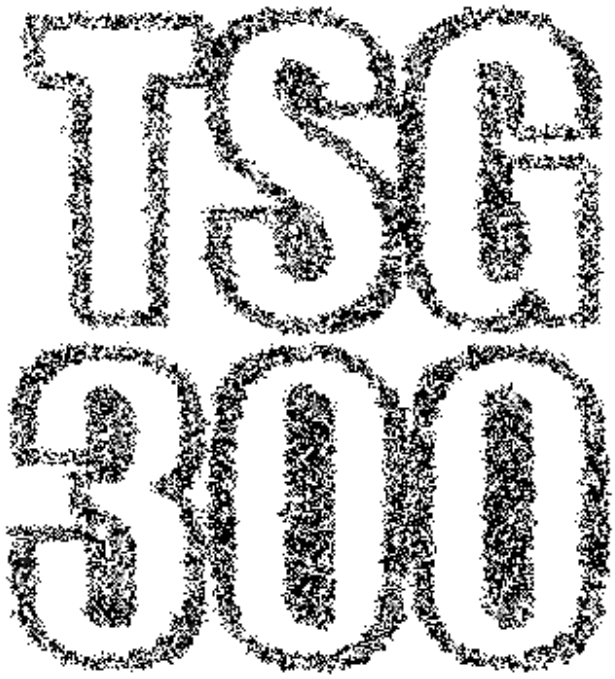
Theoretical Science Group

理論科学グループ



部報 300 号
— 駒場祭パンフレット号 —

目　次

ごあいさつ

理論科学グループ

本日は理論科学グループ駒場祭展示企画におこしいただき誠にありがとうございます。私たちは, 普段週に 1,2 回集まって情報交換などをしながらプログラミングなどの技量を磨いています。今回もおかげさまで部員たちがプログラミングしたゲーム作品を展示することができました。

この小冊子にはそんな展示企画の紹介記事を掲載しております。

今回企画紹介で使用している画像はあくまで開発途中の物であり、実際とは多少異なるかもしれません。また、印刷の都合上見づらい部分もあるかと思いますが、その点は実物を見ていただければということでご容赦ください。

なお、ここに載っていない作品や企画も当日は展示できるかもしれません。お楽しみに。

それでは、作品をお楽しみください。

TSG は 40 年以上続くサークルで, 発足当初は理学のゼミなどを
行っておりました. ガリ版刷りと思われる当初の部報も残っています.
TSG の部報は今回で 300 号の大台に乗ることになり, 嬉しい限りです.

展示企画

Polynomial Smash

Ktya

はじめに

皆様こんにちは。TSG で今年度部長をやっております Ktya というものです。駒場祭 TSG 企画にお越し頂きありがとうございます。この記事では僕が作った（未来の自分ががんばる事により展示されているとうれしいが。。）ゲームの「Polynomial Smash」について説明しておきます。

ゲームの説明

もともと iPad でダウンロードできるゲームである「Prime Smash!」をみて思いつきました。正整数に対して素数だったらタッチ、合成数だったら切ると言う動作をしてポイントを稼ぐゲームです。今回はこれを多項式に拡張してみようと言うアホな考えの元制作を決意しました。
(その他色々とアレンジを加える予定ですが)　人間の限界にチャレンジしてみましょう。

紹介

詳細についてはまだ決まっていません (すいません) 最初は $\mathbb{Z}[X]$ の範囲で既約になるまで分解するというのを考えています。ただし出てくる多項式の各次数の係数を同時に割り切る (2) 以上の整数は無いです。
さらに (実際に出来るかは分かりませんが)$\mathbb{Z}$ 以外の体でも出来たら面白いなーとか考えています。

その他

実は人生で初めて作ったゲームなので到らない点が多いと思いますがよろしくお願いします。

碁石拾い

udon

概要

碁石拾いは, ”ひろいもの”とも呼ばれる, 日本でつくられたパズルです. 江戸時代には, 問題が載った本が複数出版されるなど, 広く遊ばれていました. また, 近年, アメリカでも紹介され, TEXの作者として有名な Knuth 博士も一時期, ハマっていたとか... という話を本で読み, 興味を惹かれたので, ゲームにしてみました. ルールに従って, 碁盤上の碁石をすべて回収することができればクリアーです.

ルール

1. 碁石をクリックで選択し, 開始地点を決める.
2. その場所から, 線に沿って, 縦または横に直進する. 方向はカーソルキーで選択. 別の石にぶつかったら, その石を拾うことができる. (拾った石は盤上から取り除かれる)
3. 石を拾ったら, 進行方向に対し, 直進, 右折, 左折の三方向に進むことができる. (バックは禁止) ただし, 石がない方向には進めない.
4. 移動を繰り返していき, 進むことができなくなったらゲーム終了. このとき, 全ての石が回収済みであれば, **成功**. 拾い残しがあれば, **失敗**.

複数の解法が考えられる問題もあります.

開発環境

Visual C++/DX ライブラリ

参考文献

芦ヶ原伸之 『全天候型史上最強のパズルランド』(ベネッセコーポレーション, 1995)

Fixophony 概要

molcul

もう一度ごあいさつ

TSG 編集長の molcul です。このたびはお越しいただきありがとうございます。私の展示する作品は音楽ゲーム＝「音ゲー」です。ただし、権利上の関係でフリー音楽しかありません。

着想

そもそもの発端は自分が下手の横好きでピアノをやっていたことです。両手同時にあっちこっち動きながら鍵盤を弾くのは難しく、何人かの友人がやるようにかっこよく弾ければいいのですが、片手弾きが精一杯。

そこで考えたのが、片手を固定 (fix) して弾けるまがい物の (phony) ピアノを作ることでした。一番自分にとって自然な動きになるように、3 オクターブの音と指を対応させ、ついでに音ゲー感覚で弾けるピアノをつくろうと思ったんですね。それで fixophony です。全く意味不明ですね。

操作方法

曲を選択し、Space バーを押すと上から BeatMark という楕円がいくつか降ってきます。BeatMark が長方形に完全に入った瞬間に、「**BeatMark がある指 (キー) は押されている**」だけでなく「**BeatMark がない指は離されている**」場合、得点が入ります。親指の位置に暗い色の BeatMark が降ってくることがありますが, これは押しても押さなくても構いません。

BeatMark が長方形に入るずっと前から、指定した組み合わせの指を押しても得点は入りますが、BeatMark が長方形に入った瞬間にタイミングよく **1 つでも**指を押下すると高得点です。

ポイント

押し続け容認

親指と中指の位置に BeatMark が 3 連続で降ってきたとしましょう。このとき、

- 親指と中指をタイミングよく同時に 3 回押下する
- 親指を押し続けたまま中指をタイミングよく 3 回押下する

どちらの場合も同じ得点が入ります。2 つのキーを同時に押すより, 1 つのキーのほうがタイミングが合いやすいですよね。

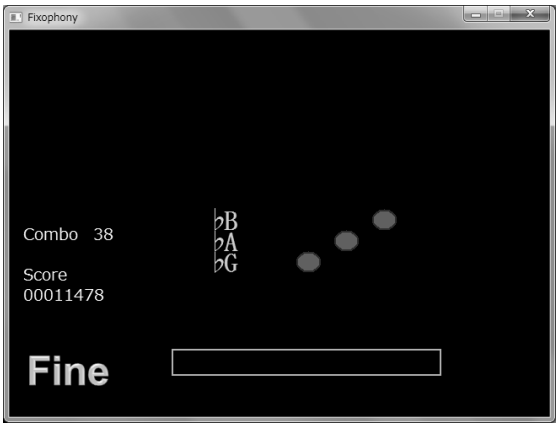
親指から小指までまとめて判定される

つまり, 2 つの指を押すべき時に 5 つの指全部同時押し！しても 1 点も入りません。

4 段階判定

Fixophony のタイミング判定は 4 種類あり、$Fail$(失敗), $Fair$(許容), $Fine$(良い), $Feat$(最適)です。$Fine$ と $Feat$ はコンボが加算され、$Fail$ はコンボが 0 にリセットされます。$Fair$ はコンボが加算されませんが、リセットもされません。

