

浙江大学



《量子计算理论基础与软件系统》 实验报告

实验名称：	Lab3 Shor and Grover Algorithm
姓 名：	王晓宇
学 号：	3220104364
电子邮箱：	3220104364@zju.edu.cn
联系电话：	19550222634
授课教师：	卢丽强/尹建伟
助 教：	储天尧

2024 年 11 月 24 日

Lab 3 Shor and Grover Algorithm

1 实验简介

1.1 Shor 算法

1.2 Grover 算法

2 实验要求

Lab 3 Shor and Grover Algorithm

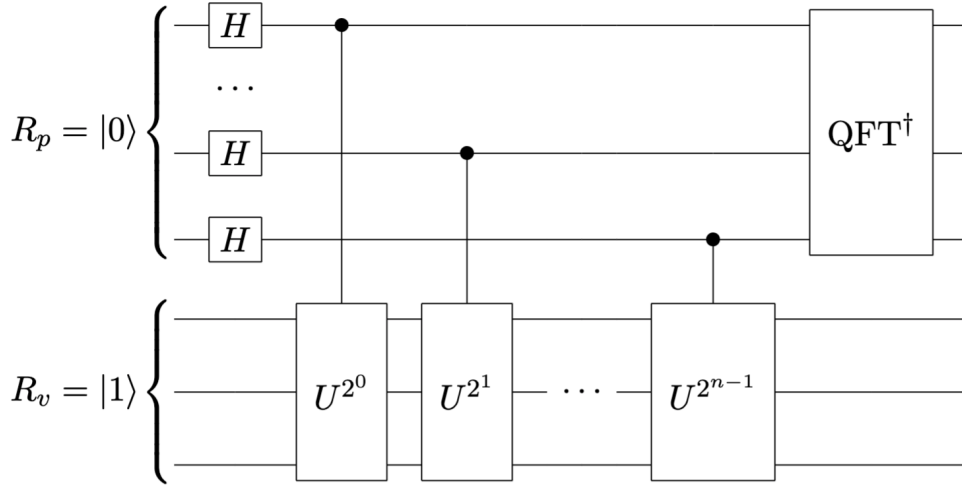
1 实验简介

本次实验中，我们将使用 `qiskit` 框架实现 Shor 算法 和 Grover 算法，并通过量子电路的模拟运行加深对这两个核心量子算法的理解。

1.1 Shor 算法

Shor 算法是一种用于大整数分解的量子算法，通过量子相位估计算法来找到周期性，从而分解大整数。它是量子计算最具革命性的算法之一，展示了量子计算在解决经典计算难题中的优势。

Shor 算法通过将大整数分解问题转化为计算阶问题，从而实现量子加速。假设 N 是待分解的大整数， $N = pq$ ，其中 p 和 q 是两个素数。任意选取整数 a ，使得 a 和 N 互质，使用量子过程计算 a 的阶 r ，使得 $a^r \equiv 1 \pmod{N}$ 。根据小端规则，基于 `qiskit` 的量子电路实现如下图所示：



其中

$$U|u_s\rangle = e^{\frac{2\pi i s}{r}}|u_s\rangle$$

$$\frac{1}{\sqrt{r}} \sum_{s=0}^{r-1} |u_s\rangle = |1\rangle$$

1.2 Grover 算法

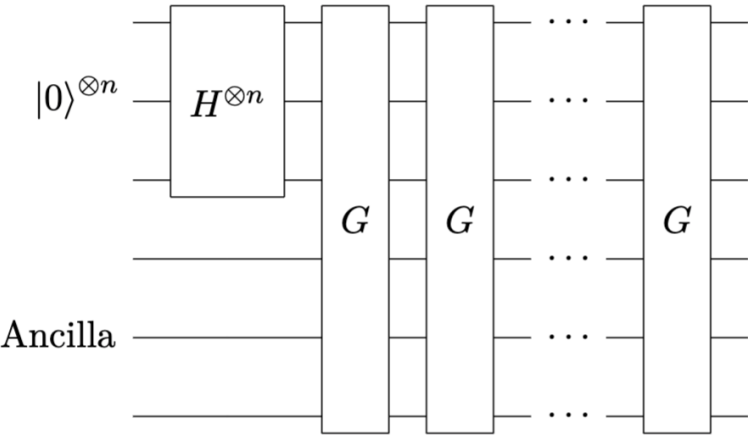
Grover 算法是量子计算中用于搜索未排序数据库的量子算法，其具有平方加速，能够在 $O(\sqrt{N})$ 的时间内找到目标项。

