Qiskitテキストブック勉強会

量子情報の基礎 - エンタングルメントの動作 –

セクション第2節「超密度符号」

Hikaru Ito, 伊藤 輝

Quantum Tokyo
2023/11/07

アジェンダ

1. 振り返り
2. アルゴリズムの解説
3. 実装

振り返り｜アリスとボブ

本章ではエンタングルメントについて掘り下げています

前提

・アリスとボブは異なる場所にいる

・アリスとボブは各々1つの量子ビットを保持しており、その２つの量子ビットはエンタングル状態$|\phi^+\rangle$にある

$$|\phi^+\rangle = \frac{1}{\sqrt{2}}(|00\rangle + |11\rangle)$$

・アリスはエンタングル状態の量子ビットを利用して、ボブに情報伝達したい

エンタングルメントの動作 (qiskit.org)

振り返り｜量子テレポーテーションと超密度符号の違い

	送信したいもの	利用する道具
量子テレポーテーション	１量子状態	古典２ビット
超密度符号	古典２ビット	１量子状態

量子テレポーテーション（前回のテーマ）

- ２ビットの古典通信を使用して、１量子ビットの状態をある場所から他の場所へ送信する

超密度符号（本日のテーマ）

- １つの量子ビット通信を使用して、２つの古典ビット情報を送信する
- エンタングルを利用することにより、量子ビットが伝送する古典ビット情報量*を倍増させることができる
  - ➢ ***ホレボ限界**
    １量子ビットを送信して１ビットより多くの古典情報を伝達することはできない

エンタングルメントの動作 (qiskit.org)

*Quantum Computation and Quantum Information; Michael A. Nielsen , Isaac L. Chuang

超密度符号

問題

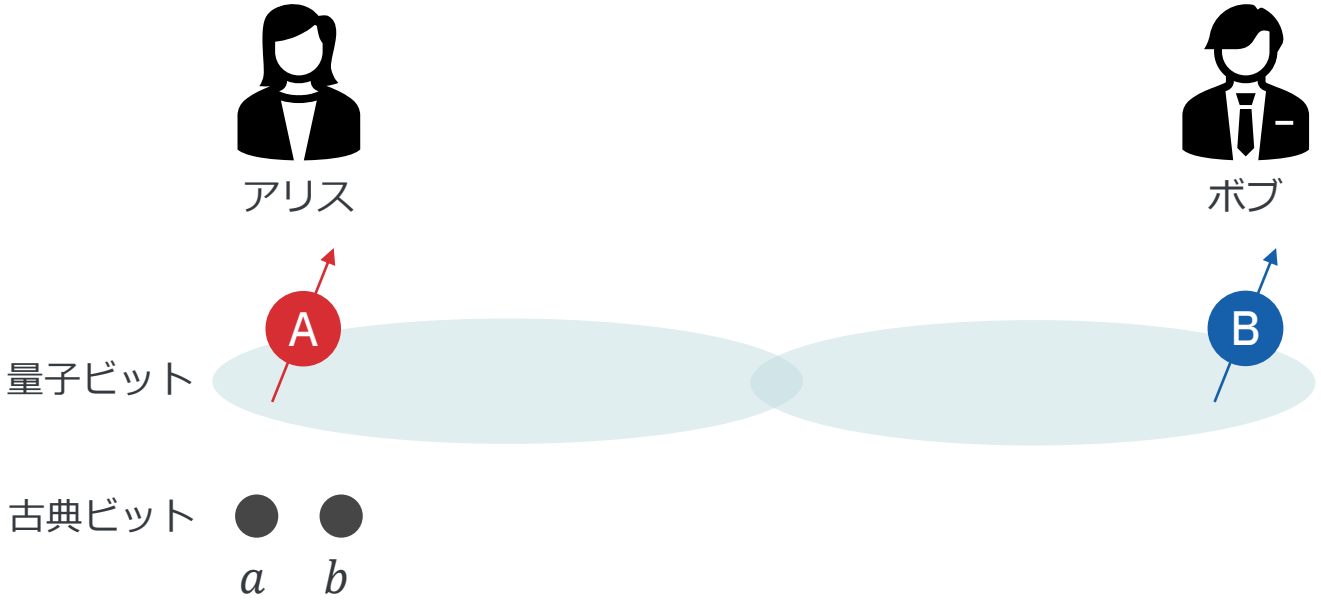
- 1つの量子ビットを使って、2つの古典ビット（$a,\ b$）の情報を伝送したい

