Lab 2 QFT and QPE Algorithm

3220106039 李瀚轩

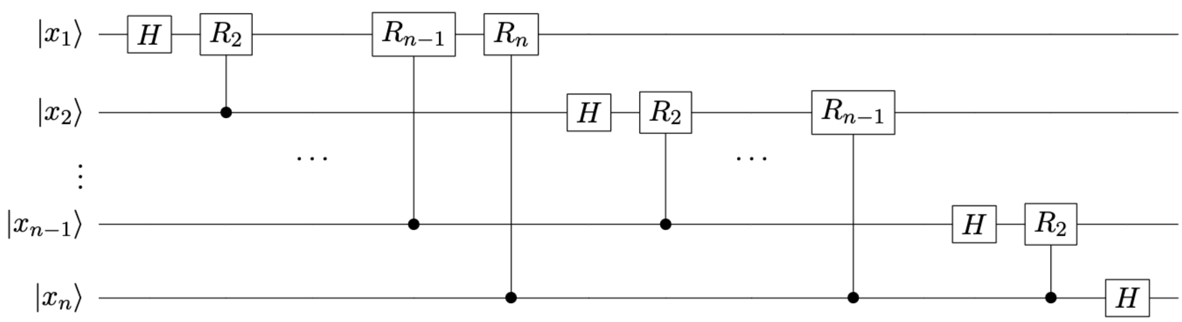
一、QFT 算法

QFT 是量子傅里叶变换，能够将一个量子态在计算基底上的表示转换为频率基底上的表示。

用公式表示为：

$$\text{QFT}|x\rangle = \frac{1}{\sqrt{2^n}} \sum_{k=0}^{2^n-1} e^{\frac{2\pi i}{2^n}xk}|k\rangle$$

QFT 的电路可以通过一系列受控 R 门和 H 门实现：



1.1 三量子比特 QFT

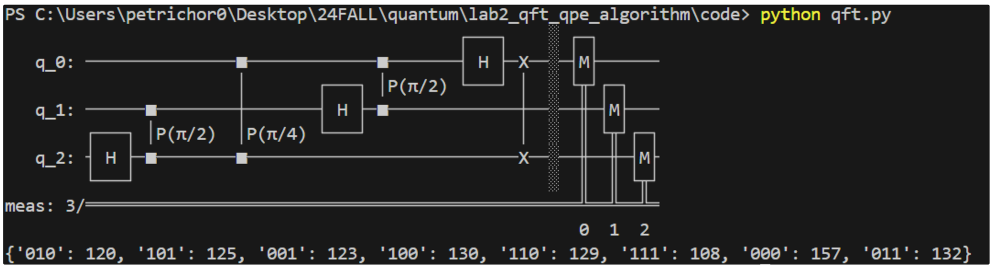
运行实验文档中给出的 QFT 算法，可以得到结果态采样的频率分布：

```python
from numpy import pi
from qiskit import QuantumCircuit, transpile
from qiskit.providers.basic_provider import BasicSimulator


def qft(qc:QuantumCircuit) -> QuantumCircuit:
    for i in range(qc.num_qubits - 1, -1, -1):
        qc.h(i)
        for j in range(i - 1, -1, -1):
            qc.cp(pi / 2 ** (i - j), j, i)
    for i in range(qc.num_qubits // 2):
        qc.swap(i, qc.num_qubits - i - 1)
    return qc


# TODO: change the number of qubits
n_qubits = 3
qc = QuantumCircuit(n_qubits)
# TODO: add quantum gate to set the initial state
qc = qft(qc)
qc.measure_all()
print(qc)


backend = BasicSimulator()
tqc = transpile(qc, backend)
result = backend.run(tqc).result()
```

```
counts = result.get_counts()
print(counts)
```

其中核心 `qft` 函数接受一个量子电路并在其上实现量子傅里叶变换。具体细节就是对每个 `bit` 先应用一个 `Hadamard` 门，然后通过内层循环对每一对 `bit` 应用一个受控 R 门，最后执行 `SWAP` 交换门。

运行上述代码，得到的结果态采样的频率分布如下图所示：

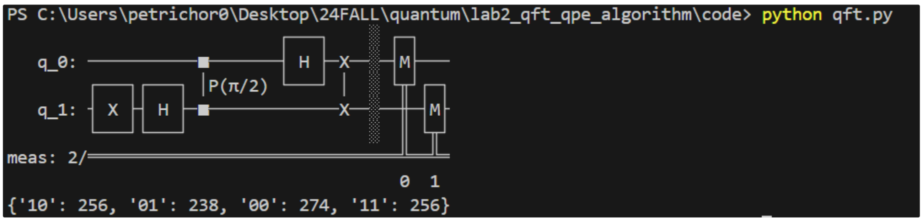


1.2 双量子比特+改变初态

我们将源代码中的 `n_qubits` 设置为 2， 并通过 `qc.x(1)` 将第一个量子比特设置为1，即将初态设置为 $|10\rangle$.
修改后的部分代码如下：

```
# TODO: change the number of qubits
n_qubits = 2
qc = QuantumCircuit(n_qubits)
# TODO: add quantum gate to set the initial state
qc.x(1)
```

