Yi-Hua Chung
r09944072@csie.ntu.edu.tw

Task 3 - Find best results

1 Problem

- The following considerations apply:
  - You are the X's.
  - The matrix as a qubit and the state of the X's is —1¿ and of the O's is —0¿, of the empty cells an unknown state.
  - What are the valid combinations to win?
  - You have at most 2 turns

1.1 Method

1.1.1 Briefly Introduction

As the hint mentioned in the question, there are only remaining 4 blanks to fill, and the game will end within two rounds. Therefore, we only need to discuss at most 24 situations and find whether the situations can make 'X' win (NOTE: There are some duplicated situations).
After discussing by brutal force, we can easily find out that there are only two reasonable solutions to this problem, including $|1100\rangle$ and $|1001\rangle$. Hence, I adopt the **"Grover algorithm"** to amplify the probabilities of our willing results by designing the **oracle** and **diffuser** of the algorithm. Since there are two reasonable answers, we only need to run oracle and diffuser for one time.

1.2 Oracle

To make the states $|1100\rangle$ and $|1001\rangle$ reflect, I adopt the circuit as below:

```
def oracle (nqubits):
    qc = QuantumCircuit(nqubits)
    qc.cz(3,2) # 1, control, target
    qc.toffoli(3,0,2) # 2, control, control, target
    qc.mct([3,1,0], 2) # 3, multi-toffoli
    # 4, CCCZ, multi-control z, place on the target
    qc.h(0)
    qc.mct([3,2,1], 0)
    qc.h(0)

    qc.mct([3,2,1], 0) # 5, multi-toffoli

    # 6, CCCZ, multi-control z, place on the target
    qc.h(0)
    qc.mct([3,2,1], 0)
    qc.h(0)

    U_w = qc.to_gate()
    U_w.name = "U$_\omega$"
    return U_w
```

- For the convenience of explanation, I will represent binary as decimal representation in below. For example: $|1100\rangle$ is represented by 12, $|1111\rangle$ is represented by 15. The willing states are **9** and **12**.
- **cz(3,2):** making states include **12, 13, 14, 15** reflect
- **toffoli(3,0,2):** making exchange between (A $\leftrightarrow$ B) for: (**9** $\leftrightarrow$ **13**) and (**11** $\leftrightarrow$ **15**). After applying this gate, the reflecting states include **12, 9, 14, 11**

- **mct([3,1,0], 2), CCCNOT, for control=[3,1,0], target=2:** making exchange between(**11** $\leftrightarrow$ **15**) After applying this gate, the reflecting states include **12, 9, 14, 15**
- **CCCZ, multi-control z, for control=[3,2,1], target=0:** making reflection on **15**, the reflecting states include **12, 9, 14**
- **mct([3,2,1], 0), CCCNOT for control=[3,2,1], target=0:** making exchange between(**11** $\leftrightarrow$ **14**) After applying this gate, the reflecting states include **12, 9, 15**
- **CCCZ, multi-control z, for control=[3,2,1], target=0:** making reflection on **15**, the reflecting states include **12, 9**, which are all we demanded

After applying the circuit above, the states $|\,1001\,\rangle$ and $|\,1100\,\rangle$ are both reflected.

1.3 Diffuser

```
def diffuser(nqubits):
    qc = QuantumCircuit(nqubits)

    for qubit in range(nqubits):
        qc.h(qubit)

    for qubit in range(nqubits):
        qc.x(qubit)

    # Do multi-controlled-Z gate
    qc.h(nqubits-1)
    qc.mct(list(range(nqubits-1)), nqubits-1)
    qc.h(nqubits-1)

    for qubit in range(nqubits):
        qc.x(qubit)

    for qubit in range(nqubits):
        qc.h(qubit)

    U_s = qc.to_gate()
    U_s.name = "U$_s$"
    return U_s
```

This part is the same as the ”3-qubit example Grover Algorithm of qiskit”, as shown in the link of reference.

