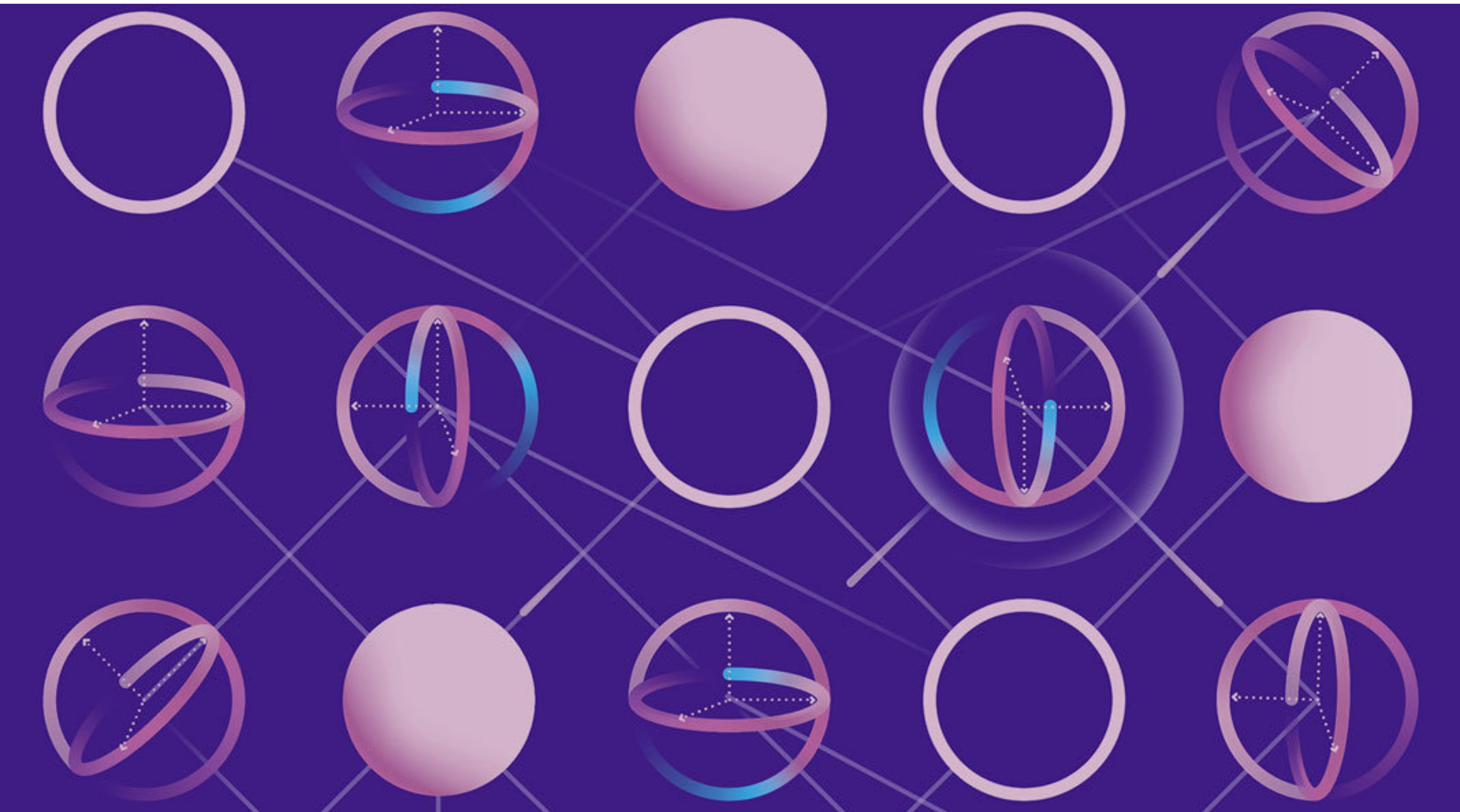




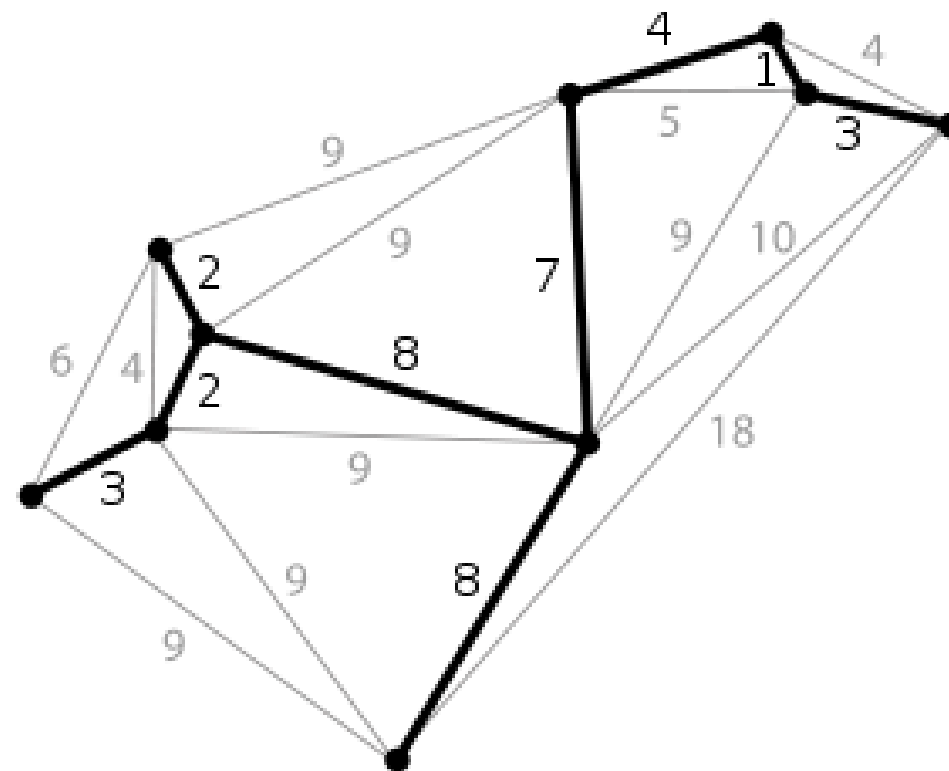
# Quantum MST



## 1

# MST - Definizione

Dato un grafo pesato non orientato, il Minimum Spanning Tree (**MST**) è un sottoinsieme dei suoi archi che **connette tutti i nodi senza cicli e con la somma dei pesi minima**.



La ricerca di un MST ha **applicazioni dirette in molti ambiti**, alcuni esempi possono essere: **path finding in reti, ottimizzazioni di reti di trasporto, reti elettriche**.

Il MST può essere ricercato anche come **subroutine** di altri problemi, come per esempio il **traveling salesman problem**.

## 2

# MST - Algoritmi classici

Gli **algoritmi più usati** per trovare il MST e le relative **complessità temporali** sono:

- **Prim**  $O(|E| + |V| * \log |V|)$
- **Borůvka**  $O(|E| * \log |V|)$
- **Kruskal**  $O(|V| * \log |V|)$

Dove **E** è il numero di archi nel grafo e **V** è il numero di nodi nel grafo.

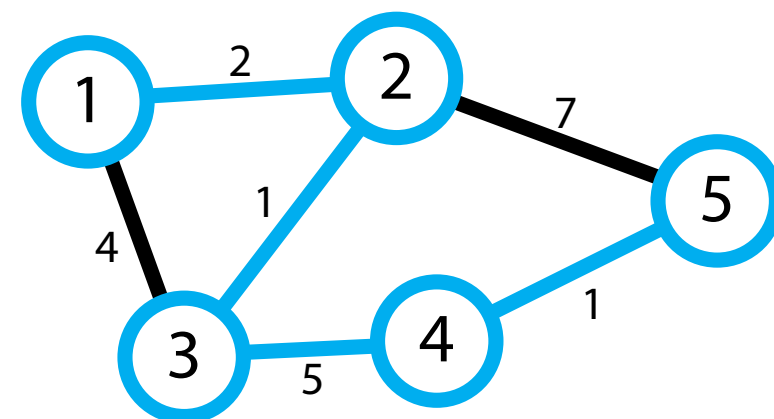
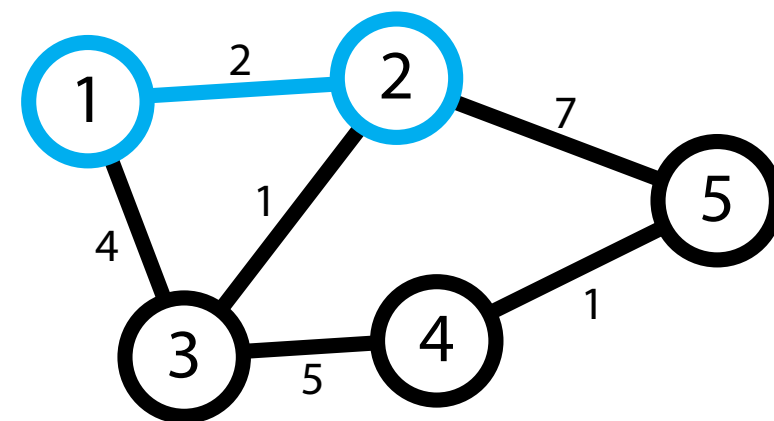
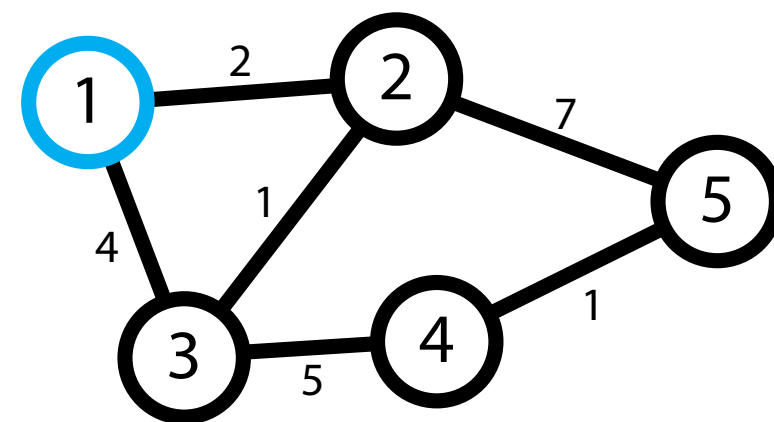
# 3

## MST - Prim

Selezioniamo un **vertice qualunque** del grafo e lo aggiungiamo al MST.

Aggiungiamo il vertice collegato al MST parziale con l'**arco di peso minore**. Questo nuovo vertice **non deve creare cicli**.

**Iteriamo il secondo passaggio** finché tutti i nodi non sono stati aggiunti al MST.



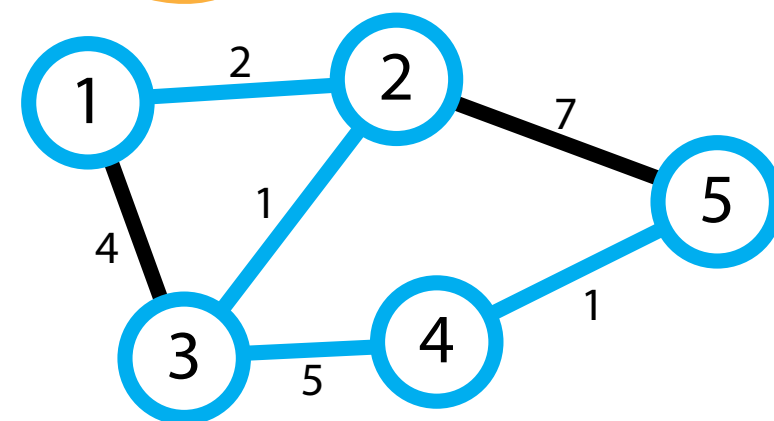
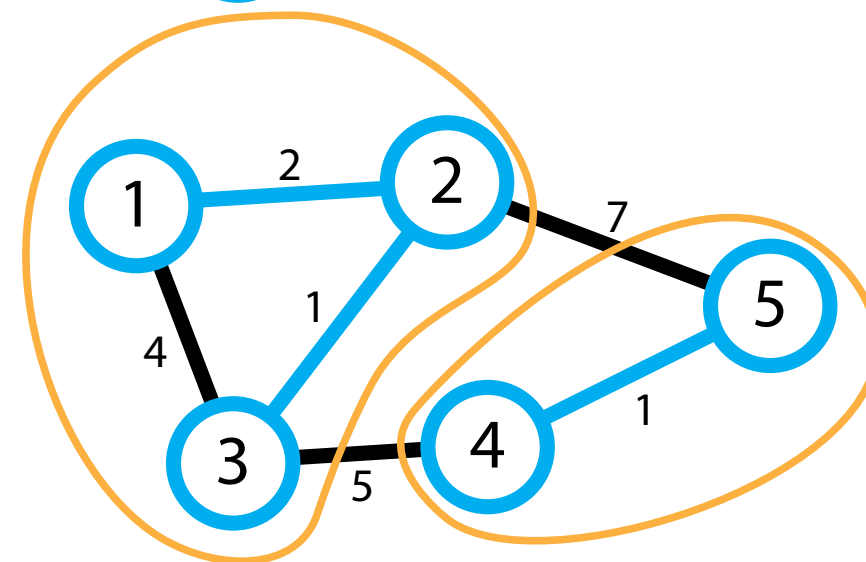
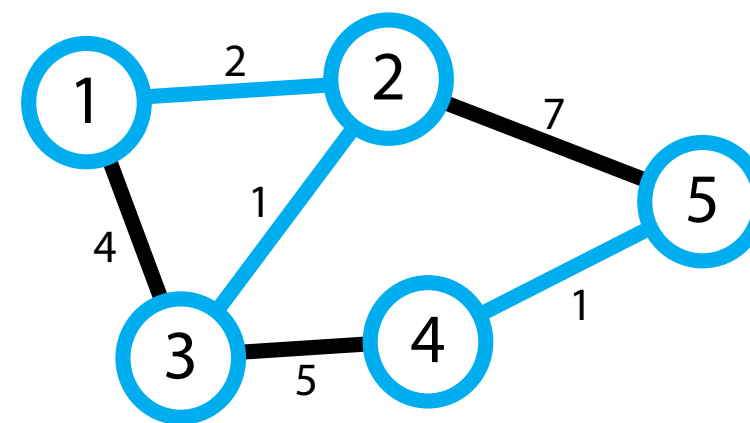
# 4

## MST - Borůvka

Per ogni nodo del grafo troviamo l'**arco uscente di minor peso** e lo aggiungiamo al MST.

Tutti i vertici collegati da archi selezionati formeranno un **supernodo**.

Iteriamo il primo passaggio sui supernodi finché l'algoritmo non converge



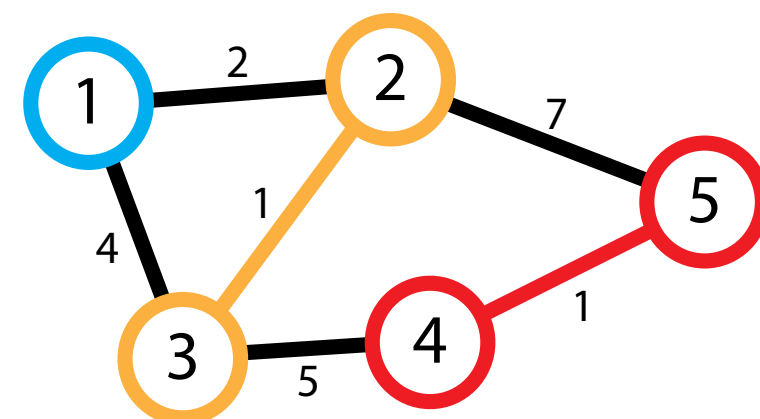
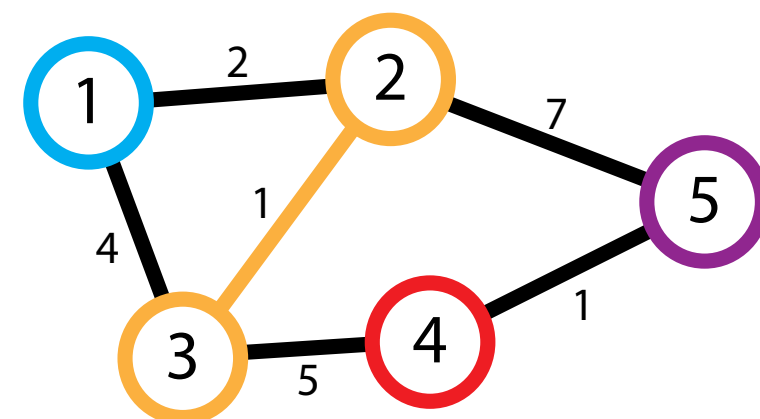
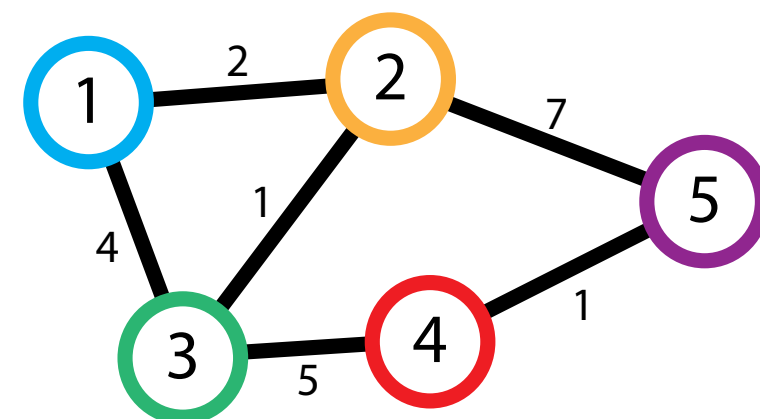
# 5

## MST - Kruskal

Consideriamo ogni **nodo** come un **albero**.

