量子計算入門

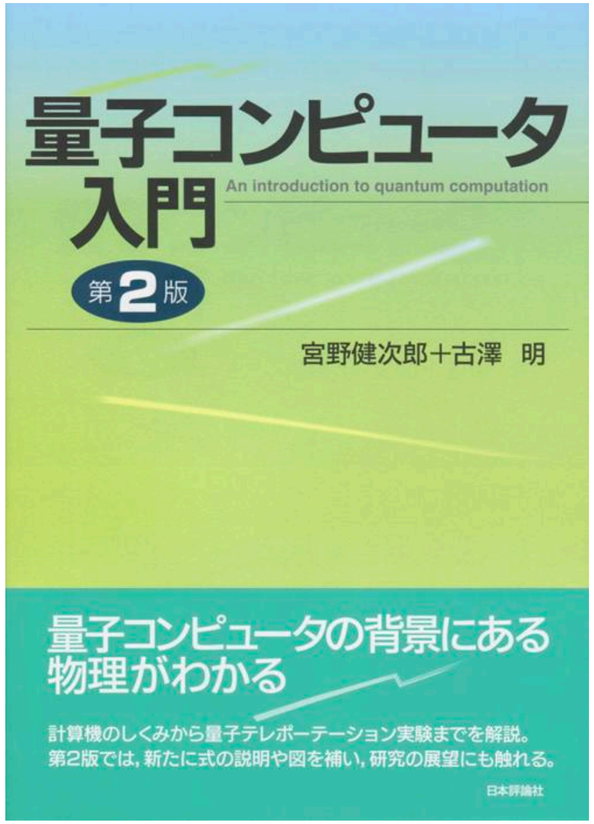
kaage(@ageprocpp) 2020/8/28

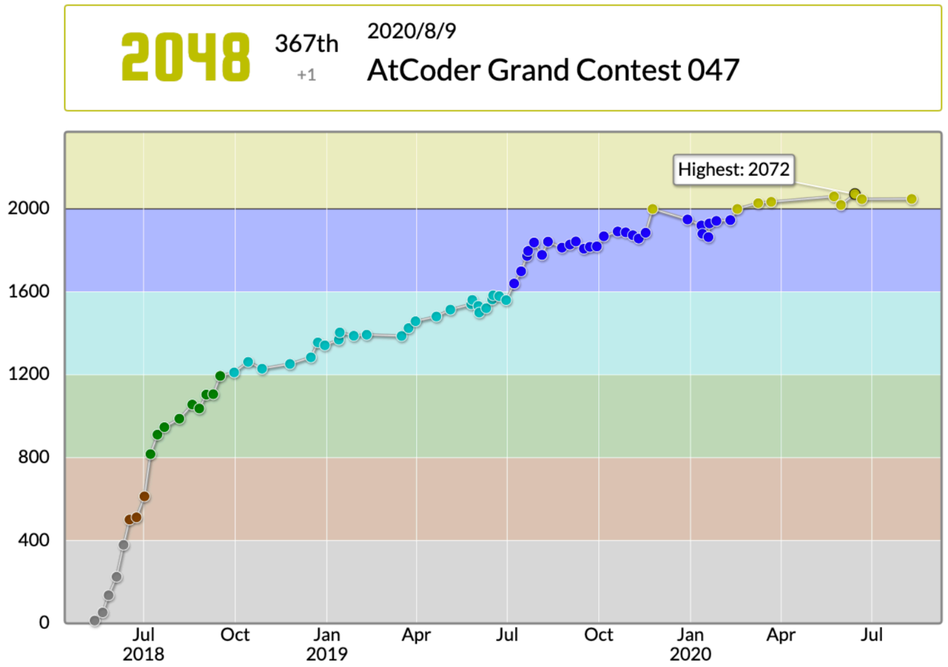
自己紹介

筑波大附属駒場中3年

AtCoder 黄

文化祭が実質中止になって泣きそうです、今





量子コンピュータとは

量子ビットを使って、離散フーリエ変換を $O(\log^2 N)$ でできたり（QFT）、

どう見ても $O(2^N)$ かかりそうな計算を $O(N)$ でやったり（Deutsch-Jozsa

Algorithm)、$O(N)$ かかりそうな計算を $O(\sqrt{N})$ でできたり（Grover's

Algorithm）するすごいコンピュータ。
様々な問題によって、実機が存在はするけど実用的な計算には至っていない。（一応 Shor's Algorithm は 15 を実際に素因数分解したらしい）