1.4 Quantum Circuit

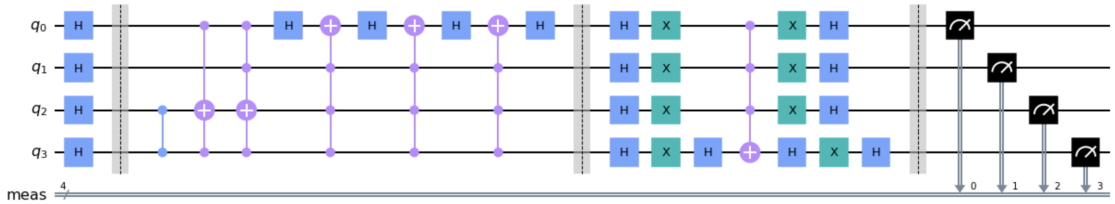

Abbildung 1: Complete quantum circuit

1.5 Result

1.5.1 Environment

Running jupyterLab on IBM Lab

1.5.2 Histogram

As shown in the plot, we can find that the output results with higher probabilities are | 1001 ⟩ and | 1100 ⟩.

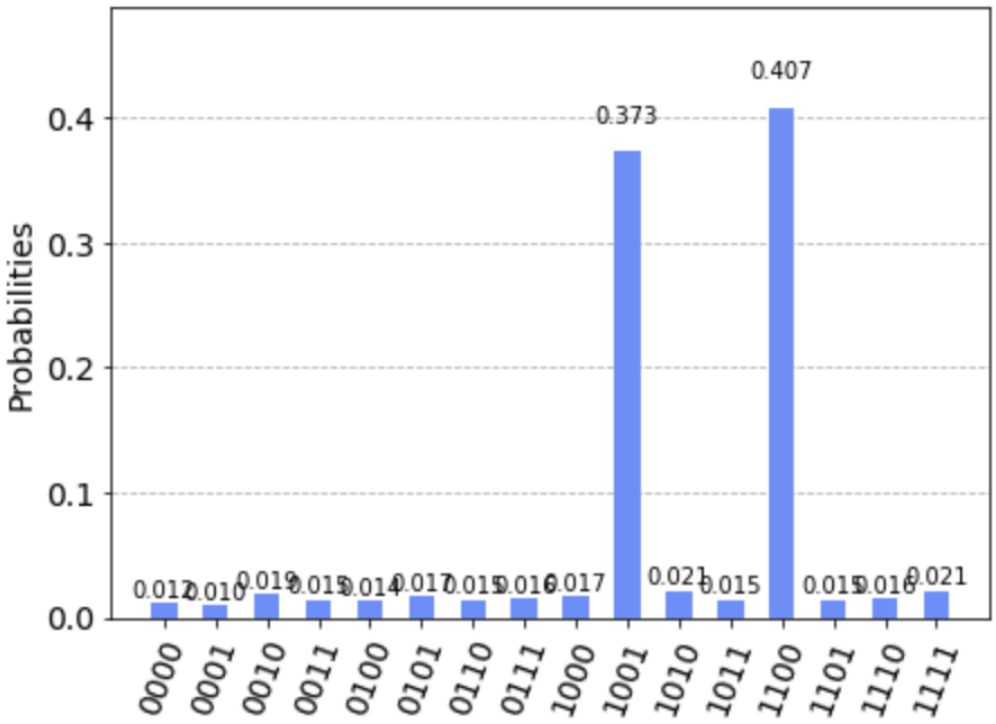

Abbildung 2: Result

2 Bonus

2.1 Method

2.1.1 Briefly Introduction

Different from the original problem, there are 5 remaining blanks that can be filled. Besides, we need to find the best decisions with **higher probability** within two rounds.
Same as solving the original problem as above, I still make a discussion at first to find out the best decisions with **higher probability** within two rounds. After discussing by brutal force, we can easily find out that there are still two solutions with the highest probabilities to win under this situation, including | 10001 ⟩, and | 11000 ⟩ (For discussion by brutal force, please ref footnote under this page).
Similarly, I adopt the **"Grover algorithm"** to amplify the probabilities of our willing results by designing the **oracle** and **diffuser** of the algorithm. Besides, same as above, there are two reasonable answers, hence, we only need to run oracle and diffuser for one time.
[1]