前提

- 送信者（アリス）が量子ビットA、受信者（ボブ）が量子ビットB を持っている
- 量子ビットA・Bはエンタングルしており、**ベル状態** $|\phi^+\rangle$ にある

$$|\phi^+\rangle = \frac{1}{\sqrt{2}}(|00\rangle + |11\rangle)$$

- アリスはボブに2つの古典ビット（$a,\ b$）を送信したいと思っており、1 qubit を送信することでこれを達成しようとしている



超密度符号

問題

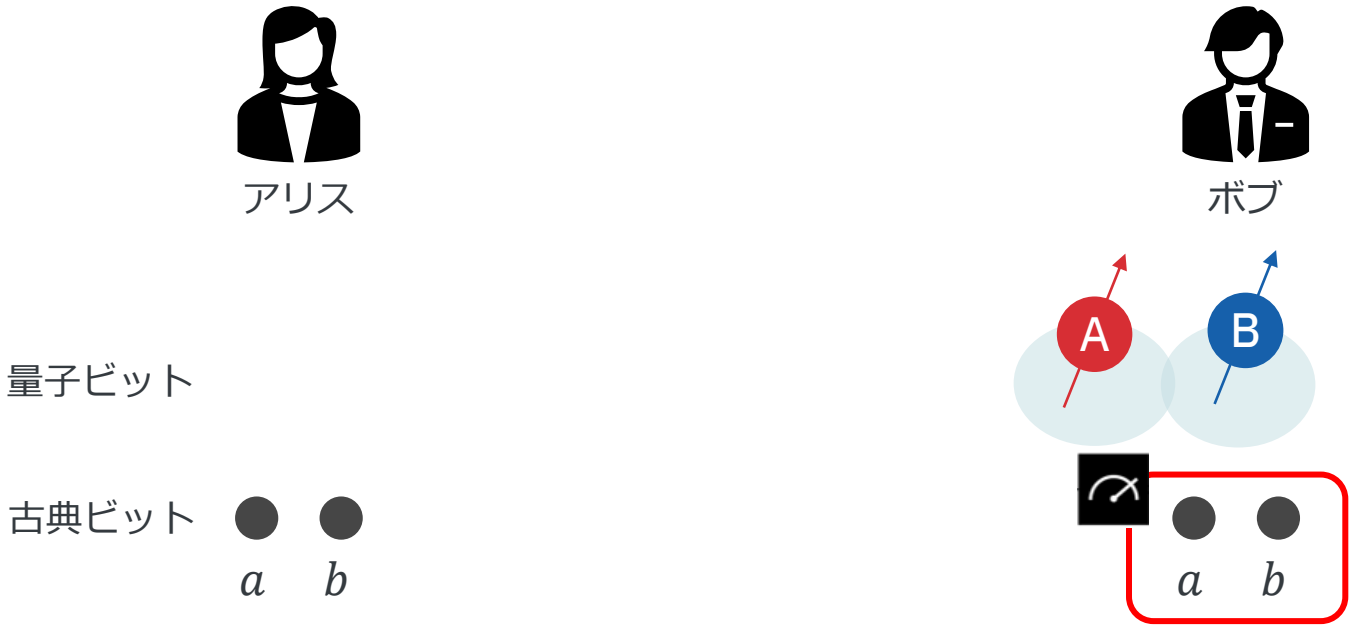
- 1つの量子ビットを使って、2つの古典ビット（$a,\ b$）の情報を伝送したい

前提

- 送信者（アリス）が量子ビットA、受信者（ボブ）が量子ビットB を持っている
- 量子ビットA・Bはエンタングルしており、ベル状態 $|\phi^+\rangle$ にある