スクリーンショット



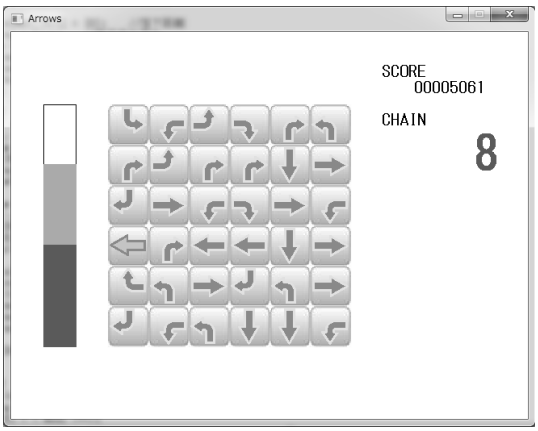
最後に

誰得なゲームですが遊んでみてください。

矢印パズルゲーム Arrows

宮田 圭介

矢印パズルゲーム　Arrows



はじめに

矢印を使ったパズルゲームを作成しました。簡単なものですがご覧いただければ幸いです。

基本ルール

３つ以上つながった矢印を左クリックすると矢印を消すことができます。消すとスコアとゲージが増え、新たな矢印が降ってきます。矢印を右クリックすると、矢印が時計回りに９０度回転します。

ゲージ

画面左部にあるのがゲージで、時間とともに減少していきます。薄いサブゲージは減るスピードが速く、矢印を消すと全回復します。サブゲージがなくなるとメインゲージが減少を始めます。

濃いメインゲージは減るスピードが遅く、矢印を消すと少し回復します。メインゲージが０になるとゲームオーバーです。

連鎖

間に挟む回転を１回以下にして矢印を続けて消すと、連鎖となります。連鎖が続いているほどスコアとメインゲージの上昇量が増えます。

オンラインタイピング対戦ブラウザゲーム

tkys

ブラウザ上で動くタイピング対戦ゲームを作りました。そのはずです。少なくとも今こうしてこの紹介が読まれている駒場祭の時点ではそうなっているはずです。世に数多ある例に倣い、原稿を書いている時点ではまだモノが出来上がっておりませんので、これから書かせていただく内容紹介は3割ほどはそうなるんだろうという予定です。といってもゲームの内容については文章でつらつらと書き続けるよりも実際に現物を触ってもらった方がいいと思うので、内容はおいといていくつかの実装上の特徴を紹介したいと思います。

○ プッシュを実現するために websocket を使用している

このゲームでは対戦する二人のプレイヤーのタイピング状況がリアルタイムでお互いに伝えられますが、通常こういうときに用いられてきた Flash は使用されていません。純粋にブラウザ上で動作します。これは HTML5 の一環として策定が始められた、websocket のおかげです。モダンブラウザではこれを利用することで外部プラグインなしにサーバからのプッシュ通信を実現します。（SPDY とか http2.0 とかはとりあえず置いときましょうよ）ちなみに websocket サーバの実装には ruby の em-websocket を使用しています。(無知ゆえに php でやってみようとかいう無謀な試みも一度やりましたが …)

○ アニメーションなどを CSS3 で実装している

jQuery の animate メソッドなどではなく CSS3 のアニメーションを用いています。Flash のキーフレームを作る感覚で実装できるのでなかなか書きやすかったです。ちなみに過去のブラウザとの互換性云々は websocket を使うと決めたときにもうどうしようもなくなっていたので、CSS3 を使うことには迷いはありませんでした。

要するにこの二つです。つまり HTML5 & CSS3 という流行りのミーハーな技術を使ってみたということですね。タイピングで腕に覚えがある方もそうでない方も、どうか一度遊んでみてください。

一般記事

トゥーンレンダリングっぽい何か

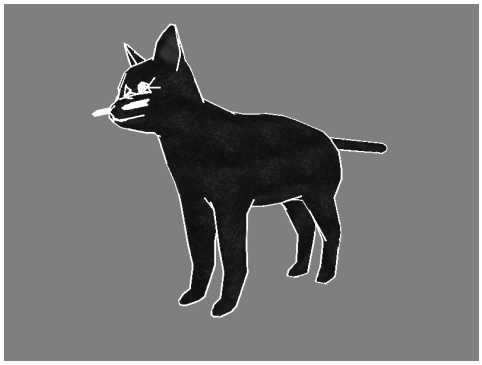
wleaf

トゥーンレンダリング技術は、3DCG 独特の「3Dっぽさ」を、画像処理によってアニメっぽく変換する技術である。トゥーンレンダリング技術の専ら応用先は、映画やアニメであり、次いでゲームとなる。一般的にトゥーンレンダリングにおいて見るに耐える精度のものを実現するためには、複雑な過程を踏まねばならない。また 3D ゲームというジャンルの関係から、プレイ時に不自然な状態になりやすく、採用タイトルは多くない (トゥーンレンダリングを採用しているタイトルとしては、NARUTO －ナルト－疾風伝 ナルティメットストーム や二ノ国あたりが有名か) 。今回作ったものは、リアルタイム性や安定性が求められるゲーム用ではなく、CG 製作などへの応用を考えた物である。

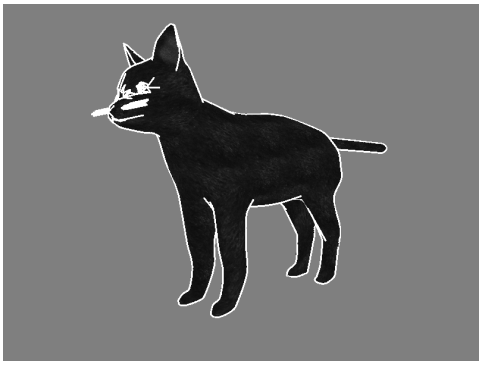
トゥーンレンダリングというと、キャラクターに輪郭を付けた上で、陰影を階調分けして塗る手法として認知されている場合が多い。輪郭を付ける場合、レンダリング対象のポリゴンを少し大きくして、そのはみ出し分をエッジとして利用する方法が元も簡単であり、よく用いられる。影の階調分けは、陰影の値を階調分けすれば容易に実装可能である。しかし、上記の方法では、輪郭はポリゴン数に依存するため、輪郭線がなかなか滑らかにならない。こうした輪郭線のカクカクは特に拡大した時に顕著になる。輪郭は滑らかであった方が良い。また、陰影の階調分けという (芸術的な意味での) 表現技法は、いわゆるアニメ塗りに限った話であり、アニメ以外の CG などでは、もっと色の連続を大切にしている。というわけで、輪郭を滑らかな曲線で補間 (今回の場合は 3 次のベジエ曲線) し、かつ色のグラデーションを付けるような形でのレンダリングは面白いかもしれない。色のグラデーションは、より暗い部分だけ塗られるよう、色の補正をトーンカーブで実装する。リアルタイム性は必要ないので、速度は一切追求していない。コードは書きやすさを重視したため、5 重 for 文なども存在する。

まず、輪郭抽出について考える。次の画像は、猫のモデルから線分の輪郭を抽出している (ちなみに輪郭の抽出は、先述のポリゴンを少し大きくして〜というような方法ではなく、輪郭となる線分を一つずつ判定して描画している) 。

目のあたりなどに不自然な線が見られるのは、目のあたりの形が複雑なためである。左足のところに線が 2 重に見えるのは、私のモデリング技術不足である。滑らかにモデルを作るのは、素人には難しい。先ほどの猫のモデルの輪郭をベジエ曲線で補間したものが次の画像である。線が少し太いのは、出力設定の問題である。よく目を凝らすと、たしかにお尻の辺りとかが滑らかであるような気がする。でも、一見しただけでは違いがよく分からない。ということで、ベジエ曲線はイマイチであった。描画サイズに比べて、この猫のモデルのポリゴンがそこそこ大きく (1000



