**Grover Algorithm**

Grover algorithm was developed by Grover[[1]](#footnote-1) at 1996, is designed to find an element within a list of unorganized elements. This algorithm is well-known in the quantum computing realm for its efficiency, surpassing classical search algorithms. Grover's Algorithm comprises three key steps: initialization, oracle application, and amplitude amplification. Let's explore these steps in more detail.

1. **First Step: The initialization:**

In this initial phase, we prepare our system to implement Grover's Algorithm. This involves bringing all the qubits into a superposition state by applying a Hadamard gate to each qubit. Consequently, the state of our system becomes:

where N is the total number of states.

1. **Second Step: Oracle application**

Next, we apply an oracle to the element we wish to search. This is done by using a reflection operator on the target element. For instance, if ∣w⟩ is the element we seek, the oracle function reflects the state in such a way to mark it as the target:

Thus, if the searched element is ∣w⟩, its phase will be inverted.

1. **Third Step: Amplitude amplification**

Finally, we transform our quantum space by reflecting about the average amplitude ∣s⟩. This is achieved by applying the operator . As a result, the amplitudes of all qubits are adjusted, except for the state ∣w⟩, which is amplified.

Lets explore an example for a two qubit system in our Qinterpreter library:

# First of all, we import our libraries:

Firstly, we import the necessary libraries from quantumgateway:

from quantumgateway.quantum\_circuit import QuantumCircuit, QuantumGate

from quantumgateway.quantum\_translator.braket\_translator import BraketTranslator

from quantumgateway.quantum\_translator.cirq\_translator import CirqTranslator

from quantumgateway.quantum\_translator.qiskit\_translator import QiskitTranslator

from quantumgateway.quantum\_translator.pennylane\_translator import PennyLaneTranslator

from quantumgateway.quantum\_translator.pyquil\_translator import PyQuilTranslator

from quantumgateway.main import translate\_to\_framework, simulate\_circuit

# Second: The initialization

We start from the initial state , and apply Hadamard gates to each qubit to create a superposition state:

Where H is the Hadamard gate represented as:

In code, this initialization is implemented as:

circuit = QuantumCircuit(2,2)

def initialize\_grover(qc, qubits):

    """Apply a H-gate to 'qubits' in qc"""

    for q in qubits:

        qc.add\_gate(QuantumGate("h", [q]))

    return qc

circuit = initialize\_grover(circuit, [0,1])

# Oracle application.

We apply an oracle for the state ∣11⟩ using the controlled-Z gate as our oracle matrix :

When we apply it to our system, then we get:

Then in our code this is done by:

circuit.add\_gate(QuantumGate("cz", [0,1]))

# Amplitude amplification:

Finally we apply the diffusion operator , to increase the probability of our target state. The operation sequence is as follows:

* After Oracle Application:
* Applying the Diffusion Operator:

As we see the state has the gighest probability.

In code, the diffusion operator is implemented as:

# Diffusion operator (U\_s)

circuit.add\_gate(QuantumGate("h", [0]))

circuit.add\_gate(QuantumGate("h", [1]))

circuit.add\_gate(QuantumGate("z", [0]))

circuit.add\_gate(QuantumGate("z", [1]))

circuit.add\_gate(QuantumGate("cz", [0,1]))

circuit.add\_gate(QuantumGate("h", [0]))

circuit.add\_gate(QuantumGate("h", [1]))

# Measurement and simulation:

Finally, we add measurement gates and choose the desired framework for executing our Grover algorithm. In Qinterpreter, this is achieved with:

circuit.add\_gate(QuantumGate("MEASURE", [0,0]))

circuit.add\_gate(QuantumGate("MEASURE", [1,1]))

selected\_framework = 'pennylane'  # Change this to the desired framework

translated\_circuit = translate\_to\_framework(circuit, selected\_framework)

translated\_circuit.print\_circuit()

#Simulate the circuit and print the result

print("The results of our simulated circuit are: ")

print(simulate\_circuit(circuit, selected\_framework))

Algoritmo para un circuito de 4 Qubits

A continuación se presenta el algoritmo de grover en un circuito de 4 Qubits, en el cual deseamos encontrar el estado.