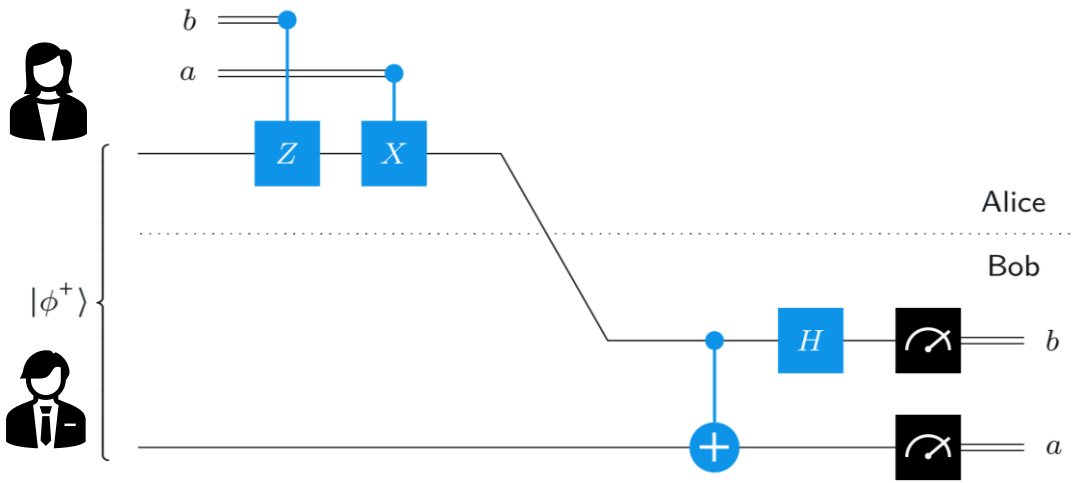
$$|\phi^+\rangle = \frac{1}{\sqrt{2}}(|00\rangle + |11\rangle)$$

- アリスはボブに**2つの古典ビット（$a,\ b$）を送信したい**と思っており、1 qubit を送信することでこれを達成しようとしている



プロトコル

- 1つのqubit で2つの古典ビットを伝送する
- 必要なもの：古典ビット(a, b), 量子ビット(A, B)



Step 1 ：アリスが行うことは次のとおり

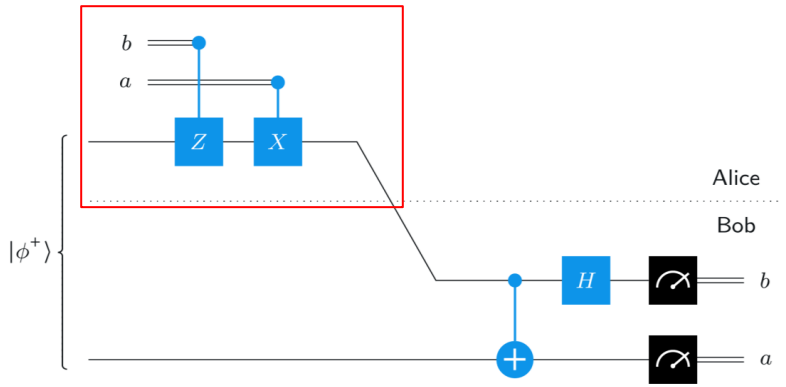
1. 古典ビットの状態確認
   - $b = 1$の場合、qubit A に Zゲートを実行
   - $a = 1$の場合、qubit A に Xゲートを実行
2. アリスは自分のqubit A をボブに送信

Step 2 ：ボブが行うことは次のとおり

3. 量子ビットの操作
   - CNOT ゲートを実行
   - qubit A にアダマールゲートを適用
4. 量子ビットを測定し、古典ビットの情報取得
   - qubit A を測定してbを取得
   - qubit B を測定してaを取得

プロトコル｜Step 1 ｜アリスの操作

1. 古典ビットの状態確認
   - $b = 1$の場合、qubit A に Zゲートを実行
   - $a = 1$の場合、qubit A に Xゲートを実行
2. アリスは自分のqubit A をボブに送信



アリスが送りたい古典ビット情報	2量子ビット初期状態
$ab = 00$	$\|\phi^+\rangle = \frac{1}{\sqrt{2}}(\|0_B 0_A\rangle + \|1_B 1_A\rangle)$
$ab = 01$	
$ab = 10$	
$ab = 11$	