(a) 加工前

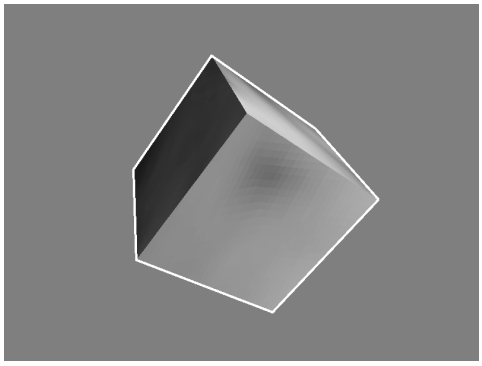


(b) 加工後

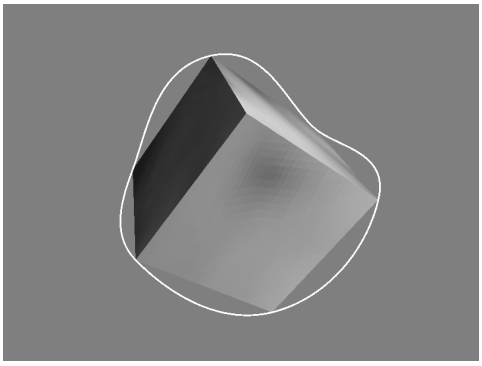
ポリゴンぐらい) 、違いがあまり見えなかったのだろうか。

さて、上でしたようなベジエ曲線での補間にはまだ問題がある。例えば、以下のような立方体の輪郭をベジエで補間する場合を考える。

もちろん、単純に実装すると以下のような出力が得られる。



(c) 加工前

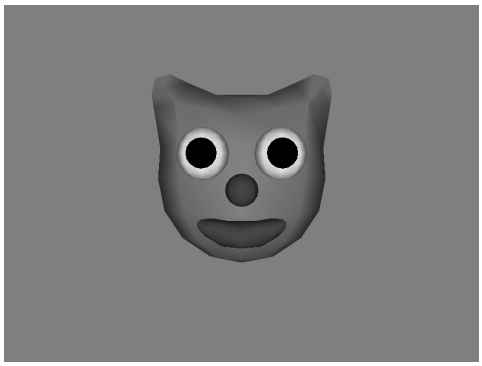


(d) 加工後

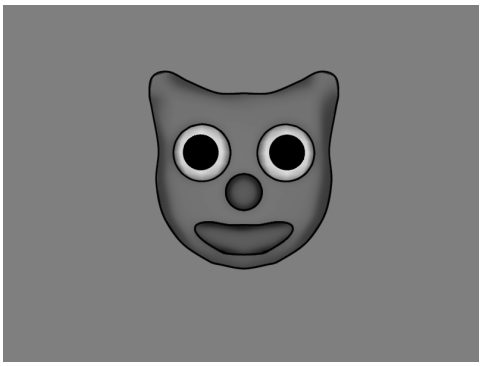
このような輪郭線の出力を防ぐためには、十分モデルを構成するポリゴンが密である必要がある。しかし十分ポリゴンが密なら、曲線で補間する必要がなくなってしまう。このベジエ曲線で増えたり減ったりした面の隙間を埋めることも考えたが、3D 描画においては出力用の色バッファだけでなく、ステンシルバッファや深度バッファなどについても考慮する必要がある。それらの整合性を保ちながら面の隙間を埋めるのは困難である。明らかなのは、ベジエ曲線での補間が全てのモデルに対して適用できるわけではない、というのことである。

以上の結果を受けた上で、よりベジエ曲線による補間を活かしながらも、今度はさらに色のグラディエーションを利用したいと思う。次に描画する対象は以下のようなモンスターである。DirectX SDK 付属の X ファイルビューアで見たものを、背景色の統一のために加工した物である。ここで

重要なのはモンスターのデザインではなく、顔の輪郭である。顔の輪郭部分にはいわゆる 3Dっぽい、カクカクが見える。ピクセルシェーダなどを利用して描画しない面単位の処理で計算をしているのため、陰影もガタガタである。かなり状態の悪いモデルと、グラフィックの質を求めるには不適切なビューアという、最悪に近い組み合わせであり、これと比較を行うのもどうかと思うが、他に用意できなかったので、この画像をサンプルとする。さて、次はこのモンスターを、輪郭を補間した上で、グラデーションをつけて表示すると、以下のような結果が得られる。



(e) 加工前



(f) 加工後

見ての通り確かに輪郭が丸くなり、また陰影も滑らかになったような気がする。しかし、一般的なトゥーンレンダリングをかけたのと同じような結果になっている。ベジエ曲線も陰影も効果は確かにあったが、一般的なトゥーンレンダリングと比較して特段優れているところはない。ちなみに、耳の辺りの輪郭線が少し太いように見えるのは、ベジエ補間によって輪郭が膨らむことによって生じた隙間から背景の灰色が見えるためである。

このレンダリング技法は、輪郭点の抽出がカギであるが、この抽出は結構時間がかかり、またカメラやモデルの位置を変更するたびに抽出し直さなければならない。このように、レンダリングに時間がかかるのがこの技法の欠点である。そもそも、一般的なトゥーンレンダリングの方法に対する描画的メリットが今回は見つからなかったため、採用するメリットは今のところ皆無である。大変残念である。とりあえずは高速化を目指したいところである。

量子コンピューター覚え書き

kobae964

下準備

分数の近似

以下の問題を考えます.

ある正の整数 M と有理数 q が存在します. この q は $0 \leq q < 1$ と,

$$\exists r, s \in \mathbb{Z}, 0 \leq r < s \leq M \wedge q = \frac{r}{s}$$

を満たします. このとき,r, s を知るためには q をどのくらいの精度で知ればよいでしょうか?
ここで, **精度 ε である数 q を知る**とは,

$$n\varepsilon \leq q < (n+1)\varepsilon$$

を満たす整数 n を知る, という意味です.(n が整数に限られることに注意してください)

これに対して,$\varepsilon = 1/(M(M-1))$ の精度で q を知れば十分なことが証明できます.
略証:
二つの候補 $r_1/s_1, r_2/s_2$ について, **それらが異なるとすれば**その差は

$$\left|\frac{r_1}{s_1} - \frac{r_2}{s_2}\right| = \left|\frac{r_1 s_2 - r_2 s_1}{s_1 s_2}\right| \geq \frac{1}{\mathrm{lcm}(s_1, s_2)} \geq \frac{1}{M(M-1)} = \varepsilon$$

を満たすので,

$$n\varepsilon \leq q < (n+1)\varepsilon$$

が成り立つとすれば, 二つの候補がともに

$$n\varepsilon \leq \frac{r}{s} < (n+1)\varepsilon$$

を満たすことはありません.

□

さて, ここで興味のあるのは $M = 2^m \quad (m \in \mathbb{Z})$ が成り立つときであり, その場合は

$$\varepsilon = \frac{1}{M(M-1)} \geq \frac{1}{M^2} = 2^{-2m}$$

であるため,q の精度は小数点以下 $2m$ 桁で十分です.