[1] *NOTE: If you are interested in the discussion to find the best solution, you can reference the file 'discu_brufo' under the same GitHub repository. However, this file is just the calculation process when I solved this problem which is not organized. It is only provided as a supplementary that the content is relatively messy.*

2.2 Oracle

```
# oracle: make |10001> and |11000> reflect
def oracle (nqubits):
    qc = QuantumCircuit(nqubits)
    qc.cz(4,3) # 1, control, target
    qc.toffoli(4,0,3) # 2, CCNOT, toffoli

    qc.mct([4,3,2,1], 0) # 3-1, CCCCX
    # 3-2, CCCCZ
    qc.h(0)
    qc.mct([4,3,2,1], 0)
    qc.h(0)

    qc.mct([4,2,1,0], 3) # 4-1, CCCCX
    # 4-2, CCCCZ
    qc.h(0)
    qc.mct([4,3,2,1], 0)
    qc.h(0)

    qc.mct([4,3,1], 2) # 5, CCCX
    qc.mct([4,3,2,1], 0) # 6-1, CCCCX
    # 6-2, CCCCZ
    qc.h(0)
    qc.mct([4,3,2,1], 0)
    qc.h(0)

    qc.mct([4,1,0], 3) # 7, CCCX
    qc.mct([4,3,1,0], 2) # 8-1, CCCCX
    # 8-2, CCCCZ
    qc.h(0)
    qc.mct([4,3,2,1], 0)
    qc.h(0)

    qc.mct([4,3,2], 0) # 9, CCCX
    qc.mct([4,3,2,0], 1) # 10-1, CCCCX
    # 10-2, CCCCZ
    qc.h(0)
    qc.mct([4,3,2,1], 0)
    qc.h(0)

    qc.mct([4,2,0], 1) # 11, CCCX
    qc.mct([4,2,1,0], 3) # 12-1, CCCCX
    # 12-2, CCCCZ
    qc.h(0)
    qc.mct([4,3,2,1], 0)
    qc.h(0)

    U_w = qc.to_gate()
    U_w.name = "U$_\omega$"
    return U_w
```

- During designing of the oracle of bonus part, I
- For the convenience of explanation, I will represent binary as decimal representation below. For example: $|\ 10000\ \rangle$ is represented by 16, $|\ 11000\ \rangle$ is represented by 24. The willing states are **17** and **24**.
- In this part, I first use CZ gate to reflect 8 state-vectors (24-31).
  - **cz(4,3):** At this point, only 24 of the reflected state is what we willing to be reflected.
  - **qc.toffoli(4,0,3):** Next, I use the CCX gate to exchange the amplitude of (**17** $\leftrightarrow$ **25**), (**19** $\leftrightarrow$ **27**), (**21** $\leftrightarrow$ **29**) and (**23** $\leftrightarrow$ **31**). At this time, there are a total of reflected state-vectors are **24, 17, 26, 19, 28, 21, 30, 23** with only **24 and 17** are the desired states need to reflect.

