Introduction to Quantum Algorithms.

1. What is Quantum Algorithm?
2. Why we need Quantum Algorithm?
3. How can we use/design Quantum Algorithm?

Now, we answer the above questions one by one.

First, we need know quantum computer. while in classical computing, bits can represent either a "0" or a "1", quantum bits or qubits can exist in superpositions of these values.

classical $\Rightarrow$ quantum Schrödinger's cats

$\{0, 1\}$ $\Rightarrow$ $[0, 1]$

Second, quantum register with several qubits to store and process exponentially more information than classical register with an equivalent number of bits. However, harnessing this property of quantum computers presents significant challenges.

Peter Shor: quantum polynomial-time factoring and discrete log algorithm.

public-key cryptography $\xrightarrow{\text{vulnerability}}$ quantum computer.

Lov Grover: quadratic speedup for unstructured search problem. $O(N) \Rightarrow O(\sqrt{N})$

HHL: properties of solutions of large sparse linear system (exp speedup).

classical $\xrightarrow{\text{speedup}}$ quantum

(big data, machine learning).

Third, how to use / design Quantum Algorithm need all this book. In Chatpter I, I will introduce these following three topics:

1.1 Deterministic ⇔ probability
P ⇔ NP

1.2 Boolean ⇔ Circuit.

1.3 reversible circuits.

1.1. Turing machine model (1963).

* Def 1.1. An alphabet is a finite nonempty set. ($\mathbb{Q}, \mathbb{Z}, \mathbb{R}, \mathbb{N}$).

* Logic operation:

Table 1.1.3. permitted Logic operations.

Name	Logic operator	
AND	$\wedge$	
OR	$\vee$	
NOT	$\neg$	
NAND	$\uparrow$	$\neg(x \wedge y)$
NOR	$\downarrow$	$\neg(x \vee y)$
XOR	$\oplus$	$(x \vee y) \wedge \neg(x \wedge y)$

important!

ab	$a \wedge b$	$a \vee b$	$\neg a$	$a \uparrow b$	$a \downarrow b$	$a \oplus b$
0 0	0	0	1	1	1	0
0 1	0	1	1	1	0	1
1 0	0	1	0	1	0	1
1 1	1	1	0	0	0	0

* Deterministic Algorithms.

Algorithm 1.1.16 Euclidean algorithm.

Input $a, b \in \mathbb{Z}$

Output gcd(a,b)

1. gcd(a,b) (Greatest common divisor)
2. $a \leftarrow |a|$
3. $b \leftarrow |b|$
4. while $b \neq 0$ do
   $r \leftarrow a \bmod b$
   $a \leftarrow b$
   $b \leftarrow r$
   end while.
   return a
   end

* Time and space complexity.

Time complexity:
running time of O(size(a)).

Size(a) —time→ output.

space complexity

Size(a) —space→ input.

memory space.

* probabilistic Algorithms.

(1) Monte Carlo algorithms: They always terminate but may not always be successful.

(2) Las Vegas algorithms. They may not terminate but when they terminate, they are successful.

Algorithm 1.2.2. (Monte carlo)

Input $k \in \mathbb{N}$

Output $s \in \{0,1\}^k$

1. random String (k)
2. for i:=0 to k-1 do
3. $s_i \leftarrow$ CoinToss
4. end for
5. return $\vec{s} = (s_0 \ldots s_{k-1})$
6. end

Algorithm 1.2.1o CLas vegas factoring).

Input. $a \in \mathbb{N}_{>1}$

Output. A proper divisor $b \in \mathbb{N}$ of a.

1. IvFactor (a)
2. repeat
3. $b \leftarrow$ mc Factor (a).
4. until $b \neq$ "Failure"
5. return b.
6. end.

* Analysis of probabilistic algorithms.

Let $k \in \mathbb{N}_0$, $k \leq l$, and let $\vec{r} = (r_0, \ldots r_{k-1})$ bea prefix of a random sequence of a run of A with input a.. Also, for $0 \leq i < k$ denote by p_i the probability for the return value r_i to occur. Then we see

$$Pr_{A,a}(\vec{r}) = \prod_{i=0}^{k-1} p_i \qquad \vec{r} \in \{0,1\}^k \quad Pr_{A,a}(\vec{r}) = \frac{1}{2^k}$$

We denote the set of all random sequences of runs of A with input a by Rand (A, a) and the set of finite string in Rand (A, a) by FR(A,a)

Lemma 1.3.2. Let $a \in$ Input(A). The sum

$$\sum_{\vec{r} \in FRand(A,a)} Pr_{A,a}(\vec{r})$$

converges in $[0,1]$. $Pr_{A,a}(\infty) = 1 - \sum_{\vec{r} \in FRand(A,a)} Pr_{A,a}(\vec{r}) = 0 \Rightarrow$

$(FRand(A,a), Pr_{A,a})$ is a discrete probability space.