次に考えることは,$q = r/s$ を満たす r, s を知ることはどのくらいの時間がかかるのか, ということです. これは量子コンピュータにおける素因数分解 ($m = \log_2 M$ の多項式時間) に使われるので,$m = \log_2 M$ の多項式時間以内に終了する必要があります.
この条件を満たすアルゴリズムは存在し, 以下のようになります.

q を, 問題となる有理数とする.
$s_{\max}$ を,s の最大値 (上の場合では M) とおく. floor(y) は、y の値以下の最大の整数とする.(floor$(2.5) = 2$, floor$(-2.8) = -3$)

1. y_0 を実数とし, その値を q とする.
2. a_{00} を整数とし, その値を 0 とする.
3. a_{10} を整数とし, その値を 1 とする.
4. n を 0 とする.
5. 以下 6. から 12. までの演算を繰り返す.
6. $z_n := y_n - \mathrm{floor}(y_n)$
7. $z_n < 0.5/{s_{\max}}^2$ ならば a_{1n} を返して終了する
8. $y_{n+1} \leftarrow 1/z_n$
9. $a_{1(n+1)} \leftarrow \mathrm{floor}(y_{n+1})a_1 + a_0$
10. $a_{1(n+1)} \geq s_{\max}$ ならば a_{1n} を返して終了する
11. $a_{0(n+1)} \leftarrow a_{1n}$
12. n を 1 増やす
13. 6. に戻る

($\leftarrow$ は代入を表す)

これによって返された値が, 与えられた有理数 $q = r/s$ の分母 s となります.
これはユークリッドの互除法というアルゴリズムを少し改造したものになっています. このアルゴリズムがうまくいくことを $q = r/s(r, s$ は互いに素) として証明します.
(注:実際には q は正確に s/r に等しいわけではなく、あくまで近似でしかないことに注意してください. これは単なる略証です.)
まずは,6. から 12. までのループの中で何が行われているかを見ることにします. 簡単のために $u_n = \mathrm{floor}(y_n)$ と記し, $y_n = c_n/d_n(c_n, d_n$ は整数) とすると,

$$y_{n+1} = \frac{1}{y_n - \mathrm{floor}(y_n)} = \frac{1}{c_n/d_n - u_n} = \frac{d_n}{c_n - u_n d_n}$$

であるため,

$$\begin{pmatrix} c_{n+1} \\ d_{n+1} \end{pmatrix} = \begin{pmatrix} 0 & 1 \\ 1 & -u_n \end{pmatrix} \begin{pmatrix} c_n \\ d_n \end{pmatrix}$$

$$\begin{pmatrix} a_{0(n+1)} \\ a_{1(n+1)} \end{pmatrix} = \begin{pmatrix} 0 & 1 \\ 1 & u_{n+1} \end{pmatrix} \begin{pmatrix} a_{0n} \\ a_{1n} \end{pmatrix}$$

これを変形すると, 任意の n に対して

$$\begin{pmatrix} (-1)^{n+1} c_{n+1} \\ (-1)^{n+2} d_{n+1} \end{pmatrix} = \begin{pmatrix} 0 & 1 \\ 1 & u_n \end{pmatrix} \begin{pmatrix} (-1)^{n} c_{n} \\ (-1)^{n+1} d_{n} \end{pmatrix}$$

となります. よって, 任意の n に対して,

$$\begin{pmatrix} (-1)^{n+2} c_{n+2} & a_{0(n+1)} \\ (-1)^{n+3} d_{n+2} & a_{1(n+1)} \end{pmatrix} = \begin{pmatrix} 0 & 1 \\ 1 & u_{n+1} \end{pmatrix} \begin{pmatrix} (-1)^{n+1} c_{n+1} & a_{0n} \\ (-1)^{n+2} d_{n+1} & a_{1n} \end{pmatrix}$$

両辺の行列式をとると,

$$\begin{vmatrix} (-1)^{n+2} c_{n+2} & a_{0(n+1)} \\ (-1)^{n+3} d_{n+2} & a_{1(n+1)} \end{vmatrix} = (-1) \begin{vmatrix} (-1)^{n+1} c_{n+1} & a_{0n} \\ (-1)^{n+2} d_{n+1} & a_{1n} \end{vmatrix}$$

ここで,

$$\begin{vmatrix} (-1)^{1} c_{1} & a_{00} \\ (-1)^{2} d_{1} & a_{10} \end{vmatrix} = \begin{vmatrix} -s & 0 \\ d_1 & 1 \end{vmatrix} = -s$$

なので, 結局

$$\begin{vmatrix} (-1)^{n+1} c_{n+1} & a_{0n} \\ (-1)^{n+2} d_{n+1} & a_{1n} \end{vmatrix} = (-1)^{n+1} s \tag{8.1}$$

が言えます. このアルゴリズムは必ず終了し, そのときの n について,$y_n = \mathrm{floor}(y_n)$, つまり $d_n = 1$ が成立しています. このとき y_{n+1} は定義できませんが, 強引に c_{n+1}, d_{n+1} を考えると, $(c_{n+1}, d_{n+1}) = (d_n, c_n$ を d_n で割った余り $) = (1, 0)$ となるので, 式 (8.1) より,

$$\begin{vmatrix} (-1)^{n+1} & a_{0n} \\ 0 & a_{1n} \end{vmatrix} = (-1)^{n+1} s$$

よって

$$a_{1n} = s$$

となる.

次に, このアルゴリズムが任意の s に対して終了することを s に関する帰納法により示します.
(i)$s = 1$ のとき

1 回目の 6. の時点で $\mathrm{floor}(y) = \mathrm{floor}(q) = q$ であるため,$z = 0$ となります. $0 < 0.5/{s_{\max}}^2$ なのでここで終了します. (ii)$s < s'$ となる任意の s についてこのアルゴリズムが正しいと仮定します.$s = s'$ のとき
$q = ts' + g(t$ は商,g は余り,$0 < g < s')$ とおくと,$\mathrm{floor}(y) = t$ なので,6. において $z = y - t = g/s'$ となります. $g/s' \geq 1/s_{\max} > 0.5/s_{\max}^2$ であるため, まだ終了しません.
8. では y に $1/z = s'/g$ が代入されます.
9. では a_2 に $\mathrm{floor}(y)a_1 + a_0$ が代入されます. ここで, 今度は $s' = ug + v(u$ は商,v は余り$)$ とおくと,$\mathrm{floor}(s'/g) = u$ であるため, $a_2 = ua_1 + a_0$ となります. 10. では終了せず (証明略),11. と 12. を行うと $a_0 \leftarrow a_1, a_1 \leftarrow a_2$ となるので, 結局,6. から 12. までを一回行うと y の分母は必ず小さくなっているので, 帰納法の仮定より, このアルゴリズムは終了します.

以上より, このアルゴリズムが目的の性質を持つことが示されました.

量子ビットの基礎

この文章ではアルゴリズムについての解説を主にしたいため, 物理的な実現方法についての解説は割愛します. また, 厳密な理論も割愛します. この解説では, 以下のような理解で十分です.

量子状態

量子状態は,| と $\rangle$ で両側を囲み,$|0\rangle, |101\rangle, |\psi\rangle$ などと表記します.
厳密な定義を避け, 具体例により示します.

1bit の場合

複素数 c_0, c_1 が $|c_0|^2 + |c_1|^2 = 1$ を満たすとき,

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

は,1bit の量子 bit を表します (ただし外部からはどのような状態かはわかりません). この c_0, c_1 を**振幅**といい,$|0\rangle, |1\rangle$ を基底ベクトルと言います.(つまり量子 1bit は 2 次元の線型空間の部分集合に属しているということです.) ある絶対値 1 の複素数 z に対して,$a = cz, b = dz$ が成り立つとき, $a|0\rangle + b|1\rangle \equiv c|0\rangle + d|1\rangle$(二つの量子 bit は等しい) と言います.
この量子 bit に対し, **観測**という (不可逆の) 操作を考えることができます.
観測:

- 確率 $|c_0|^2$ で 0 を, 確率 $|c_1|^2$ で 1 を選択する.(これは観測者が知ることができる)

- 観測した量子 bit を, 選択が 0 ならば $|0\rangle$ に,1 ならば $|1\rangle$ に変更する (この操作は非可逆である).

また, 可逆な操作としては, 次のもののみを考えることができます (状態がどうなったかを知ることはできない).