- After reflecting those needed state-vectors, there are some state-vector that don't need to be reflected. Hence, we need to reflect them back to be positive.
- The approach I took is to swap each state-vector with only three qubits are measured in $| 1 \rangle$ by using the CCCX gate to swap with their corresponding state-vectors with 4 qubits are measured in $| 1 \rangle$.
- Then exchanges the state-vector with 4 qubits are measured in $| 1 \rangle$ with $| 11111 \rangle$ by using the CCCCX gate.
- Finally, we use the CCCCZ gate to reflect $| 11111 \rangle$ as a positive state-vector. Then we can successfully make the state-vector reflected back to the positive amplitude.
- The processes in detail are shown below:
  - making the state **30** reflected back to positive, the remaining negative state-vectors are **24, 17, 26, 19, 28, 21, 23**
    - qc.mct([4,3,2,1], 0) # 3-1, CCCCX
    - qc.h(0) # 3-2, CCCCZ
    - qc.mct([4,3,2,1], 0)
    - qc.h(0)
  - making the state **23** reflected back to positive, the remaining negative state-vectors are **24, 17, 26, 19, 28, 21**
    - qc.mct([4,2,1,0], 3) # 4-1, CCCCX
    - # 4-2, CCCCZ
    - qc.h(0)
    - qc.mct([4,3,2,1], 0)
    - qc.h(0)
  - making the state **26** reflected back to positive, the remaining negative state-vectors are **24, 17, 19, 28, 21**
    - qc.mct([4,3,1], 2) # 5, CCCX
      - Making exchange between **26** $\leftrightarrow$ **30** and **27** $\leftrightarrow$ **31**. The exchange between 27 and 31 have no impact.
    - qc.mct([4,3,2,1], 0) # 6-1, CCCCX
      - Making exchange between **30** $\leftrightarrow$ **31**.
    - # 6-2, CCCCZ
    - qc.h(0)
    - qc.mct([4,3,2,1], 0)
    - qc.h(0)
      - Making reflection on **31**.
  - making the state **19** reflected back to positive, the remaining negative state-vectors are **24, 17, 28, 21**
    - qc.mct([4,1,0], 3) # 7, CCCX
      - Making exchange between **19** $\leftrightarrow$ **27** and **23** $\leftrightarrow$ **31**. The exchange between 23 and 31 have no impact.
    - qc.mct([4,3,1,0], 2) # 8-1, CCCCX
      - Making exchange between **27** $\leftrightarrow$ **31**.
    - # 8-2, CCCCZ
    - qc.h(0)
    - qc.mct([4,3,2,1], 0)
    - qc.h(0)
      - Making reflection on **31**.
  - making the state **28** reflected back to positive, the remaining negative state-vectors are **24, 17, 21**
    - qc.mct([4,3,2], 0) # 9, CCCX
      - Making exchange between **28** $\leftrightarrow$ **29** and **30** $\leftrightarrow$ **31**. The exchange between 30 and 31 have no impact.

        - qc.mct([4,3,2,0], 1) # 10-1, CCCCX
            - Making exchange between **29** ↔ **31**.
        - # 10-2, CCCCZ
        - qc.h(0)
        - qc.mct([4,3,2,1], 0)
        - qc.h(0)
            - Making reflection on **31**.

    - making the state 21 reflected back to positive, the remaining negative state-vectors are **24, 17**
        - qc.mct([4,2,0], 1) # 11, CCCX
            - Making exchange between **21** ↔ **23** and **29** ↔ **31**. The exchange between 29 and 31 have no impact.
        - qc.mct([4,2,1,0], 3) # 12-1, CCCCX
            - Making exchange between **23** ↔ **31**.
        - # 12-2, CCCCZ
        - qc.h(0)
        - qc.mct([4,3,2,1], 0)
        - qc.h(0)
            - Making reflection on **31**.

- Finally, we got the only the state-vectors **24, and 17** are reflected in negative.

2.3 Diffuser

This part is the same as the ”3-qubit example Grover Algorithm of qiskit”, as shown in the link of reference.