P: polynomial time algorithm (deterministic).

NP: give a answer, can be verified in polynomial.

Example: (1742) Goldbanch conjecture: Every even integer ≥ 4 is the sum of two odd prime numbers.

$4=1+3 \quad 6=3+3 \quad 8=3+5$.

We give a; and prime number p. If a-p is also a prime number.

However, we can not prove Goldbanch conjecture in a polynomial time.

1. million $ $\Leftarrow$ P $\Leftrightarrow$ NP. seven millenium prize problem.

§1.2. Boolean $\Leftrightarrow$ Circuit.

Table 1.5.1

Name	Logic	circuit.
AND	$\wedge$	
OR	$\vee$	
NOT	$\neg$	
NAND	$\uparrow$	
NOR	$\downarrow$	
XOR	$\oplus$	

Theorem 1.5.7. The set {NOT, AND, OR} is universial for classical computation.

$\begin{cases} NOR = NOT(OR) \\ NAND = NOT(AND) \\ XOR = [(OR)] AND (NOT(AND)). \end{cases}$

Theorem 1.5.9 The gate set {NAND} is universial for classical computation.

$NOT = NAND(b,b) \quad AND(a,b) = NAND(NAND(a,b), NAND(a,b))$.

$NAND(NAND(b,b), NAND(a,a)) = OR(a,b)$.

$\Rightarrow$ {NOR} is universial for classical computation.

$NOR(NOR(a,b), NOR(a,b)) = NAND$.

Definition 1.5.1 Let G be a universal set of logic gates and let f be a Boolean function. The circuit size complexity of f with respect to G is the minimum size of any circuit that computes f.

Theorem 1.6.6 For all function $f: \{0,1\}^* \to \{0,1\}^*$, computation of problems CP_f and language L, there is a circuit family that computes f. solves CP_f or decides L.

Boolean $\Leftrightarrow$ circuit.

$\Downarrow$

Boolean $\Leftarrow$ quantum circuit.

★ 1.3. Reversible circuits.

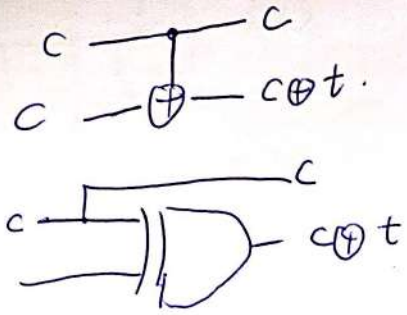
Section 4.7

CNOT
SWAP
CCNOT
FANOUT
Fredkin (CSWAP)

quantum

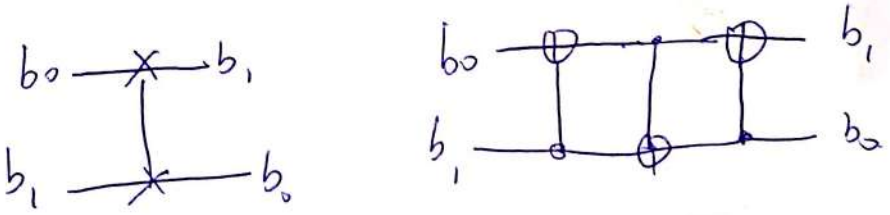
CNOT
SWAP
CCNOT
FANOUT
Fredkin

CNOT



NOT operation under control c.

SWAP



Proposition 1.7.2. Let $n \in \mathbb{N}$ and let $\pi \in S_n$. Then the map.