from quantumgateway.quantum\_circuit import QuantumCircuit, QuantumGate

from quantumgateway.quantum\_translator.braket\_translator import BraketTranslator

from quantumgateway.quantum\_translator.cirq\_translator import CirqTranslator

from quantumgateway.quantum\_translator.qiskit\_translator import QiskitTranslator

from quantumgateway.quantum\_translator.pennylane\_translator import PennyLaneTranslator

from quantumgateway.quantum\_translator.pyquil\_translator import PyQuilTranslator

from quantumgateway.main import translate\_to\_framework, simulate\_circuit

import math

# Quantum circuit initialization

circuit = QuantumCircuit(4, 4)

# Grover Initialization

for i in range(4):

    circuit.add\_gate(QuantumGate("h", [i]))

# Function to define the oracle with the object state |1111>

def apply\_oracle(circuit):

    pi = math.pi

    # Secuencia de puertas para el Oracle

    circuit.add\_gate(QuantumGate("cphase", [0, 3], [pi/4]))

    circuit.add\_gate(QuantumGate("cnot", [0, 1]))

    circuit.add\_gate(QuantumGate("cphase", [1, 3], [-pi/4]))

    circuit.add\_gate(QuantumGate("cnot", [0, 1]))

    circuit.add\_gate(QuantumGate("cphase", [1, 3], [pi/4]))

    circuit.add\_gate(QuantumGate("cnot", [1, 2]))

    circuit.add\_gate(QuantumGate("cphase", [2, 3], [-pi/4]))

    circuit.add\_gate(QuantumGate("cnot", [0, 2]))

    circuit.add\_gate(QuantumGate("cphase", [2, 3], [pi/4]))

    circuit.add\_gate(QuantumGate("cnot", [1, 2]))

    circuit.add\_gate(QuantumGate("cphase", [2, 3], [-pi/4]))

    circuit.add\_gate(QuantumGate("cnot", [0, 2]))

    circuit.add\_gate(QuantumGate("cphase", [2, 3], [pi/4]))

# Function for the Amplitude Amplification

def apply\_amplification(circuit):

    for i in range(4):

        circuit.add\_gate(QuantumGate("h", [i]))

        circuit.add\_gate(QuantumGate("x", [i]))

    pi = math.pi

    circuit.add\_gate(QuantumGate("cphase", [0, 3], [pi/4]))

    circuit.add\_gate(QuantumGate("cnot", [0, 1]))

    circuit.add\_gate(QuantumGate("cphase", [1, 3], [-pi/4]))

    circuit.add\_gate(QuantumGate("cnot", [0, 1]))

    circuit.add\_gate(QuantumGate("cphase", [1, 3], [pi/4]))

    circuit.add\_gate(QuantumGate("cnot", [1, 2]))

    circuit.add\_gate(QuantumGate("cphase", [2, 3], [-pi/4]))

    circuit.add\_gate(QuantumGate("cnot", [0, 2]))

    circuit.add\_gate(QuantumGate("cphase", [2, 3], [pi/4]))

    circuit.add\_gate(QuantumGate("cnot", [1, 2]))

    circuit.add\_gate(QuantumGate("cphase", [2, 3], [-pi/4]))

    circuit.add\_gate(QuantumGate("cnot", [0, 2]))

    circuit.add\_gate(QuantumGate("cphase", [2, 3], [pi/4]))

    for i in range(4):

        circuit.add\_gate(QuantumGate("x", [i]))

        circuit.add\_gate(QuantumGate("h", [i]))

# Now we apply the oracle and the amplitude amplification

apply\_oracle(circuit)

apply\_amplification(circuit)

# Measuring the Qubits

for i in range(4):

    circuit.add\_gate(QuantumGate("measure", [i, i]))

# Finally we simulate the Grover algorithm

#

selected\_framework = 'qiskit'  # Change this to the desired framework

translated\_circuit = translate\_to\_framework(circuit, selected\_framework)

translated\_circuit.print\_circuit()

#Simulate the circuit and print the result

print("The results of our simulated circuit are: ")

print(simulate\_circuit(circuit, selected\_framework))

1. Lov K. Grover, “A Fast Quantum Mechanical Algorithm for Database Search,” *Proceedings of the Annual ACM Symposium on Theory of Computing* Part F1294 (May 1996): 212–219, doi:10.1145/237814.237866. [↑](#footnote-ref-1)