Selezioniamo dall'insieme degli **archi** quello con **peso minore**. Se questo unisce **due alberi distinti, allora viene aggiunto al MST**. Il nuovo arco unisce due alberi in uno unico.

**Iteriamo il secondo passaggio** finché tutti i gli alberi sono stati fusi in uno unico.



## 1

# Qubit - Definizione

Un bit quantistico, o **qubit**, è l'equivalente nella computazione quantum di un bit classico. Come un bit è l'**unità minima di informazione** nella computazione classica, il qubit lo è nel mondo quantum.

**Bit**  
(Classical Computing)

0



1

**Qubit**  
(Quantum Computing)

0



1

In un sistema classico, un bit può essere in uno stato solo. Al contrario, la meccanica quantistica permette ai qubit di essere in uno stato di **superposizione**. Questa proprietà dà alla computazione quantum un nuovo strumento per sbloccare una **nuova classe di algoritmi più efficienti**.



## 2

# Qubit - Rappresentazione

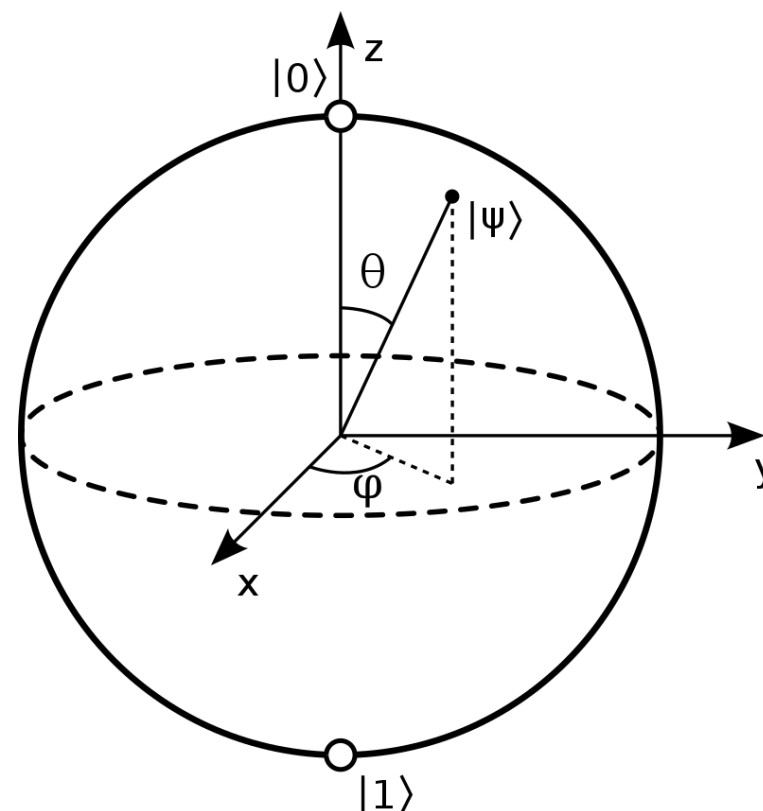
## Braket notation

Un qubit può essere interpretato come un **vettore** le cui componenti sono una **sovrapposizione della sua base ortonormale**. La base, nel nostro caso, è composta dai **possibili valori del qubit** dopo la misurazione, 0 o 1.

$$|\psi\rangle = \alpha |0\rangle + \beta |1\rangle \quad \alpha, \beta \in \mathbb{C}$$

## Bloch sphere

Un qubit può essere interpretato come un **vettore unitario** in uno spazio 3D. Gli stati della base sono antipodali. Tutti gli **stati intermedi** sulla sfera unitaria sono una **superposizione degli stati della base**, 0 e 1.





## 3

# Quantum gate - Definizione

I quantum gate possono essere considerati come un **equivalente quantum delle porte logiche classiche**. Questi gate cambiando lo **stato del sistema** manipolando la sua **distribuzione di probabilità** piuttosto che applicare una **funzione logica**.

Questi gate devono essere **reversibili** in modo tale da preservare le proprietà della computazione quantistica. Quindi i gate saranno rappresentati da **matrici unitarie**.

$$U |a\rangle = \begin{bmatrix} u_{11} & u_{12} \\ u_{21} & u_{22} \end{bmatrix} \begin{bmatrix} a_1 \\ a_2 \end{bmatrix}$$

Queste matrici unitarie possono agire su **uno o più qubit** alla volta. Se agiscono su **n qubit** avranno una dimensione di  **$2^n \times 2^n$** .

## 4

# Quantum gate - Singolo qubit

**Hadamart gate**

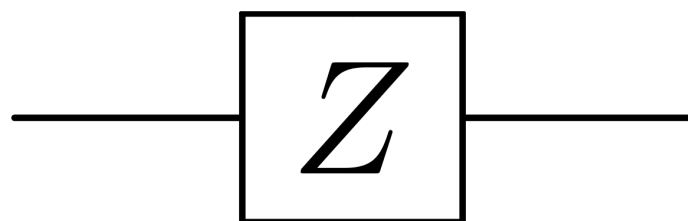
Pone il qubit in ingresso in uno stato di superposizione



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

**Pauli-Z**

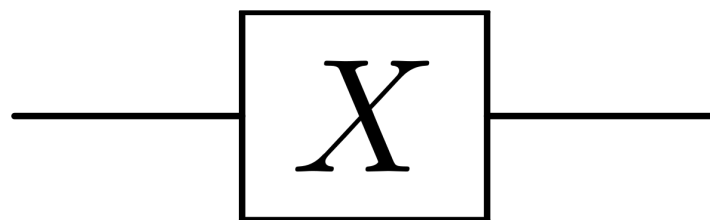
Ruota la fase del qubit in ingresso di 180°



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

**Pauli-X**

Inverte lo stato del qubit in ingresso



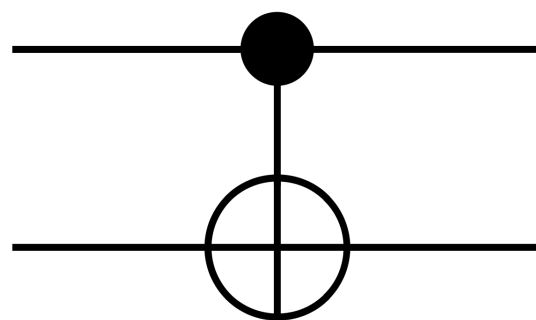
$$X = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}$$

## 5

# Quantum gate - Controllati

## CNOT

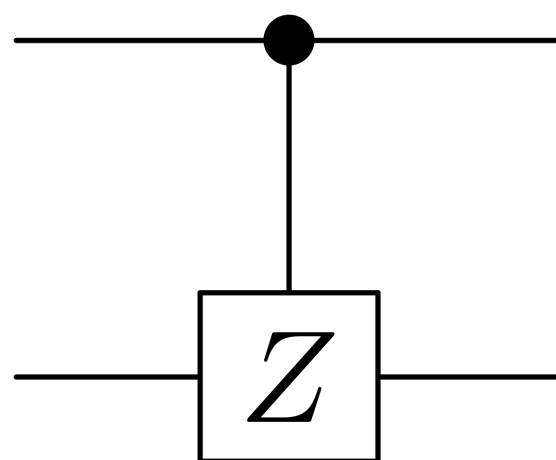
Inverte il qubit in ingresso se il qubit di controllo è posto a 1



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

## Controlled Z

Ruota la fase del qubit in ingresso di 180° se il qubit di controllo è posto a 1



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

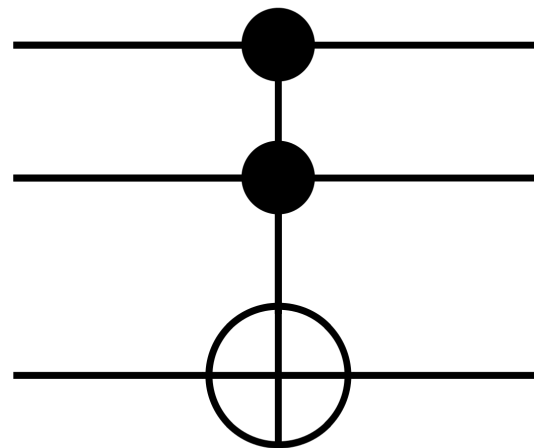
## 6

# Quantum gate - Controllati a più qubit

I gate controllati possono essere **estesi ad un numero arbitrario di qubit** di controllo.

## CCNOT/ Toffoli gate

Inverte il qubit in ingresso se i qubit di controllo sono posti a 1.



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

Se pur di estrema utilità i gate controllati a più bit sono di **difficile realizzazione pratica** in quanto richiedono un **numero esponenziale di gate** per la loro realizzazione.

Da notare che il **CCNOT è un gate universale** per la computazione classica, il che vuol dire che ogni **gate classico** come AND, OR, XOR, ecc... può essere **ricostruito dal CCNOT**.

Questo prova che i computer quantistici sono almeno **Touring completi**.

## 7

# Quantum speed up - Parallelismo

Il vantaggio dei computer quantistici è la loro abilità di **parallelizzare la computazione**. Consideriamo una funzione computabile con  $k$  gate classici.

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

Questo circuito può essere implementato da  $n$  **CCNOT gate**. Chiameremo questo circuito  $U$ . Se **appliciamo l'operazione  $U$**  ad un ipotetico **registro posto in superposizione** avremo

$$|\psi\rangle = \frac{1}{\sqrt{2^n}}(|0\rangle + |1\rangle + |2\rangle + \dots + |2^n - 1\rangle)$$

$\downarrow U$

$$|\psi\rangle = \frac{1}{\sqrt{2^n}}(|f(0)\rangle + |f(1)\rangle + |f(2)\rangle + \dots + |f(2^n - 1)\rangle)$$

Quindi la computazione della funzione viene eseguita un **numero di volte esponenziale** (possibili combinazioni dei qubit) facendo lo stesso numero di operazioni per **elaborare un singolo valore**.

## 1

# Grover - Definizione

L'algoritmo di Grover è un **algoritmo quantistico** che trova con alta **probabilità** l'input ad una **black box** che produce un particolare output.

$$f : \{0, 1, \dots, N - 1\} \rightarrow \{0, 1\}, \quad \text{where} \quad N = 2^n, \quad n \in \mathbb{N}$$

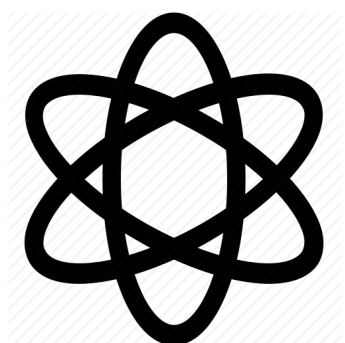
$$f(x) = \begin{cases} 0 & \text{if } x \neq w \\ 1 & \text{if } x = w \end{cases}$$

Per esempio si ipotizzi una funzione che mappa un insieme di interi all'insieme  $\{0, 1\}$ . La funzione sarà associata alla blackbox e Grover troverà quei input che avranno come risultato 1.

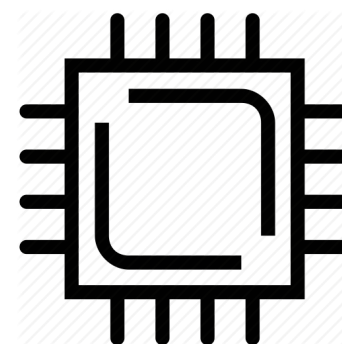
## 2

# Grover - Vantaggi (complessità)

Il più grande vantaggio nell'utilizzo di Grover è la sua **complessità** rispetto ad un algoritmo classico.



$$O(\sqrt{N}) * k$$



$$O(N)$$

Da notare che la **complessità** degli **algoritmi quantistici** viene misurata in **chiamate all'oracolo e non temporalmente**. Inoltre bisogna considerare anche la complessità della **creazione dell'oracolo**, la quale dipende dal problema che si sta affrontando.



## 3

# Grover - Fasi dell'algoritmo

## Super position setup

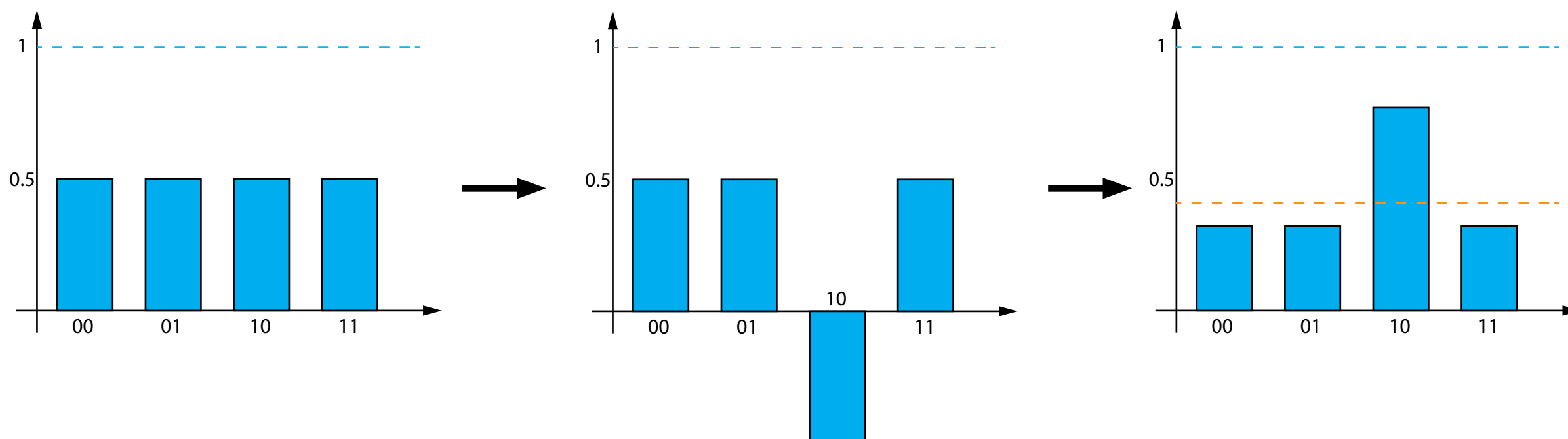
Il registro di input viene posto in **superposizione** attraverso i gate H. quindi ogni stato del registro sarà equiprobabile.

## Phase inversion

Il registro di input viene elaborato dalla blackbox, detta anche **oracolo**, **invertendo la fase** dello stato ricercato dalla funzione codificata nell'oracolo.

## Inversion about the mean

Tutti i possibili stati vengo **invertiti** rispetto all'**ampiezza media** degli stati. Così facendo si **aumenta la probabilità** che il registro si trovi nello stato ricercato



## 4

# Grover - Iterazioni

Per ottenere un risultato migliore le ultime due fasi dell'algoritmo devono essere **iterate un numero preciso di volte**.

**Numero iterazioni ottime  
singolo match**

$$\frac{\pi}{4} \sqrt{N}$$

**Numero iterazioni ottime  
m match**

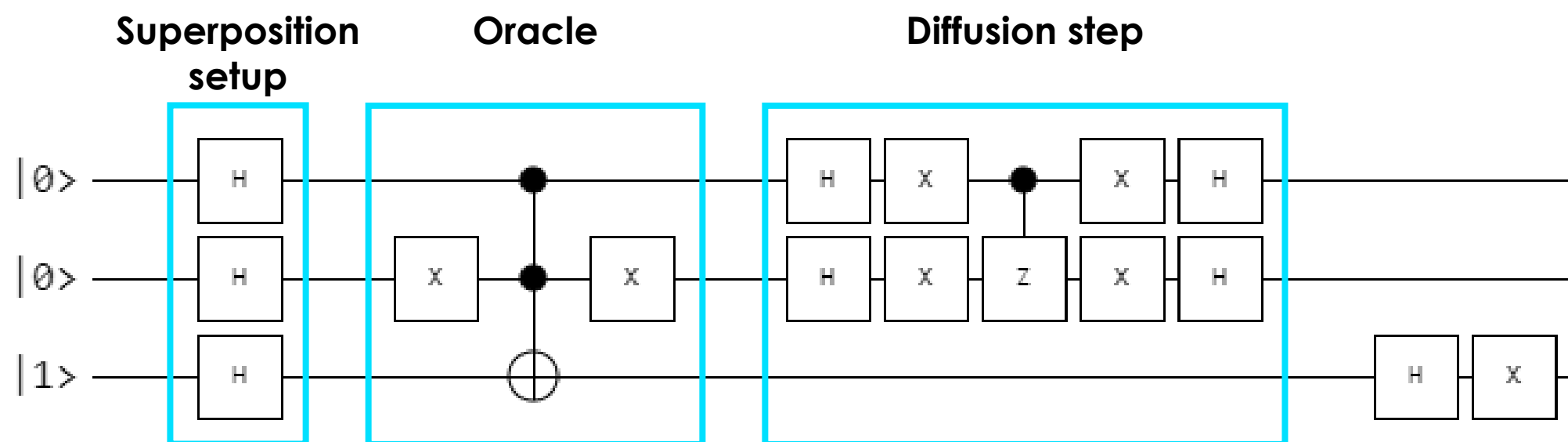
$$\frac{\pi}{4} \sqrt{\frac{N}{M}}$$

Il numero di iterazioni dipende da due fattori la **cardinalità** dei possibili stati del registro di input e il **numero di match**/stati ricercati. Spesso il numero di iterazioni ottime **non è un numero intero**. Questo porta ad una **minore probabilità di successo**.

## 5

# Grover - Implementazione

Prendendo in considerazione l'oracolo dell'esempio precedente, di seguito troviamo una possibile implementazione di Grover.