量子ビットとは

量子ビットは、基底 $|0\rangle, |1\rangle$ と複素数 c_0, c_1 を用いて、

$$|\psi\rangle = c_0|0\rangle + c_1|1\rangle \ (c_0, c_1 \in \mathbb{C}, |c_0|^2 + |c_1|^2 = 1)$$

と表される**状態**（光子・スピンなどの物理的実体で表現される）

古典コンピュータでのビット（古典ビット）と異なる点がいくつかある

・観測による波束収縮
・量子もつれ
・クローニング不可能性

量子ビットとは-観測による波束収縮

量子ビット $|\psi\rangle = c_0|0\rangle + c_1|1\rangle$ は、「観測」を行うと $|0\rangle, |1\rangle$ のどちらかにそれぞれ確率 $|c_0|^2, |c_1|^2$ で遷移してしまう。

観測以外で量子ビットに行える操作は限られており（後述）、c_0, c_1 を途中で知ることはできない。
観測を一度行ってしまうと、その後何度観測しても状態は変わらない。
（が、そもそも量子ビットを破壊せずに観測する方法が存在はするもののまだ確立・実用化されていない）
うまく使えばとても便利な性質だが、これが量子計算の複雑さの一つの要因である。

量子ビットとは-量子もつれ

たとえば、ふたつの量子ビットが独立で双方

$$\frac{1}{\sqrt{2}}(|0\rangle + |1\rangle)$$

である場合、この2量子ビットの状態はまとめて直積で表されて、

$$\frac{1}{\sqrt{2}}(|0\rangle + |1\rangle) \otimes \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle) = \frac{1}{2}(|00\rangle + |01\rangle + |10\rangle + |11\rangle)$$

と書ける。

つまり、4つの状態を均等な確率で観測する可能性がある。

量子ビットとは-量子もつれ

しかし、「もつれた」状態の2量子ビットでは、たとえばふたつまとめて次のように表されることがありうる。

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

この状態では、どちらのビットも観測するまでどうなるかは分からないが、仮に片方観測して $|0\rangle$ となれば、もう片方も必ず同じになる。

つまり、片方を観測してしまえばもう片方のビットの振る舞いを決定することができる。
これは古典的な物理では決して起こらないことで、このような「もつれ」の状態を使うと効率的に計算を行うことができる。

量子ビットとは-量子もつれ

例 Microsoft Q# Coding Contest Summer 2018-B1

N 量子ビットが与えられる。これらは、次の2つの状態のどちらかである

$$|0\cdots0\rangle$$

$$\frac{1}{\sqrt{N}}(|1000\cdots0\rangle + |0100\cdots0\rangle + |0010\cdots0\rangle + \cdots + |0000\cdots1\rangle)$$

与えられたものがどちらか判定せよ。

これは、全部順番に観測してみて全部 $|0\rangle$ かどうかを見るだけで解ける。

量子ビットとは-クローニング不可能性

量子ビットは観測すると収縮して壊れてしまうため、ある量子ビットの状態を覗き見たり、量子ビットをコピーしたりすることはできない。
これは暗号的な面でも、扱いやすさの面でも重要になる。
なお、ある量子ビットの状態を、古典ビットを通じて他の量子ビットに移し替えることはできる。（量子テレポーテーション）
また、量子ビットどうしのスワップもできる。
一方、ユニタリ変換の枠を超えるような、あるふたつの量子ビットが等しいかの判定はできないし、観測できるのも一回だけなので、慣れないうちは気をつけて扱い方を見ないといけない。

