Q8

QX08-GroversAlgorithm.ipynb
File

Necesito que me expliques detalladamente cada una de las celdas del siguiente proyecto.

A continuación tienes una explicación paso a paso de cada celda del notebook **QX08-Grover's Algorithm**, con el detalle del código y la teoría subyacente.

Celda 1: Importaciones y configuración de gráficos

```python
from qiskit import *
from qiskit.tools.visualization import plot_histogram
%matplotlib inline
```

- **`from qiskit import *`**: importa el SDK de Qiskit, que incluye clases como **`QuantumCircuit`**, **`Aer`**, **`execute`**, etc.
- **`from qiskit.tools.visualization import plot_histogram`**: trae la función para graficar histogramas de resultados de medida.
- **`%matplotlib inline`**: magia de Jupyter que permite que las figuras de Matplotlib se muestren directamente debajo de la celda.

  Teoría
  Qiskit es un framework de IBM para diseñar, simular y ejecutar circuitos cuánticos. El histograma de resultados mostrará las probabilidades de medir cada estado clásico tras la ejecución.

Celda 2: Sección de búsqueda clásica

```python
#Classical Search
```

- Simplemente un comentario que delimita la parte de **búsqueda lineal clásica**.

Celda 3: Definición de la lista a buscar

```python
myList = [5, 4, 6, 9, 1, 2, 3, 7, 8, 0]
```

- Se crea una lista de 10 números en la que vamos a buscar un "número ganador" (**`winningNumber`**).

Celda 4: Implementación de la función *oracle* clásica

```python
def oracle(number):
    winningNumber = 8
    if number == winningNumber:
        response = True
    else:
        response = False
    return response
```

- **Entrada**: un **`number`** entero.
- **Salida**: devuelve **`True`** si **`number`** coincide con **`winningNumber = 8`**, **`False`** en caso contrario.
- Esta función simula el "oráculo" clásico: comprueba igualdad y marca el elemento buscado.

Celda 5: Búsqueda lineal usando el oráculo clásico

```python
for index, number in enumerate(myList):
    if oracle(number) is True:
        print(f"winning number index: {index}")
        print(f"execution count: {index+1}")
        break
```

- **`enumerate(myList)`** recorre la lista con índice y valor.
- Llama a **`oracle(number)`** para cada elemento; al encontrar **`True`**:
  - Imprime el índice donde se encontró (**`index`**).
  - Imprime cuántas comparaciones se han hecho (**`index+1`**).
  - Sale del bucle con **`break`**.

  Complejidad: en el peor caso examina todos los elementos — $O(N)$ comparaciones.

Celda 6: Construcción del oráculo cuántico

```python
#Quantum Model with Grover's Algorithm
#Oracle Circuit (WinningNumber = 11)

oracleCircuit = QuantumCircuit(2, name='oracleCircuit')
oracleCircuit.cz(0, 1)
```

```
oracleCircuit.to_gate()
oracleCircuit.draw(output='mpl')
```

- Creamos un circuito de 2 qubits (`QuantumCircuit(2)`) llamado `oracleCircuit`.
- `oracleCircuit.cz(0, 1)` aplica una puerta CZ (controlled-Z) entre los qubits 0 y 1.
  - En la base computacional, la CZ invierte la fase **–1** sólo en el estado $|11\rangle$.
- `.to_gate()` convierte el circuito en una *puerta reutilizable*.
- `.draw(output='mpl')` muestra el diagrama del oráculo.

Teoría
En Grover, el oráculo marca el estado buscado invirtiendo su fase. Aquí, "estado ganador" es $|11\rangle$ (binario 3). Aunque la búsqueda clásica usaba **8**, en este ejemplo cuántico se simula para 2 qubits.