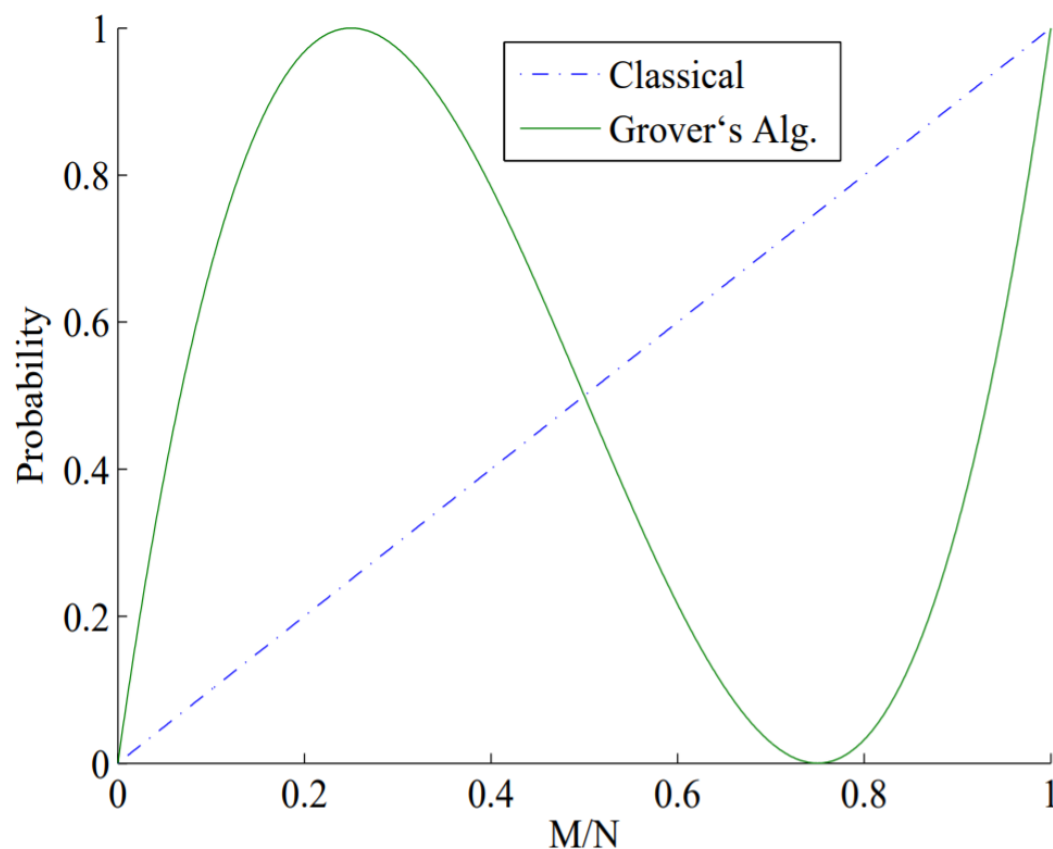


L'oracolo in questo caso codifica la funzione che ha come risultato 1 se l'input è 10. Quindi il compito dell'**oracolo** è quello di **evidenziare gli stati ricercati** e il **diffusion step** **aumenta la probabilità** che il registro sia nello stato ricercato.

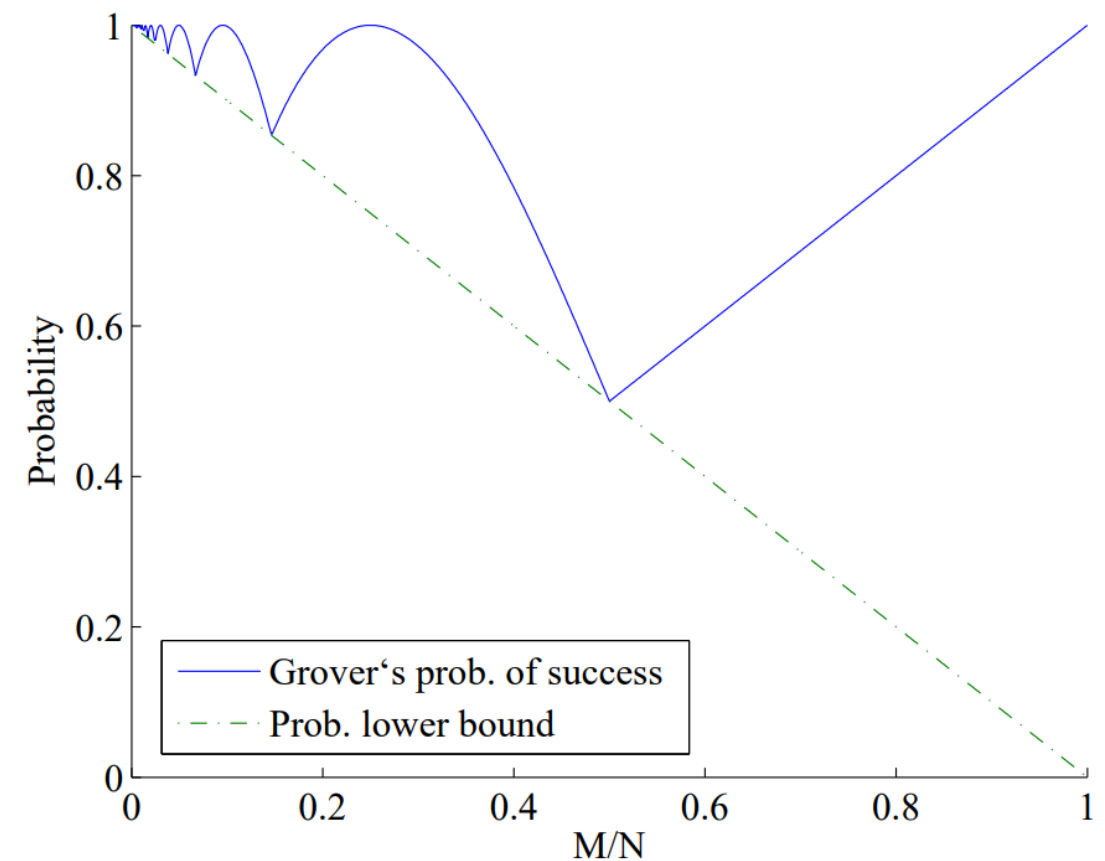
## 6

# Grover - Problematiche

Essendo un **algoritmo probabilistico** è bene aumentare il più possibile le possibilità di successo, ma non è sempre possibile. Infatti nel caso in cui **non si conoscono i numero di match**, l'algoritmo potrebbe essere iterato un numero errato di volte, portando ad un **risultato non attendibile** sistematicamente. Come si può vedere dai grafici sottostanti **all'aumentare dei match** Grover ha la **stessa probabilità di successo di un random guesser classico**.



Probabilità di successo di una singola iterazione  
vs.  
Random guesser classico



Probabilità di successo con iterazione ottimale

## 7

# Grover - Problema del phase shift

Classicamente Grover applica uno phase shift, però è **difficilmente implementabile in modo diretto**. Per ovviare a questo problema è utile **trasporre il problema** da un phase shift all'implementazione di una **funzione booleana** nel seguente modo.

## Phase shift diretto

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

## Phase shift attraverso funzione booleana

$$|x\rangle |0\rangle \rightarrow |x\rangle |f(x)\rangle \xrightarrow{Z} (-1)^{f(x)} |x\rangle |f(x)\rangle$$

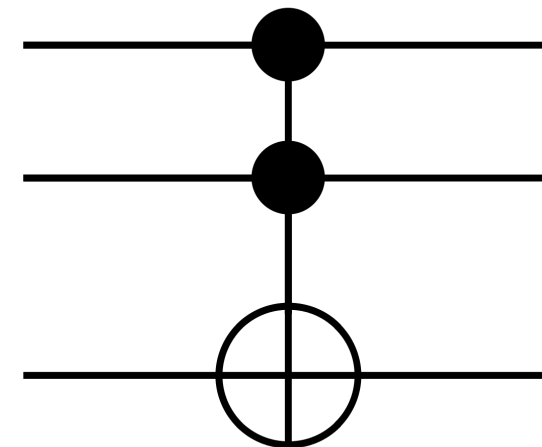
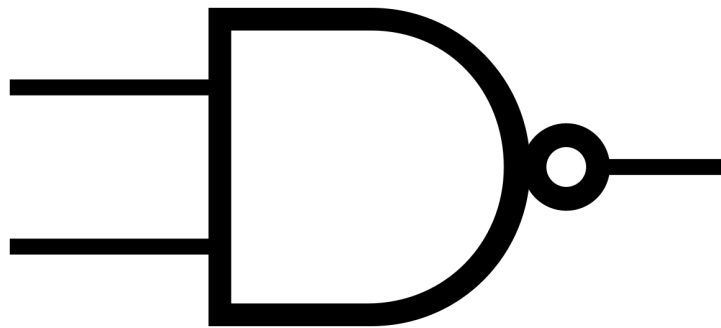
La funzione viene applicata direttamente per poi invertirla attraverso un gate Z

## 1

# Funzioni bool - Costruzione

Per poter implementare la ricerca in Kruskal avremo bisogno di poter rappresentare **funzioni booleane attraverso quantum gate**. Sapendo che nel mondo classico il **CCNOT è un gate universale**, possiamo costruire funzioni booleane arbitrarie.

Gate universali (computazione classica)

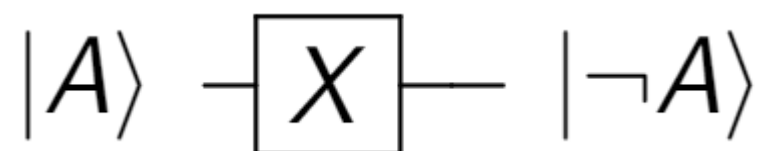
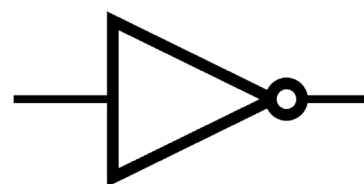


Come per il NAND gate, una combinazione di CCNOT gate può descrivere una **qualunque funzione booleana**.

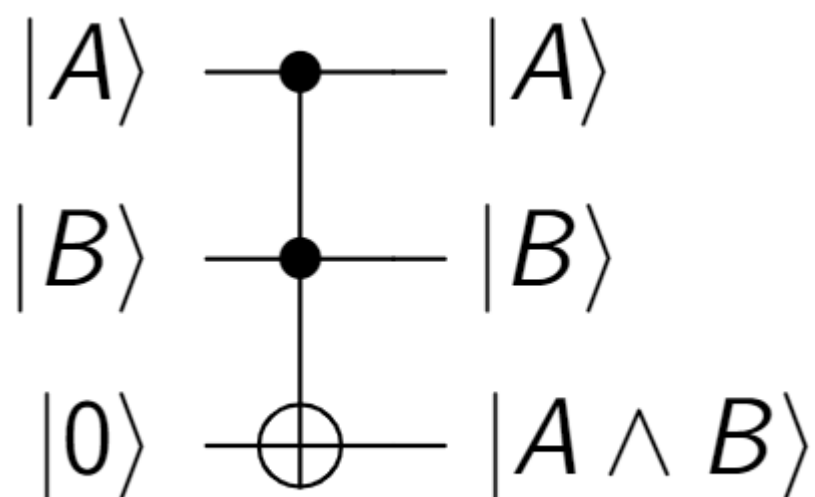
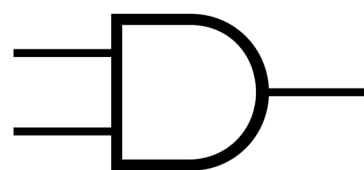
## 2

# Funzioni bool - Porte logiche classiche

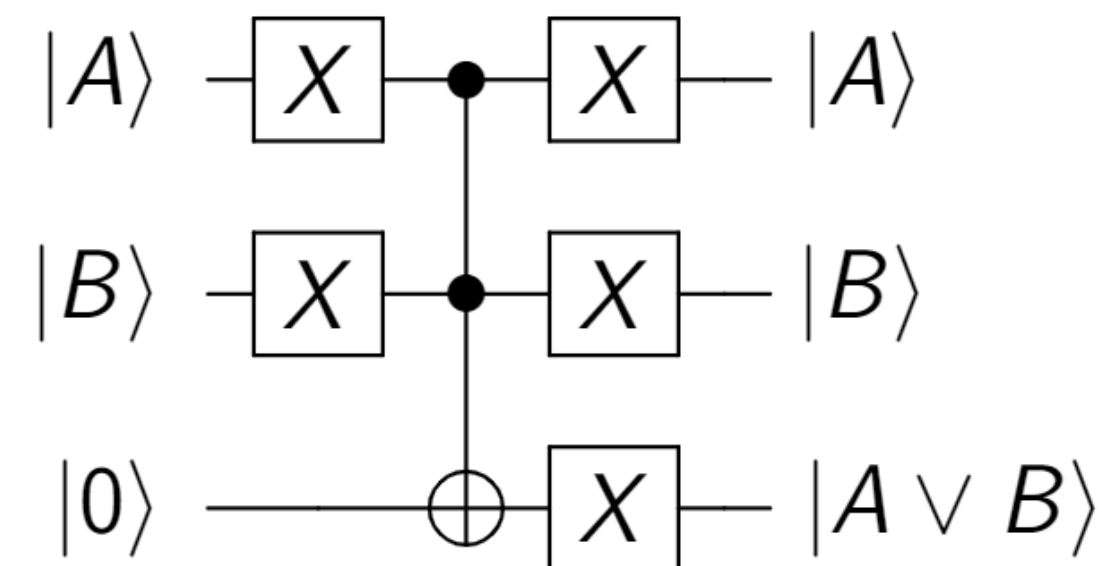
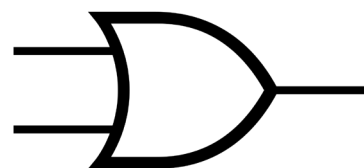
NOT gate



AND gate



OR gate

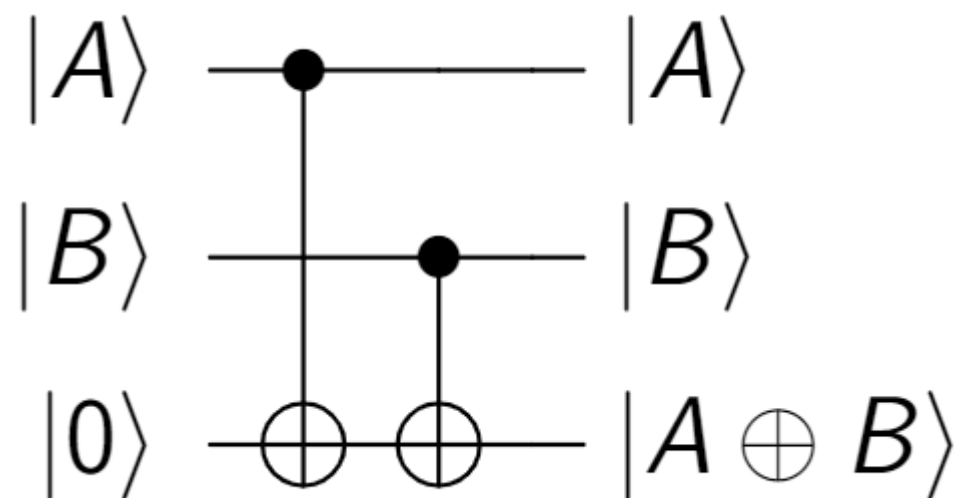
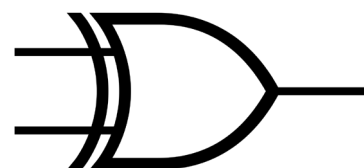