待搜索的数据库由 **Oracle** 实现：

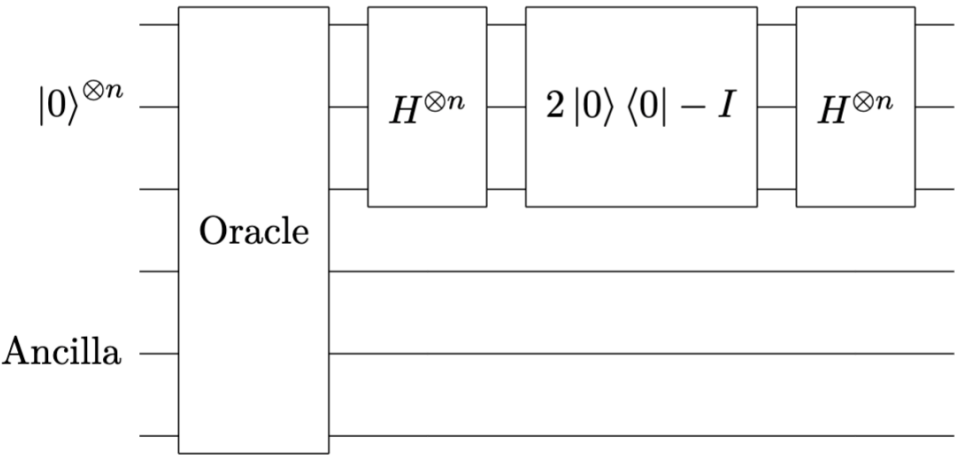
$$\text{Oracle}|x\rangle = (-1)^{f(x)}|x\rangle$$

其中 x 为目标项时 $f(x) = 1$ ，否则 $f(x) = 0$ 。

Grover 算法通过反复作用 **Grover** 算子来增加目标项的振幅，从而实现搜索。其量子电路如下图所示：



其中 **Grover** 算子 G 如下图所示：



其中 **Ancilla** 为实现 **Oracle** 可能使用的辅助量子比特。

2 实验要求

1. Shor 算法

```
1  import numpy as np
2  from qiskit import QuantumCircuit, transpile
3  from qiskit.circuit.library import UnitaryGate, QFT
4  from qiskit.providers.basic_provider import BasicSimulator
5
6
7  def shor_circuit(N, a, n_p, n_v):
8      qc = QuantumCircuit(n_p + n_v, n_p)
9
10     for q in range(n_p):
11         qc.h(q)
12     qc.x(n_p)
13
14     for q in range(n_p):
15         exponent = 2 ** q
16         ctrl_mod = mod_exp_circuit(a, exponent, N,
n_v).to_gate().control(1)
17         qc.append(ctrl_mod, [q] + list(range(n_p, n_p + n_v)))
18
19     qc.append(QFT(n_p, inverse=True), range(n_p))
20
21     qc.measure(range(n_p), range(n_p))
22     return qc
23
24 def mod_exp_circuit(a, power, N, n_v):
25     qc = QuantumCircuit(n_v)
26     for _ in range(power):
27         qc.append(mod_circuit(a, N, n_v), range(n_v))
28     return qc
29
30 def mod_circuit(a, N, n_v):
31     matrix = np.zeros((2 ** n_v, 2 ** n_v), dtype=int)
32     # TODO complete modular multiplication circuit
```

```

33     return UnitaryGate(matrix)
34
35
36 N = 15
37 a = 7
38 n_p = 3 # number of qubits in period register
39 n_v = 4 # number of qubits in value register
40 qc = shor_circuit(N, a, n_p, n_v)
41 print(qc.draw())
42
43 backend = BasicSimulator()
44 tqc = transpile(qc, backend)
45 result = backend.run(tqc).result()
46 counts = result.get_counts()
47 print("counts:", counts)
48
49 r = len(counts)
50 print(f"r: {r}")
51
52 if r % 2 == 0 and pow(a, r // 2, N) != N - 1:
53     factor1 = np.gcd(pow(a, r // 2) - 1, N)
54     factor2 = np.gcd(pow(a, r // 2) + 1, N)
55     print(f"N = {factor1} * {factor2}")
56 else:
57     print("Invalid a!")