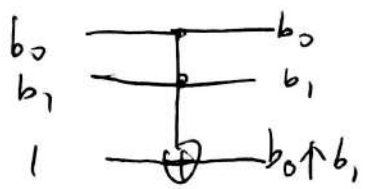
(1.7.1) $f_\pi: \{0,1\}^n \to \{0,1\}^n$, $(b_0, \dots b_{n-1}) \mapsto (b_{\pi(0)}, \dots b_{\pi(n-1)})$

Prof: Each permutation π is the product of at most $n-1$ transposition. □

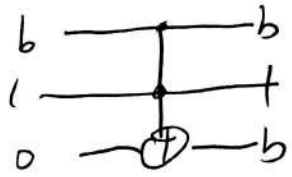
Toffoli or CCNOT: $\{0,1\}^3 \to \{0,1\}^3$ $(c_0, c_1, t) \mapsto (c_0, c_1, c_0 \wedge c_1 \oplus t)$

Use CCNOT to derive NAND and FAN OUT. gates.

(always b)



NAND

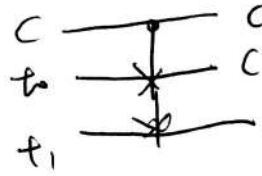


FANOUT

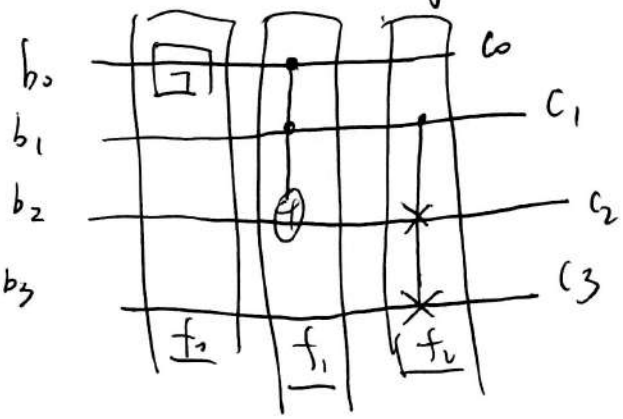
$(b \wedge 1) \oplus 0 = b \oplus 0 = (b \vee 0) \wedge \neg(b \wedge 0)$
$= b \vee 0 = b$

$((b_0 \wedge b_1) \vee 1) \wedge \neg(b_0 \wedge b_1) \wedge 1) = 1 \wedge \neg(b_0 \wedge b_1) = \neg(b_0 \wedge b_1)$
$= b_0 \uparrow b_1$

Fredkin (CSWAP). $\{0,1\}^3 \to \{0,1\}^3$ $(c, t_0, t_1) \mapsto (c, (\neg c \wedge t_0) \vee (c \wedge t_1), (c \wedge t_0) \vee (\neg c \wedge t_1))$



1.7 construction of reversible circuits.



$f = f_2 \circ f_1 \circ f_0$

$f_0 = (NOT, I, I, I)$

$f_1 = (CCNOT, I)$

$f_2 = (I, CSWAP)$.

$(0,1,0,0) \xrightarrow{f_0} (1,1,0,0) \xrightarrow{f_1} (1,1,1,0) \xrightarrow{f_2} (1,1,0,1)$

ancilla bits

1.7.3. f may not reversible.

find h reversible s.t. $h(y) = f(\vec{x})$, $\forall \vec{x} \in \{0,1\}^n$.

Theorem 1.7.6. For all Boolean function f, there exist. $a \in \{0,1\}^p$ $(p \le 2|f|_F)$

a reversible Cr (use CCNOT), $g : \{0,1\}^n \to \{0,1\}^{n+p-m}$,

(1.7.7) $h : \{0,1\}^n \times \{0,1\}^p \to \{0,1\}^m \times \{0,1\}^{n+p-m}$.

with (1.7.8) $h(\vec{x}, \vec{a}) = (f(\vec{x}), g(\vec{x}))$. → garbage

$|f|_F = 0$. $p=0$ $c_r = c$ $\vec{a} = ()$. $g : \{0,1\}^n \to \{0,1\}^0$, $\vec{x} \to ()$.

$|f|_F > 0$. $f : \{0,1\}^n \to \{0,1\}^m$, ~~$\exists$~~ $\forall f'$ $|f'|_F < k$. Th 1.7.10 holds