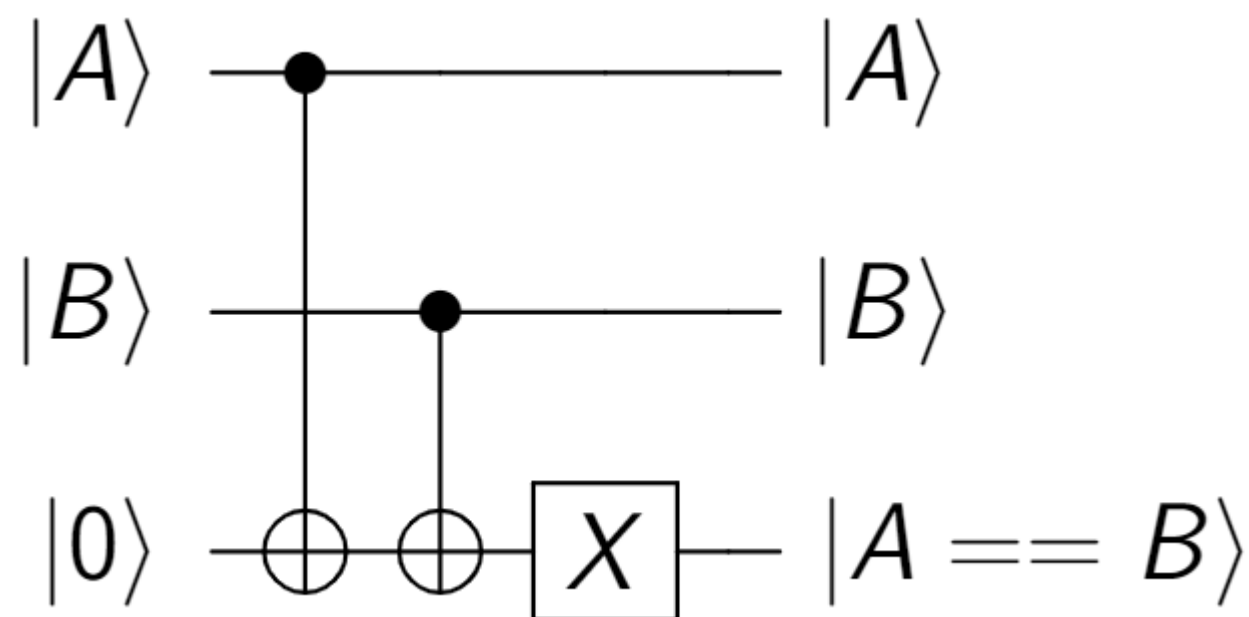
# 3

## Funzioni bool - Porte logiche classiche

XOR gate



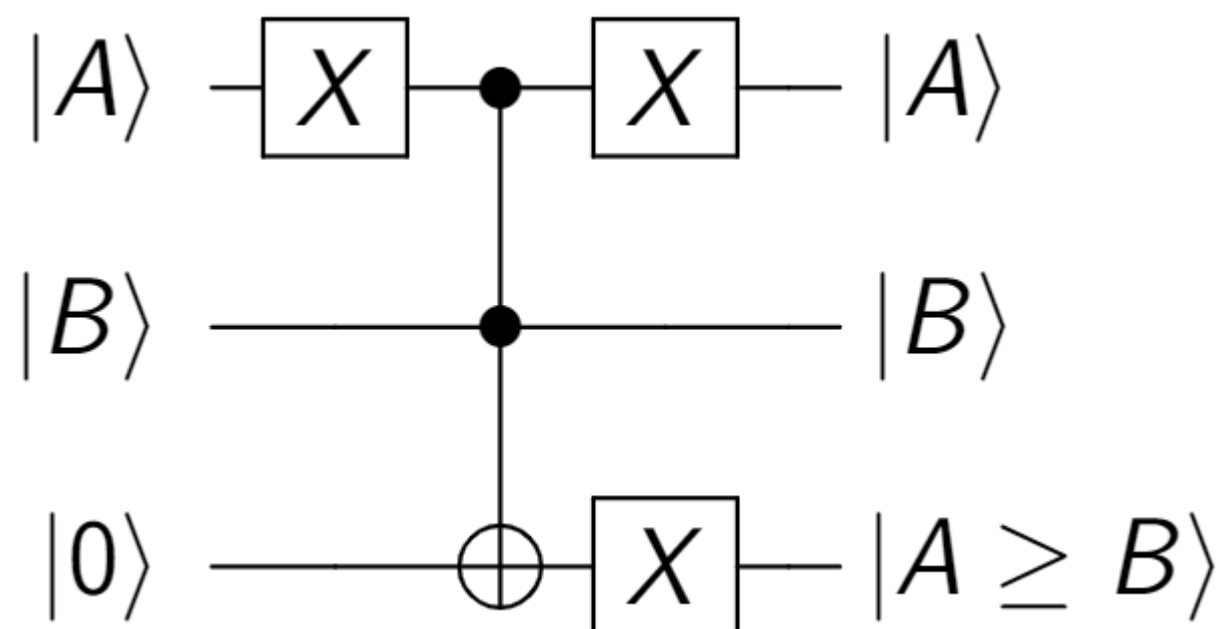
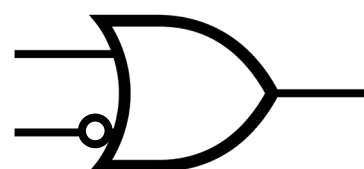
XNOR gate  
Uguaglianza



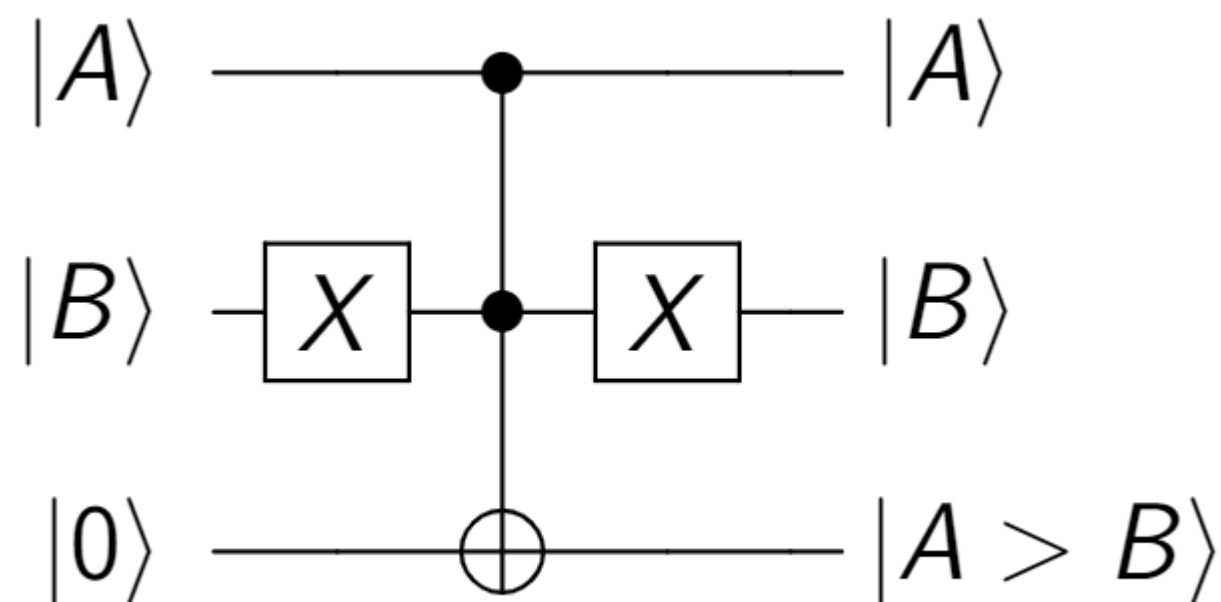
## 4

## Funzioni bool - Porte logiche classiche

A OR NOT(B)  
Maggiore uguale



A AND NOT(B)  
Maggiore

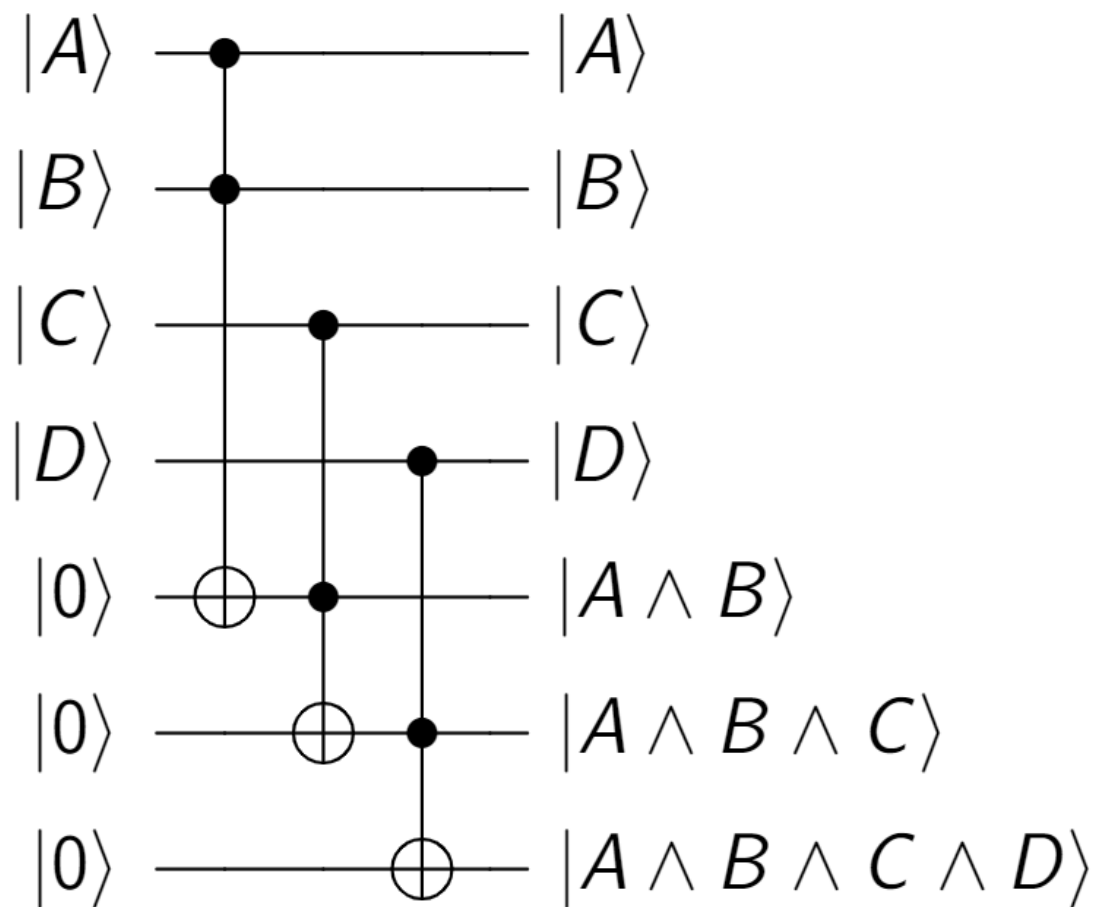


## 5

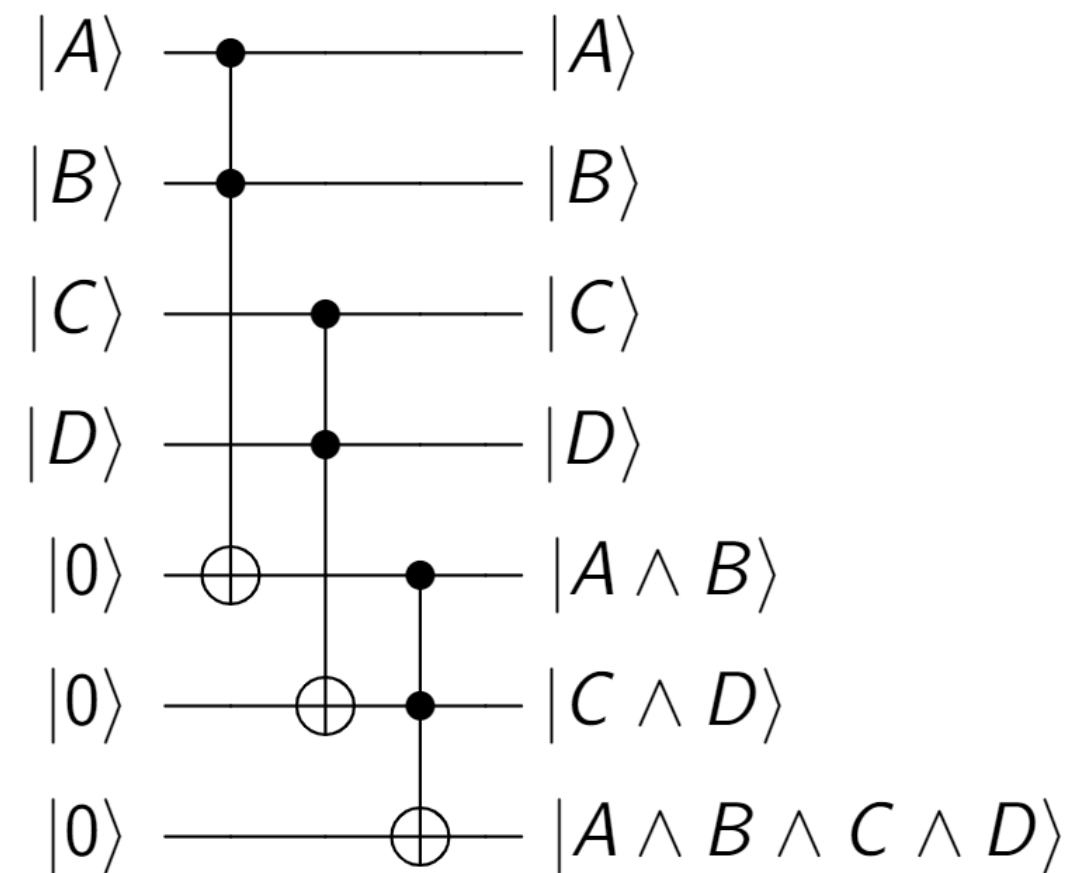
# Funzioni bool - Porte logiche classiche

Nella nostra computazione avremo bisogno di **gate a N qubit**, ma attualmente **non ne esistono implementazioni fisiche**. Nonostante ciò, queste possono essere composte da gate più semplici. Alcune librerie **simulano queste porte** nei seguenti modi.

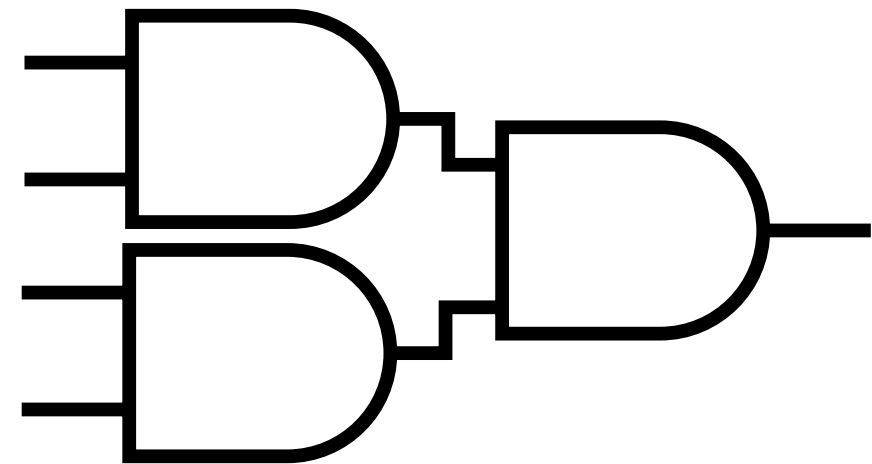
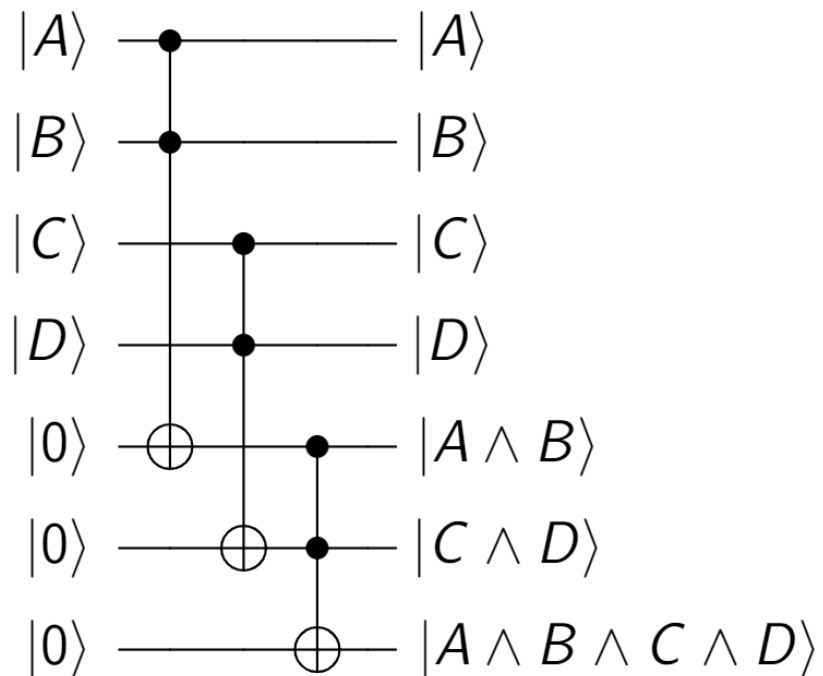
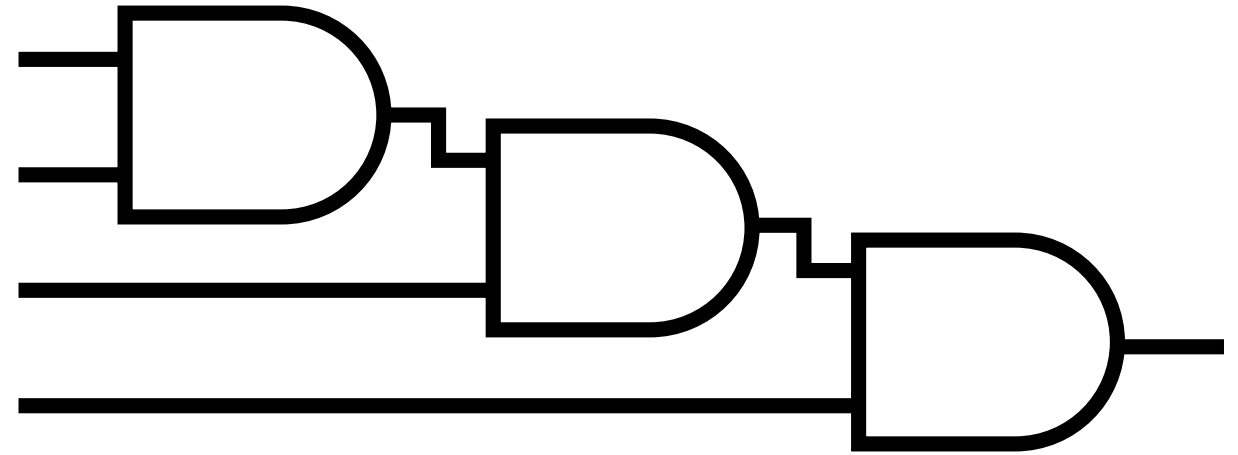
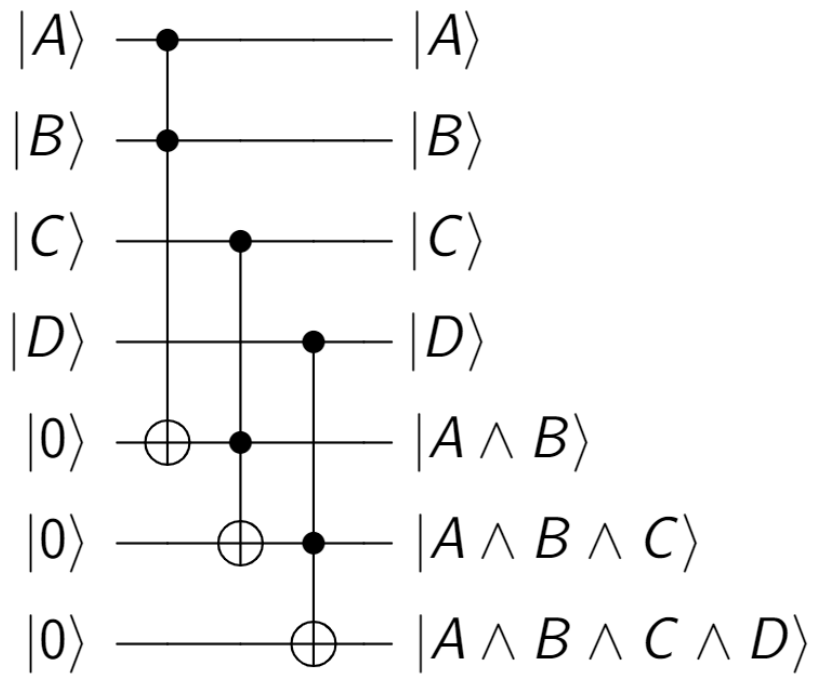
Ladder