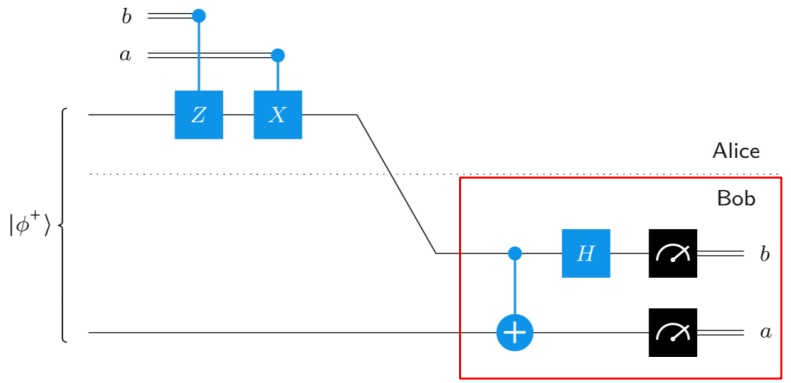
アリスの操作	アリスがボブに送信する量子ビット情報
$(\mathbb{I}_B \otimes \mathbb{I}_A)\|\phi^+\rangle$	$\|\phi^+\rangle = \frac{1}{\sqrt{2}}(\|00\rangle + \|11\rangle)$
$(\mathbb{I}_B \otimes \mathbb{Z}_A)\|\phi^+\rangle$	$\|\phi^-\rangle = \frac{1}{\sqrt{2}}(\|00\rangle - \|11\rangle)$
$(\mathbb{I}_B \otimes \mathbb{X}_A)\|\phi^+\rangle$	$\|\psi^+\rangle = \frac{1}{\sqrt{2}}(\|01\rangle + \|10\rangle)$
$(\mathbb{I}_B \otimes \mathbb{X}_A\mathbb{Z}_A)\|\phi^+\rangle$	$\|\psi^-\rangle = \frac{1}{\sqrt{2}}(\|01\rangle - \|10\rangle)$

例） $ab = 10$のとき、アリスの操作は下記のようになる

$$(\mathbb{I}_B \otimes \mathbb{X}_A)\,|\phi^+\rangle = \begin{pmatrix} 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix} \begin{pmatrix} 1 \\ 0 \\ 0 \\ 1 \end{pmatrix} = \frac{1}{\sqrt{2}} \begin{pmatrix} 0 \\ 1 \\ 1 \\ 0 \end{pmatrix} = \frac{1}{\sqrt{2}}(|01\rangle + |10\rangle) = |\psi^+\rangle$$

プロトコル｜Step 2 ｜ボブの操作

3. 量子ビットの操作
   - CNOT ゲートを実行
   - qubit A にアダマールゲートを適用
4. 量子ビットを測定し、古典ビットの情報取得
   - qubit A を測定してbを取得。qubit B を測定してaを取得



ボブが受信した情報	ボブの操作1	出力1	ボブの操作2	出力2	復号後の古典ビット
$\|\phi^+\rangle = \frac{1}{\sqrt{2}}(\|00\rangle + \|11\rangle)$	$CX_{\mathrm{A}\to\mathrm{B}}$	$\frac{1}{\sqrt{2}}\|0\rangle_B(\|0\rangle_A + \|1\rangle_A)$	$\mathbb{I}_{\mathrm{B}}\otimes H_{\mathrm{A}}$	$\|0_{\mathrm{B}}0_{\mathrm{A}}\rangle$	$ab = 00$
$\|\phi^-\rangle = \frac{1}{\sqrt{2}}(\|00\rangle - \|11\rangle)$		$\frac{1}{\sqrt{2}}\|0\rangle_B(\|0\rangle_A - \|1\rangle_A)$		$\|0_{\mathrm{B}}1_{\mathrm{A}}\rangle$	$ab = 01$
$\|\psi^+\rangle = \frac{1}{\sqrt{2}}(\|01\rangle + \|10\rangle)$		$\frac{1}{\sqrt{2}}\|1\rangle_B(\|1\rangle_A + \|0\rangle_A)$		$\|1_{\mathrm{B}}0_{\mathrm{A}}\rangle$	$ab = 10$
$\|\psi^-\rangle = \frac{1}{\sqrt{2}}(\|01\rangle - \|10\rangle)$		$\frac{1}{\sqrt{2}}\|1\rangle_B(\|1\rangle_A - \|0\rangle_A)$		$\|1_{\mathrm{B}}1_{\mathrm{A}}\rangle$	$ab = 11$

例） $ab = 10$のとき、ボブの操作は下記のようになる

1. $CX_{\mathrm{A}\to\mathrm{B}}|\psi^+\rangle = \frac{1}{\sqrt{2}}(|11\rangle + |10\rangle)$
2. $(\mathbb{I}_{\mathrm{B}}\otimes H_{\mathrm{A}})\frac{1}{\sqrt{2}}(|11\rangle + |10\rangle) = \frac{1}{2}\{|10\rangle - |11\rangle + |10\rangle + |11\rangle\} = |10\rangle$

実装１ | a = 1, b = 0 を送信する

```
from qiskit import QuantumCircuit, QuantumRegister, ClassicalRegister
```