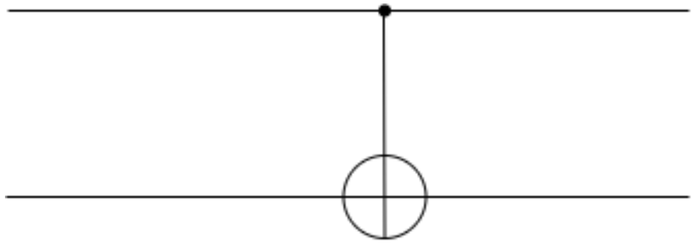
量子ビットに行える操作

量子ビットを通せるゲート（行える操作）のうち、重要なものを挙げる。

・制御 NOT(CNOT) ゲート

2量子ビットを入力にとり、1つ目のビット（制御ビット）が $|1\rangle$ のときのみ、2つ目のビットの NOT をとる。1つ目のビットは変化しない。

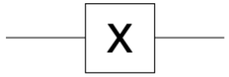
制御 NOT ゲートと、ユニタリ行列による変換で、すべての可能な操作が行えることが知られている。



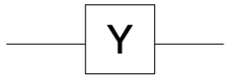
・パウリゲート

X, Y, Z の3種類があり、それぞれ以下の変換をする。

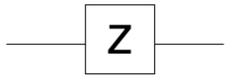
X: $|0\rangle \to |1\rangle, |1\rangle \to |0\rangle$



Y: $|0\rangle \to i|1\rangle, |1\rangle \to -i|0\rangle$



Z: $|0\rangle \to |0\rangle, |1\rangle \to -|1\rangle$



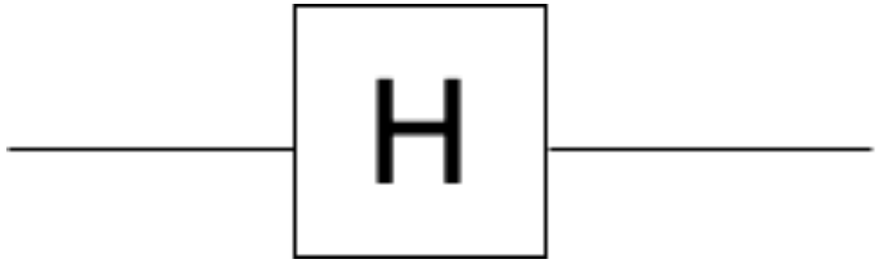
量子ビットに行える操作

・アダマール(Hadamard)ゲート

一番よく使う（要出典）。

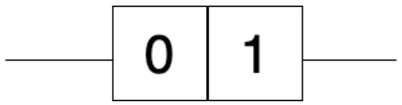
$$|0\rangle \rightarrow \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle)$$

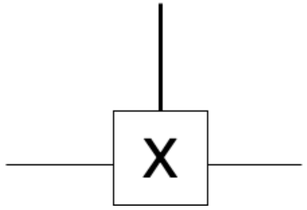
$$|1\rangle \rightarrow \frac{1}{\sqrt{2}}(|0\rangle - |1\rangle)$$



・古典ゲート、古典制御ゲート

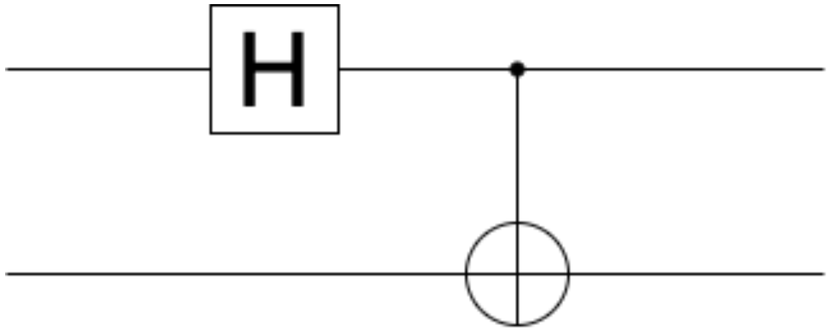
それぞれ、測定をしたり、CNOT ゲートみたいな感じで古典情報を用いて変換を制御したりする。





