Laboratorinis darbas: Tikrinių reikšmių fazės kvantinio paieškos algoritmo realizavimas ir rezultatai

R. Čiegis

Vilniaus Gedimino technikos universitetas
e-mail: rc@vgtu.lt

Kovo 10 d., 2026, Vilnius

Turime unitarųjį kvantinį operatorių U.

Žinome jo tikrinį vektorių $|\psi\rangle$:

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

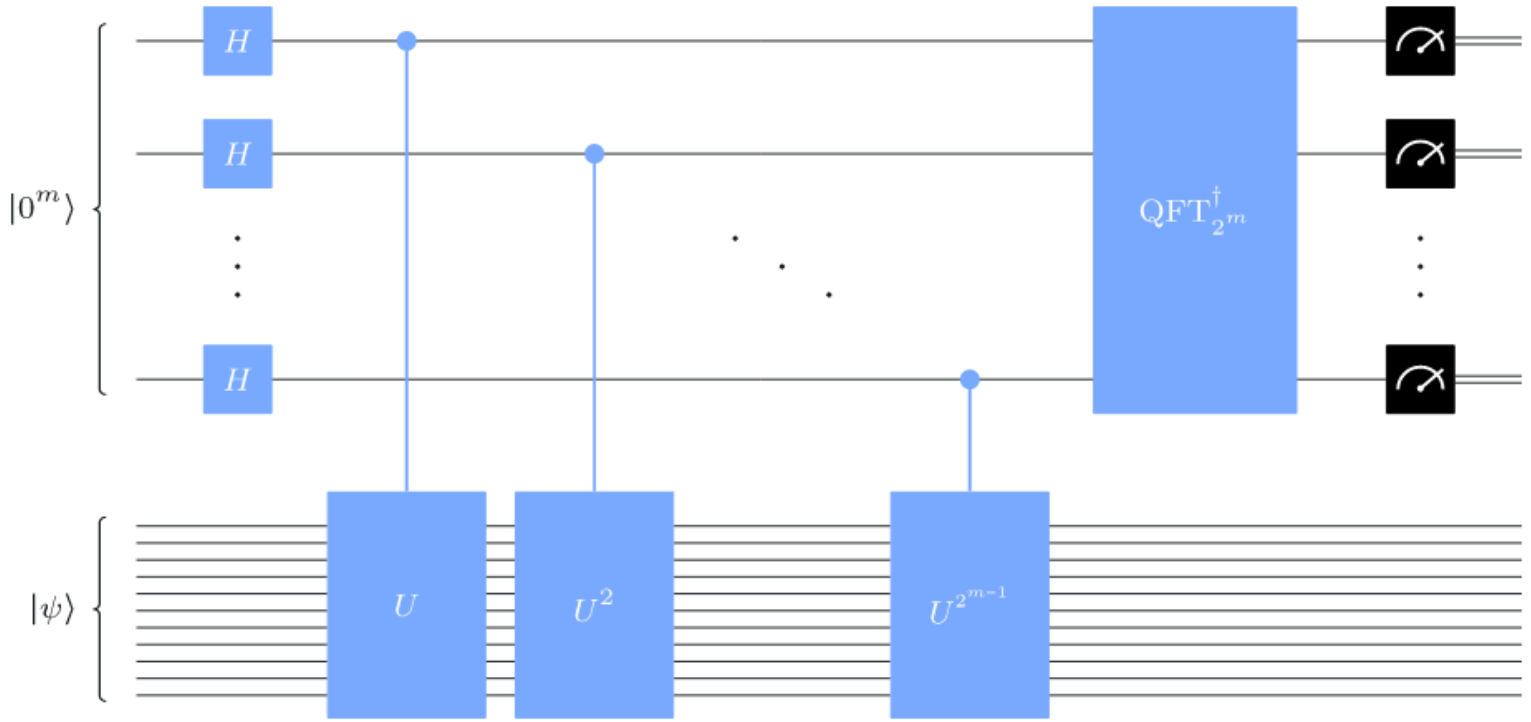
Turime unitarųjį kvantinį operatorių U.

Žinome jo tikrinį vektorių $|\psi\rangle$:

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

Norime apskaičiuoti fazės θ aproksimaciją

$$\theta \approx y/2^m, \quad y \in 0, 1, \ldots, 2^{m-1}.$$


$|0^m\rangle$
$|\psi\rangle$
H
H
H
U
U^2
$U^{2^{m-1}}$
$\mathrm{QFT}^{\dagger}_{2^m}$