运行代码，得到如下结果：



现在我们进行理论的推导：

根据量子傅里叶变换的张量积形式，我们有：

$$\mathrm{QFT}|\overline{j_1 j_2 \cdots j_n}\rangle = \frac{1}{\sqrt{2^n}}(|0\rangle + e^{2\pi i \overline{0.j_n}}|1\rangle)(|0\rangle + e^{2\pi i \overline{0.j_{n-1}j_n}}|1\rangle)\cdots(|0\rangle + e^{2\pi i \overline{0.j_1\cdots j_{n-1}j_n}}|1\rangle)$$

将本题的情况代入，可得：

$$\begin{aligned}\mathrm{QFT}|10\rangle &= \frac{1}{2}(|0\rangle + |1\rangle)(|0\rangle - |1\rangle) \\ &= \frac{1}{2}(|00\rangle - |01\rangle + |10\rangle - |11\rangle)\end{aligned}$$

可以看到，测量结果应该大致均匀分布在双量子比特的四个态上，实验电路的测量结果输出和理论推导一致。

1.3 复杂度分析

`QFT` 电路的代码分析如下：

```python
def qft(qc: QuantumCircuit) -> QuantumCircuit:
    for i in range(qc.num_qubits - 1, -1, -1):  # 外层循环: 从最高位量子比特开始
        qc.h(i)  # 对量子比特 i 应用 Hadamard 门
        for j in range(i - 1, -1, -1):  # 内层循环: 对低于 i 的量子比特应用受控相位旋转门
            qc.cp(pi / 2 ** (i - j), j, i)
    for i in range(qc.num_qubits // 2):  # SWAP门
        qc.swap(i, qc.num_qubits - i - 1)
    return qc
```

可以观察到:

- 每个量子比特都应用一个 H 门, 对于 n 个量子比特, 总共有 n 个 H 门,复杂度为 $O(n)$
- 对于受控旋转门, 我们可以得到总数为 $\sum_{i=1} i = \frac{n(n-1)}{2} = O(n^2)$.
- 对于 SWAP 门, 复杂度也为 $O(n)$.

因此量子傅里叶变换复杂度为 $O(n^2)$.

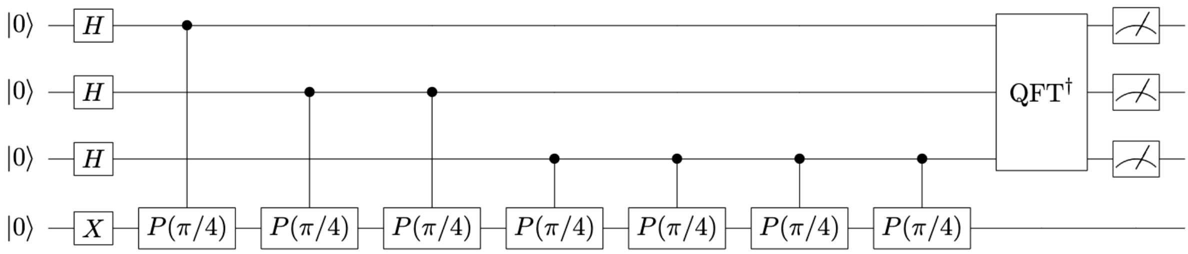
二、QPE 算法

量子相位估计主要用于估计酉矩阵本质态对应本征值的相位。

首先对于酉矩阵 U 的本征态 $|\psi\rangle$ 对应本征值的相位 ϕ, 即:

$$U|\psi\rangle = e^{2\pi i\phi}|\psi\rangle$$

QPE 算法使用两部分量子寄存器实现相位的估计, 电路如下图所示:



2.1 构建 QPE 电路

QPE 电路代码如下:

```python
from qiskit import QuantumRegister, ClassicalRegister, QuantumCircuit, transpile
from qiskit.circuit.library import QFT
from qiskit.providers.basic_provider import BasicSimulator

qr = QuantumRegister(4)
cr = ClassicalRegister(3)
qc = QuantumCircuit(qr, cr)

qc.h([0, 1, 2])

qc.x(3)

pi = 3.141592653589793
theta = pi / 4
qc.cp(theta, 0, 3)
qc.cp(2 * theta, 1, 3)
qc.cp(4 * theta, 2, 3)
```

```python
qc = qc.compose(QFT(3, inverse=True), [0, 1, 2])

qc.measure([0, 1, 2], [0, 1, 2])

backend = BasicSimulator()
tqc = transpile(qc, backend)
result = backend.run(tqc).result()
counts = result.get_counts()
print("QPE counts for P(pi/4):", counts)
plot_histogram(counts)
```

首先我们创建一个四量子比特的寄存器 `qr` ，前三个量子比特用于相位估计，最后一个用于表示本征态。并且我们还创造一个经典寄存器 `cr` 用于记录前三个量子比特的测量结果。接着我们根据 QPE 的算法流程依次对初始化后的量子比特进行操作，包括，对前三个量子比特施加 H门，对最后一个比特赋值为本征态的值，依次施加受控 $P(\theta)$ 门，以及逆傅里叶变换。最后进行测量。

电路的运行结果如下：

```
PS C:\Users\petrichor0\Desktop\24FALL\quantum\lab2_qft_qpe_algorithm\code> python qpe.py
QPE counts for P(pi/4): {'001': 1024}
```

该电路估计 $P(\frac{\pi}{4})$ 门的相位，理论上结果为 $\frac{1}{8}$，二进制近似值为 001_2,因此测量结果集中在 001 ,符合预期。

2.2 更改角度并验证电路输出

在上述代码的基础上，将 θ 改为 $\frac{2\pi}{3}$，并修改相应的 $P(\theta)$ 门操作，部分代码如下：

```python
theta = 2 * pi / 3
qc.cp(theta, 0, 3)
qc.cp(2 * theta, 1, 3)
qc.cp(4 * theta, 2, 3)
```

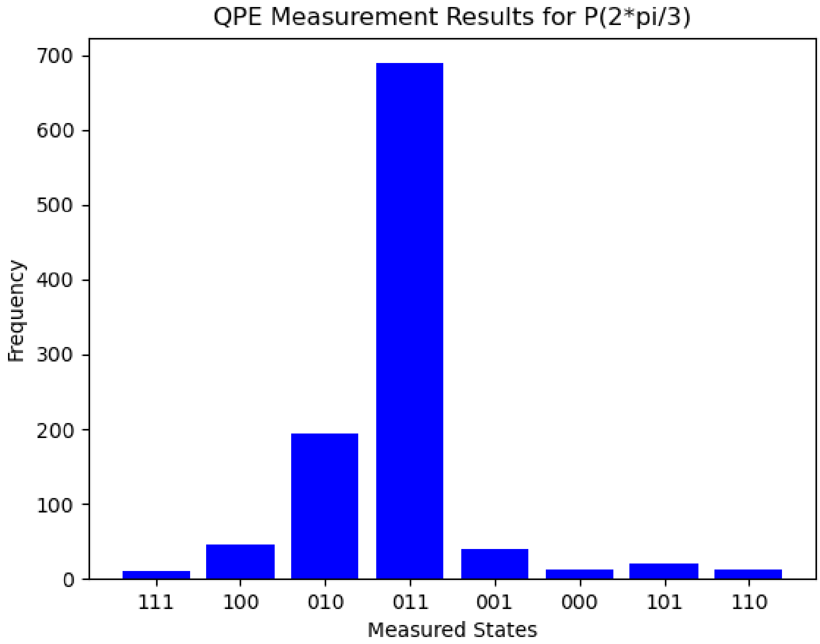
运行该电路，得到结果如下：

```
PS C:\Users\petrichor0\Desktop\24FALL\quantum\lab2_qft_qpe_algorithm\code> python qpe.py
QPE counts for P(pi/4): {'111': 10, '100': 45, '010': 195, '011': 689, '001': 40, '000': 13, '101': 20, '110': 12}
```

为了直观起见，使用如下代码画出频率分布直方图：

```python
import matplotlib.pyplot as plt

counts = {'111': 10, '100': 45, '010': 195, '011': 689, '001': 40, '000': 13, '101':
20, '110': 12}

labels = list(counts.keys())
values = list(counts.values())

plt.bar(labels, values, color='blue')
plt.xlabel('Measured States')
plt.ylabel('Frequency')
plt.title('QPE Measurement Results for P(2*pi/3)')
plt.show()
```

得到结果：



下面我们进行理论的推导:

对于相位旋转门 $P(\frac{2\pi}{3}) = \begin{bmatrix} 1 & 0 \\ 0 & e^{\frac{2\pi}{3}i} \end{bmatrix}$,当本征态为 $|1\rangle$ 时,我们有:

$$P(\frac{2\pi}{3})|1\rangle = e^{\frac{2\pi}{3}i}|1\rangle = e^{2\pi i\phi}$$

可以解得 $\phi = \frac{1}{3}$.但是我们发现 $\frac{1}{3}$ 不能由二进制精确表示,这也是我们上述的电路运行结果中,各个值都有出现的原因,我们不能像 $P(\frac{\pi}{4})$ 那样准确估计出相位值。观察上述的频率分布直方图,可以看到 $(011)_2 = \frac{3}{8}$ 的频率最高,因为这个结果与 $\frac{1}{3}$ 最接近。因此可以得出,实验结果和理论分析是相符合的。

2.2 复杂度分析

- 受控 U 门:我们需要对每个控制量子比特应用 U^{2^j} 操作,受控 U 门的数量为 $\frac{n(n-1)}{2} = O(n^2)$.
- 逆量子傅里叶变换:复杂度为 $O(n^2)$.

因此 QPE 的量子门复杂度为 $O(n^2)$.

2.3 选做部分

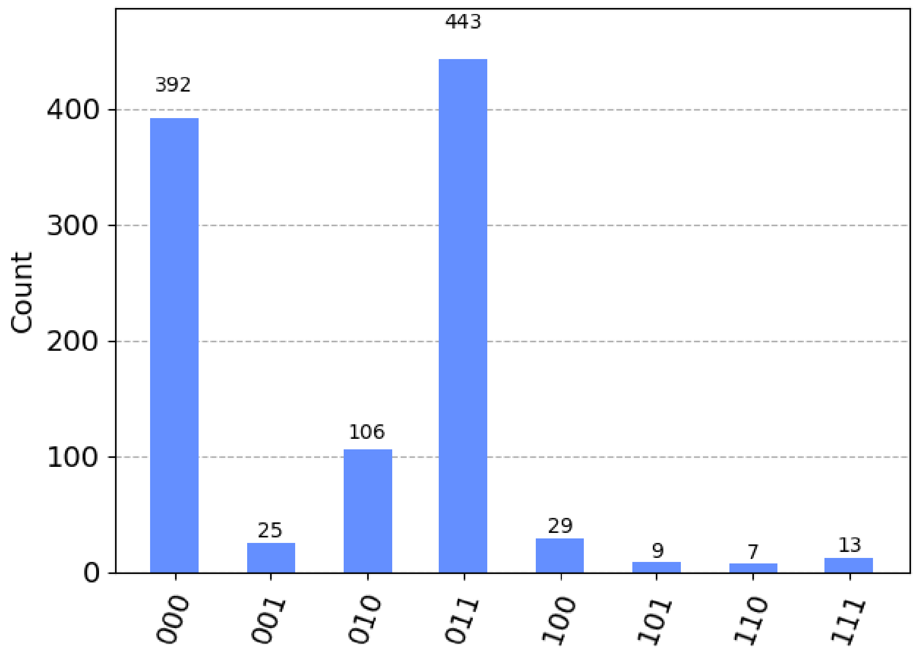
更改特征寄存器的初态为非本征态 $|\varphi\rangle = [3/5, 4/5]^\top$,

代码如下:

```
from qiskit import QuantumRegister, ClassicalRegister, QuantumCircuit, transpile
from qiskit.circuit.library import QFT
from qiskit.providers.basic_provider import BasicSimulator
from numpy import pi
import numpy as np
import matplotlib.pyplot as plt
from qiskit.visualization import plot_histogram

qr = QuantumRegister(4)
cr = ClassicalRegister(3)
qc = QuantumCircuit(qr, cr)

qc.h([0, 1, 2])
```

```
theta_non_eigen = 2 * np.arctan(4/3)
qc.ry(theta_non_eigen, 3)

theta = 2 * pi / 3
qc.cp(theta, 0, 3)
qc.cp(2 * theta, 1, 3)
qc.cp(4 * theta, 2, 3)

qc = qc.compose(QFT(3, inverse=True), [0, 1, 2])

qc.measure([0, 1, 2], [0, 1, 2])

simulator = BasicSimulator()
compiled_circuit = transpile(qc, simulator)
result = simulator.run(compiled_circuit).result()

counts = result.get_counts(qc)
print("QPE counts with non-eigenstate initial state:", counts)

plot_histogram(counts)
plt.show()
```

得到结果的频率分布直方图如下图所示：



使用非本征态时，测量结果可能不会集中在某个确定的二进制结果上，而是更分散。