量子回路について

量子回路を読むときは、量子もつれに気をつけながら読まないといけない。



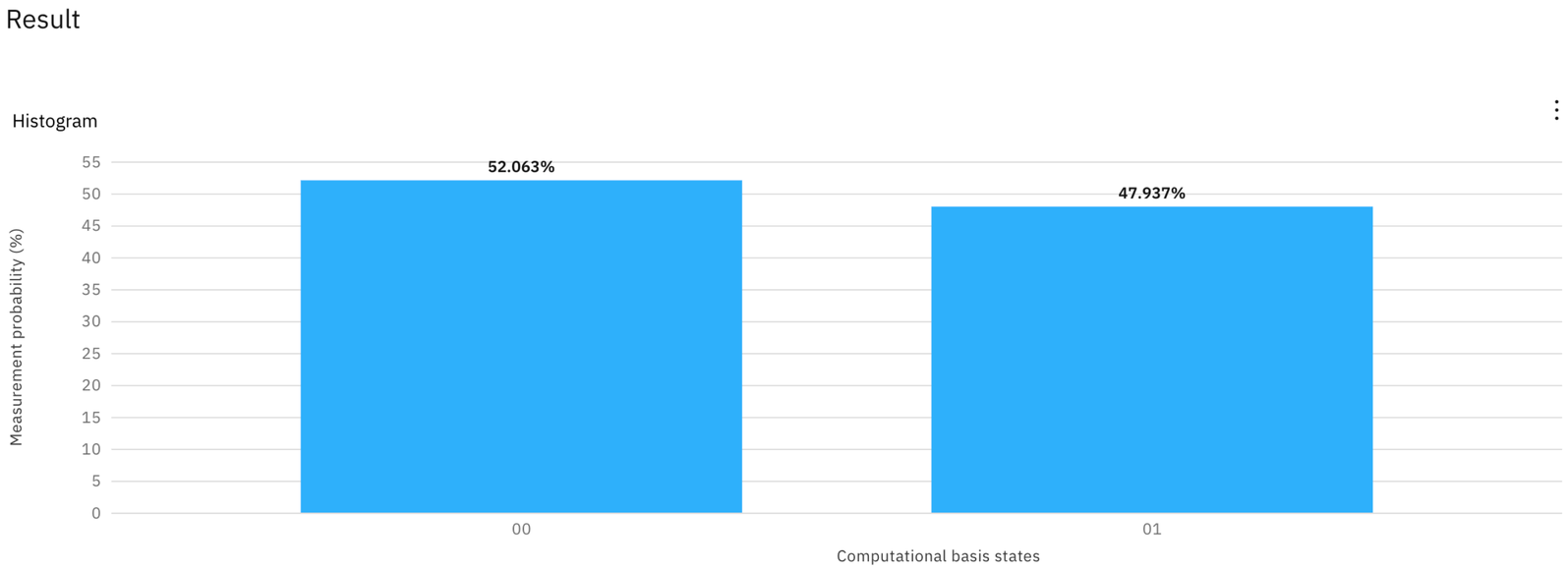
たとえば、この回路にそれぞれ $|0\rangle$ を通したあと、下のビットを観測すると、量子もつれの影響で上のビットも収縮してしまう。
このように、CNOT ゲートの制御ビットは、通った時点で変化しなかったとしても、もつれを生んで他の観測によって結果的に変化してしまう場合があることに注意しないといけない。

量子コンピュータのエラー

上のようなゲートをたくさん使って、高速に素因数分解したり、離散フーリエ変換したりするような スゴイ=アルゴリズム を実装することができる
が、現実の量子コンピュータは、まだそのようなアルゴリズムを使える段階とはいえない
CNOT に使える量子ビットの組が少なかったり、量子ビットが途中でエラーを起こして反転してしまったりする。
仮に精度が十分に高まっても、エラーを起こす可能性は否定できず、これを計算にそのまま使うのは問題があるといえる。

量子コンピュータのエラー

ロンドンにある IBM の量子コンピュータでアダマール変換した量子ビットを観測してみたが、画像のように 2% 以上の誤差が出た。
Grover’s Algorithm とかは、誤差のせいでもはや動かない（らしい）。