Kiekvienam sveikam skaičiui N, apibrėžiame aibę $\mathbb{Z}_N$, kur

$$\mathbb{Z}_N = \{0, 1, \ldots, N-1\}.$$

Kiekvienam sveikam skaičiui N, apibrėžiame aibę $\mathbb{Z}_N$, kur

$$\mathbb{Z}_N = \{0, 1, \ldots, N-1\}.$$

Svarbus poaibis

$$\mathbb{Z}_N^* = \{a \in \mathbb{Z}_N : \ \gcd(a, N) = 1\}.$$

Kiekvienam sveikam skaičiui N, apibrėžiame aibę $\mathbb{Z}_N$, kur

$$\mathbb{Z}_N = \{0, 1, \ldots, N-1\}.$$

Svarbus poaibis

$$\mathbb{Z}_N^* = \{a \in \mathbb{Z}_N : \ \gcd(a, N) = 1\}.$$

Kiekvienam $a \in \mathbb{Z}_N^*$ egzistuoja mažiausias r, toks, kad $a^r \equiv 1 \pmod{N}$. Šis skaičius r vadinamas elemento a eile moduliu N.

Kiekvienam sveikam skaičiui N, apibrėžiame aibę $\mathbb{Z}_N$, kur

$$\mathbb{Z}_N = \{0, 1, \ldots, N-1\}.$$

Svarbus poaibis

$$\mathbb{Z}_N^* = \{a \in \mathbb{Z}_N : \ \gcd(a, N) = 1\}.$$

Kiekvienam $a \in \mathbb{Z}_N^*$ egzistuoja mažiausias r, toks, kad $a^r \equiv 1 \pmod{N}$. Šis skaičius r vadinamas elemento a eile moduliu N.

Žinodami r galime faktorizuoti N.

Apibrėžiame daugybos operatorių

$$M_a \left|x\right\rangle = \left|ax \left(\mod N \right)\right\rangle, \ \forall x \in \mathbb{Z}_N.$$

Apibrėžiame daugybos operatorių

$$M_a \left|x\right\rangle = \left|ax \left(\mod N \right)\right\rangle, \ \forall x \in \mathbb{Z}_N.$$

Šis operatorius yra unitarusis ir jo tikrinės reikšmės yra

$$\omega_r^j = e^{2\pi i j/r}, \quad j \in \{0, \ldots, r-1\}.$$

Apibrėžiame daugybos operatorių

$$M_a |x\rangle = |ax \ (\mod N)\rangle , \ \forall x \in \mathbb{Z}_N .$$

Šis operatorius yra unitarusis ir jo tikrinės reikšmės yra

$$\omega_r^j = e^{2\pi i j/r}, \quad j \in \{0, \ldots, r-1\}.$$

Algoritme naudosime tikrinių vektorių superpoziciją

$$|1\rangle = \frac{1}{\sqrt{r}} \sum_{k=0}^{r-1} |\psi_k\rangle .$$

Reikia efektyviai skaičiuoti operatoriaus M_a eksponentes moduliu N

$$M_a^k, \quad k = 1, 2, 4, \ldots, 2^{m-1}.$$

Tai atliksime ne daugindami operatorių M_a reikiamą skaičių $k-1$ kartų, bet skaičiuodami

$$b = a^k \mod N$$

ir į algoritmo kvantinę grandinę įtrauksime operatoriaus M_b kvantinę realizaciją.

Reikia efektyviai skaičiuoti operatoriaus M_a eksponentes moduliu N

$$M_a^k, \quad k = 1, 2, 4, \ldots, 2^{m-1}.$$

Tai atliksime ne daugindami operatorių M_a reikiamą skaičių $k-1$ kartų, bet skaičiuodami

$$b = a^k \mod N$$

ir į algoritmo kvantinę grandinę įtrauksime operatoriaus M_b kvantinę realizaciją.