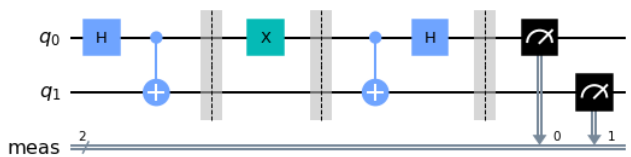
```
a = "1"
b = "0"
```

```
protocol = QuantumCircuit(2)

# Prepare ebit used for superdense coding
protocol.h(0)
protocol.cx(0, 1)
protocol.barrier()

# Alice's operations
if b == "1":
    protocol.z(0)
if a == "1":
    protocol.x(0)
protocol.barrier()

# Bob's actions
protocol.cx(0, 1)
protocol.h(0)
protocol.measure_all()

protocol.draw()
```

qubit A, Bを初期状態$|\phi^+\rangle$にする

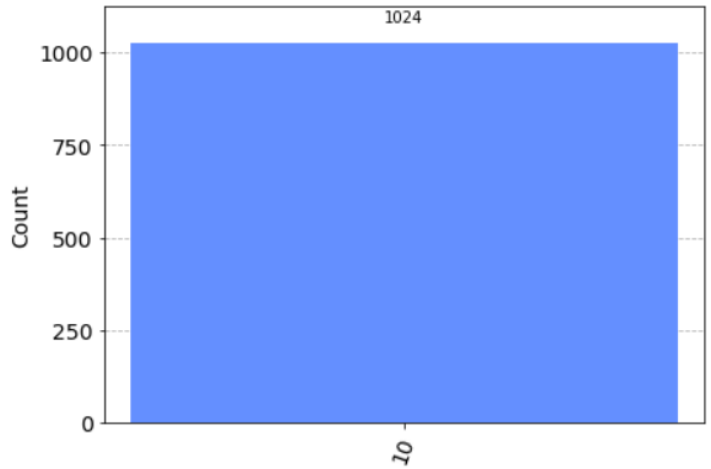
アリスの操作

ボブの操作




```
from qiskit import Aer, transpile
from qiskit.visualization import plot_histogram

# Transpile for simulator
simulator = Aer.get_backend('aer_simulator')
circ = transpile(protocol, simulator)

# Run and get counts
result = simulator.run(circ).result()
counts = result.get_counts(circ)
plot_histogram(counts, title='Bell-State counts')
```