```

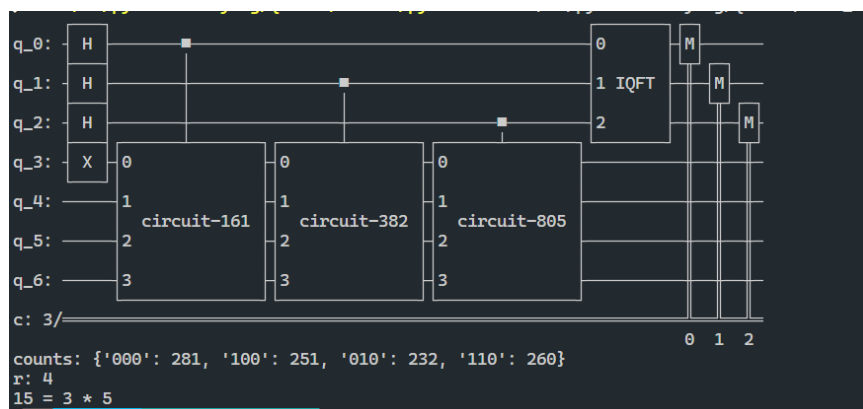
- 以上代码使用 `qiskit` 部分实现了 Shor 算法，请补全 `mod_circuit` 函数中量子门 U 对应的酉矩阵定义，以构造量子门 $U|y\rangle = |ay(\bmod n)\rangle$ 。

```

1 def mod_circuit(a, N, n_v):
2     matrix = np.zeros((2 ** n_v, 2 ** n_v), dtype=int)
3     for i in range(2 ** n_v):
4         matrix[i][i] = (a * i)%N
5     U, S, V_dagger = np.linalg.svd(matrix)
6     return UnitaryGate(U)

```

- 运行补全的代码，取 $a = 7$ ，分解整数 $N = 15$ ，观察输出的计数结果，验证分解结果是否正确。



可以看到阶数为4.符合我们用7分解15的阶数，最终得到 $15=3*5$ ，结果正确。

- 修改代码中的部分参数，选取合适的 a ，分解整数 $N = 21$ 。

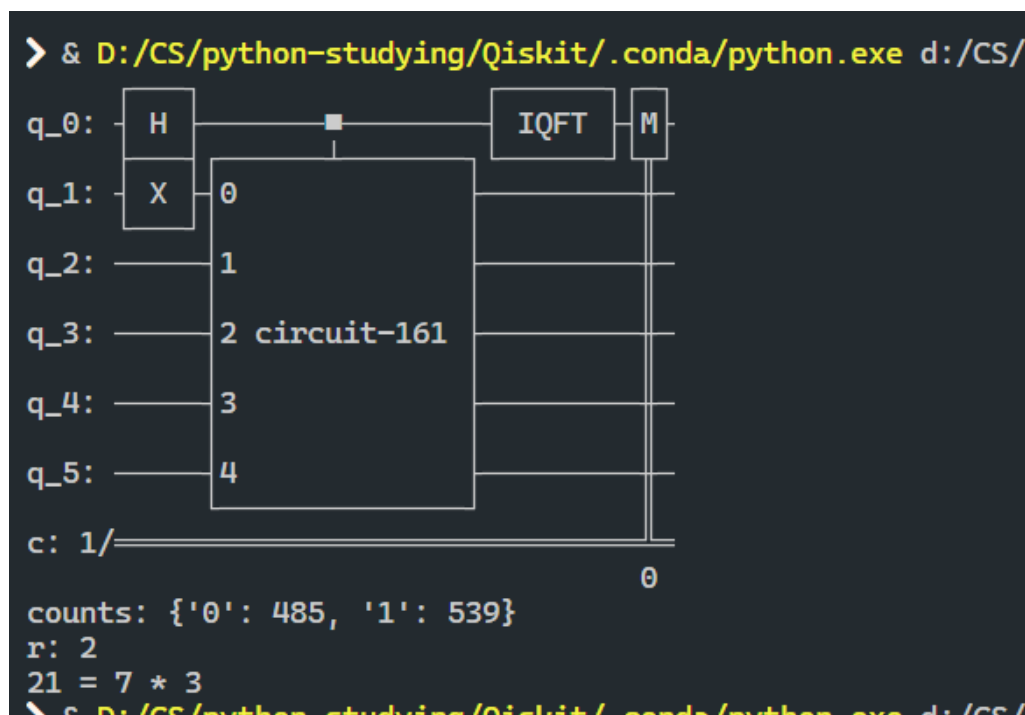
修改参数如下：

```

N = 21
a = 8
n_p = 1 # number of qubits in period register
n_v = 5 # number of qubits in value register