Tree



# Funzioni bool - Porte logiche classiche

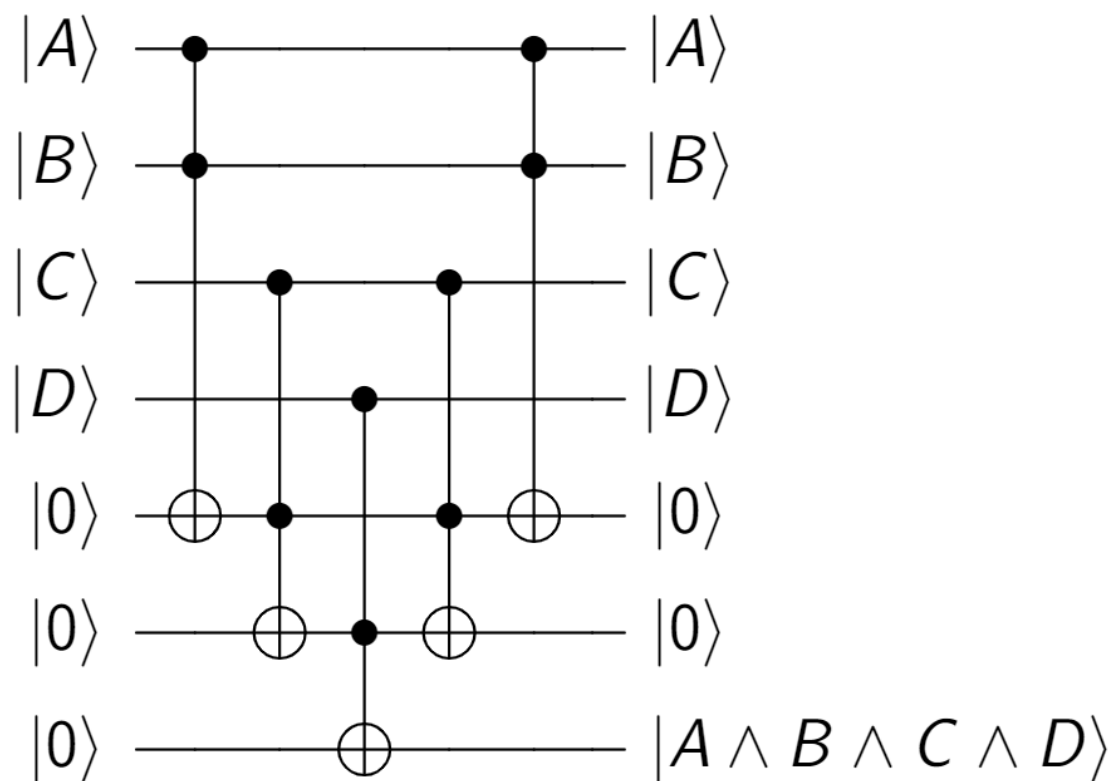


## 7

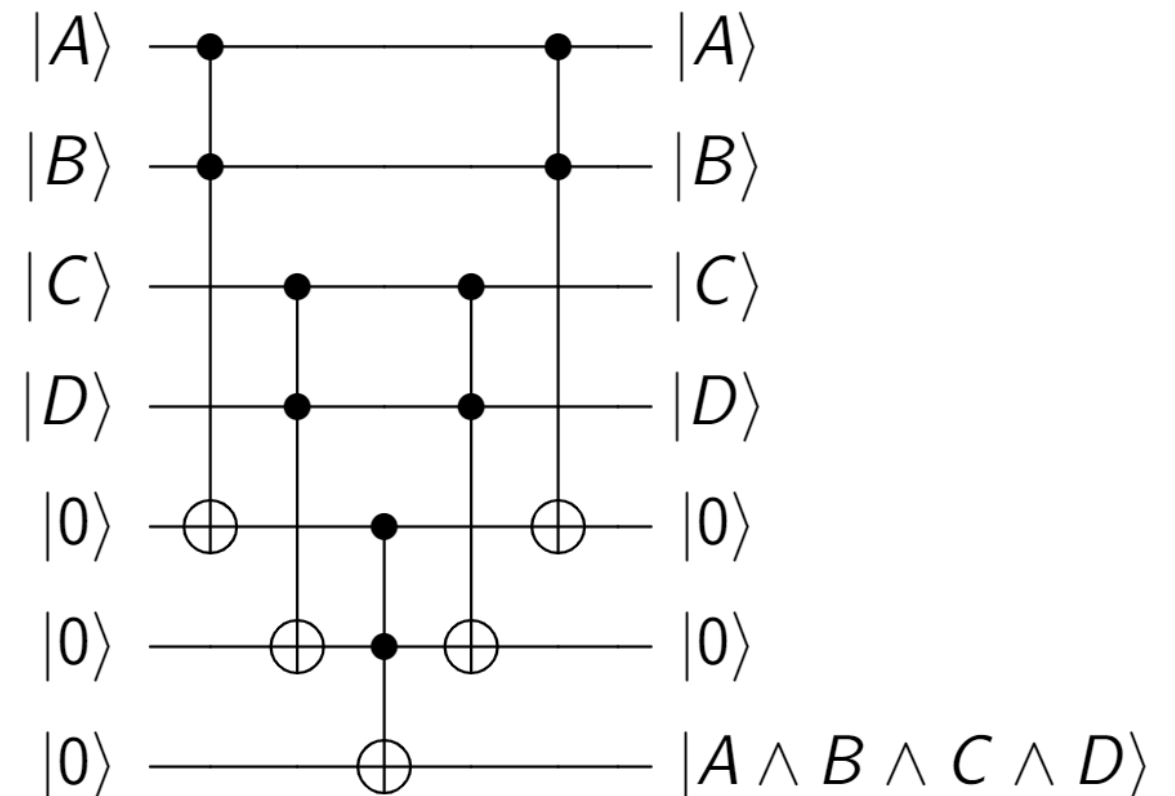
# Funzioni bool - Porte logiche classiche

Solitamente è buona norma **porre il risultato soltanto nell'ultimo qubit**, lasciando gli **altri qubit invariati alla fine della computazione**.

Ladder clean



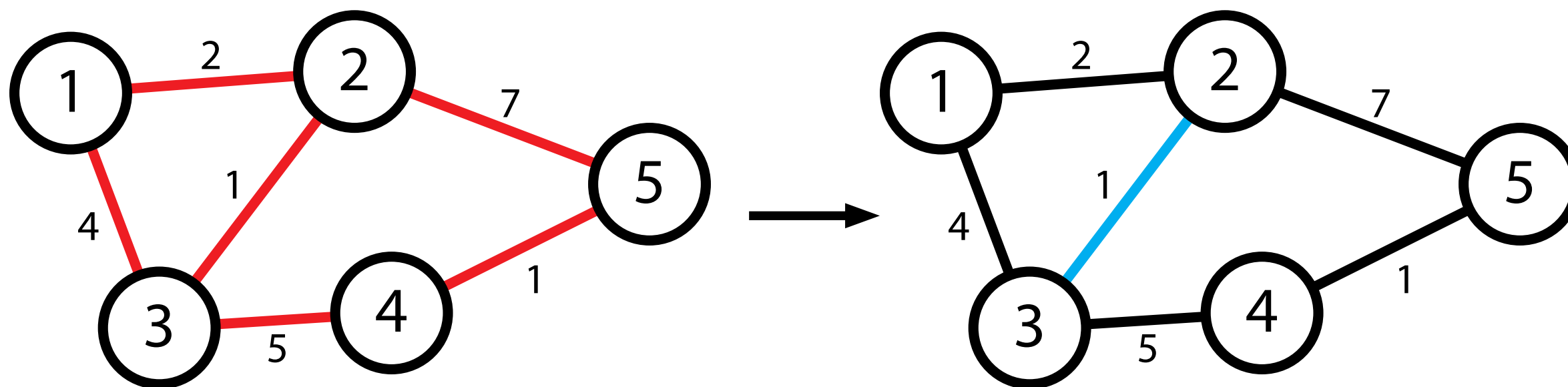
Tree clean



## 1

# MST quantum - Kruskal quantum

Creiamo due insiemi: l'insieme degli **archi appartenenti al MST** e l'insieme di **nodi visitati**.



Per  $|V| - 1$  volte selezioniamo l'**arco di minor peso incidente** su un nodo che non appartiene al set di nodi già visitati. Ogni volta che si seleziona un **arco** questo viene aggiunto all'**insieme degli archi del MST** e il **nodo** appena visitato all'insieme dei **nodi visitati**.

Il mondo quantum può fornirci un **vantaggio nella fase di ricerca di un arco incidente**. Un possibile algoritmo di ricerca quantum è **Grover**.

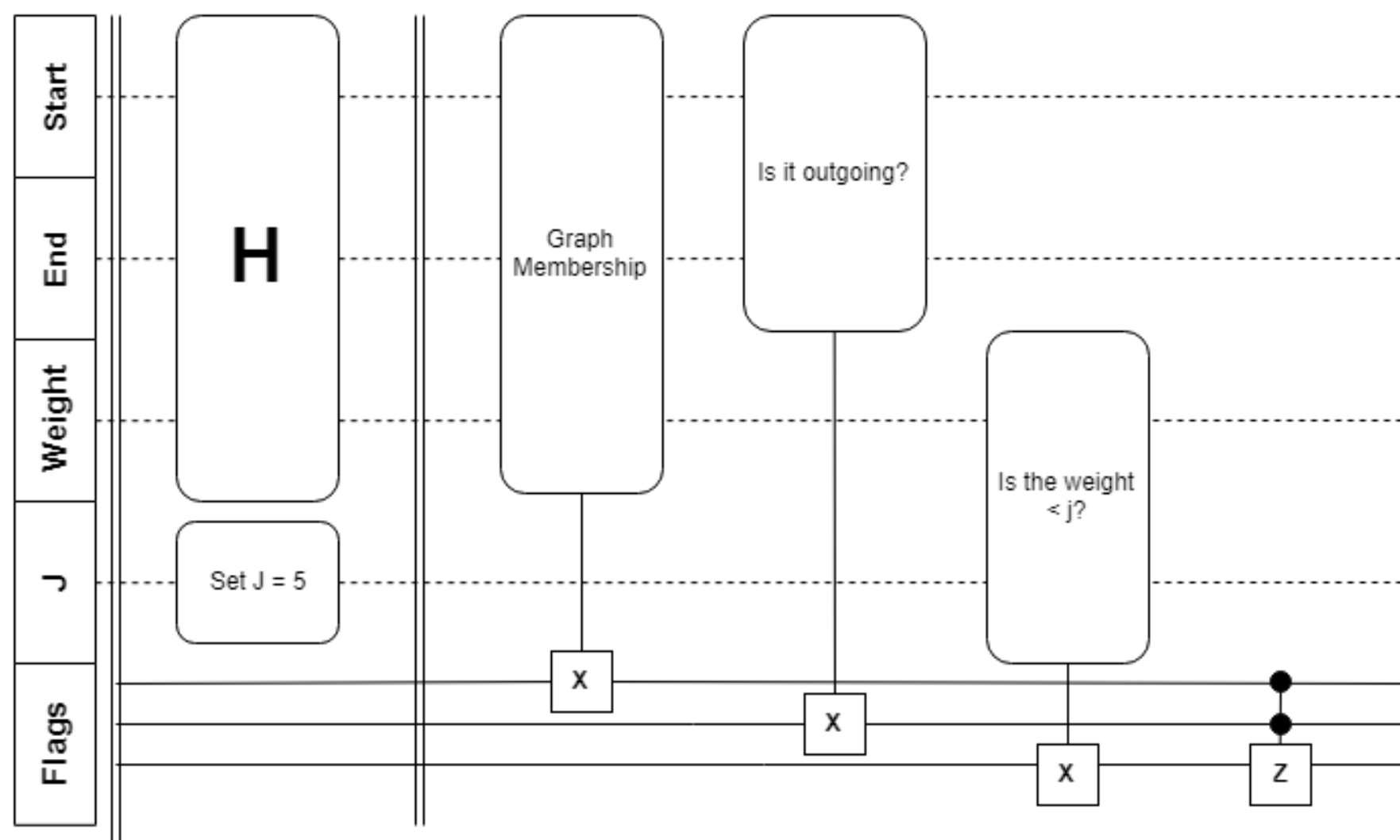
## 2

# MST quantum - Circuito MST

Grover può essere utilizzato nel calcolo di un MST per la **ricerca di archi** e la **valutazione dei loro pesi**. L'oracolo sarà una funzione che avrà come parametri l'**appartenenza** di un arco al grafo, la sua **direzione** e il suo **peso**

Registers Setup

Oracle





# 3

## MST quantum - Creazione oracolo

L'oracolo può essere descritto come una **combinazione di funzioni booleane**, in particolare

$f(x)$  = Appartenenza al grafo && Arco uscente && Peso minimo

Per la **valutazione dei pesi** sorge un problema nella trasposizione a funzione logica del minimo poiché **non è implementabile attraverso una funzione booleana**.

Nei computer classici, dove si hanno a disposizione tanti registri per salvare dati, la ricerca del minimo è un compito facile. Al contrario negli attuali computer quantistici ciò non è possibile, soprattutto considerando che spesso i **valori sono salvati attraverso superposizioni e non direttamente in registri**.

# 4

## MST quantum - Ricerca minimo

Per fare ciò si fissa un **valore j** e si trovano tutti i valori inferiori. Se se ne trova qualcuno si **itera** di nuovo la comparazione **finché il sistema non converge**. Si può dimostrare che la ricerca del minimo convergerà in  $\sqrt{N}$  passaggi.

### Pseudocodice

- 1) Scegliamo un valore intero casuale j
- 2) Iteriamo fino a convergenza
  - a) Se esiste un  $i < j$
  - b) Poniamo  $j = i$

### Numero ieterazioni

$$O(\sqrt{N})$$

## 5

# MST quantum - Setup registri

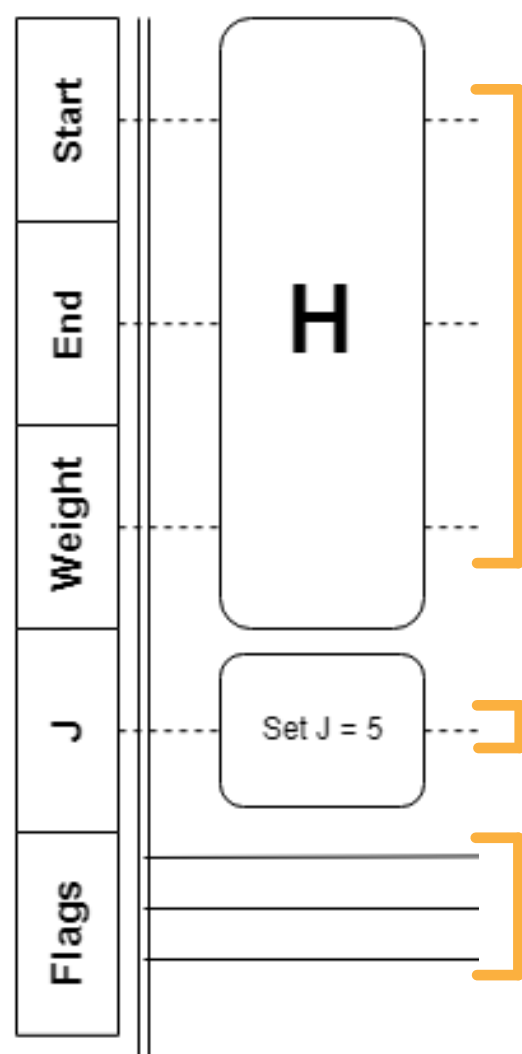
**Start e end** sono registri che descrivono rispettivamente il nodo di partenza e di arrivo di un arco.