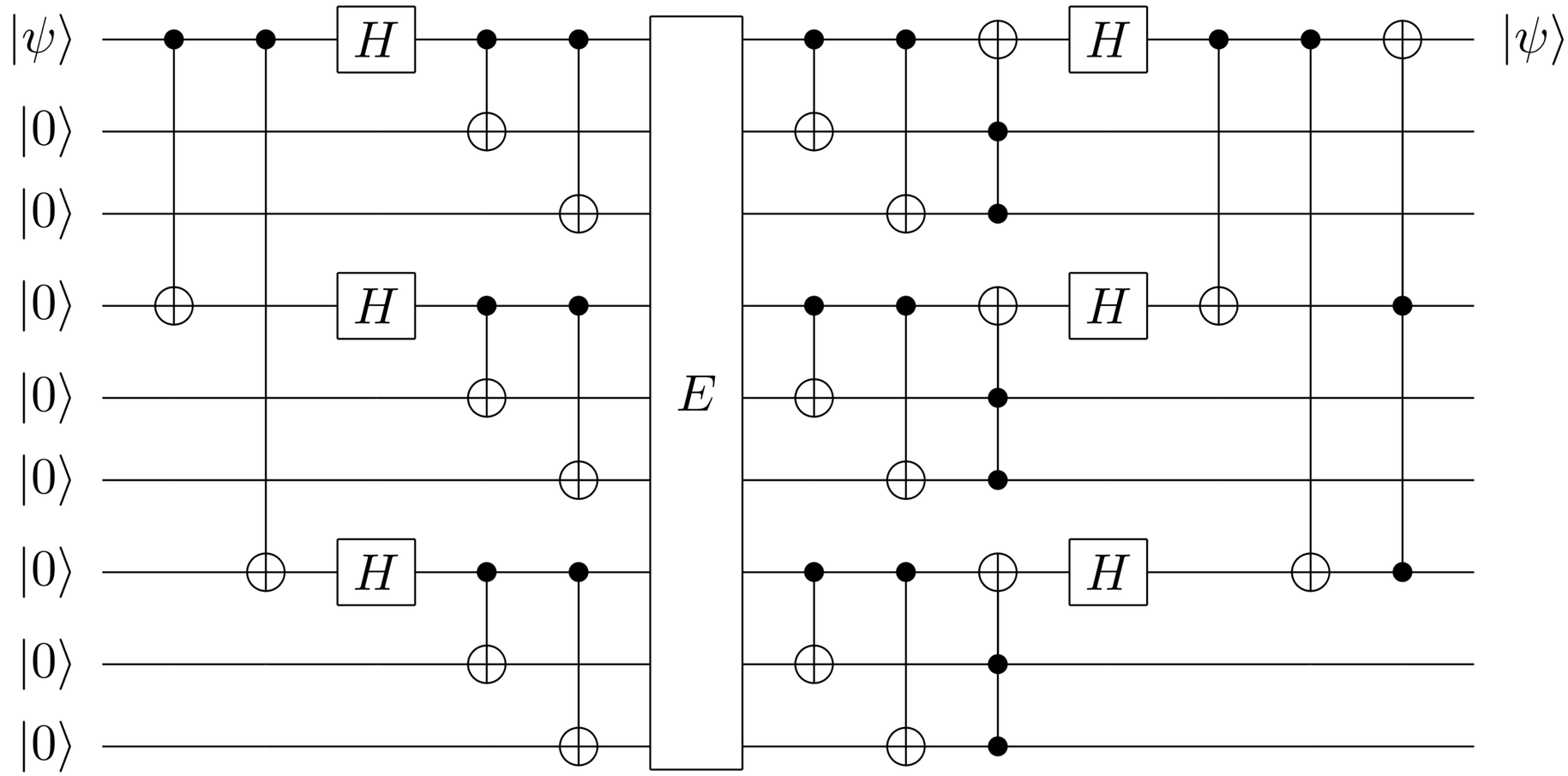
この誤差を（まだ実装できなかったとしても理論上だけでも）なんとかしないと、量子計算を正確に行うことはできない。

量子エラーコレクション

通常のコンピュータでは、同じ計算を何度も行ったり、もっと一般的に言えば、同じ物理過程を何度も通すことで正しい計測を行える。
しかし、量子ビットはコピーが不能で、かつ比較をすることも不可能である。
量子ビットに生じるエラーを外側から観測するのは不可能に見える。
ところが、Shor’s Algorithm で知られる Shor の発明した Shor の9量子ビットコードエラーコレクションを使えば、量子ビットをエラーから保護し、量子ビットが壊れるのを防ぐことができる。
Shor の9量子ビットコードエラーコレクションでは、量子もつれを使ってエラーを4つの状態に振り分け、観測して状態を確定させることで、振り分けられたエラーを修正できるようになっている。