ユニタリー変換

(複素係数)2= 2^1 次正方行列 U に対して,

$$UU^\dagger = U^\dagger U = E_2$$

($U^\dagger$ は U の複素共役の転置行列である)
を満たすものを, ユニタリー行列といいます. $c_0|0\rangle + c_1|1\rangle$ に対して,

$$\begin{pmatrix} x \\ y \end{pmatrix} = U \begin{pmatrix} c_0 \\ c_1 \end{pmatrix}$$

としたとき, その量子 bit を $x|0\rangle + y|1\rangle$ に変更することを**ユニタリー変換**といいます.
重要な性質として, **量子 bit に行える可逆な操作はユニタリー変換に限る**というものがあります.
以下, 実例を見ましょう.
(i-i) 否定 (X 変換)
量子 bit $c_0|0\rangle + c_1|1\rangle$ に対して, それを $c_1|0\rangle + c_0|1\rangle$ に変更する操作を**否定**と言います.
(注 1:簡単にわかる通り, この操作は可逆です. 実際 2 回行うと元に戻ります.)
(注 2:量子 bit$|0\rangle, |1\rangle$ に対して否定を行うと, それぞれ $|1\rangle, |0\rangle$ になります. これは古典的な bit と似ているとともに, $c_0|0\rangle + c_1|1\rangle (c_0 \neq 0, c_1 \neq 0)$ に否定を行うというのは, ある意味では $|0\rangle, |1\rangle$ の両方に否定を行っているとみることもできます.)
(注 3:

$$\sigma_x = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}$$

に対応します.)
(i-ii) アダマール変換
量子 bit$c_0|0\rangle + c_1|1\rangle$ に対して, それを

$$\frac{c_0 + c_1}{\sqrt{2}}|0\rangle + \frac{c_0 - c_1}{\sqrt{2}}|1\rangle$$

に変換する操作を, **アダマール (Hadamard) 変換**といいます. (

$$U = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix}$$

に対応します.)
(i-iii)Y 変換

行列 σ_y:

$$\sigma_y = \begin{pmatrix} 0 & -i \\ i & 0 \end{pmatrix}$$

によってあらわされる変換です.
(i-iv)Z 変換
行列 σ_z:

$$\sigma_z = \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix}$$

によってあらわされる変換です.
これはのちに述べる Conditional Phase の一例となっています.

2bit の場合

任意の (2bit の) 量子 bit は,

$$|c_{00}|^2 + |c_{01}|^2 + |c_{10}|^2 + |c_{11}|^2 = 1$$

を満たす複素数 $c_{00}, c_{01}, c_{10}, c_{11}$ を用いて,

$$c_{00}|00\rangle + c_{01}|01\rangle + c_{10}|10\rangle + c_{11}|11\rangle$$

と表すことができます.
この量子 bit に対しても, 観測という操作を考えることができます.
観測 (全体):

- 確率 $|c_{00}|^2$ で 00 を,$|c_{01}|^2$ で 1 を,$|c_{10}|^2$ で 10 を,$|c_{11}|^2$ で 11 を選択する.
- 観測した量子 bit を選択に応じて 1bit の場合と同じように変更する (非可逆)

これは 1bit の場合に似ていますが,1bit のみ観測するという部分的な観測というのもあります.
観測 (部分, 下位 1bit を観測する例):

- 確率 $|c_{00}|^2 + |c_{10}|^2$ で 0 を選択し, 確率 $|c_{01}|^2 + |c_{11}|^2$ で 1 を選択する.
- 0 を選択した場合は状態は

$$\frac{c_{00}|00\rangle + c_{10}|10\rangle}{\sqrt{|c_{00}|^2 + |c_{10}|^2}} = \frac{c_{00}}{\sqrt{|c_{00}|^2 + |c_{10}|^2}}|00\rangle + \frac{c_{10}}{\sqrt{|c_{00}|^2 + |c_{10}|^2}}|10\rangle$$

となる. 同様に,1 の場合は

$$\frac{c_{01}|01\rangle + c_{11}|11\rangle}{\sqrt{|c_{01}|^2 + |c_{11}|^2}} = \frac{c_{01}}{\sqrt{|c_{01}|^2 + |c_{11}|^2}}|01\rangle + \frac{c_{11}}{\sqrt{|c_{01}|^2 + |c_{11}|^2}}|11\rangle$$

となる.(下位 bit が確定し, もともとの状態に応じて残りの bit が決まる)

1bit→2bit:
ある量子 1bit$|\psi\rangle, |\phi\rangle$

$$|\psi\rangle = a|0\rangle + b|1\rangle, |\phi\rangle = c|0\rangle + d|1\rangle$$

があるとき, この 2bit をまとめて表記すると,

$$|\psi\rangle \otimes |\phi\rangle = (a|0\rangle + b|1\rangle) \otimes (c|0\rangle + d|1\rangle) = ac|00\rangle + ad|01\rangle + bc|10\rangle + bd|11\rangle$$

となります. 二つの量子 bit をまとめて表記するときにはこのように $\otimes$ 記号を使います. (また $|01\rangle$ などは $|0\rangle \otimes |1\rangle$ の略記です)

量子 2bit にも同様に, 以下のような可逆変換のみが考えられます.
ユニタリー変換
(複素係数)4= 2^2 次正方行列 U に対しても同じように

$$UU^\dagger = U^\dagger U = E_2$$

を満たすものを, ユニタリー行列といいます. $c_{00}|00\rangle + c_{01}|01\rangle + c_{10}|10\rangle + c_{11}|11\rangle$ に対して,

$$\begin{pmatrix} x \\ y \\ z \\ w \end{pmatrix} = U \begin{pmatrix} c_{00} \\ c_{01} \\ c_{10} \\ c_{11} \end{pmatrix}$$

としたとき, その量子 bit を $x|00\rangle + y|01\rangle + z|10\rangle + w|11\rangle$ に変更することを**ユニタリー変換**といいます.
その実例として, 以下のようなものがあります. (ii-1) 制御 NOT(cNOT)
上位 bit が 1 のベクトル ($|10\rangle, |11\rangle$) に対し, 下位 bit を反転させる操作を, **制御 NOT**(Controlled-NOT) といいます. つまり,

$$x|00\rangle + y|01\rangle + z|10\rangle + w|11\rangle$$

が

$$x|00\rangle + y|01\rangle + w|10\rangle + z|11\rangle$$

になります.
(

$$U = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix}$$

に対応します.)

3bit の場合

ここでは一つの操作を紹介します.
(iii-i) 制御制御 NOT(ccNOT)
上位 2bit が 11 である場合に下位 1bit を反転します.

$$|110\rangle \to |111\rangle, |111\rangle \to |110\rangle$$

n-bit(一般) の場合

ユニタリー変換:
先と同様に,2^n 次ユニタリー行列で表現できる変換です. 量子 bit に行える可逆変換はユニタリー変換に限ります.
その実例:
(iv-i)Conditional Phase
α を実数とします. $n$bit 全てが 1 のベクトル $|111\cdots11\rangle$ に対して, 位相を α だけずらす ($e^{i\alpha}$ 倍する) 操作を,Conditional Phase(CPhase と略される) といいます.
$n = 1$ のとき, これはユニタリー行列

$$U = \begin{pmatrix} 1 & 0 \\ 0 & e^{i\alpha} \end{pmatrix}$$

によって表現できます. $n = 2$ のときも同様に

$$U = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & e^{i\alpha} \end{pmatrix}$$

で表現できます.

