RELAZIONE ALGORITMO BERSTEIN-VAZIRANI

Lavoro svolto da:
-Magno Alessandro : 4478234

Funzionamento

Ho una scatola con all'interno un numero formato da 6 bits. Con un computer classico, mi occorrono 6 tentativi per indovinare il numero correttamente. Se il numero fosse di 60 bits, mi servono 60 tentativi. Quindi il numero di bit mi implica il numero di tentativi. Con un computer quantistico utilizzando l'algoritmo di Berstein-Vazirani, posso determinare il numero segreto con un tentativo indipendentemente dalla dimensione del numero.

Es. numero segreto 101001

Il numero è all'interno di una black box, il cui scopo è quello di dare una funzione a cui do un numero e mi risponde con un si, se il numero è lo stesso all'interno della scatola altrimenti no. Quindi interrogo la black box (l'oracolo) e chiedo il numero è 000000? →NO

000001? →NO

….

Il computer classico esegue questo procedimento applicando una porta AND,

101001 AND 000001 → 000001 no

AND 000010 → 000000 no

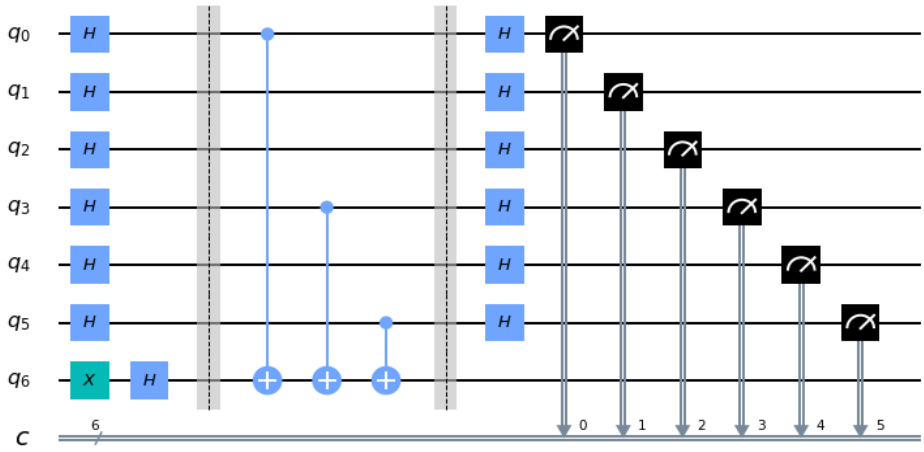
… ne occorrono 6

Classicamente sono necessarie n chiamate dell'oracolo, con un computer quantistico, si può risolvere questo problema con una sola chiamata all'oracolo. Si ha quindi uno speed-up polinomiale (lineare).

Implementazione algoritmo Bernstain-Vazirani

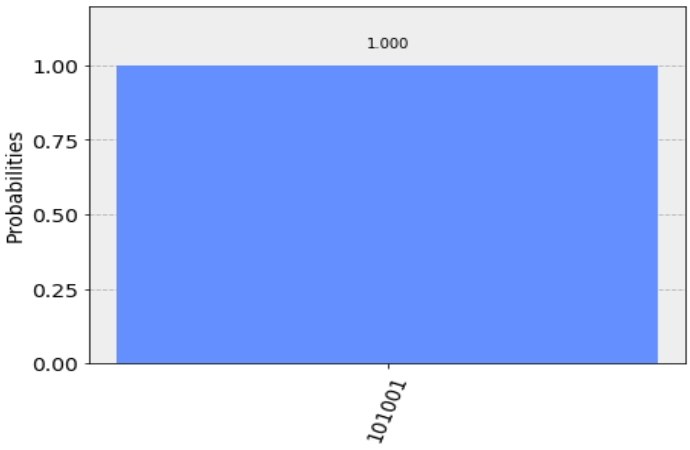
```
1.  from qiskit import *
2.  %matplotlib inline
3.  from qiskit.tools.visualization import plot_histogram
4.
5.  secretnumber = '101001'
6.
7.  #costruisco il circuito con 6 qubits (numero dei bits che ho nel numero segreto)+ 1 qubit

8.  #6 bits classici in cui salvare i risultati
9.  qc = QuantumCircuit(6+1, 6)
10.
11. #applico hadamard a i primi 6 qubits
12. qc.h([0,1,2,3,4,5])
13.
14. #applico prima trasformazione x e poi hadamard all'ultimo qubit
15. qc.x(6)
16. qc.h(6)
17.
18. qc.barrier()
```

```
19.
20. #costruisco la scatola che contiene il numero
21. #per ogni 1 che vedo nel numero segreto partendo dal fondo, applico CNOT
22. for ii, yesno in enumerate(reversed(secretnumber)):
23.     if yesno == '1':
24.         qc.cx(ii, len(secretnumber))
25.
26. qc.barrier()
27.
28. #applico nuovamente a i primi 6 qubits hadamard
29. qc.h([0,1,2,3,4,5])
30.
31. #eseguo la misura sui primi 6 qubits e salvo i risultati nei bits classici
32. qc.measure([0,1,2,3,4,5],[0,1,2,3,4,5])
33.
34. #stampa
35. qc.draw(output='mpl')
```



Si capisce vedendo il circuito che il numero segreto è codificato all'interno della scatola, leggendo (nella parte centrale divisa dalle due barriere) 1 quando si vede un CNOT e 0 zero quando non si vede. Nel disegno sopra, partendo da q5 si legge 1, q4 0, q3 1, q2 0, q1 0, q0 1.

```
1. #simulazione circuito, con un tentativo
2. simulator = Aer.get_backend('qasm_simulator')
3. result = execute(qc, backend = simulator, shots = 1).result()
4. counts = result.get_counts()
5. plot_histogram(counts)
```



Si può vedere che con un tentativo, si ottiene con probabilità certa (100%), il numero segreto.