量子エラーコレクション

Shor's Quantum Error Correction の全体像（Wikipedia より）



量子エラーコレクション

よくわからないので、量子ビットに起こりうるエラーを順に考えていく。

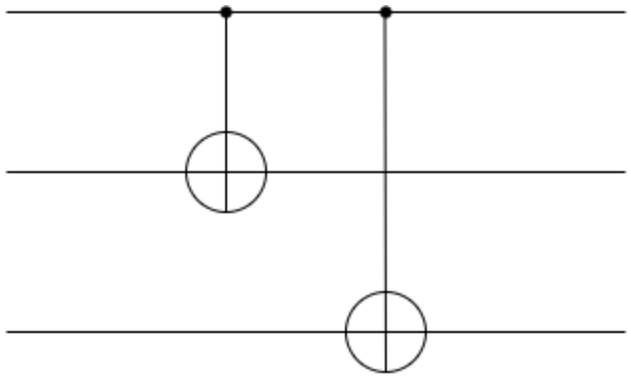
・ビットフリップエラー

ビットフリップエラーは、

$$c_0|0\rangle + c_1|1\rangle \to c_1|0\rangle + c_0|1\rangle$$

というようなエラーである。

このエラーをコレクションするために、次のような回路に、上から、守りたいビット、$|0\rangle, |0\rangle$ の順に量子ビットを通す。



量子エラーコレクション

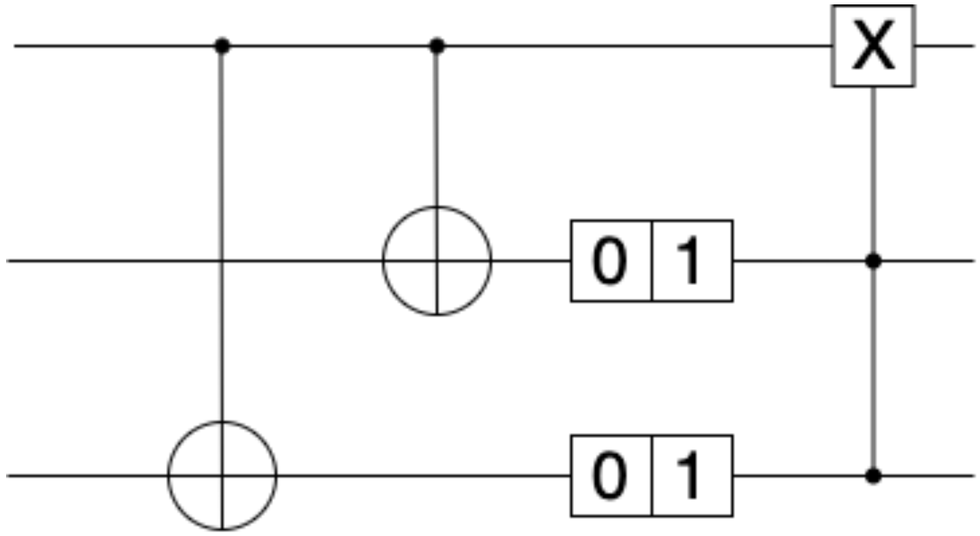
これで、3つの量子ビットは次のような状態になる。

$$(c_0|0\rangle + c_1|1\rangle) \otimes |00\rangle \rightarrow c_0|000\rangle + c_1|111\rangle$$

これで、3つの量子ビットが完全にもつれた。
実は、このあとこのうちどれか1つがフリップエラーを起こしたとしても、「多数決」によって、守りたい第1ビットを修正することができる。
つまり、第2ビットと第3ビットの双方と第1ビットが異なれば、第1ビットにフリップエラーが起こったとして第1ビットを修正してやればいい。
ふたつのビットが異なるかどうかは CNOT ゲートで計算できて、条件を複数とる CCX ゲートと呼ばれるゲートを使えば、次ページの図の回路で第1ビットを修正できる。

量子エラーコレクション

観測を行ってからそれを制御ビットにしてやれば、もつれの影響で必ず第1ビットが収縮するため、エラーを次の回路で間違いなく修正できる。
こうして、必ずビットフリップエラーを修正できることがわかった。