```

由 $8^2 \bmod 21 = 1$ 可以知道此时的阶数为2，由此即可算出两个质因数7、3



2. Grover 算法 (选做 Bonus)

```

1 def oracle():
2     qc = QuantumCircuit(4)
3     qc.cz(0, 3)
4     return qc

```

- 以上代码定义了一个四量子比特的 **Oracle** 电路，分析该函数标记的目标态。
由Z门的酉矩阵：

$$\begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix}$$

我们可以得知仅将 $|1\rangle$ 的相位翻转，结合受控Z门`qc.cz(0, 3)`我们可以得知，它将 $|1xx1\rangle$ 即： $|1001\rangle$ 、 $|1011\rangle$ 、 $|1101\rangle$ 、 $|1111\rangle$ 相位翻转（这里用了小端排序，与Qiskit一致），即标记的目标态 $|9\rangle$ 、 $|11\rangle$ 、 $|13\rangle$ 、 $|15\rangle$ 进行了相位翻转，我们也可以推算出，这个**Oracle**算子的作用其实是将0-15中筛选出9、11、13、15。

- 根据实验简介中的量子电路图，基于 `qiskit` 实现 **Grover** 算法，完成该 **Oracle** 目标项的搜索。

这里给出代码实现：

```

1 import numpy as np
2 from qiskit import QuantumCircuit, transpile
3 from qiskit.circuit.library import UnitaryGate
4 from qiskit.providers.basic_provider import BasicSimulator
5
6
7 def grover(n,times):
8     qc = QuantumCircuit(n,n)
9     qc.h(range(n))
10
11     for _ in range(times):
12         qc.append(G(n), range(n))
13
14     qc.measure(range(0,n), range(n))
15     return qc
16
17 def G(n):

```

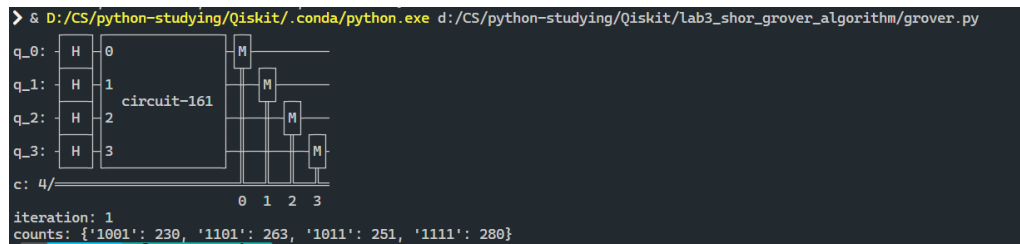


```

18     qc = QuantumCircuit(n)
19     qc.append(oracle(), range(n))
20     qc.h(range(n))
21
22     qc.append(rotate(n), range(n))
23
24     qc.h(range(n))
25     return qc
26
27 def rotate(n):
28     matrix = np.zeros((2 ** n, 2 ** n), dtype=int)
29     for i in range(2**n):
30         matrix[i][i] = -1
31     matrix[0][0] = 1
32     return UnitaryGate(matrix)
33
34 def oracle():
35     # Q2.2
36     qc = QuantumCircuit(4)
37     qc.cz(0, 3)
38     return qc
39
40 N = 4
41 times = 1
42 qc = grover(N, oracle, times)
43 print(qc.draw())
44 print("iteration:", times)
45
46 backend = BasicSimulator()
47 tqc = transpile(qc, backend)
48 result = backend.run(tqc).result()
49 counts = result.get_counts()
50 print("counts:", counts)

```

这里设置了迭代次数为1，得到正解 $|1001\rangle$ 、 $|1011\rangle$ 、 $|1101\rangle$ 、 $|1111\rangle$ ，关于迭代次数的选择解释在后文。



我们先逐步解释代码：

先解释主函数 `grover`：

`grover` 函数接受两个参数：`n` 表示量子比特的数量，`times` 表示Grover算法的迭代次数。

- 函数首先创建一个包含 `n` 个量子比特和 `n` 个经典比特的量子电路 `qc`。然后，对所有量子比特应用 **Hadamard** 门，初始化为叠加态。
- 接下来，函数进入一个循环，迭代 `times` 次。在每次迭代中，调用辅助函数 `G`，`G` 函数生成 **Grover** 算法的扩散操作。
- 在循环结束后，`grover` 函数对前 `n` 个量子比特进行测量，并将测量结果存储在相应的经典比特中。