性質

量子もつれ (エンタングル)

ある量子 2bit$|\psi\rangle$ が

$$|\psi\rangle = (a|0\rangle + b|1\rangle) \otimes (c|0\rangle + d|1\rangle) = ac|00\rangle + ad|01\rangle + bc|10\rangle + bd|11\rangle$$

(ただし,$a|0\rangle + b|1\rangle, c|0\rangle + d|1\rangle$ はそれぞれ量子 bit であり, 特に

$$|a|^2 + |b|^2 = 1, |c|^2 + |d|^2 = 1$$

を満たす) と表せる場合を考えます. このとき, 下位 bit を”観測”します. すると $|\psi\rangle$ は
(0 の場合, 確率:$|ac|^2+|bc|^2=|c|^2$)

$$\frac{ac|00\rangle+bc|10\rangle}{|ac|^2+|bc|^2}\equiv a|00\rangle+b|10\rangle=(a|0\rangle+b|1\rangle)\otimes|0\rangle$$

(1 の場合, 確率:$|ad|^2+|bd|^2=|d|^2$)

$$\frac{ad|00\rangle+bd|10\rangle}{|ad|^2+|bd|^2}\equiv a|00\rangle+b|10\rangle=(a|0\rangle+b|1\rangle)\otimes|0\rangle$$

(どちらの場合も, それぞれ $|ac|^2+|bc|^2, |ad|^2+|bd|^2$ は 0 より大きいとしてよい. もし 0 に等しいならばそれが観測の結果である確率は 0 である) に変化します. 上位 bit を観測する場合も同様です. つまり,2bit が複数の 1bit の”積”として表せるときには, 観測結果によって観測していない bit が変化することはありません (ここ重要).
これに対して, 次のような状態 $|\phi\rangle$ を考えます.

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

これを 2 つの量子 1bit の”積”として表すことはできません. $|\phi\rangle$ の下位 bit を観測してみましょう. すると, (0 の場合, 確率 0.5)$|00\rangle$
(1 の場合, 確率 0.5)$|11\rangle$
となり, 観測の結果により, 上位 bit が影響を受けてしまいました. この様に, 複数の量子状態 (この場合は $\phi\rangle$ の上位 bit と下位 bit) が相関を持つことを**量子もつれ**といいます.

線型性

今まで何の気なしに $\otimes$ や $+$ などの積や和を連想させる記号を使ってきましたが, これは以下の線型性により正当化されます.
線型性: 任意の量子状態 $|\psi\rangle$,$|\phi\rangle$, および任意のユニタリー変換 U に対して,

$$aU|\psi\rangle+bU|\phi\rangle=U(a|\psi\rangle+b|\phi\rangle)$$

が成り立ちます. また, 量子状態の組み合わせについて,

$$|\psi\rangle\otimes(|\phi_1\rangle+|\phi_2\rangle)=|\psi\rangle\otimes|\phi_1\rangle+|\psi\rangle\otimes|\phi_2\rangle$$

$$(|\psi_1\rangle+|\psi_2\rangle)\otimes|\phi\rangle=|\psi_1\rangle\otimes|\phi\rangle+|\psi_2\rangle\otimes|\phi\rangle$$

が成り立ちます.

量子アルゴリズムの基礎

(注意: 簡単のために,

$$M_n = \{0, 1, \cdots, 2^n - 1\}(\text{元が } 2^n \text{ 個}) = \mathbb{Z}/2^n(\text{巡回群, 加法群})$$

とおきます. また, 場合によって基底ベクトルの書き方として 2 種類を使い分けます.
(1)2 進数を用いる表し方

$$|000\rangle, |10101\rangle, |111\rangle \cdots$$

(2)10 進数を用いる表し方

$$|0\rangle, |21\rangle, |7\rangle, \cdots$$

)

量子アルゴリズムにおける演算には, 基底ベクトルについての演算が多いです.(ただし例外あり)
(例えば,(上位と下位入れ替え)

$$|00\rangle \mapsto |00\rangle, |01\rangle \mapsto |10\rangle, |10\rangle \mapsto |01\rangle, |11\rangle \mapsto |11\rangle$$

とする変換など.)
先に見たように, 量子 $n$bit は 2^n 種類の bit を同時に表現でき, それらに対する操作も同時に一度に行えるため, n が大きくなるにつれて計算の速度が飛躍的に向上する可能性を持っています.

ただし, 基本的なことですが, これで実現できる操作は ($M_n \to M_n$ の) 全単射, つまり逆変換ができるものに限られます. なぜなら, もし $f(m) = f(n)$ ならば,f を実現する操作を U とすると

$$U|m\rangle = |f(m)\rangle, U|n\rangle = |f(n)\rangle$$

ですが,U は逆変換ができるので

$$|m\rangle = U^{-1}|f(m)\rangle = U^{-1}|f(n)\rangle = |n\rangle$$

より,$m = n$ となるからです. ところが実用的な (量子とは限らない) アルゴリズムにはほぼ確実に乗算, 条件分岐などが必要となりますので, このままでは実現不可能です. そのため, 以下の性質を使います.

性質
m, n を任意の自然数 (1 以上),f を任意の $M_m \to M_n$ とする. このとき,

$$\begin{aligned} f^* : M_m \times M_n &\to M_m \times M_n \\ (x, y) &\mapsto (x, y + f(x) \mod 2^n) \end{aligned}$$

は全単射である.(逆写像 ${f^*}^{-1} : (x, y) \mapsto (x, y - f(x) \mod 2^n)$ が存在する)

こういった工夫を用いれば, 必要な関数はすべて定義できます.
以降, 必要な関数を順次定義していきます.

基礎

Conditional NOT

量子 1bit $|\psi\rangle$ と, 量子 m-bit$|c\rangle$ に対して, $|c\rangle$ がすべて 1 の場合に $|\psi\rangle$ に否定を行う操作です.($|c\rangle$ は変わりません)
この処理は帰納的に定義することができます.
(i)$m = 1$ の場合 $|c\rangle \otimes |\psi\rangle$ に対して制御 NOT をすればよいです.
(ii)$m = m'$ でできているとして,$m = m' + 1$ の場合
別の量子 1bit$|q\rangle$ を用意します. それに対して $|c\rangle$ の下位 m'-bit を使って Conditional NOT を行います. そのあと $|q\rangle \otimes (|c\rangle$ の上位 1bit$) \otimes |\psi\rangle$ に ccNOT を行えばよいです. さて,$|q\rangle$ の処理ですが, このまま $|q\rangle$ を破棄 (観測) してしまうと, $|q\rangle$ は $|c\rangle$ や $|\psi\rangle$ ともつれているので, 影響を与えてしまいます.

Conditional Increment

n-bit の量子 bit$|\psi\rangle$ と,m-bit の量子 bit$|c\rangle$ に対して, $|c\rangle$ がすべて 1 の場合にその値を 1 増やす操作です.(m はたいていの場合 1 ですが, 別に 1 である必要はありません)
$m = 1$,$|c\rangle = |1\rangle$ とすると, $n = 1$ のときは

$$x|0\rangle + y|1\rangle \to y|0\rangle + x|1\rangle$$

という変換になり,$n = 2$ のときには

$$x|00\rangle + y|01\rangle + z|10\rangle + w|11\rangle \to w|00\rangle + x|01\rangle + y|10\rangle + z|11\rangle$$

という変換になります. これは, 以下のような手順で帰納的に実現できます.
(i)$n = 1$ のとき
普通の NOT(X 変換) と同じです.
(ii)$n = n'$ で実現できているとして,$n = n' + 1$ のとき
(以降 $|d\rangle[e, f]$ で, 量子 bit$|d\rangle$ の上位 e-bit から f-bit($(f - e + 1)$bit) を表すものとします (最上位が 0 番目とします). また $|d\rangle[e]$ は $|d\rangle[e, e]$ と同じ意味です. また $|d\rangle$ の bit 数を $|d|$ と表記します.)
$|\psi\rangle[1, n']$ を使って $|\psi\rangle[0]$ に Conditional NOT を行います.
その後 $|\psi\rangle[1, n']$ を新しい ψ とみなして ($|c\rangle$ はそのままで),
$n = n'$ とした Conditional increment を行います.

lt

(less than の略)
$(\mathrm{lt}(a, |b\rangle, |f\rangle, |j\rangle))$
これが一番の鬼門と言えるでしょう.n-bit の $a, |b\rangle$ に対して,$a < b$ ならば f を反転するのですが, この大小比較という操作は曲者です. 例えば最上位の bit の時点で差があったらそれ以降の比較は意味をなさないのですが,b の値によって条件分岐をしてある操作を行ったり行わなかったりすることはできないので, 結局すべての bit をチェック (して, その結果を無視) することになります. そのため,$(n-1)$-bit の $|j\rangle$(j は junk の意) という量子 bit が必要になります. この $|j\rangle$ は観測してはいけません (ほかの bit ともつれています).