**Weight** serve per la gestione dei pesi degli archi.

**J** serve come registro di comparazione per la ricerca del minimo.

**Flag** sono qubits che memorizzeranno i risultati delle sottofunzioni dell'oracolo.

## Registers Setup



Applichiamo H a tutti i qubit dei registri di start, end e pesi

Applichiamo H per descrivere ogni possibile combinazione di nodi e pesi. **Un arco è descritto dalla tupla(start, end, peso)**

Poniamo a un valore intero il registro J

A un valore a cui verranno confrontati gli altri pesi

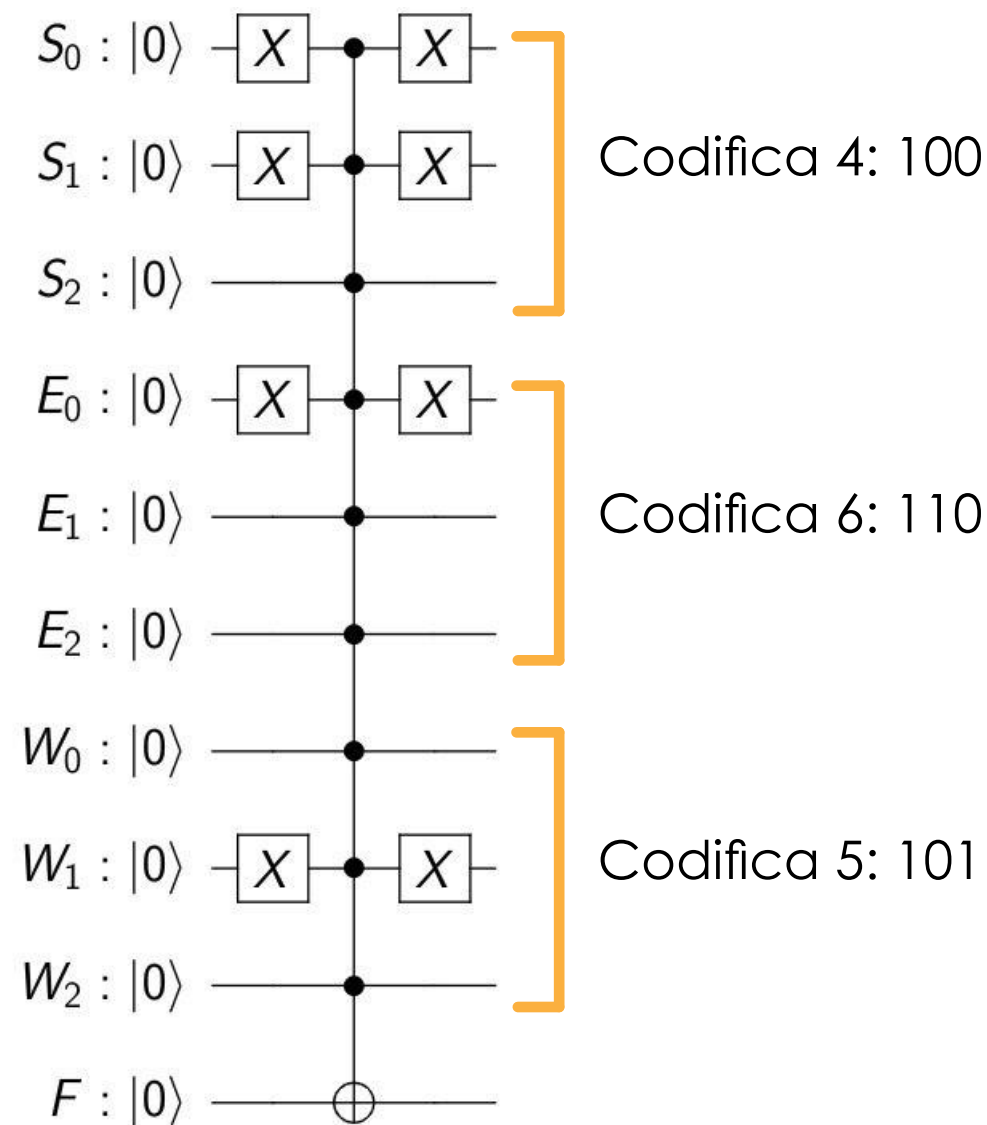
Poniamo a 0 i qubit di flag

A 0 poiché dovranno memorizzare in modo deterministico il risultato delle varie sub-routine

## 6

# MST quantum - Appartenenza al grafo

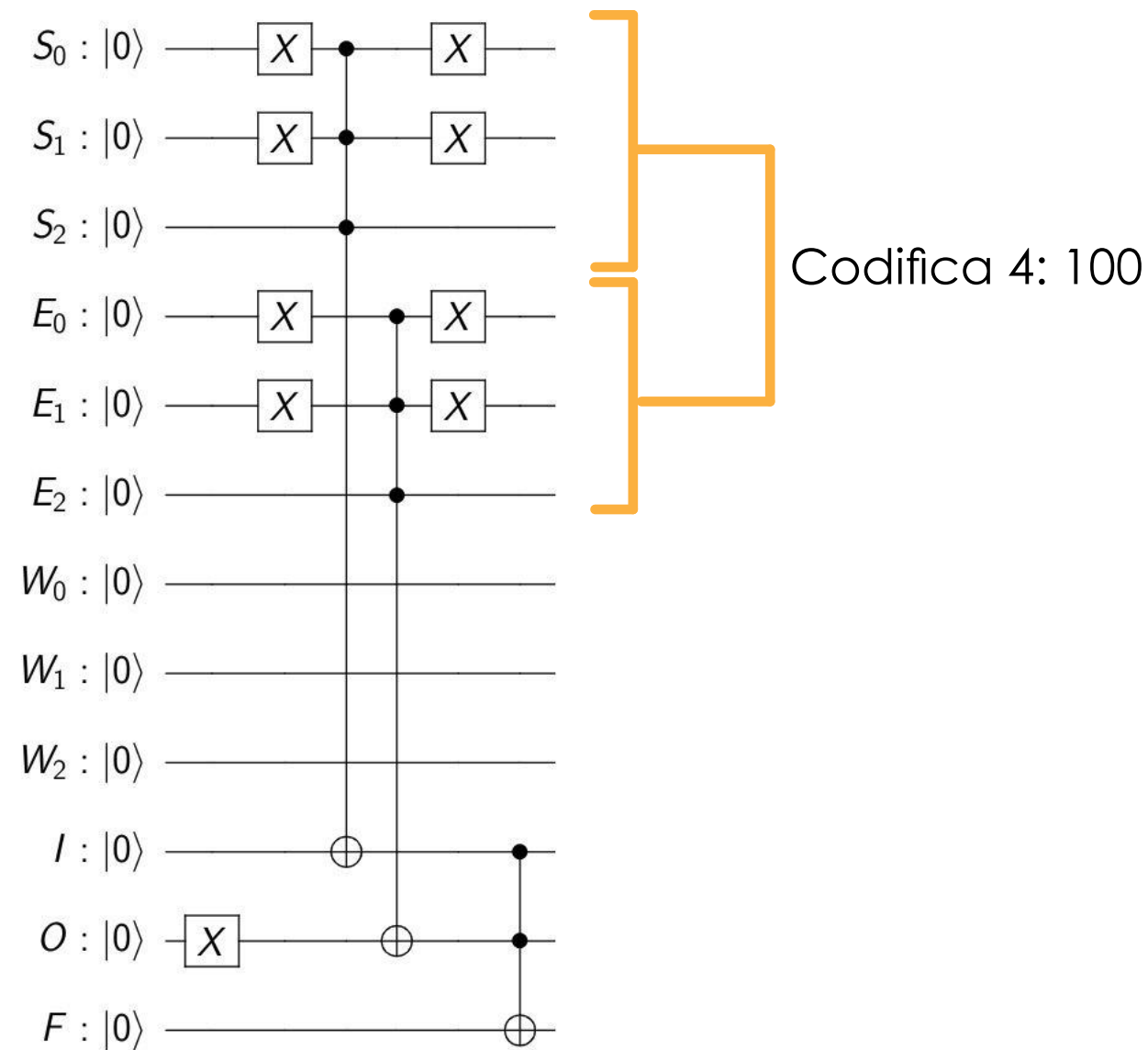
Codifichiamo gli archi del grafo attraverso il nodo di partenza, di arrivo e il peso dell'arco associato. I nodi vengono codificati attraverso i **gate X**, **per porre il registro al valore del nodo (id)**, e il multi-qubit X ha come controllo i qubit dei registri dei nodi e dei pesi e pongono **a 1 un flag se esiste un arco tra i nodi considerati**. Per resettare i registri **si pone un gate X dove era stato precedentemente applicato**.



## 7

# MST quantum - Arco uscente

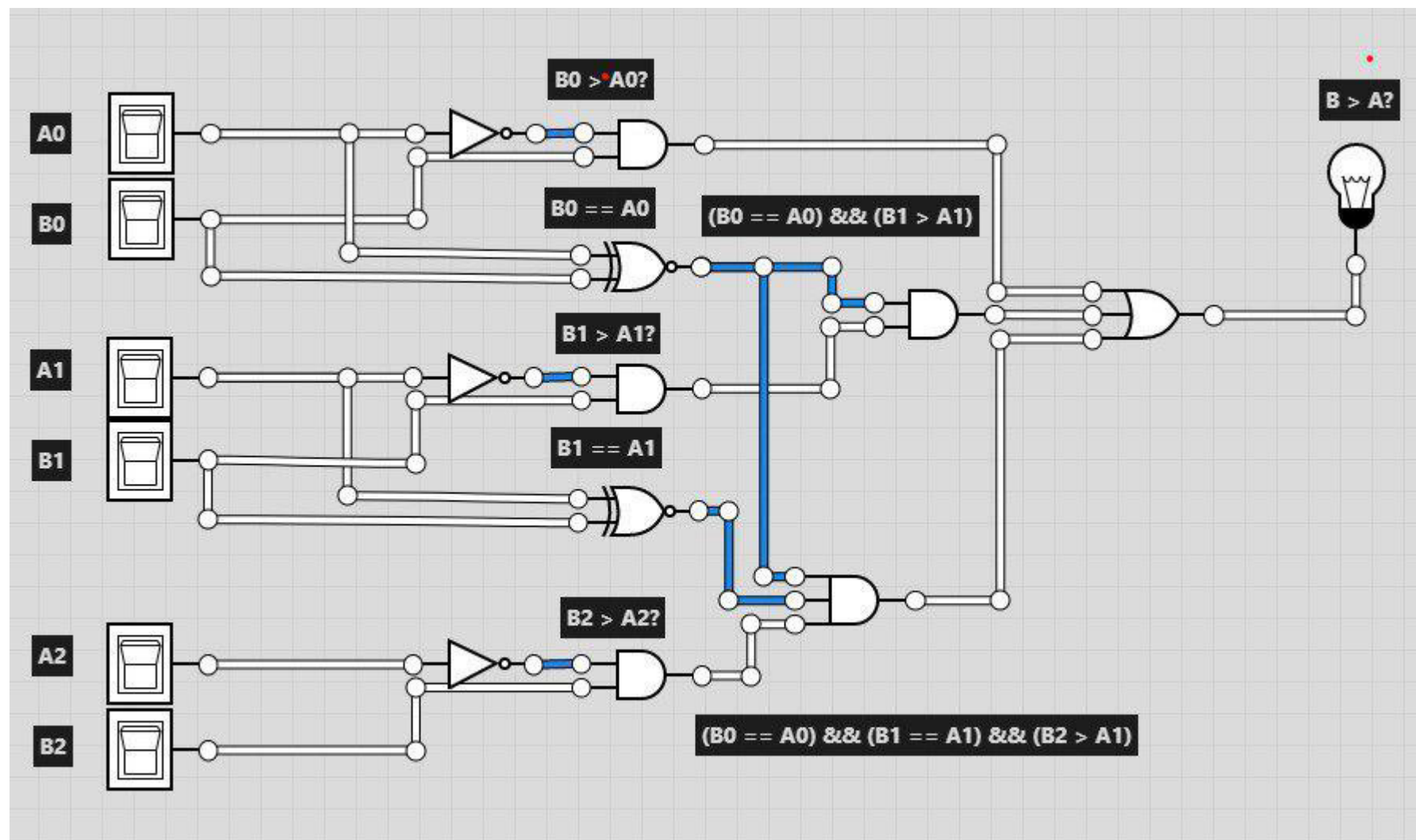
Controlliamo se un determinato **nodo è già stato visitato**. Con il registro start controlliamo che il nodo sia stato visitato e con il registro end controlliamo che il nodo non sia stato visitato. I due risultati **vengono comparati e il risultato messo in output sul qubit di flag**.



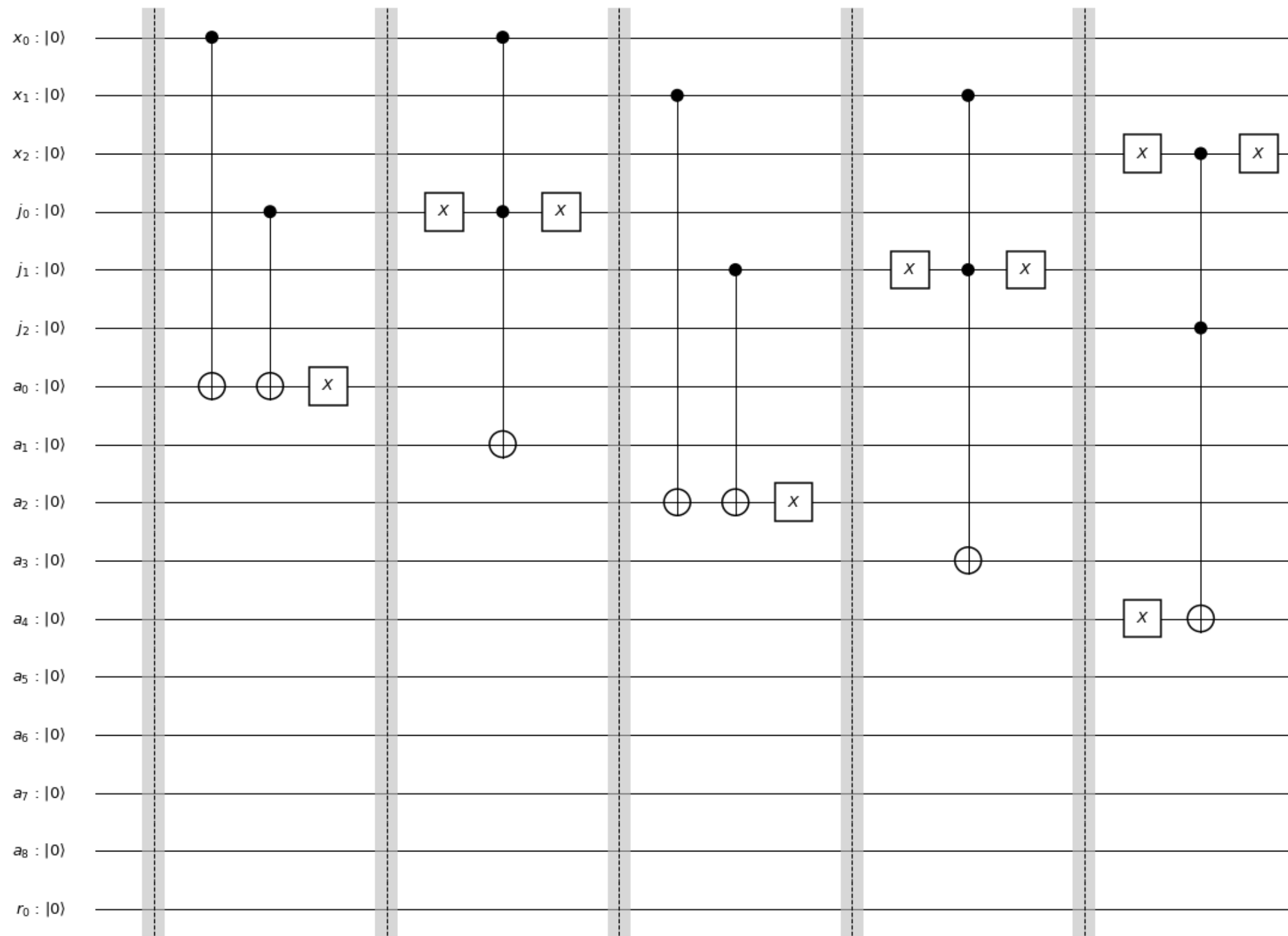
## 8

# MST quantum - Ricerca minimo

Come già detto in precedenza codifichiamo la ricerca del minimo come una **iterazioni di comparazioni**. La **comparazione di due numeri a N bit** può essere descritta come funzione logica nel seguente modo.