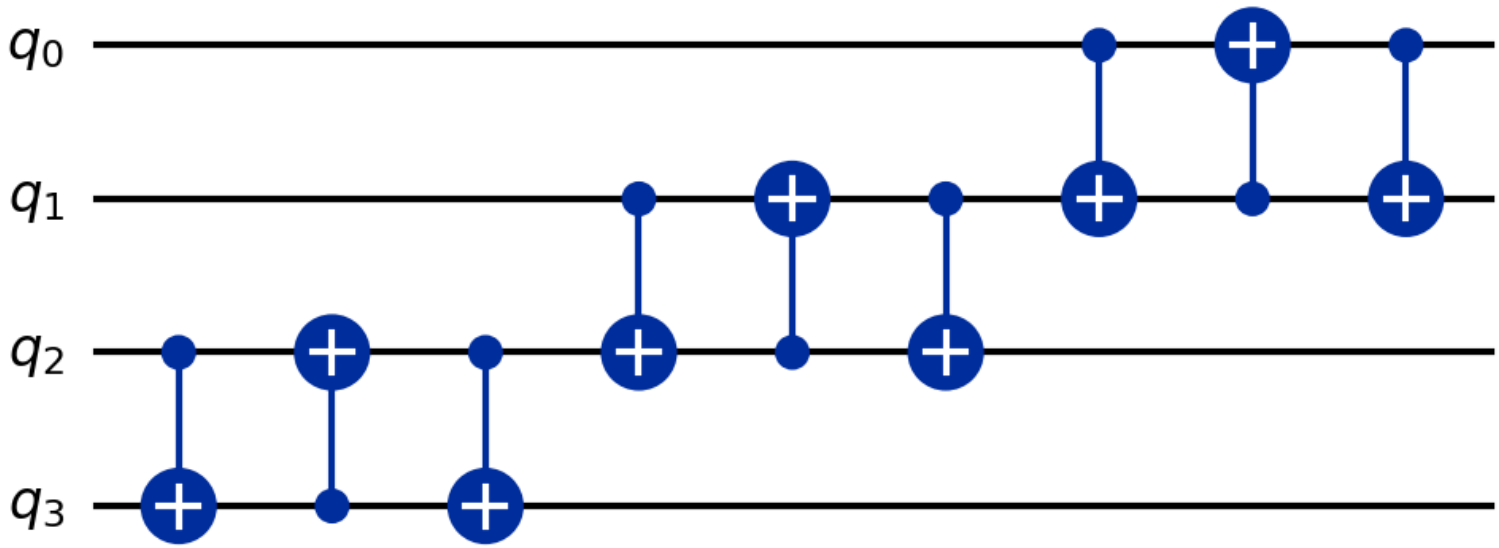
Parametrą b galime labai greitai skaičiuoti ir klasikiniais kompiuteriais.

$N = 15$ ir $a = 2$

$$M_2 |0\rangle = |0\rangle , \; M_2 |1\rangle = |2\rangle , \; M_2 |2\rangle = |4\rangle , \; M_2 |3\rangle = |6\rangle ,$$
$$M_2 |4\rangle = |8\rangle , \; M_2 |5\rangle = |10\rangle , \; M_2 |6\rangle = |12\rangle , \; M_2 |7\rangle = |14\rangle ,$$
$$M_2 |8\rangle = |1\rangle , \; M_2 |9\rangle = |3\rangle , \; M_2 |10\rangle = |5\rangle , \; M_2 |11\rangle = |7\rangle ,$$
$$M_2 |12\rangle = |9\rangle , \; M_2 |13\rangle = |11\rangle , \; M_2 |14\rangle = |13\rangle .$$

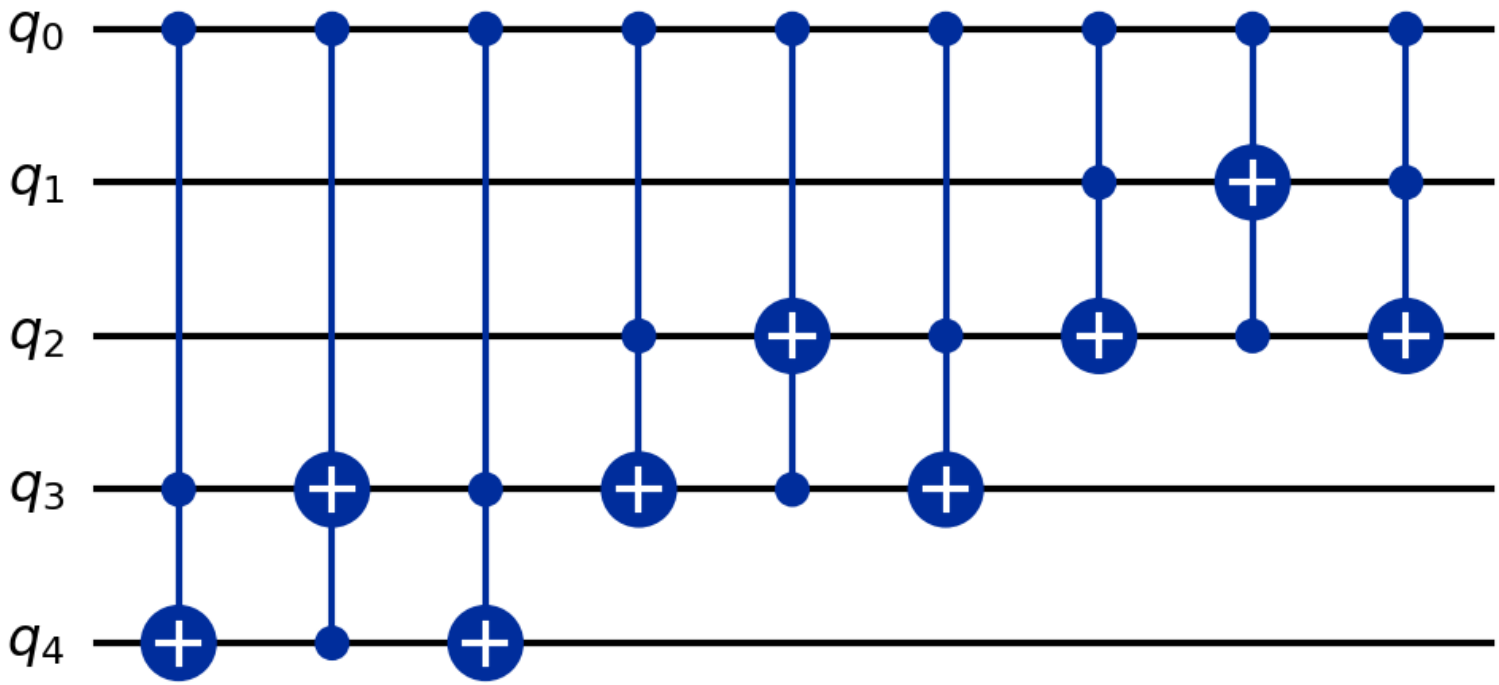
```
def M2mod15():
    """
    M2 (mod 15)
    """
    b = 2
    U =QuantumCircuit(4)
    U.swap(2, 3)
    U.swap(1, 2)
    U.swap(0, 1)
    U = U.to_gate()
    U.name = f"M_b"
    return U
```

```
# Get the M2 operator
M2 = M2mod15()
# Add it to a circuit and plot
circ = QuantumCircuit(4)
circ.compose (M2, inplace=True)
circ.decompose (reps=2).draw(output="mpl", style="iqp",
            filename="circ1.png" )
```