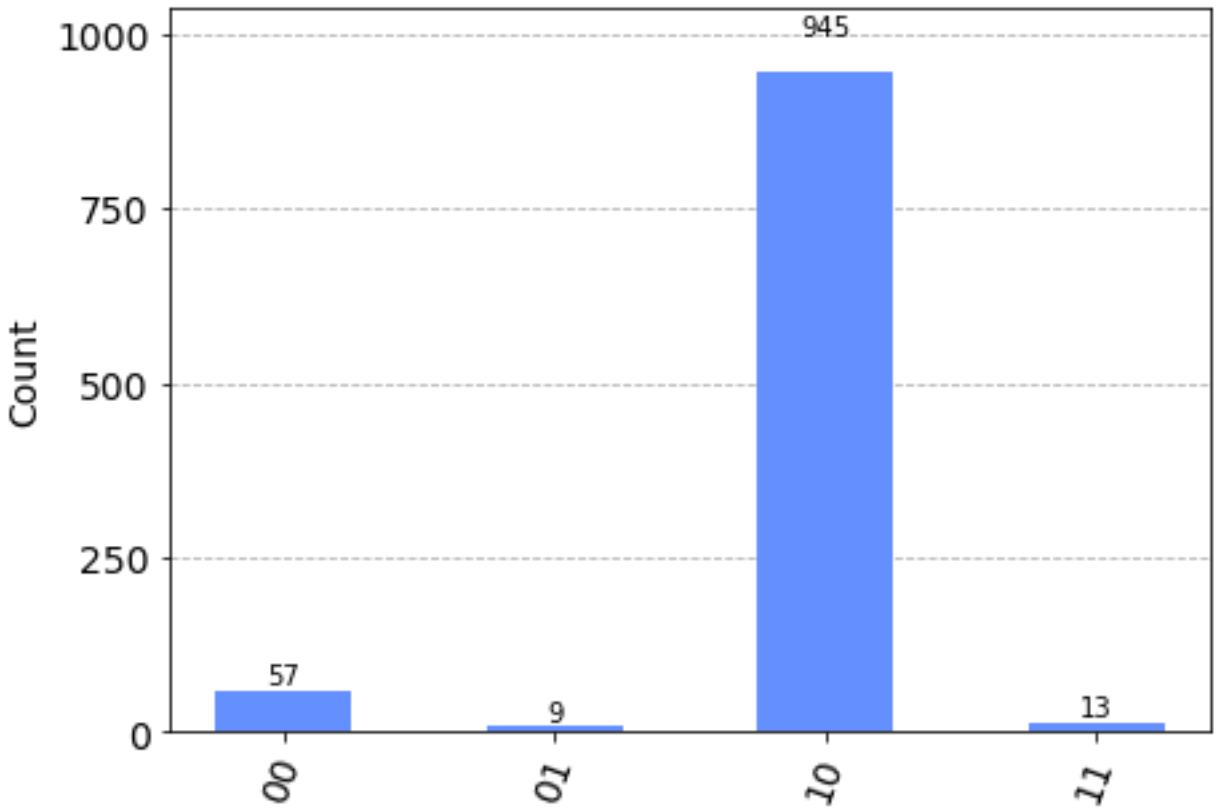
シミュレータでの実行結果



実装１ | a = 1, b = 0 を送信する | ハードウェアでの実行

```
# 実機での動作確認
# アカウント情報をロードします
from qiskit import *
from qiskit_ibm_provider import IBMProvider

# MY_API_TOKEN = '
# IBMProvider.save_account(token=MY_API_TOKEN)
provider = IBMProvider()

# 使うデバイスを指定します
real_backend = provider.get_backend('ibm_nairobi')#ibm_perth,  ibm_lagos

# 実機のバックエンドでの実行に最適な回路に変換します
from qiskit import transpile
qc_compiled = transpile(circuits=circ, backend=real_backend)

# 動的回路「dynamic=True」を入れて実行します
job = real_backend.run(qc_compiled, shots=1024, dynamic=True)

# ジョブの実行状態を確認します
from qiskit.tools.monitor import job_monitor
print(f"Job ID: {job.job_id()}")
job_monitor(job)

# 結果を確認します
real_result= job.result()
print(real_result.get_counts(circ))
plot_histogram(real_result.get_counts(circ))
```

```
Job ID: cmjpvbx4acc0008hzvpg
Job Status: job has successfully run
{'00': 57, '01': 9, '10': 945, '11': 13}
```



実装２｜ランダムな a, b を送信する

```
from qiskit import QuantumCircuit

rbg = QuantumRegister(1, "randomizer")
ebit0 = QuantumRegister(1, "qubit A")
ebit1 = QuantumRegister(1, "qubit B")

Alice_a = ClassicalRegister(1, "alice_a")
Alice_b = ClassicalRegister(1, "alice_b")

test = QuantumCircuit(rbg, ebit0, ebit1, Alice_b, Alice_a)


# Use the 'randomizer' qubit twice to generate Alice's bits a and b.
test.h(rbg)
test.measure(rbg, Alice_a)
test.h(rbg)
test.measure(rbg, Alice_b)
test.barrier()

# Initialize the ebit
test.h(ebit0)
test.cx(ebit0, ebit1)
test.barrier()

# Now the protocol runs, starting with Alice's actions, which depend on her bits.
with test.if_test((Alice_b, 1), label="Z"):
    test.z(ebit0)
with test.if_test((Alice_a, 1), label="X"):
    test.x(ebit0)
test.barrier()

# Bob's actions
test.cx(ebit0, ebit1)
test.h(ebit0)
test.barrier()

Bob_a = ClassicalRegister(1, "bob_a")
Bob_b = ClassicalRegister(1, "bob_b")
test.add_register(Bob_b)
test.add_register(Bob_a)
test.measure(ebit1, Bob_a)
test.measure(ebit0, Bob_b)

test.draw('mpl')
```