Celda 7: Preparación del circuito principal y aplicación del oráculo

```
python

mainCircuit = QuantumCircuit(2, 2)
mainCircuit.h([0, 1])
mainCircuit.append(oracleCircuit, [0, 1])
mainCircuit.draw(output='mpl')
```

1. `QuantumCircuit(2, 2)` crea un circuito con 2 qubits y 2 bits clásicos (para medida).
2. `mainCircuit.h([0, 1])` aplica puertas Hadamard a ambos qubits, generando la **superposición uniforme**:

$$\frac{1}{2}\sum_{x=0}^{3}|x\rangle$$

3. `mainCircuit.append(oracleCircuit, [0,1])` inserta el oráculo en los mismos qubits.
4. `.draw()` muestra el circuito hasta este punto.

Teoría
El Hadamard inicial distribuye amplitudes por igual; luego el oráculo invierte la fase sólo del estado candidato.

Celda 8: Construcción del operador de difusión (reflexión)

```
python

reflectionCircuit = QuantumCircuit(2, name="reflectionCircuit")
reflectionCircuit.h([0, 1])
reflectionCircuit.z([0, 1])
reflectionCircuit.cz(0, 1)
reflectionCircuit.h([0, 1])
reflectionCircuit.to_gate()
reflectionCircuit.draw(output='mpl')
```

Este bloque implementa la **inversión alrededor de la media** (diffusion operator):

1. **H** en todos los qubits: transforma de la base computacional a la base $|+\rangle, |-\rangle$.
2. **Z** en cada qubit: invierte la fase de $|1\rangle \to -|1\rangle$.
3. **CZ** entre qubits: invierte la fase de $|11\rangle$.
4. **H** de nuevo: retorna a la base computacional.

Matemáticamente, este conjunto de puertas realiza

$$D = 2\,|\psi\rangle\langle\psi| - I$$

donde $|\psi\rangle$ es la superposición uniforme.

Celda 9: Montaje final del circuito y medida

```
python


mainCircuit.append(reflectionCircuit, [0, 1])
mainCircuit.measure([0, 1], [0, 1])
mainCircuit.draw(output='mpl')
```

1. Añade el **diffuser** al circuito principal.
2. Mide cada qubit en su bit clásico correspondiente.
3. Dibuja el circuito completo con oracle + difusión + medida.

> **Nota**: En este ejemplo se realiza **una sola iteración** de Grover. Para $N = 4$ elementos, la fórmula sugiere $\lfloor \frac{\pi}{4}\sqrt{N} \rfloor = 1$ iteración.

Celda 10: Simulación y visualización de resultados

```
python


backend = Aer.get_backend('qasm_simulator')
result = execute(mainCircuit, backend=backend, shots=1).result()
counts = result.get_counts(mainCircuit)
plot_histogram([counts])
```

- **`Aer.get_backend('qasm_simulator')`** selecciona el simulador de circuitos de Qiskit.
- **`execute(..., shots=1)`** ejecuta una única corrida del circuito.
- **`get_counts`** devuelve un diccionario con el número de veces que se midió cada resultado (aquí será algo como **`{'11': 1}`**).
- **`plot_histogram`** dibuja la probabilidad (en este caso, 100 % en $|11\rangle$).

Interpretación
El histograma confirma que el estado marcado $|11\rangle$ es el resultado medido, mostrando la **eficacia de Grover** al concentrar amplitud en el elemento buscado.

Celda 11: (vacía)

- Celda en blanco, sin código.

Conclusiones generales

- **Búsqueda clásica**: $O(N)$ comparaciones.
- **Grover**: $O(\sqrt{N})$ iteraciones cuánticas → velocidad cuadrática.
- El notebook ilustra paso a paso cómo:
  1. Preparar superposición uniforme con Hadamard.
  2. Aplicar oráculo de fase para marcar el estado objetivo.
  3. Realizar difusión (inversión alrededor de la media).
  4. Medir y obtener el estado buscado con alta probabilidad.

Si tienes dudas sobre algún paso teórico (por ejemplo, la derivación del operador de difusión o el porqué del número de iteraciones), ¡avísame y profundizamos en ello!