q_0
q_1
q_2
q_3

```
def controlled_M2mod15():
    """
    Controlled M2 (mod 15)
    """

    b = 2
    U = QuantumCircuit(4)
    U.swap(2, 3)
    U.swap(1, 2)
    U.swap(0, 1)
    U = U.to_gate()
    U.name = f"M_b"
    c_U = U.control()
    return c_U
```

```
# Get the controlled - M2 operator
M2 = controlled_M2mod15()
# Add it to a circuit and plot
circ = QuantumCircuit(5)
circ.compose (controlled_M2, inplace=True)
circ.decompose (reps=1).draw(output="mpl", style="iqp",
            filename="circ2.png" )
```


q_0
q_1
q_2
q_3
q_4

Operatoriai

$$M_b, \quad b = a^{2^k} \ (\mod N), \quad k = 0, 1, \ldots, 7,$$

$$a = 2, \quad N = 15.$$

Gauname b :

$$[2,\ 4,\ 1,\ 1,\ 1,\ 1,\ 1,\ 1].$$

Operatoriai

$$M_b, \quad b = a^{2^k} \ (\mod N), \quad k = 0, 1, \ldots, 7,$$

$$a = 2, \quad N = 15.$$

Gauname b :

$$[2,\ 4,\ 1,\ 1,\ 1,\ 1,\ 1,\ 1].$$

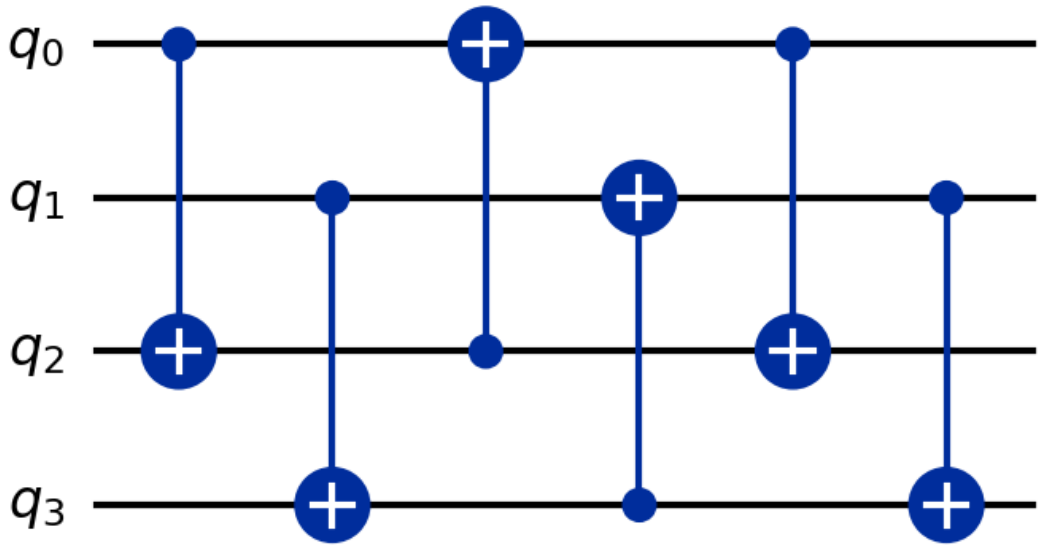
$$N = 15 \text{ ir } b = 4$$

$$M_4 |0\rangle = |0\rangle, \ M_4 |1\rangle = |4\rangle, \ M_4 |2\rangle = |8\rangle, \ M_4 |3\rangle = |12\rangle,$$
$$M_4 |4\rangle = |1\rangle, \ M_4 |5\rangle = |5\rangle, \ M_4 |6\rangle = |9\rangle, \ M_4 |7\rangle = |13\rangle,$$
