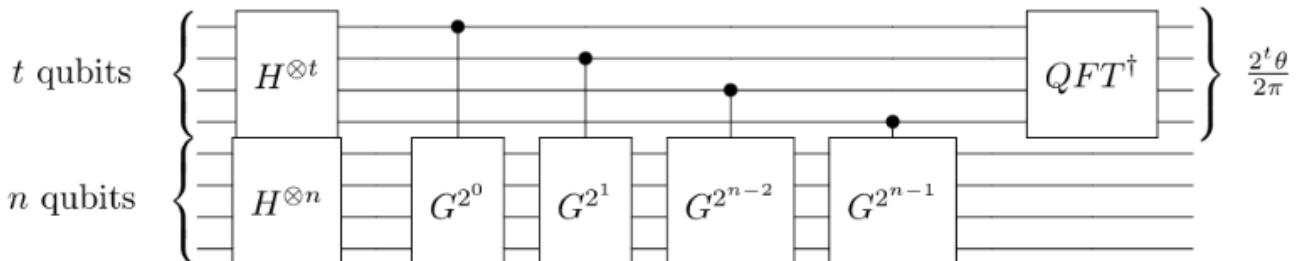
Consideriamo un set finito $\{0,1\}^n$ di dimensione $N=2^n$ e un set B di soluzioni. Definiamo:

$$\begin{cases} f : \{0,1\}^n \rightarrow \{0,1\} \\ f(x) = \begin{cases} 1 & x \in B \\ 0 & x \notin B \end{cases} \end{cases}$$

In altre parole f è la funzione indicatore di B .

Calcola il numero di soluzioni $M = |f^{-1}(1)| = |B|$

L'algoritmo:



1. L'input è costituito da due registri: i p qubits in alto rappresentano il primo registro, gli n qubit in basso il secondo registro;
2. Lo stato iniziale del sistema è $|0\rangle^{\otimes p}|0\rangle^{\otimes n}$. Dopo l'applicazione del gate H a ciascun registro separatamente otteniamo:

$$\frac{1}{2^{p/2}}(|0\rangle + |1\rangle)^{\otimes p} \quad \text{per il primo registro, e}$$

$$\frac{1}{2^{n/2}}(|0\rangle + |1\rangle)^{\otimes n} = \frac{1}{\sqrt{N}} \sum_{x=0}^{N-1} |x\rangle \quad \text{per il secondo registro.}$$

3. Poiché la dimensione dello spazio è $|\{0,1\}^n| = 2^n = N$ e il numero di soluzione è $|B| = M$, possiamo definire gli stati normalizzati:

$$|\alpha\rangle = \frac{1}{\sqrt{N-M}} \sum_{x \notin B} |x\rangle, \quad \text{and} \quad |\beta\rangle = \frac{1}{\sqrt{M}} \sum_{x \in B} |x\rangle.$$

Notare che:

$$\sqrt{\frac{N-M}{N}}|\alpha\rangle + \sqrt{\frac{M}{N}}|\beta\rangle = \frac{1}{\sqrt{N}} \sum_{x=0}^{N-1} |x\rangle,$$

che è lo stato del secondo registro dopo la trasformata di Hadamard.

La visualizzazione geometrica dell'algoritmo di Grover mostra che nello spazio bidimensionale attraversato da $|\alpha\rangle$ e $|\beta\rangle$, l'operatore di Grover è una rotazione in senso antiorario; quindi, può essere espresso come:

$$G = \begin{bmatrix} \cos \theta & -\sin \theta \\ \sin \theta & \cos \theta \end{bmatrix}$$

Dalle proprietà delle matrici di rotazione sappiamo che G è una matrice unitaria con i due autovalori $e^{\pm i\theta}$.

4. Da qui in poi, seguiamo lo schema dell'algoritmo di stima di fase quantistica: applichiamo gli operatori di Grover controllati seguiti dalla trasformata di Fourier quantistica inversa; e secondo l'analisi, troveremo la migliore approssimazione di p-bit al numero reale θ .
5. Supponendo che la dimensione N dello spazio sia almeno il doppio del numero di soluzioni ($M \leq \frac{N}{2}$) un risultato dell'analisi dell'algoritmo di Grover è:

$$\sin \frac{\theta}{2} = \sqrt{\frac{M}{N}}.$$

Quindi, se troviamo θ , possiamo anche trovare il valore di M (perché N è noto).