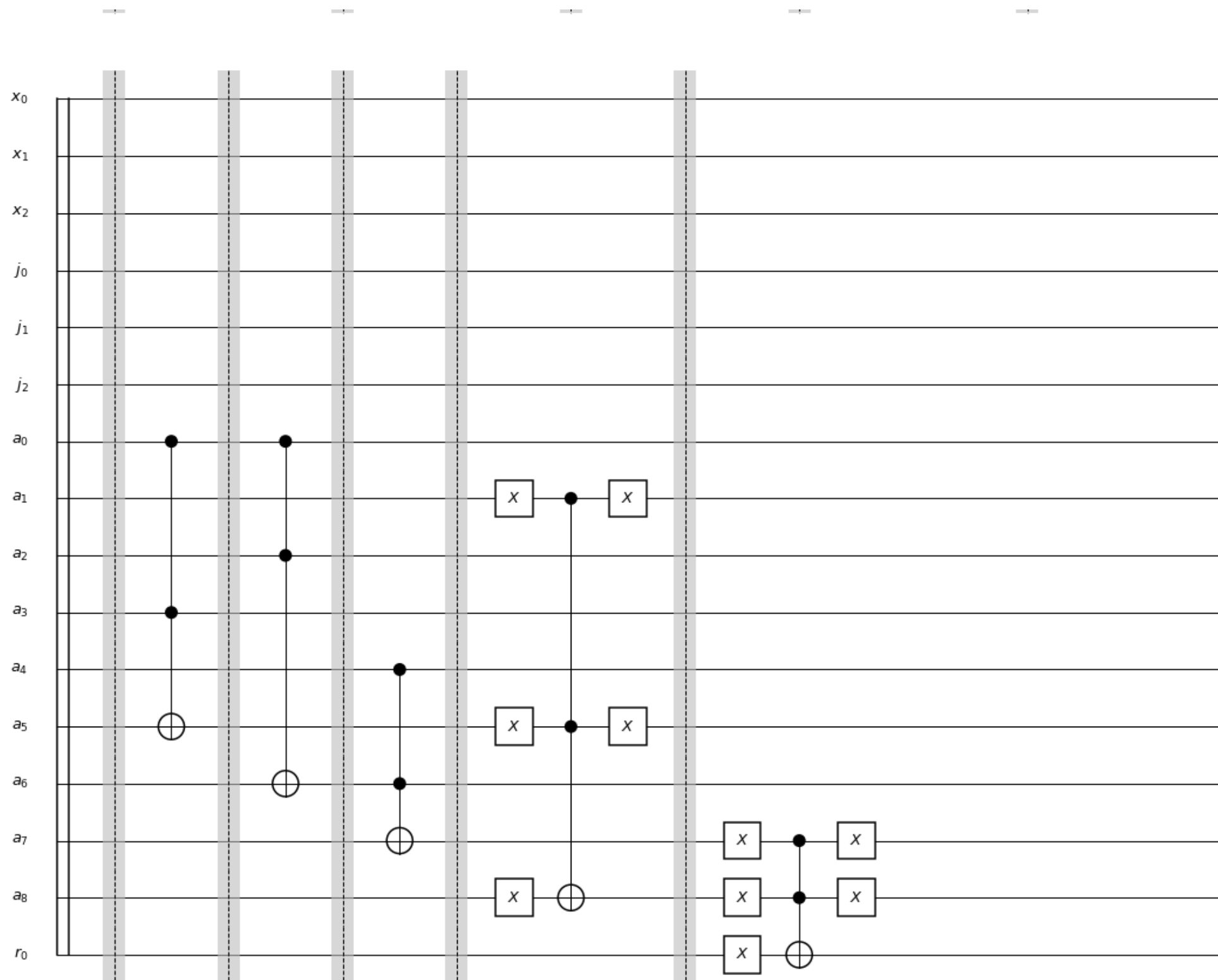
# MST quantum - comparazione





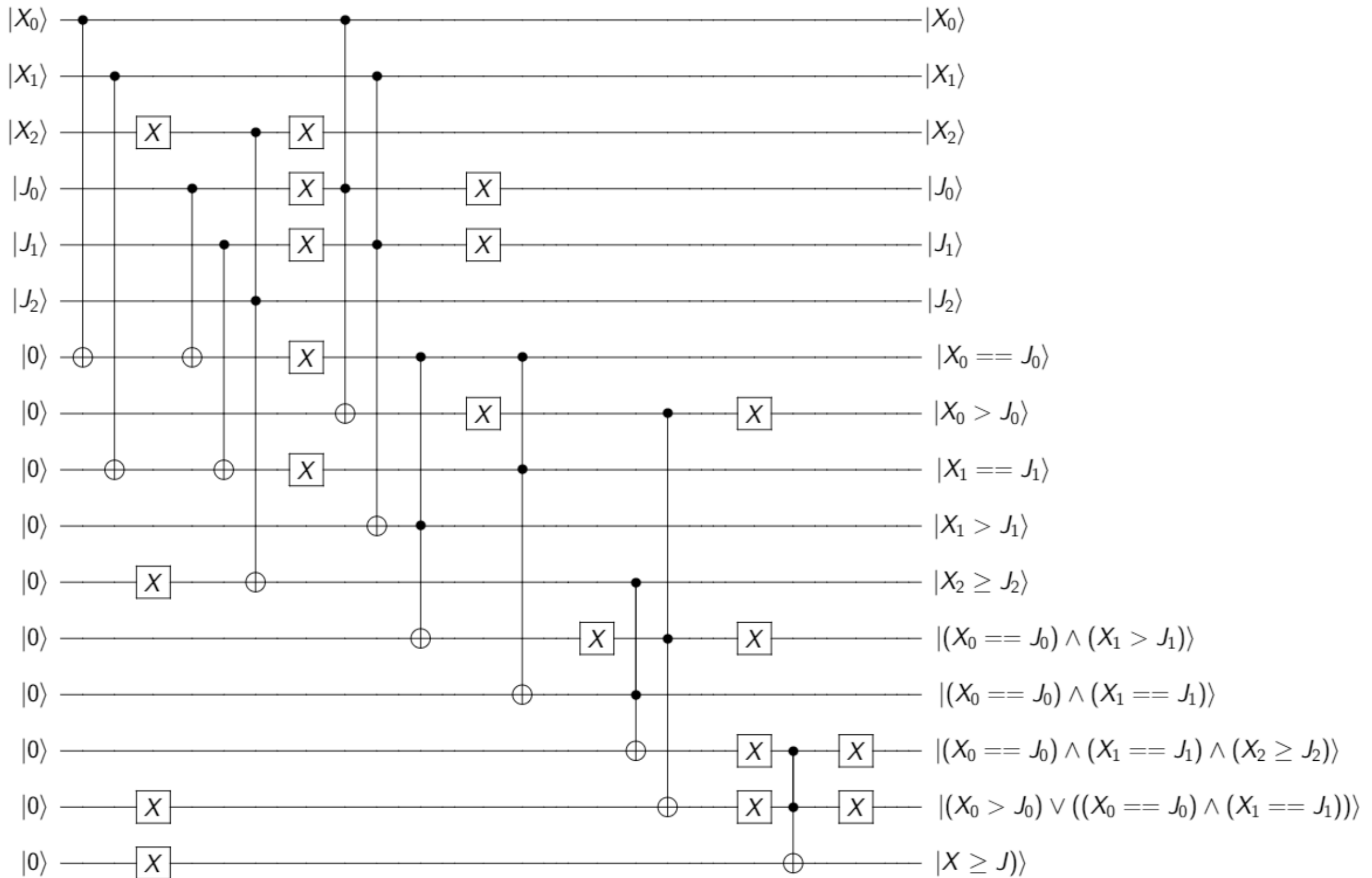
10

# MST quantum - comparazione



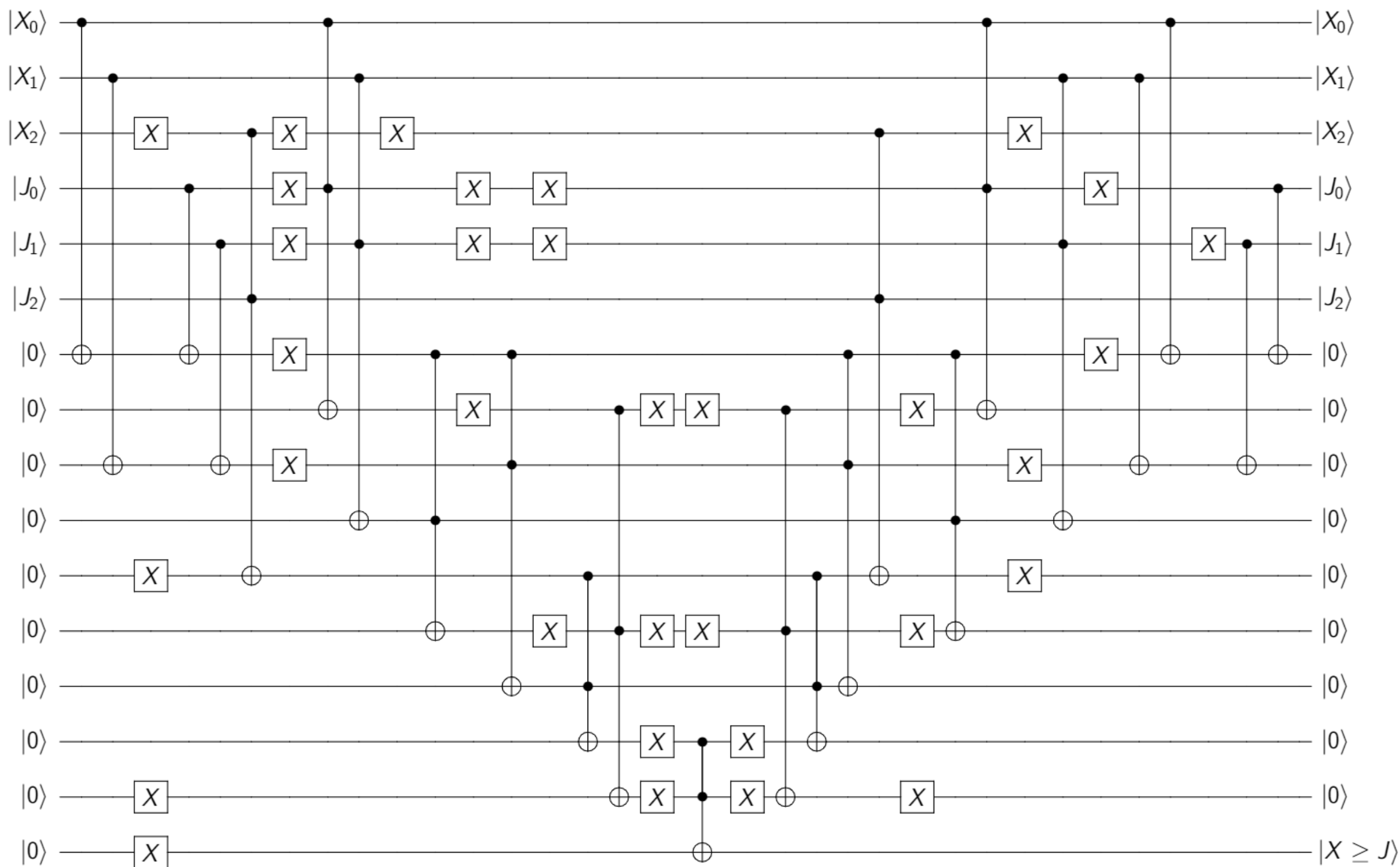
# 11

## MST quantum - Comparazione



## 12

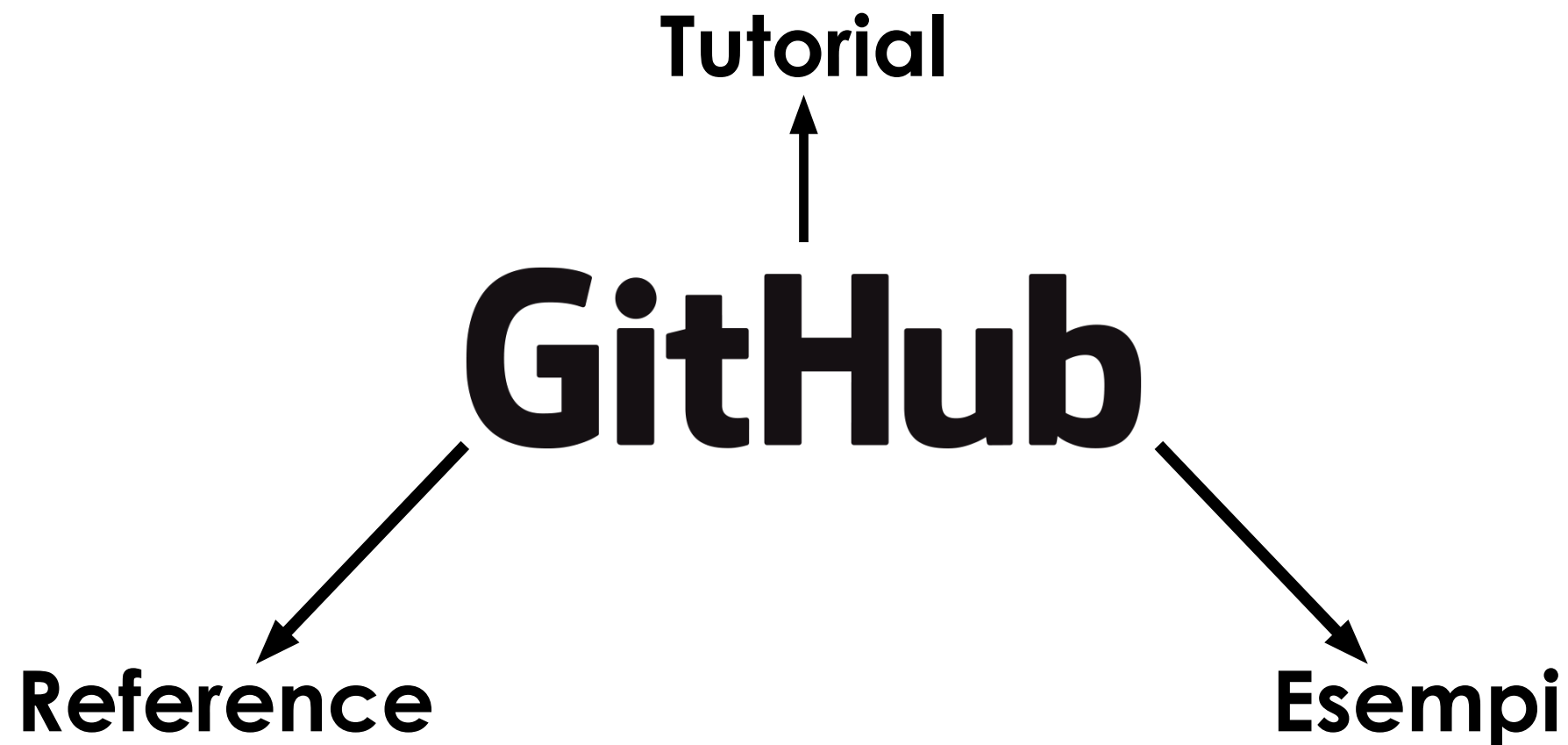
## MST quantum - Comparazione clean



# 1

## Qiskit - Documentazione

La maggior parte della documentazione di questa piattaforma si può trovare su Github. Qui si possono trovare **reference**, **tutorial ed esempi**. Gli esempi sono di grande importanza poiché la **documentazione è molto complicata** e mirata ad una **utenza più esperta del settore**, come scienziati e ricercatori.



# 2

## Qiskit - Caratteristiche

Qiskit è una piattaforma opensource sviluppata da **IBM basata su C++ e Python**

Qiskit è diviso in **4 macro sezioni**



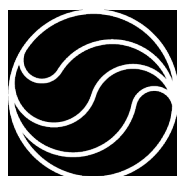
### Terra

Contiene tutti gli strumenti di base per lo sviluppo di applicazioni quantum.



### Aer

Simulatore di macchine quantum ad “alte prestazioni”.



### Aqua

Libreria di algoritmi quantum, utile per implementazioni rapide di algoritmi attualmente ben conosciuti.



### Ignis

Racchiude un insieme di strumenti per la simulazione del rumore e l'analisi della computazione.


# 3

## Qiskit - Terra

All'interno di Terra troviamo un'insieme di strumenti per la creazione di applicazioni quantum **a livello di circuiti e segnali**.

Essendo strettamente legata all'hardware sviluppato da IBM, il **set di istruzioni è limitato** rispetto ad altre piattaforme. Per esempio è **impossibile definire direttamente gate controllati a multi qubit** come per esempio i  $C^N$ NOT.

Questo suo stretto legame con hardware fisico però ha come vantaggio di poter disporre di una **piattaforma reale con cui testare il codice**.

Qiskit terra  IBM Q Experience

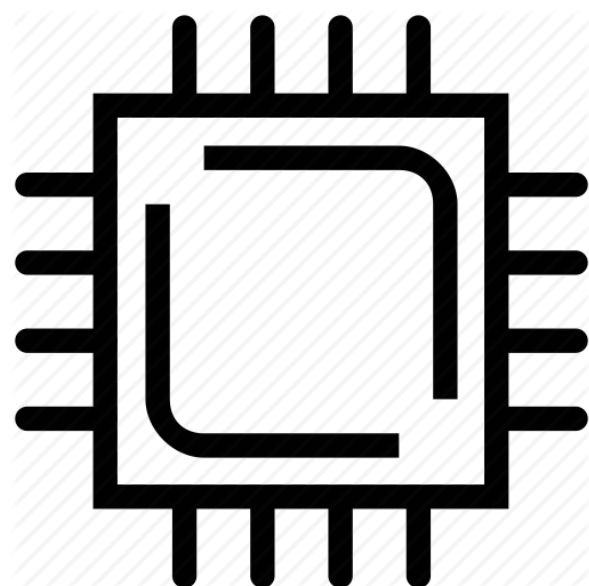
# 4

## Qiskit - Aer

Simulatore di macchine quantum ad “alte prestazioni”. Tra le sue funzionalità troviamo anche la **simulazione del rumore**.

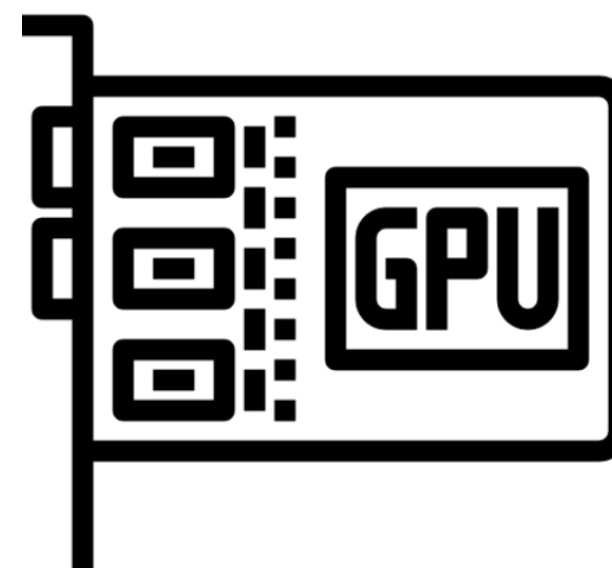
Attualmente **manca il supporto** per eseguire il simulatore su più **processi paralleli**. Questo ovviamente **rallenta notevolmente la simulazione** dei circuiti.