$$M_4 |8\rangle = |2\rangle, \ M_4 |9\rangle = |6\rangle, \ M_4 |10\rangle = |10\rangle, \ M_4 |11\rangle = |14\rangle,$$
$$M_4 |12\rangle = |3\rangle, \ M_4 |13\rangle = |7\rangle, \ M_4 |14\rangle = |11\rangle.$$

```
def M4mod15():
    """
    M4 (mod 15)
    """
    b = 4
    U =QuantumCircuit(4)
    U.swap(1, 3)
    U.swap(0, 2)
    U = U.to_gate()
    U.name = f"M_b"
    return U
```

```
# Get the M4 operator
M4 = M4mod15()
# Add it to a circuit and plot
circ = QuantumCircuit(4)
circ.compose (M4, inplace=True)
circ.decompose (reps=2).draw(output="mpl", style="iqp",
filename="circ5.png" )
```

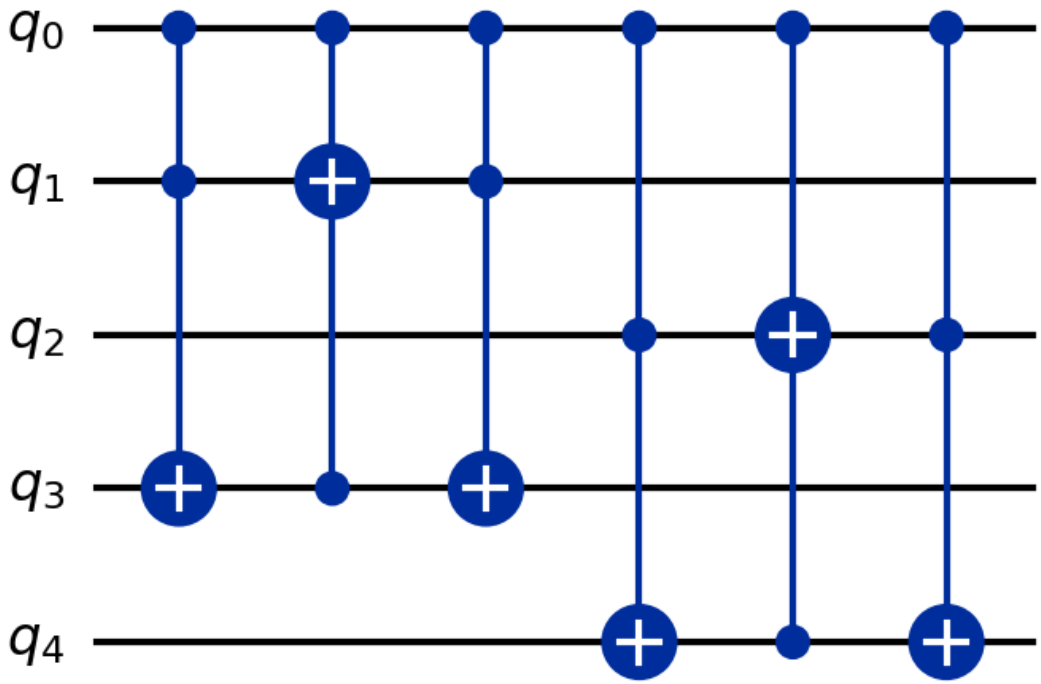
M4 circ



```
def  controlled M4mod15():
    """
    Controlled M4 (mod 15)
    """
    b = 4
    U =QuantumCircuit(4)
    U.swap(1, 3)
    U.swap(0, 2)
    U = U.to_gate()
    U.name = f"M_b"
c_U = U.control ()
    return c_U
```

```
# Get the controlled –M4 operator
controlled_M4 = controlled_M4mod15()
# Add it to a circuit and plot
circ = QuantumCircuit(5)
circ.compose (controlled_M4, inplace=True)
circ.decompose (reps=1).draw(output="mpl", style="iqp",
filename="circ6.png" )
```