量子エラーコレクション

しかし、量子ビットには他のエラーも存在する。

・位相フリップエラー

位相フリップエラーは、

$$c_0|0\rangle + c_1|1\rangle \to c_0|0\rangle - c_1|1\rangle$$

というエラーである。

しかし、ここで「基底の置き換え」を行うと、このエラーはビットフリップエラーに帰着できる。

$$|0\rangle \to \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle), |1\rangle \to \frac{1}{\sqrt{2}}(|0\rangle - |1\rangle)$$

と変換してやると、このエラーはただのビットフリップエラーとみることができる。

量子エラーコレクション

そして、この変換は明らかにアダマール変換である。
つまり、アダマール変換すれば位相フリップはビットフリップに帰着できて、ビットフリップと同様にエラーコレクションができる。
具体的には、エンコードのとき、もつれを生じさせた後にすべてのビットをアダマール変換して、デコード前にもう1度アダマール変換してからフリップエラーと同じようにコレクションしてやればよい。

~~現在 1:15 なので図を描くのはサボります~~

量子エラーコレクション

ここまで2種類の量子フリップエラーを考えてきたが、実際には任意のエラーをコレクションする必要がある。
ここで、「エラー」とは何かをちゃんと定義してやる。
任意の量子ビットの状態から任意の状態への遷移は、2次正方行列を用いて書くことができる。
ここで、任意の2次正方行列は、次の4つの行列の線形結合で書ける。

$$\begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}, \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}, \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix}, \begin{pmatrix} 0 & -1 \\ 1 & 0 \end{pmatrix}$$

この行列はそれぞれ、エラーなし・ビットフリップエラー・位相フリップエラー・その2つのエラー両方、を表す。

量子エラーコレクション

よって、すべてのエラー（もっと一般的にいえば状態遷移）は、先に挙げた2つのエラーの重ね合わせで表現できる。
そして、すべてのエラーは、コレクションの補助ビットを観測する過程で、**必ず**2つのエラーのどちらかに収縮する。
つまり、任意のエラーは、観測による収縮という量子力学的な性質を用いて、2つのエラーのどちらかだったことにすることができる。
こうして、エンコード後に任意の1ビットに起こった任意のエラーを、量子ビットの性質を用いて修正することができた。
ただし、実際の量子コンピュータでは、複数のビットに同時に少しずつエラーが起こることの方が多いため、このコレクション方法が万能というわけではない。

おまけ

Q# でさっきのエラーコレクションを実装してみました

偏角が $\frac{\pi}{2}$ ずれてますが、これは本質的には同じ量子ビットです

・コレクションなしの場合（ぶっ壊れてる）

```
# wave function for qubits with ids (least to most significant): 0
|0⟩:     0.289601 +  0.456217 i  ==     ******                [ 0.292003 ]     +/  [  1.00519 rad ]
|1⟩:     0.721786 +  0.432461 i  ==     ****************      [ 0.707997 ]      /- [  0.53980 rad ]
# wave function for qubits with ids (least to most significant): 0
|0⟩:    -0.512673 +  0.416624 i  ==     *********             [ 0.436410 ]  \      [  2.45919 rad ]
|1⟩:    -0.646894 + -0.380944 i  ==     ************          [ 0.563590 ] -/      [ -2.60939 rad ]
```

・コレクションありの場合（正しい）

```
# wave function for qubits with ids (least to most significant): 0
|0⟩:     0.949983 +  0.263736 i  ==     ********************* [ 0.972025 ]      /- [  0.27080 rad ]
|1⟩:    -0.131812 + -0.102959 i  ==     *                     [ 0.027975 ]  /      [ -2.47848 rad ]
# wave function for qubits with ids (least to most significant): 0
|0⟩:    -0.263736 +  0.949983 i  ==     ********************* [ 0.972025 ]  \-     [  1.84160 rad ]
|1⟩:     0.102959 + -0.131812 i  ==     *                     [ 0.027975 ]      \  [ -0.90768 rad ]
```

おまけ

Q# は書きにくい書きにくいって無限人が言ってますが、自分はエミュレータの中では格段に書きやすい（タイプ数は多いけど）方だし、文法もしっかりしている方だと思っています。