実際にやる方法:
(注意:$|j\rangle$ の上位の bit がすべて 1 のときにはまだ比較の結果が確定していないことに注意して読み進めてください.)

(i) まずは最上位 bit を処理します.
a の上位 0bit 目 (最上位, 今後 $a[0]$ と記す) が 1 のとき

$$\mathrm{cNOT}(|b\rangle[0] \otimes |j\rangle[0]), \sigma_x(|b\rangle[0]), \mathrm{cNOT}(|b\rangle \otimes |f\rangle)$$

の 3 つの操作を順に行います. これにより,$b[0]$ が 1 ならば未定なので $j[0]$ を 1 にして続行, そうでなければもう a の方が大きいことがはっきりしているので $j[0]$ を 0 にしてそれ以降の判定を無意味にすることになります.
$a[0]$ が 0 のときは,

$$\sigma_x(|b\rangle[0]), \mathrm{cNOT}(|b\rangle[0] \otimes |j\rangle[0])$$

の 2 つの処理を順に行います.

(ii) 次にそれ以外の bit を処理します. $i = 1, 2, \cdots, n-2$ として $a[i], b[i]$ を比較します. このときその比較が必要ならば,$j[0, i-1]$ のすべての bit が 1, 特に $j[i-1]$ が 1 であることに注意すると, これは上の 2 箇所の $\mathrm{cNOT}(|b\rangle[0] \otimes |j\rangle[0])$ を $\mathrm{ccNOT}(|j\rangle[i-1] \otimes |b\rangle[i] \otimes |j\rangle[i]$ に変更することでできます.

(iii) 最後の bit を処理します.
(ii) とほとんど同じですが, 最後に $\mathrm{ccNOT}(|j\rangle[i-1] \otimes |b\rangle[i] \otimes |j\rangle[i]$ を行わない点が異なります.(また, そのため)$a[n-1]$ が 0 のときは何もする必要がありません.

これで終わりです. 特に,$|b\rangle$ は変更されたままであることに注意! またこの操作は逆にたどれます.(これも後で使います)

加減

add

$(\mathrm{add}(|s\rangle, |d\rangle))$
量子 m-bit$|s\rangle$ を, 量子 n-bit$|d\rangle$ に加えます. あいている場所は 0 で埋めます. これは先の Conditional increment を $|s\rangle$ の bit 毎に繰り返せば実現できます.

muxadd

$(\mathrm{muxadd}(a_0, a_1, |c\rangle, |d\rangle))$
n-bit の整数 a_0, a_1(古典 bit で表現されている), 量子 1bit$|c\rangle$ に対してを n-bit の $|d\rangle$ に,c が 0 ならば a_0 を,c が 1 ならば a_1 を, 加えます. これも先の Conditional increment を各 bit 毎に行うことでできます.

addn

$(\mathrm{addn}(a, n, |b\rangle, |f\rangle, |s\rangle))$
$(a+b) \mod n$ を計算して, その結果を s に格納しますが, この写像を全単射にするために $a+b \geq n$ のときは f を反転します. このため,

$$(f,b) = (0,0), \cdots, (0, n-a-1) \mapsto (f,s) = (0,a), \cdots, (0,n-1)$$
$$(f,b) = (0,n-a), \cdots, (0,n-1) \mapsto (f,s) = (1,0), \cdots, (1,b-1)$$
$$(f,b) = (1,0), \cdots, (1,n-a-1) \mapsto (f,s) = (1,a-n+2^{|b|}), \cdots, (1,2^{|b|}-1)$$
$$(f,b) = (1,n-a), \cdots, (1,n-1) \mapsto (f,s) = (0,n), \cdots, (0,n+a-1)$$

というように,$f \neq 0$ の場合はおかしい対応になります.
その実現方法ですが, まず $|j\rangle$($(n-2)$-bit) を用意します.
また $|f'\rangle$(lt の結果を保存するための量子 bit だが,$|s\rangle[0]$ でよい) を用意します.
そして,$\mathrm{lt}(n-a, |b\rangle, |f'\rangle, |s\rangle[1, |s|-1])$ を実行します.
ここからが肝心ですが, そのあと

$$\mathrm{cNOT}(|f'\rangle \otimes |f\rangle)$$

を行って,f' の内容を f にコピーしておきます. そのあと,

$$\mathrm{lt}^{-1}(n-a, |b\rangle, |f'\rangle, |s\rangle[1, |s|-1]) \quad \text{(逆変換)}$$

を行って,$|s\rangle$ を元の状態 ($|0\rangle$) に戻します. そして,

$$\mathrm{muxadd}(a, 2^{|b|}+a-n, |f\rangle, |s\rangle), \mathrm{add}(|b\rangle, |s\rangle)$$

を行います. すると $|s\rangle$ には目的の値が入っています.($|f\rangle$ も変更されています)

oaddn

(overwriting addn の略)
(oaddn$(a, n, |s\rangle)$)
s の値に a を加えます (s は変更されます).
実現方法
まず量子 1bit$|f\rangle$ を用意します. また量子 $|s|$-bit$|j\rangle$ も用意します (これは $(a+b) \mod n$ を一時的に格納するために使われます). その後,

$$\mathrm{addn}(a, n, |s\rangle, |f\rangle, |j\rangle), \sigma_x(|f\rangle), \mathrm{addn}^{-1}(n-a, n, |s\rangle, |f\rangle, |j\rangle)$$

を順番に行います.
これでうまくいく理由は今回は割愛します.

乗法

muln

(muln$(a, n, |b\rangle, |p\rangle)$)
$|p\rangle$(0 の場合が多い) に $a * b \mod n$ を加えます.

omuln

(omuln$(a, n, |p\rangle)$)
$|p\rangle$ に a を掛けて,n で割った余りを求めます. 全単射であるために a, n は互いに素である必要があります. 実現方法は oaddn と似ているので省略します.

expn

(expn$(a, n, |b\rangle, |x\rangle = |0\rangle)$)
$|x\rangle$ を $|1\rangle$ に変更し, その後 a^b を掛け,n で割った余りを求めます.a, n は互いに素です. 具体的には,b の各 bit に対して,x に $a^{2^?}$ あるいは 1 を掛ければよいです.
このアルゴリズムはのちに解説するショアのアルゴリズムの中核をなします.

離散フーリエ変換 (DFT)

このアルゴリズムについては詳細は割愛します.

$$|\psi\rangle = |\psi_0\rangle \otimes |\psi_1\rangle \otimes \cdots \otimes |\psi_{n-1}\rangle$$

(各 ψ_j は 0 または 1 とする, したがって $|\psi\rangle$ は基底ベクトルである.
これは,$\psi = 2^{n-1}\psi_0 + 2^{n-2}\psi_1 + \cdots + 2^1\psi_{n-2} + \psi_{n-1}$ と言っているに等しい.)