controlled_M4 circ

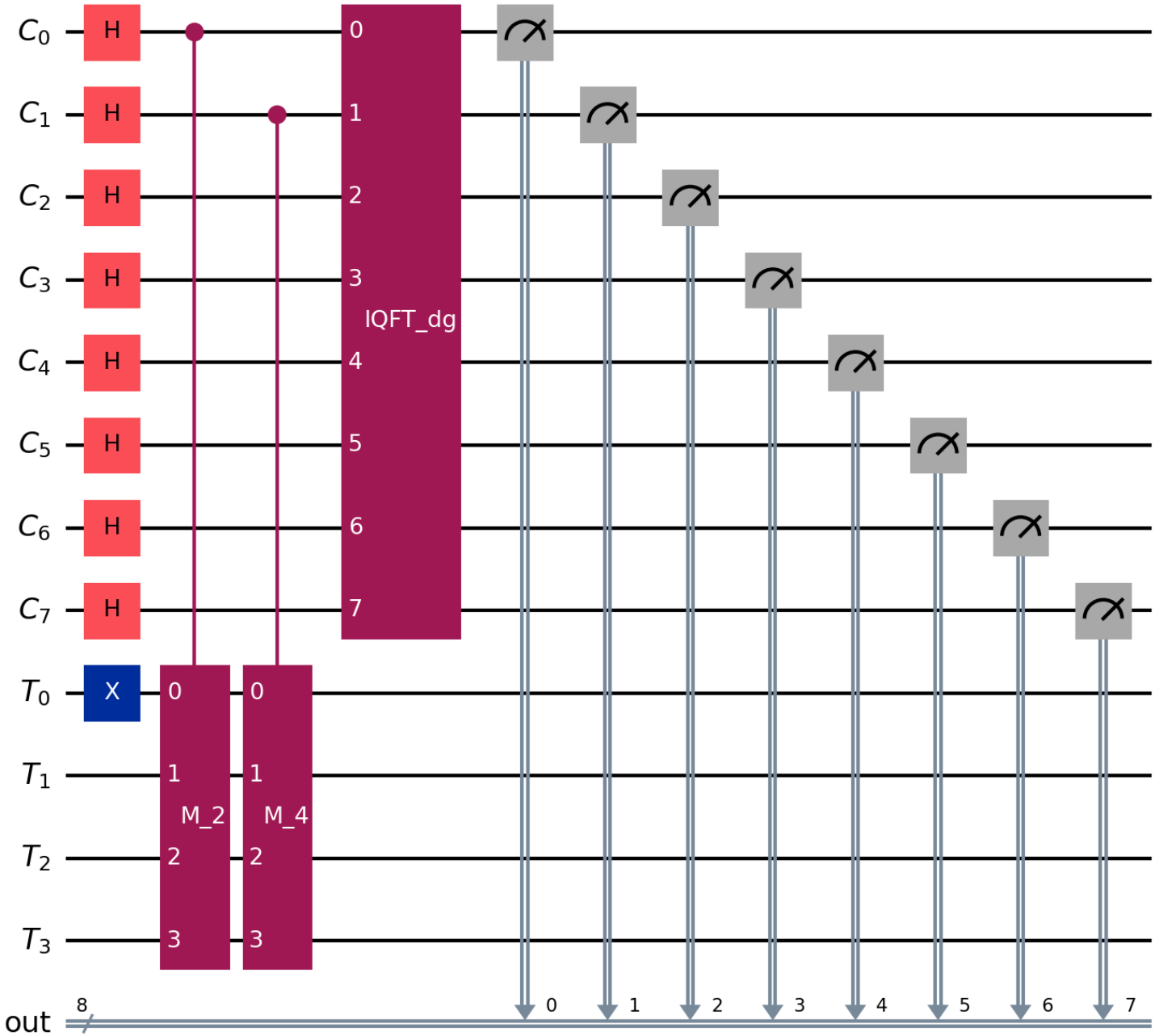


```
# Order finding problem for N = 15 with a = 2
N = 15
a = 2

# Number of qubits
num_target = floor ( log (N - 1, 2)) + 1 # for modular
        exponentiation operators
num_control = 2 * num_target # for enough precision of
        estimation

# List of M_b operators in order
k_list = range(num_control)
b_list = [a2kmodN(2, k, 15) for k in k_list]

# Initialize the circuit
control = QuantumRegister(num_control, name="C")
```

```
target = QuantumRegister(num_target, name="T")
output = ClassicalRegister(num_control, name="out")
circuit = QuantumCircuit(control, target, output)

# Initialize the target register to the state |1⟩
circuit.x(num_control)

# Add the Hadamard gates and controlled versions of the
# multiplication gates
for k, qubit in enumerate(control):
    circuit.h(k)
    b = b_list[k]
```

```
if b == 2:
    circuit.compose(
        M2mod15().control(), qubits=[qubit] + list(target),
            inplace=True )
elif b == 4:
    circuit.compose(
        M4mod15(). control(), qubits=[qubit] + list(target),
            inplace=True)
else:
    continue    # M1 is the identity operator
```

```
# Apply the inverse QFT to the control register
circuit.compose( QFT(num_control, inverse=True),
            qubits=control, inplace=True)
# Measure the control register
circuit.measure (control, output)

circuit.draw("mpl", fold=-1)
```