~~Pythonのモジュールでエミュレートなんてしたくない（宗教的主張）~~

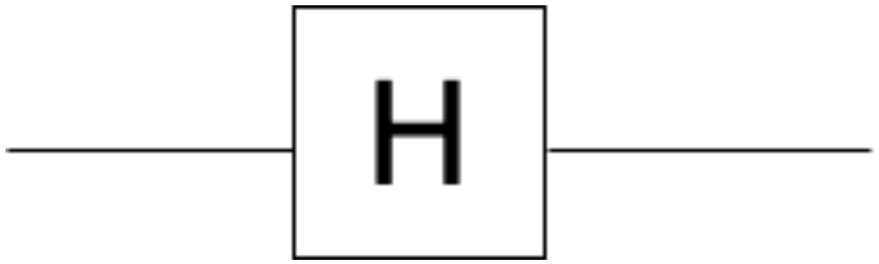
```
operation ErrorCollectionEncode(q:Qubit,qs:Qubit[]):Unit{
    CNOT(q,qs[2]);CNOT(q,qs[5]);
    H(q);H(qs[2]);H(qs[5]);
    CNOT(q,qs[0]);CNOT(q,qs[1]);
    CNOT(qs[2],qs[3]);CNOT(qs[2],qs[4]);
    CNOT(qs[5],qs[6]);CNOT(qs[5],qs[7]);
}

operation ErrorCollectionDecode(q:Qubit,qs:Qubit[]):Unit{
    CNOT(q,qs[1]);CNOT(q,qs[0]);
    CNOT(qs[2],qs[4]);CNOT(qs[2],qs[3]);
    CNOT(qs[5],qs[7]);CNOT(qs[5],qs[6]);
    H(q);H(qs[2]);H(qs[5]);
    CNOT(q,qs[5]);CNOT(q,qs[2]);
    for(i in 0..7){let tmp=M(qs[i]);}
    Controlled Z([qs[0],qs[1]],q);
    Controlled Z([qs[3],qs[4]],q);
    Controlled Z([qs[6],qs[7]],q);
    Controlled X([qs[2],qs[5]],q);
}
```

```
operation QuantumTeleportation(x:Qubit,z:Qubit):Unit{
    using(y=Qubit()){
        SetQubitState(Zero,y);
        SetQubitState(Zero,z);
        H(y);
        CNOT(y,z);
        CNOT(x,y);
        H(x);
        let xres=M(x);
        let yres=M(y);
        if(yres==One){X(z);}
        if(xres==One){Z(z);}
        Reset(y);
    }
}
```

おまけ

あと、私事ですが、登校が再開されないという仮定のもと夏休みを過ごしていたため、やっていなかった夏休みの自由研究に夏季セミで使ったノートを提出（英断）する予定です。

みなさんはこうはならず、$\frac{1}{\sqrt{2}}(|0\rangle + |1\rangle)$ のように、ふたつの可能性を両方考えて行動しましょう。



おまけ

あと、深夜テンションで back number を聴きながらスライドを書いていたらネタを思いついたんですが、140字を超えたのでここで供養します。

くだらない話は思い付くのに君を抱き締めていい理由だけが見付からない？wフッフッフw心配することなかれwwこんな時にも役に立つのが量子コンピュータ、伝家の宝刀†Grover's Algorithm†を使わせていただきますぞ！wwwまずは初期状態を用意して...ウオー！O(N^(1/2)) で計算している間に好きな人がいなくなってしまった...悲しい... しかも Grover's Algorithm は最適な量子アルゴリズムであることが証明されている...悲しい...

おまけ

なお、Grover’s Algorithm は、このあと autumn_eel さんが解説してくれるはずです。

くだらない話は思い付くのに君を抱き締めていい理由だけが見付からない？wフッフッフw心配することなかれwwこんな時にも役に立つのが量子コンピュータ、伝家の宝刀†Grover's Algorithm†を使わせていただきますぞ！wwwまずは初期状態を用意して...ウオー！O(N^(1/2)) で計算している間に好きな人がいなくなってしまった...悲しい... しかも Grover's Algorithm は最適な量子アルゴリズムであることが証明されている...悲しい...

ご清聴ありがとうございました