L'errore

$$\frac{|\Delta M|}{N} = \left| \sin^2\left(\frac{\theta + \Delta\theta}{2}\right) - \sin^2\left(\frac{\theta}{2}\right) \right|$$

è determinato dall'errore nella stima del valore di θ . L'algoritmo di stima di fase quantistica trova, con alta probabilità, la migliore approssimazione p-bit di θ ; questo significa che se p è abbastanza grande, avremo $\Delta\theta \approx 0$ quindi $|\Delta M| \approx 0$.

Algoritmo di ricerca di Grover per un numero di soluzioni inizialmente sconosciuto:

$$\frac{\pi}{4} \sqrt{\frac{N}{M}}$$

Nell'algoritmo di ricerca di Grover, il numero di iterazioni da eseguire è

Pertanto, se N è noto e M è calcolato dall'algoritmo di conteggio quantistico, il numero di iterazioni per l'algoritmo di Grover è facilmente calcolabile.

Problema di esistenza quantistica:

Il problema dell'esistenza quantistica è un caso speciale di conteggio quantistico in cui non vogliamo calcolare il valore di M, ma desideriamo solo sapere se $M \neq 0$ o meno. Una banale soluzione a questo problema è usare direttamente l'algoritmo di conteggio quantistico: l'algoritmo produce M, quindi controllando se $M \neq 0$ otteniamo la risposta al problema di esistenza. Questo approccio comporta alcune informazioni generali perché non siamo interessati al valore di M.

L'uso di una configurazione con un numero ridotto di qubit nel registro superiore non produrrà una stima accurata del valore di θ , ma sarà sufficiente per determinare se M è uguale a zero o meno.

Implementazione:

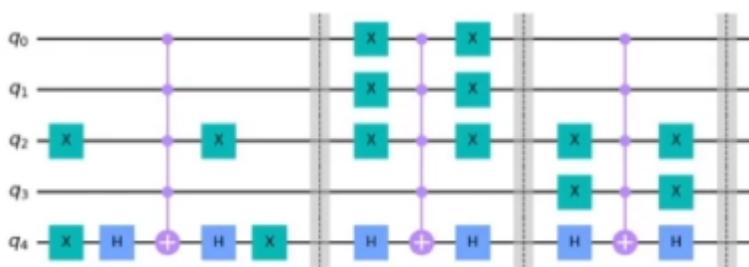
Creiamo una funzione che genera il phase oracle dell'operatore di Grover:

```
def GeneralPhaseOracle(wl):
    n = len(wl[0])
    qc = QuantumCircuit(n)

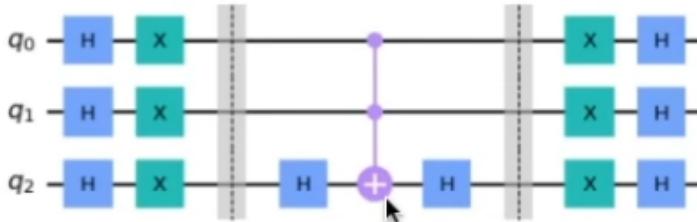
    for sol in wl:
        for i in range(n):
            if(sol[i]=='0'):
                qc.x(i)
        qc.h(n-1)
        qc.mcx(list(range(n-1)),n-1)
        qc.h(n-1)
        for i in range(n):
            if(sol[i]=='0'):
                qc.x(i)
        #qc.barrier()
    qc.draw_circuit(qc)
    return qc
```

Facciamo un esempio. Diamo in input:

```
GeneralPhaseOracle(['11010','00011','11001'])
```



Applichiamo adesso la riflessione (chiamata anche diffusione) che segue questo schema:



Il metodo lo creiamo in funzione del metodo precedente:

```
def diffuser(n):
    s = '0'*n
    xr = QuantumRegister(n)
    qc = QuantumCircuit(xr)
    qc.h(xr)
    qc = qc.compose(GeneralPhaseOracle([s]))
    qc.h(xr)
    #qc.draw_circuit(qc)
    return qc
```

Definiamo l'operatore generico di Grover, unendo phase oracle e operatore di riflessione:

```
def GGroverOperator(wl):
    n = len(wl[0])
    qc = QuantumCircuit(n)
    qc = qc.compose(GeneralPhaseOracle(wl))
    qc = qc.compose(diffuser(n))
    qc.draw_circuit(qc)
    qc = qc.to_gate()
    qc.label = 'Grov'
    return qc
```