C_0
C_1
C_2
C_3
C_4
C_5
C_6
C_7
T_0
T_1
T_2
T_3
out
H
X
0
1
2
3
4
5
6
7
IQFT_dg
M_2
M_4
8
0
1
2
3
4
5
6
7

```
simulator = AerSimulator()
compiled_circuit = transpile(circuit, simulator)
job = simulator.run(compiled_circuit, shots=1024)
result = job.result()
counts = result.get_counts(circuit)
print(counts)
```

```
simulator = AerSimulator()
compiled_circuit = transpile(circuit, simulator)
job = simulator.run(compiled_circuit, shots=1024)
result = job.result()
counts = result.get_counts(circuit)
print(counts)
```

```
{'10000000': 249, '11000000': 282, '01000000': 239, '00000000':
254 }
```

{'10000000': 249, '11000000': 282, '01000000': 239, '00000000': 254 }

{'10000000': 249, '11000000': 282, '01000000': 239, '00000000': 254 }

Register	Output	Phase
0	10000000(bin) = 128(dec)	128/256 = 0.50
1	11000000(bin) = 192(dec)	192/256 = 0.75
2	00000000(bin) = 0(dec)	0/256 = 0.00
3	01000000(bin) = 64(dec)	64/256 = 0.25

{'10000000': 249, '11000000': 282, '01000000': 239, '00000000': 254 }

Register	Output	Phase
0	10000000(bin) = 128(dec)	128/256 = 0.50
1	11000000(bin) = 192(dec)	192/256 = 0.75
2	00000000(bin) = 0(dec)	0/256 = 0.00
3	01000000(bin) = 64(dec)	64/256 = 0.25

Phase	Fraction	Guess for r
0	0.00	0/1 1
1	0.25	1/4 4
2	0.50	1/2 2
3	0.75	3/4 4

```
def mod_mult_gate(b, N):
"""
Modular multiplication gate from permutation matrix.
"""
n = floor(log(N - 1, 2)) + 1
U = np.full ((2**n, 2**n), 0)

for x in range(N):
    U[b*x%N][x] = 1
for x in range(N, 2**n):
    U[x][x] = 1

G = UnitaryGate(U)
G.name = f"M_b"
return G
```

```
def mod_mult_gate(b, N):
"""
Modular multiplication gate from permutation matrix.
"""
n = floor(log(N - 1, 2)) + 1
U = np.full ((2**n, 2**n), 0)

for x in range(N):
    U[b*x%N][x] = 1
for x in range(N, 2**n):
    U[x][x] = 1

G = UnitaryGate(U)
G.name = f"M_b"
return G

M2_other = mod_mult_gate(2, 15)
```

M2_other

qubits: 4

2q-depth: 94

2q-size: 96

Operator counts: OrderedDict('cx': 45, 'swap': 32)

M2_other

qubits: 4

2q-depth: 94

2q-size: 96

Operator counts: OrderedDict('cx': 45, 'swap': 32)