Multi core CPU



Supporto futuro

Schede grafiche



Da poco si è cominciato a sviluppare una **nuova versione per processori multi-core e schede grafiche**.

## 5

# Qiskit - Aqua

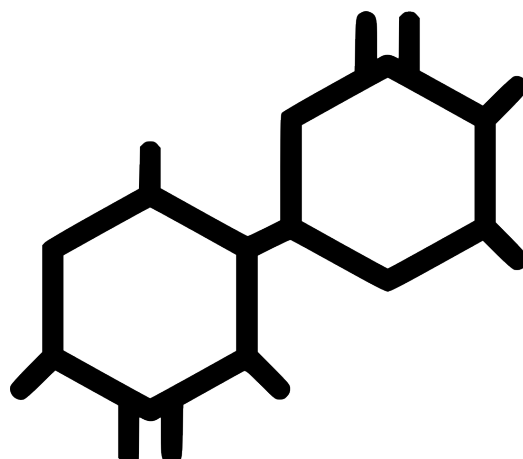
Qiskit Aqua contiene una **libreria di algoritmi quantistici** su più domini, sulla quale si possono **costruire applicazioni per computer quantistici verosimilmente disponibili a breve termine**.

Possibili domini di applicazione possono essere

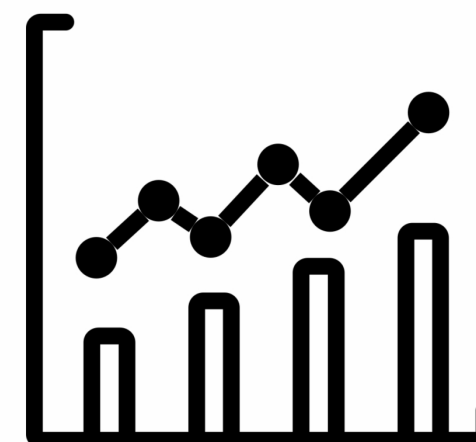
AI



Chimica



Finanza



Con Qiskit Aqua abbiamo anche la possibilità di fare **override di alcune operazioni di Qiskit Terra**. In questo modo possiamo **implementare gate multi qubit** come il  $C^N$ NOT.



Qiskit Ignis è un framework per **comprendere e mitigare il rumore nei circuiti e dispositivi quantum**. Questo framework contiene **esperimenti per generare circuiti di benchmark**, che posso essere seguiti su macchine quantum reali. Ignis ha anche strumenti per l'**analisi dei risultati**.



## Caratterizzazione

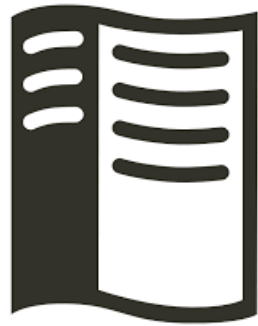
Gli esperimenti di caratterizzazione servono a testare la **vita utile dei qubit e il fenomeno del dephasing**. Test mirati all'analisi del **rumore del sistema**.

## Verifica

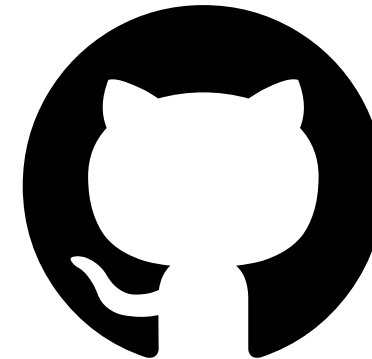
Gli esperimenti di verifica servono a **testare piccoli circuiti attraverso piccoli benchmark**.

## Mitigazione

Gli esperimenti di mitigazione servono a **calibrare i sistemi**, creando **routine** che possono essere applicate alla macchina.



La **documentazione ufficiale** è reperibile sulla piattaforma **Read the Docs**



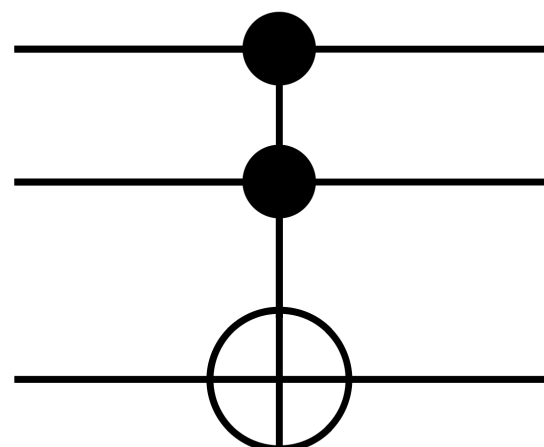
Alcuni esempi di implementazione su **GitHub**

Documentazione estremamente **tecnica e specifica**. **Esempi** disponibili **poco numerosi e poco significativi**: sono presenti infatti solo esempi estremamente banali, quali l'aggiunta di gate al circuito, ed esempi estremamente complessi, quali l'implementazione di algoritmi avanzati.

## 2

# Cirq - Caratteristiche

Cirq è un Framework opensource sviluppato da **Google**  
È disponibile come libreria **Python**.  
Si tratta di una **libreria relativamente giovane**, essendo disponibile in **versione Alpha da meno di un anno**.

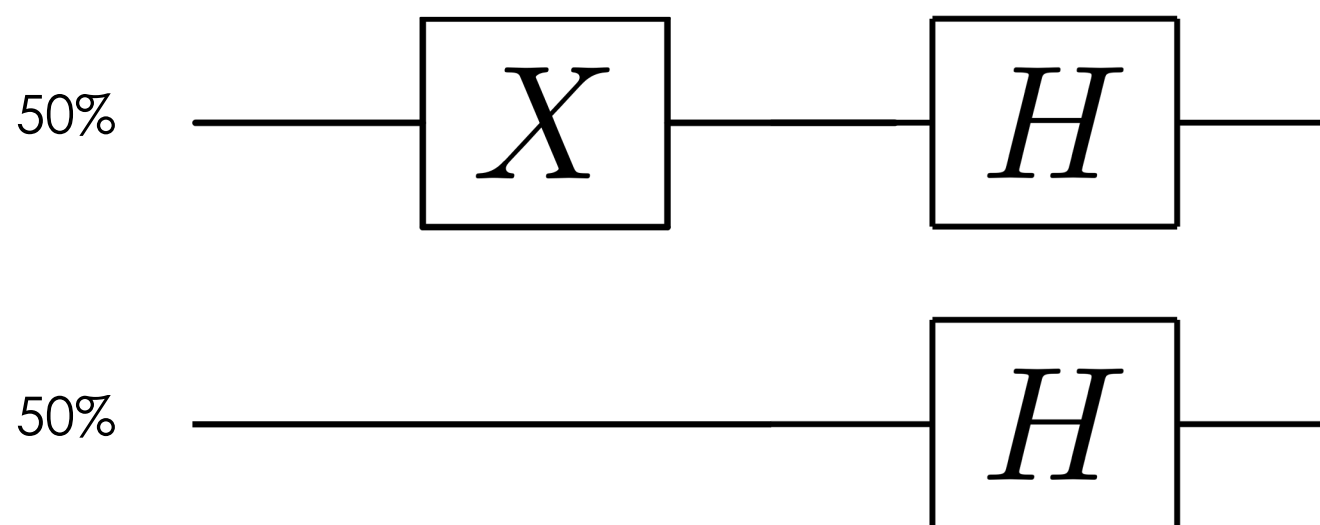


Questo framework offre la possibilità di **creare Gate controllati, scegliendo** arbitrariamente i **bit controllati e i bit controllori**.  
Questo offre un grande vantaggio rispetto ai framework che non permettono di generare automaticamente Gate controllati, poiché permette di **creare tali Gate senza bisogno di implementarli ex-novo** come combinazione di altri Gate.

## 3

# Cirq - Simulatore

Cirq permette di simulare l'esecuzione di un computer quantistico.



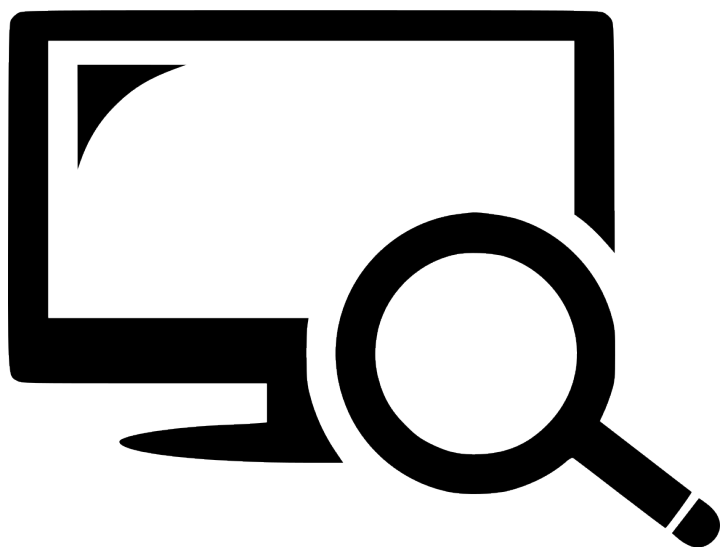
Questo framework offre la possibilità di **aggiungere rumore al circuito**.

Uno dei metodi possibili che vengono utilizzati da Cirq consiste nell'**aggiungere casualmente** al circuito possibili **evoluzioni con probabilità diverse**. Un esempio di possibile evoluzione consiste nell'**aggiungere un Gate X che verrà applicato con una certa probabilità**.

# 4

## Cirq - Simulatore

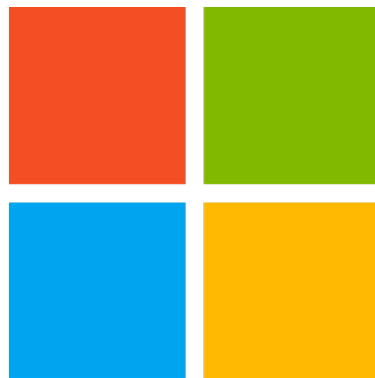
Cirq offre la possibilità di monitorare la **simulazione del circuito passo passo**, mostrando dunque i **risultati intermedi della computazione**.



Attraverso un **iteratore**, è possibile percorrere ogni passo dell'esecuzione a uno a uno. L'iteratore **restituisce lo stato del circuito alla fine del passo** a cui è riferito, **insieme ad eventuali misurazioni** avvenute durante tale passo.

# 1

## QDK - Documentazione



QDK download

Tutorial per la configurazione  
del QDK

Reference su Q# e algoritmi  
quantum

# GitHub

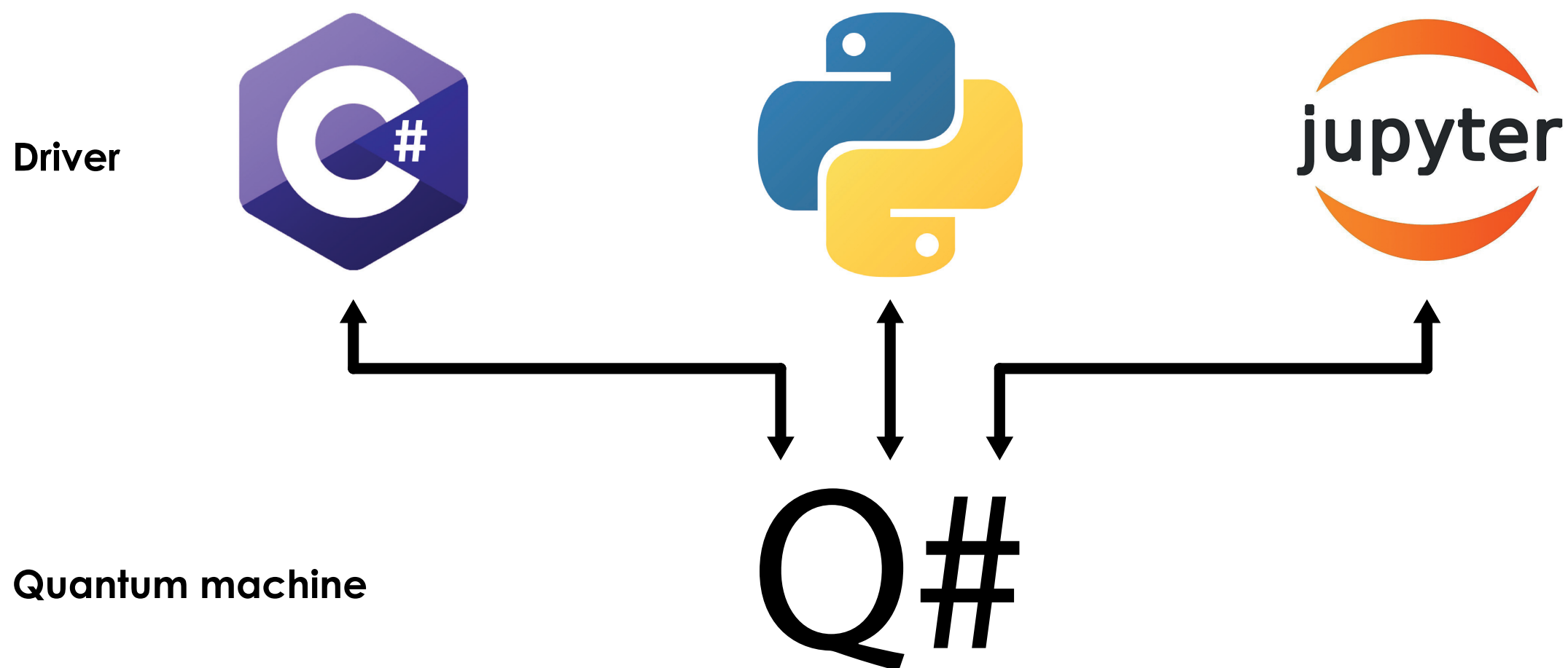
Ricca libreria di esempi con  
algoritmi già implementati e  
esempi di utilizzo di alcune  
librerie

Tutorial di configurazione più  
approfonditi

La presenza di esempi su **GitHub** viene sfruttata al meglio nel momento in cui si utilizza Visual Studio, il quale permette di **importare repository direttamente dal IDE** così permettendo di avere un **accesso rapidissimo a una grande varietà di esempi**.

## 2

# QDK - Caratteristiche (driver e Q#)



**Q#** è un linguaggio di programmazione ideato appositamente **per sviluppare applicazioni quantum**. L'idea di base dietro la QDK è di avere un processo classico, chiamato **driver**, che **chiama le operazioni scritte in Q#**, le quali **restituiranno il risultato** dell'esecuzione

# 3

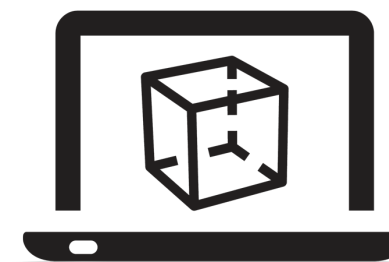
## QDK - Caratteristiche (librerie)

QDK mette a disposizione una grande **varietà di librerie** che implementano la maggior parte dei **gate classici e gli algoritmi quantum più conosciuti**. Questa lista è in continua crescita e permette agli sviluppatori di **accelerare lo sviluppo del software**.



### Microsoft.Quantum.diagnostic

Questa libreria permette di creare **test** per controllare lo stato della macchina, **quando possibile**. La sua struttura si basa sul concetto di **Assert**.



### Microsoft.Quantum.Simulation.Simulator

Da accesso a una grande **varietà di simulatori con differenti proprietà**. Questo permette agli sviluppatori di utilizzare **simulatori specializzati** per alcune applicazioni.



# 4

## QDK - Simulatori

### Full state vector simulator

Questo è il simulatore più utilizzato e simula una **quantum machine completa**. Ha la possibilità di **simulare il rumore** e la capacità di sfruttare il **parallelismo per la simulazione**.

**MAX qubit: 30 locale, 40 cloud**

### Toffoli simulator

Questo simulatore è una **versione semplificata** del full state vector simulator in cui si dispone solamente di **tre gate: X, CNOT e multi-qubit X**. Per la sua semplicità può **simulare molti più qubit**.

**MAX qubit: milioni**

### Resources estimator

**Stima le risorse necessarie** per eseguire l'algoritmo selezionato. Per far ciò **non esegue effettivamente la simulazione**, così facendo può **simulare** algoritmi che **superano il limite massimo di qubit**.

**MAX qubit: non allocati, indefinito**

### Trace-based resources estimator

Simile allo stimatore semplice ma utilizza anche **assert probabilistici** per dare informazioni più specifiche sulle risorse necessarie

**MAX qubit: non allocati, indefinito**

# 5

## QDK - Hardware

Al contrario di Rigetti e IBM che hanno costruito **quantum computer basati sulla tecnologia dei superconducting qubits**, Microsoft sta scommettendo su un nuovo tipo di qubits chiamati **topological qubits**, basati su fermioni di Majorana. Questi nuovi qubits dovrebbero garantire tempi di decoerenza migliori.

### Superconducting qubits

vs

### Topological qubits

#### PRO

Più facili da produrre

Dovrebbero garantire una decoherence migliore

#### CONTRO

Decoherence più difficile da gestire

Non esistono ancora qubit di questo tipo

Da poco è possibile utilizzare QuantumExperience di IBM per eseguire codice scritto in Q#.