に対して, これを

$$|\psi'\rangle = \sum_{j=0}^{2^n-1} \exp(-\frac{2\pi\psi j}{2^n})|j\rangle$$

$$= \sum_{0\leq k\leq n-1, j_k=0,1} \exp\left(-\frac{2\pi\psi(2^{n-1}j_0 + 2^{n-2}j_1 + \cdots 2^1 j_{n-2} + j_{n-1})}{2^n}\right)|j_0\rangle \otimes |j_1\rangle \otimes \cdots |j_{n-1}\rangle$$

$$= \sum_{0\leq k\leq n-1, j_k=0,1} \exp\left(-\frac{2\pi\psi(2^{n-1}j_0)}{2^n}\right)|j_0\rangle \otimes \exp\left(\frac{-2\pi\psi(2^{n-2}j_1)}{2^n}\right)|j_1\rangle$$

$$\otimes \cdots \otimes \exp\left(-\frac{2\pi\psi(2^1 j_{n-2})}{2^n}\right)|j_{n-2}\rangle \otimes \exp\left(-\frac{2\pi\psi j_{n-1}}{2^n}\right)|j_{n-1}\rangle$$

$$= \sum_{0\leq k\leq n-1, j_k=0,1} \exp(-2\pi\cdot 0.\psi_{n-1}\cdots j_0)|j_0\rangle \otimes \exp(-2\pi\cdot 0.\psi_{n-2}\psi_{n-1}\cdot j_1)|j_1\rangle$$

$$\otimes \cdots \otimes \exp(-2\pi\cdot 0.\psi_1\psi_2\cdots\psi_{n-1}\cdot j_{n-2})|j_{n-2}\rangle \otimes \exp(-2\pi\cdot 0.\psi_0\psi_1\cdots\psi_{n-1}\cdot j_{n-1})|j_{n-1}\rangle$$

(小数はすべて二進数である)
に変更する操作を離散フーリエ変換といいます. この変形には意味があり, 実装をするのに必要な変形なのですが, 今回は割愛します. また周期的に振幅が高い (例:$n = 5$ で

$$\frac{1}{\sqrt{5}}|3\rangle + |9\rangle + |15\rangle + |21\rangle + |27\rangle)$$

) 量子 bit を離散フーリエ変換すると, 結果として周期を r としたとき, $0, 1\cdot 2^n/r, 2\cdot 2^n/r, \cdots, (r-1)\cdot 2^n/r$ の r 個の数値に近い整数に対応する基底ベクトルの振幅が高くなります. (この例では $0, 5, 11, 16, 21, 27$ の 6 個なので,

$$|0\rangle, |5\rangle, |11\rangle, |16\rangle, |21\rangle, |27\rangle$$

の振幅が高くなる)

ショアのアルゴリズム

ここで, この記事の最大の動機であるショア (Shor) のアルゴリズムについて解説したいと思います.

ショアのアルゴリズムは, 大きな奇数の素因数分解を (その桁数の) 多項式時間以内に終わらせる非決定的アルゴリズムです. このアルゴリズムは,$a^r \equiv 1 \pmod N$ となる最小の正の整数 r を (量子コンピューターを使って) 求め, その後で古典的計算機を使って r から N の約数を求めるものです. 素因数分解する数を N(奇数) とします. ここでは $N = 15$ とします.

ショアのアルゴリズム

1. 量子 $2n$-bit$|\psi\rangle$ と, 量子 n-bit $|\phi\rangle$ を用意する.($N < 2^n$ をみたす)(ここでは $n = 4$)

2.$|\psi\rangle$ を,2^{2n} 種類のすべての基底ベクトルの振幅が等しくなるように初期化する.(これは $2n$-bit すべてを 0 で初期化した後アダマール変換すればよい)

その結果,

$$|\psi\rangle = \frac{1}{2^n}(|0\rangle + |1\rangle + \cdots + |2^{2n} - 1\rangle)$$

となる.(ここでは

$$|\psi\rangle = \frac{1}{16}(|0\rangle + |1\rangle + \cdots + |255\rangle)$$

となる.)

3. 整数 x を,$2 \leq x < N$ を満たすようにランダムにとる.x と N は互いに素としてよい.(そうでなければ, 両者の最大公約数を因数とすればよい)

(ここで x として N と互いに素なものが選ばれる確率は,$(\varphi(N) - 1)/(N - 1)$ である (φ はオイラーのトーシェント関数, **1 から** $N - 1$ までの整数のうち,N と互いに素であるものの個数). $N = 15$ の場合はこの値は 1/2 となりあまり大きくないが, 現実的な $N = pq$(p と q は巨大素数) の場合は十分 1 に近い.)

(ここでは $x = 2$ が選択されたものとする)

4.expn$(x, N, |\psi\rangle, |\phi\rangle)$ を行う. これにより,

$$|\psi\rangle \otimes |\phi\rangle = \frac{1}{2^n}(|0\rangle \otimes |1\rangle + |1\rangle \otimes |x\rangle + |2\rangle \otimes |x^2 \bmod N\rangle + \cdots + |2^{2n} - 1\rangle \otimes |x^{2^{2n}-1} \bmod N\rangle)$$

となる.

(ここでは

$$|\psi\rangle \otimes |\phi\rangle = \frac{1}{16}(|0\rangle \otimes |1\rangle + |1\rangle \otimes |2\rangle + |2\rangle \otimes |4\rangle + |3\rangle \otimes |8\rangle + \cdots + |254\rangle \otimes |4\rangle + |255\rangle \otimes |8\rangle)$$

となる.

4.$|\phi\rangle$ を観測する. ここで, 観測された値は何でもよい. 重要なのは,ϕ の値が決まれば,$a^b = \phi$ を満たすような b もある程度決まり, それはある公差 r の等差数列になるということである (そしてこの公差は

$$a^r \equiv 1 \pmod{N}$$

を満たす).

(ここでは $\phi = 8$ が観測されたとすると,$2^b \equiv 8 \pmod{15}$ を満たすのは $b = 3, 7, 11, 15, \cdots, 251, 255$ なので,

$$|\psi\rangle \otimes |\phi\rangle = \frac{1}{8}(|3\rangle \otimes |8\rangle + |7\rangle \otimes |8\rangle + \cdots + |255\rangle \otimes |8\rangle) = \frac{1}{8}(|3\rangle + |7\rangle + \cdots + |255\rangle) \otimes |8\rangle$$

となり, 実際に $|\psi\rangle$ の周期が $r = 4$ になっていることがわかる)

5.DFT の逆変換を $|\psi\rangle$ に行う. これにより,$|\psi\rangle$ を観測したときに結果が $2^{2n} \cdot s/r$ となる確率

が非常に高くなる. (ここでは　$|0\rangle, |64\rangle, |128\rangle, |192\rangle$)
6.$|\psi\rangle$ を観測し, その結果を $2^{2n}q$ とする.(q は小数点以下 $2n$ 桁までの実数)(ここでは $q = 1/4$ だったとする)
最初に紹介したアルゴリズムによって $q = s/r'$ となる分母 r' を求める.($r = r'$ となる確率は $\varphi(r)/r$ で, もし違っていた場合には次で失敗しやり直しになる)
(ここでは無事 $r' = r = 4$ となる)
7.r' が奇数ならばやり直す.r' が偶数ならば,

$$x^{r'} - 1 = (x^{r'/2} - 1)(x^{r'/2} + 1) \equiv 0 \pmod{N}$$

であるが, r が最小の周期であるため,r' も最小の周期である ($r = r'$ である) 可能性は高く, その場合

$$x^{r'/2} \pm 1 \not\equiv 0 \pmod{N}$$

が成り立つ. ことを利用して,N と $x^{r'/2} \pm 1$ の最大公約数を計算することで約数が求まる.
(ここでは $x^{r'/2} = 2^2 = 4$ であるため,

$$\gcd(N, x^{r'/2} - 1) = \gcd(15, 3) = 3$$

$$\gcd(N, x^{r'/2} + 1) = \gcd(15, 5) = 5$$

となり,3,5 という自明でない約数を持つことがわかる.)

編集後記

★ 300 号にふさわしい何かをしようと思いましたが、
それをするに足る意識がありませんでした。

★ TSG は多サークルよりもはるかに「強制される」ということが少なく
僕も個人としてそれは素晴らしいことだと思いますが、編集長としては大変です。

★ 多分、自分が編集する最後の部報であると思います。

★ TSG にお越しいただいた方に, 今一度礼を重ねて失礼します。

理論科学グループ　部報 第 300 号

2012 年 11 月 16 日　発行

発行者　河内谷耀一

編集者　久良尚任

発行所　理論科学グループ

〒 153–0041　東京都目黒区駒場 3–8–1

東京大学教養学部内学生会館 305

Telephone: 03–5454–4343

理論科学グループ部報 第 300 号
— 駒場祭パンフレット号 —
2012 年 11 月 16 日

THEORETICAL SCIENCE GROUP