M2mod15

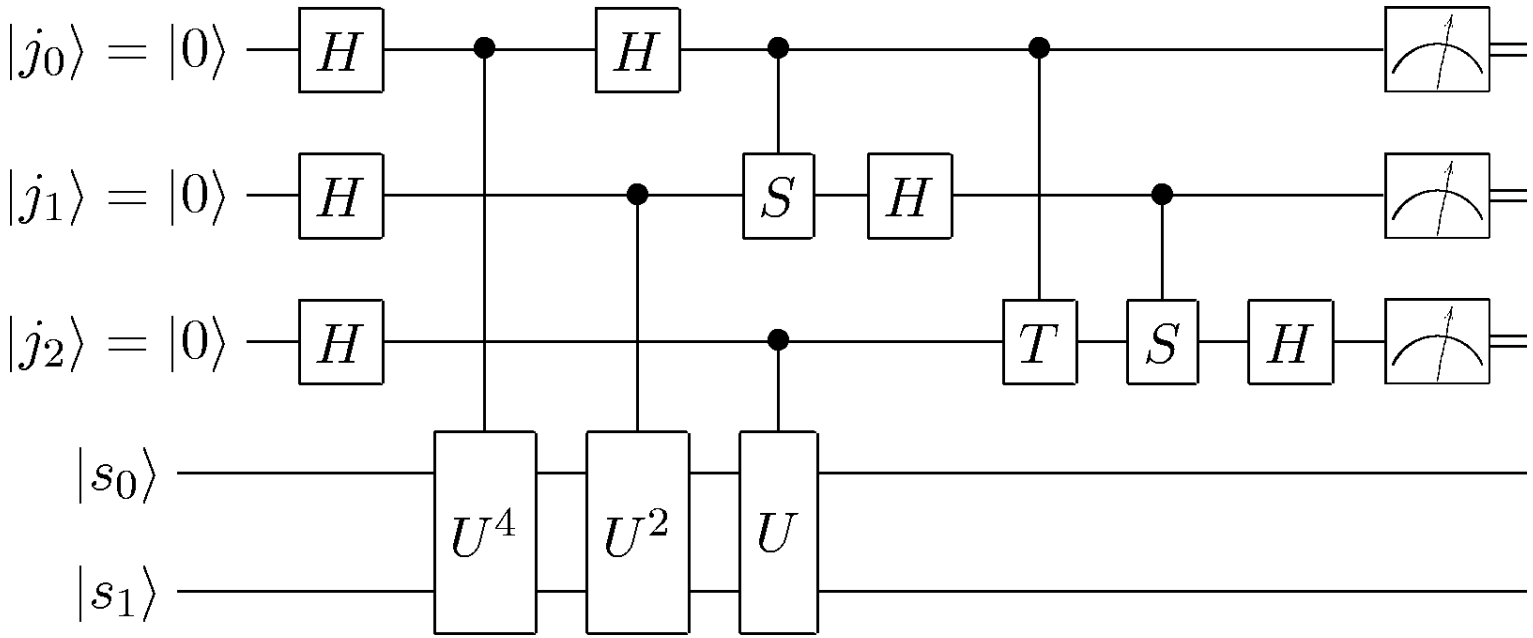
qubits: 4

2q-depth: 9

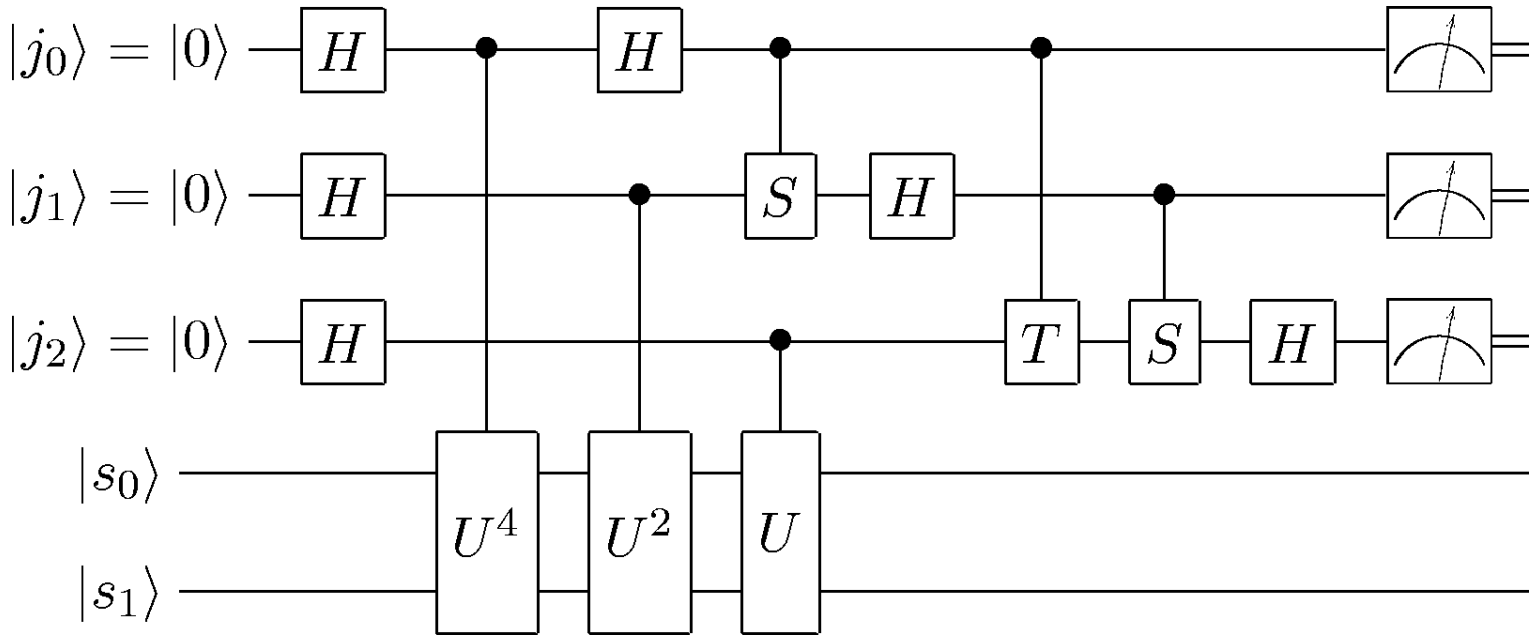
2q-size: 9

Operator counts: OrderedDict('cx': 9)

Ganto diagrama



Ganto diagrama



Diagramos gylis: $|j_2\rangle$ kubitas užbaigs darbą atlikęs 7 vartus (8, jei skaičiuosime ir matavimų vartus).

Du laiko vienetai prisidėjo, nes teko laukti, kol bus atlikta sąlyginė operacija U^2 (kubitas $|j_2\rangle$ vykdant šias dvi operacijas nedalyvauja, bet turi sulaukti, kol kiti kubitai užbaigs skaičiavimus).