接下来是主函数调用的辅助函数：

`oracle` 函数创建一个包含 4 个量子比特的量子电路 `qc`，并在第 0 个和第 3 个量子比特之间添加一个受控 **Z** 门（**CZ** 门）。

- 我们探讨过这里的作用是标记了 $|1001\rangle$ 、 $|1011\rangle$ 、 $|1101\rangle$ 、 $|1111\rangle$ 四个状态

`G` 函数接受一个参数 `n`，表示量子比特的数量。

- 函数首先创建一个包含 `n` 个量子比特的量子电路 `qc`。
- 调用 `oracle` 函数应用 **Oracle** 电路。
- 接着 H^{\otimes} 我们使用了 `qc.h(range(n))` 对所有量子比特应用 **Hadamard** 门。
- 调用 `rotate` 函数生成 $2|0\rangle\langle 0| - I$ 酉矩阵后直接应用。
- 再次对所有量子比特应用 **Hadamard** 门。

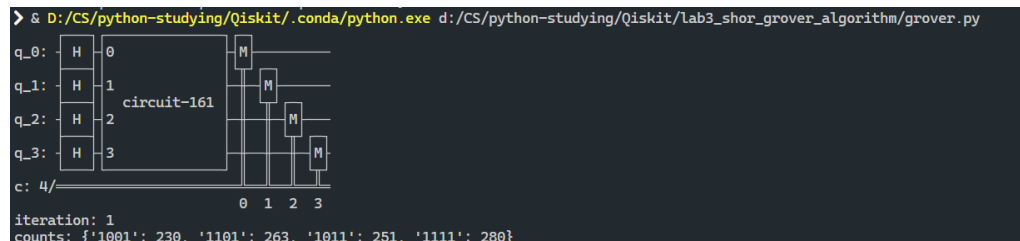
`rotate` 函数接受一个参数 `n`，表示量子比特的数量。

- 函数首先创建一个大小为 $2^n \times 2^n$ 的零矩阵 `matrix`。
- 因为这里的酉矩阵比较简单，直接对矩阵进行填充即可。将矩阵的对角线元素设置为 **-1**，特别地将第一个元素设置为 **1**。

- 将酉矩阵转换为n比特量子电路后返回。
- 根据 **Oracle** 目标态和待搜索态的数量，选取合适的 **G** 算子迭代次数，观察 **Grover** 算法的搜索结果。

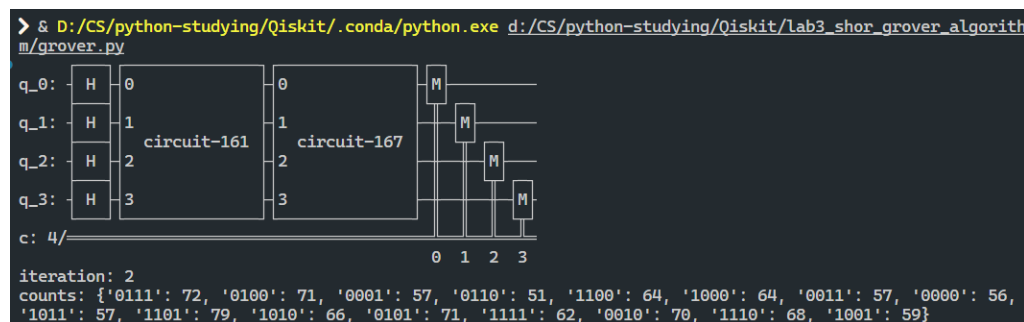
有 $\sin\theta = \frac{2\sqrt{M(N-M)}}{N} = \frac{2 \times \sqrt{4 \times (16-4)}}{16} = \frac{\sqrt{3}}{2}$ 我们可以得知 $\theta = \frac{\pi}{3}$ ，每一次应用 **G** 算子旋转固定角度即为 $\frac{\pi}{3}$ ，由于初始状态是 $\frac{\theta}{2} = \frac{\pi}{6}$ ，因此我们的迭代次数只需要一次便可以逼近正解状态 $\frac{\pi}{6} + (\frac{\pi}{3} \times n) = \frac{\pi}{2} \Rightarrow n = 1$