2.4 Quantum Circuit

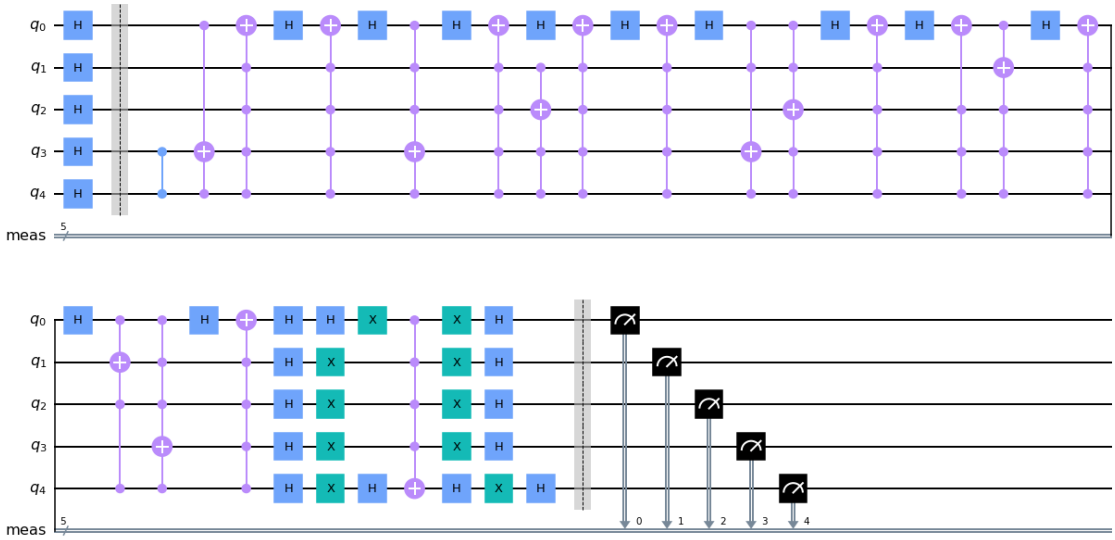

Abbildung 3: Complete quantum circuit - bonus

2.5 Result

2.5.1 Environment

Running jupyterLab on IBM Lab

2.5.2 Histogram

As shown in the plot, we can find that the output results with higher probabilities are $|\,10001\,\rangle$ and $|\,11000\,\rangle$.

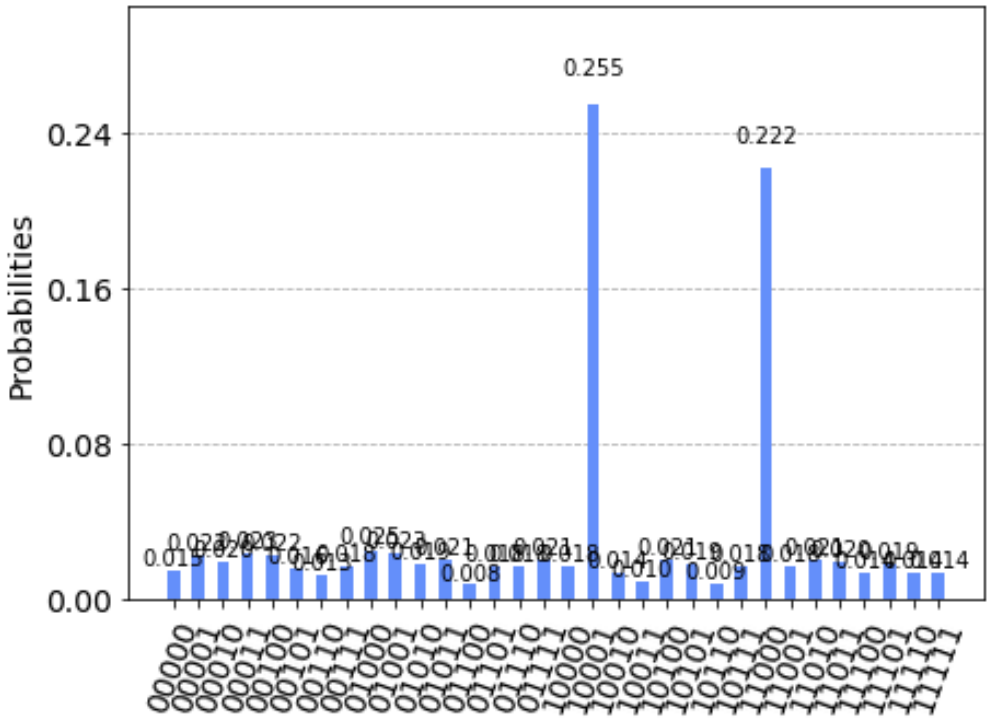

Abbildung 4: Result - bonus

3 Conclusion

In this report, we introduce the solution of **task3 of QC qosf Mentorship program cohort 5**.
This report focuses on describing how to build quantum circuits of the oracle function of Grover's Algorithm by describing everything that happens after applying each gate in the oracle.
For the program part, please see the **"qosf.ipynb"** and **"qosf_bonus.ipynb"** files under the same Github repository, which contains the complete executable program.
If there's any question, please feel free to send an email to me.

4 Reference

Qiskit - Grover's Algorithm