Q7

QX07-DeutschAlgorithm.ipynb
File

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

A continuación tienes una explicación detallada, celda por celda, de lo que hace este notebook de Qiskit:

Celda 1

```python
from qiskit import *
```

Explicación:

- Importa todo el módulo principal de Qiskit, la librería de IBM para programación de ordenadores cuánticos.
- Incluye Terra (para crear circuitos), Aer (simuladores) y herramientas de ejecución en hardware real.

Celda 2

```python
circuit = QuantumCircuit(2, 1)
```

Explicación:

- Crea un objeto `QuantumCircuit` con 2 qubits y 1 bit clásico.
- Los **qubits** son las unidades de información cuántica; los **bits clásicos** almacenan el resultado tras la medición.

Celda 3

```python
circuit.h(0)
```

Explicación:

- Aplica la puerta de Hadamard al **qubit 0**.

- La puerta H convierte el estado base $|0\rangle$ en superposición $\frac{|0\rangle+|1\rangle}{\sqrt{2}}$.
- Fundamental para generar interferencia cuántica y algorítmica (como en Deutsch, Grover, etc.).

Celda 4

```
python

circuit.cx(0, 1)
```

Explicación:

- Aplica la puerta control-NO (CNOT) con **control** en qubit 0 y **target** en qubit 1.
- Si el qubit 0 está en $|1\rangle$, invierte el qubit 1; si está en $|0\rangle$, lo deja igual.
- Aquí crea un estado entrelazado (Bell) entre qubit 0 y qubit 1.

Celda 5

```
python

circuit.measure(0, 0)
```

Explicación:

- Mide el qubit 0 y almacena el resultado (0 o 1) en el bit clásico 0.
- La medición "colapsa" la superposición al valor clásico.

Celda 6

```
python

circuit.draw(output='mpl')
```

Explicación:

- Dibuja el circuito usando Matplotlib (`'mpl'`).
- Permite visualizar la secuencia: H → CX → Measure.

Celda 7

```
python

backend = Aer.get_backend('qasm_simulator')
```

Explicación:

- Selecciona el simulador de QASM de Aer (simula circuitos cuánticos midiendo en el dominio de densidad de probabilidad).

- Ideal para probar localmente antes de usar hardware real.

Celda 8

```
python

#Real Quantum Computer
```

Explicación:

- Comentario indicando que a continuación se configura el acceso a un ordenador cuántico real de IBM.

Celda 9

```
python

IBMQ.load_account()
```

Explicación:

- Carga las credenciales de tu cuenta IBM Quantum guardadas en disco.
- Necesario para acceder a los dispositivos cuánticos en la nube.

Celda 10

```
python

provider = IBMQ.get_provider("ibm-q")
```

Explicación:

- Obtiene el "provider" (servicio) de IBM Q Experience, que agrupa los backends disponibles.

Celda 11

```
python

for backend in provider.backends():
    print(backend)
```

(asumiendo que quieres listar todos)
Explicación:

- Recorre e imprime cada uno de los backends disponibles (simuladores y máquinas reales).
- Útil para elegir el dispositivo más adecuado (por ejemplo, con menor error o cola más corta).

Celda 12

```python
quantum_computer = provider.get_backend('ibmq_belem')
```

Explicación:

- Selecciona el backend real llamado `'ibmq_belem'`.
- Belem es uno de los dispositivos cuánticos de IBM con arquitectura de 5 qubits.

Celda 13

```python
quantum_result = execute(circuit, backend=quantum_computer, shots=1024).result()
```

Explicación:

- Envía (`execute`) el circuito al backend real, pidiendo **1024 ejecuciones** ("shots").
- Cada shot mide y devuelve un bit; con muchas repeticiones podemos estimar probabilidades.
- `.result()` bloquea hasta obtener la respuesta.

Celda 14

```python
quantum_counts = quantum_result.get_counts(circuit)
```

Explicación:

- Extrae un diccionario con la cuenta de cada resultado medido (por ejemplo, `{'0': 512, '1': 512}`).
- Permite ver la distribución de 0s y 1s tras la medición del qubit 0.

Celda 15
(vacía)
Explicación:

- Celda en blanco preparatoria por si quieres añadir análisis de resultados (por ejemplo, graficar `quantum_counts`).

Con esto tienes, paso a paso, el propósito de cada línea: desde la creación del circuito y sus puertas cuánticas, hasta la ejecución local en simulador y en hardware real, y la recolección de datos. Si quieres profundizar en algún punto (por ejemplo, la teoría de la entrelazación o cómo interpretar `quantum_counts`), ¡dímelo y lo ampliamos!