我们应用**1**次迭代次数：



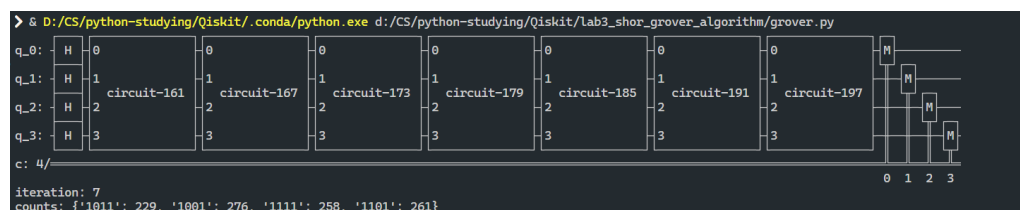
得到正解|1001⟩、|1011⟩、|1101⟩、|1111⟩

而进行**2**次迭代后：



可以看到偏离了正解集合|β⟩

根据几何关系我们可以计算让其旋转 $\frac{\pi}{6} + (\frac{\pi}{3} \times n) = \frac{\pi}{2} + 2\pi \Rightarrow n = 7$ ，即7次之后仍然得到正解相位|1001⟩、|1011⟩、|1101⟩、|1111⟩



得到正解|1001⟩、|1011⟩、|1101⟩、|1111⟩

- 修改 **Oracle** 改变目标态的数量，尝试不同的 **G** 算子迭代次数，观察 **Grover** 算法的搜索结果。

我们修改**Oracle**如下：

```

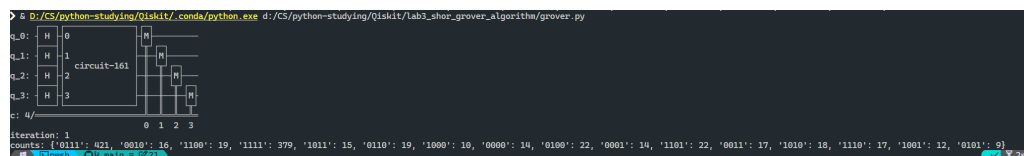
1 def oracle():
2     qc = QuantumCircuit(4)
3     qc.ccz(0,1,2)
4     return qc

```

这里应用**ccz**门，即当量子比特**0**，**1**位同时为**1**时，将第**2**位应用**Z**门，这样标记的目标态只有 **$|0111\rangle$** 、 **$|1111\rangle$** 两个，即正解个数 **$M = 2$** ，此时所有解个数为 **$N = 2^4 = 16$** ，根据**Grover**算法的迭代次数公式

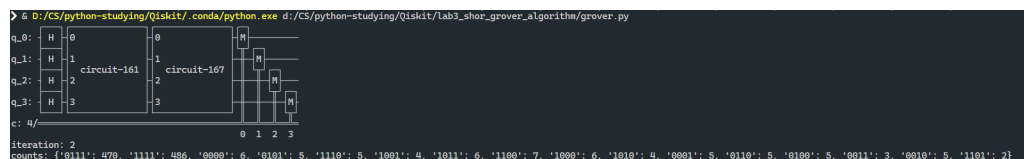
$$k \approx \frac{\pi}{4} \sqrt{\frac{N}{M}} - \frac{1}{2} = 1.72 \approx 2$$

当迭代次数为**1**时：



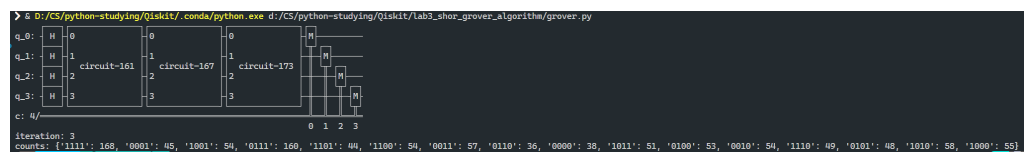
可以看到测量的量子态向正解靠近， **$|0111\rangle$** 、 **$|1111\rangle$** 的测量概率显著地排在前两位。

当迭代次数为**2**时：



此时测量结果则更接近正解，错解出现的概率变得很小，**1000**次测量只有个位数次出现， **$|0111\rangle$** 、 **$|1111\rangle$** 比例最大验证了**Grover**算法的正确性

当迭代次数为**3**时：



可以看到输出结果又向错解方向靠近，正解的概率有所下降，证明理论的迭代次数 **$1.72 \approx 2$** 是正确的