Hゲートを使って
ランダムな古典ビット a, b を発生

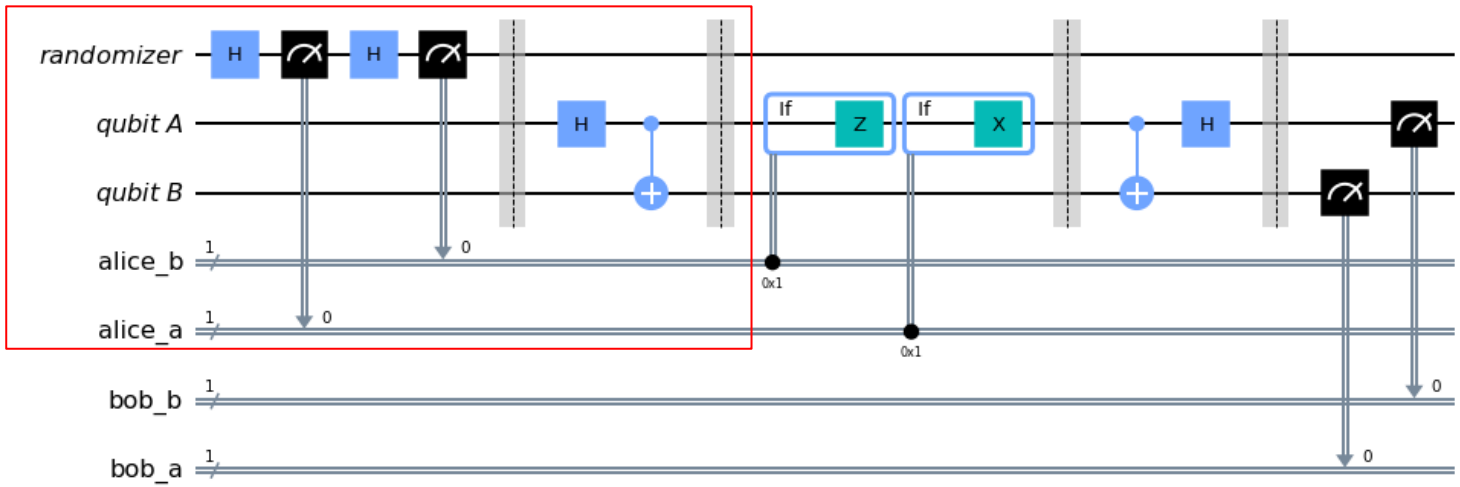
qubit A, Bを初期状態$|\phi^+\rangle$にする

アリスの操作
b=1の場合 qubit A に Zゲート実行
a=1の場合 qubit A に Xゲート実行

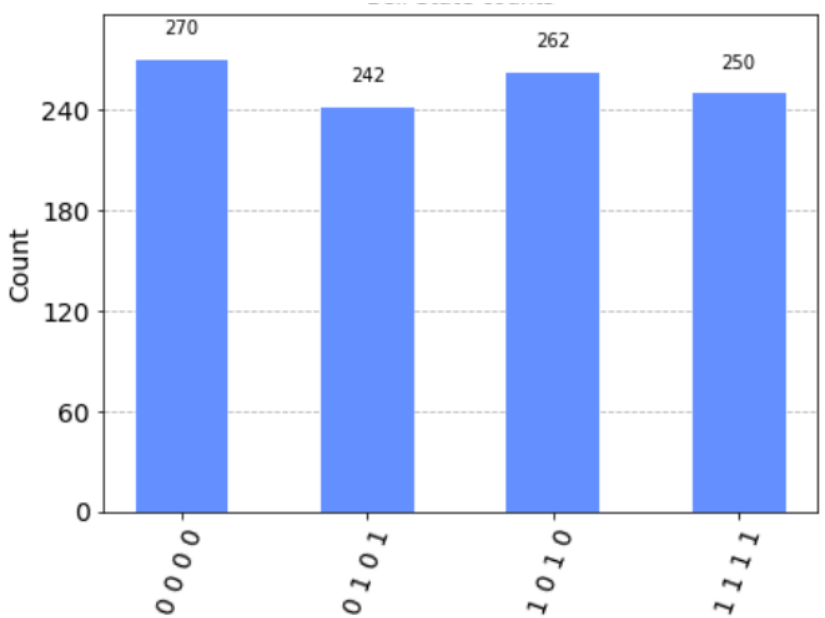
ボブの操作
CNOT ゲートを実行
qubit A にアダマールゲートを実行

測定
qubit A を測定してbを取得
qubit B を測定してaを取得

```
# Transpile for simulator
simulator = Aer.get_backend('qasm_simulator')
job_sim = simulator.run(test)
result_sim = job_sim.result()

# Run and get counts
counts = result_sim.get_counts()
plot_histogram(counts, title='Bell-State counts')
```



シミュレータでの実行結果



実装２｜ランダムな a, b を送信する｜ハードウェアでの実行

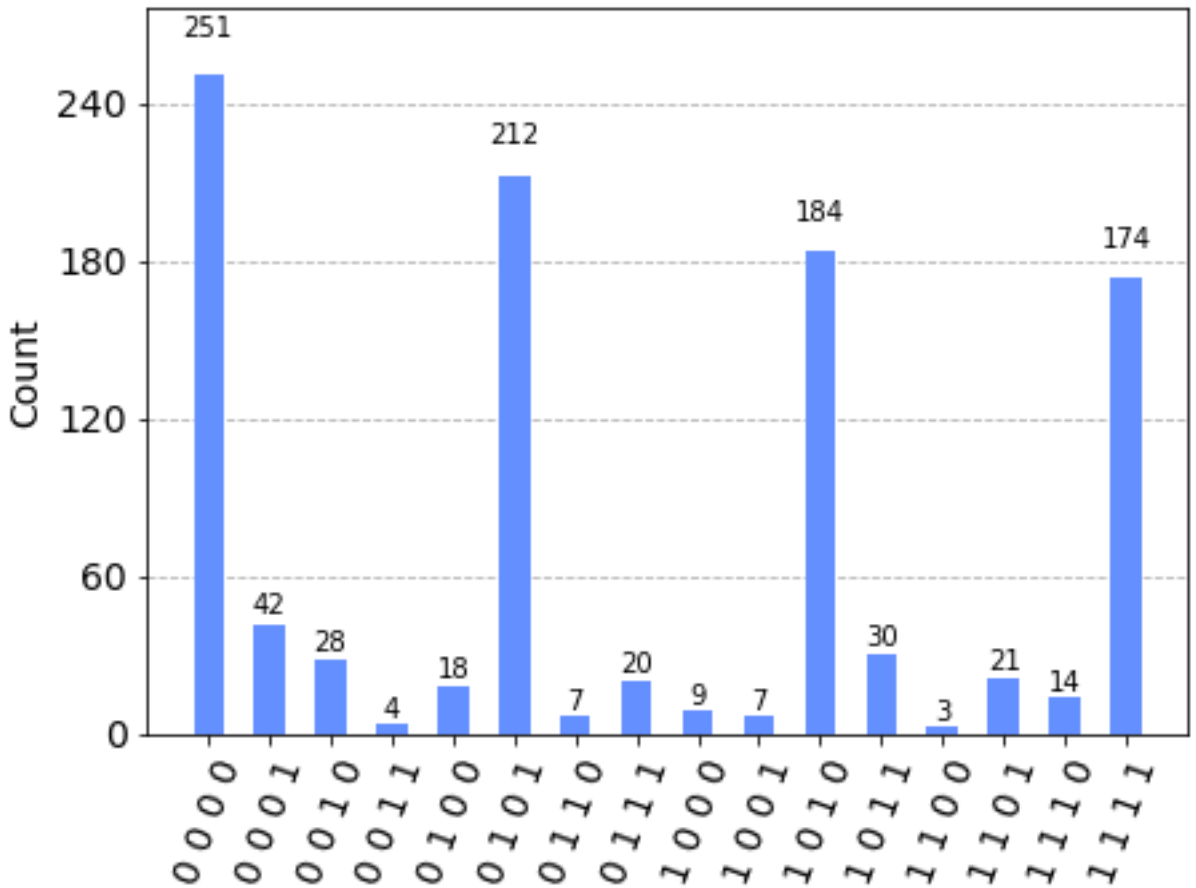
```
# アカウント情報をロードします
from qiskit import *
from qiskit_ibm_provider import IBMProvider

# MY_API_TOKEN = '
# IBMProvider.save_account(token=MY_API_TOKEN)
provider = IBMProvider()
```

```
# 使うデバイスを指定します
real_backend = provider.get_backend('ibm_nairobi')#ibm_perth,  ibm_lagos

# 実機のバックエンドでの実行に最適な回路に変換します
from qiskit import transpile
qc_compiled = transpile(circuits=test, backend=real_backend)

# 動的回路「dynamic=True」を入れて実行します
job = real_backend.run(qc_compiled, shots=1024, dynamic=True)

# ジョブの実行状態を確認します
from qiskit.tools.monitor import job_monitor
print(f"Job ID: {job.job_id()}")
job_monitor(job)
```

```
Job ID: cmjp3qe5mym0008dvmfg
Job Status: job is queued (None)
```

```
# 結果を確認します
real_result= job.result()
print(real_result.get_counts(test))
plot_histogram(real_result.get_counts(test))
```

IBM Nairobi での実行結果



まとめ

超密度符号について

- 量子ビットのエンタングルメントを活用し、
  古典ビット情報を量子情報へ符号化することで、効率よく古典情報を転送できる。

次回はCHSHがテーマです！

Back Up


Quantum
IBM Q
Tokyo