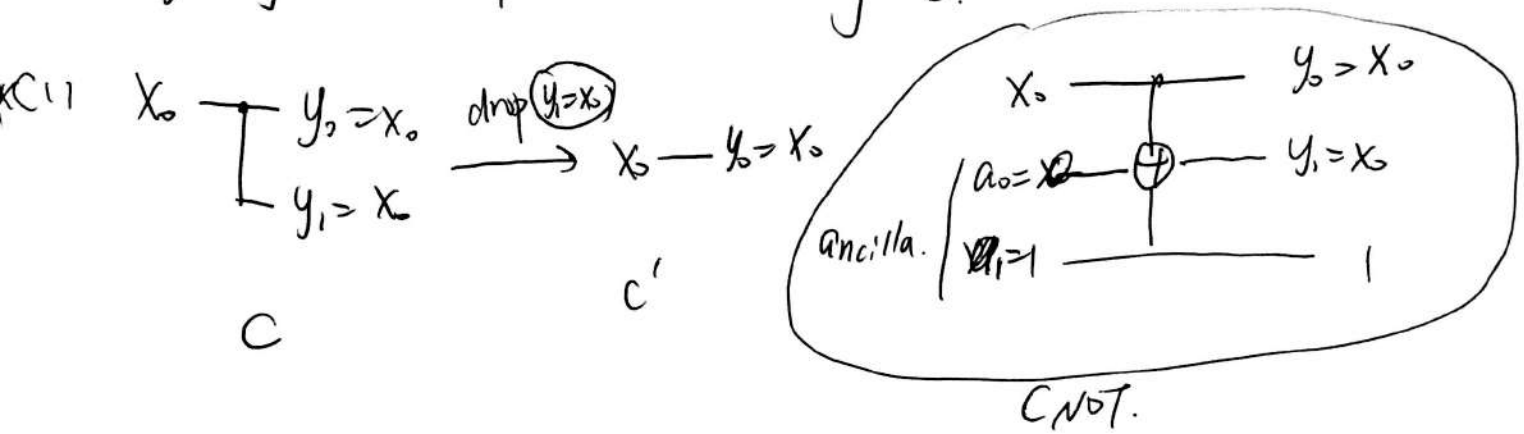
Let C be a circuit that implement f, use only NAND and FANOUT gates. $|C|_F = k$.

Then. C has ~~at~~ lease one of the following properties.

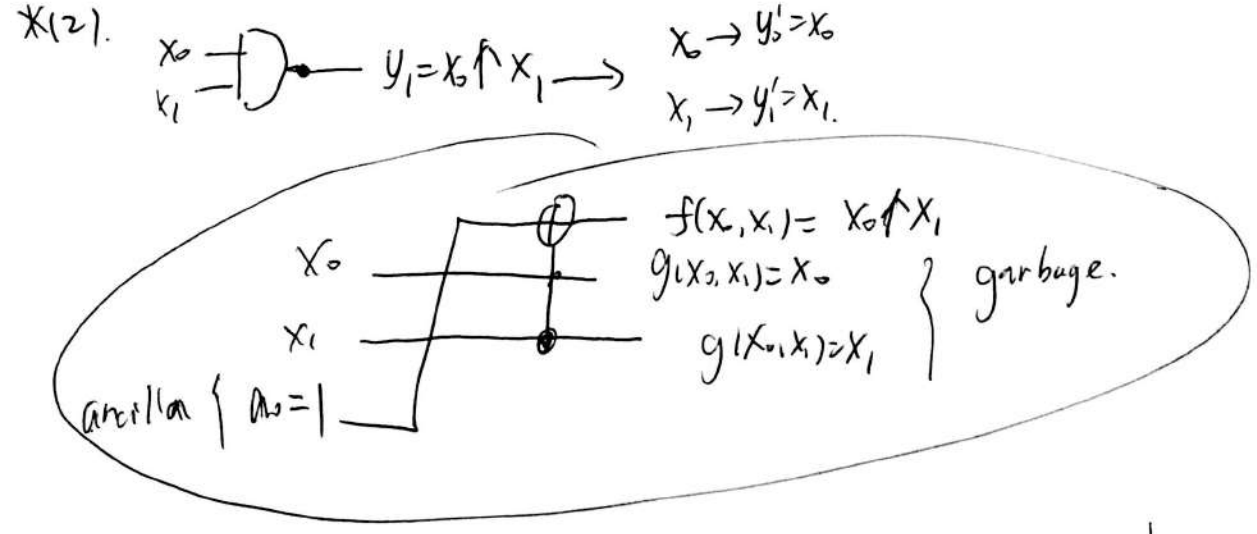
(1). $\exists$ FANOUT in C. whose out going edges are incoming edges of two output nodes y_i and y_j of C.

(2) $\exists$ NAND in C whose outgoing edges ~~is~~ the incoming edge of an output node y_i of C.

*(1)



*(2).



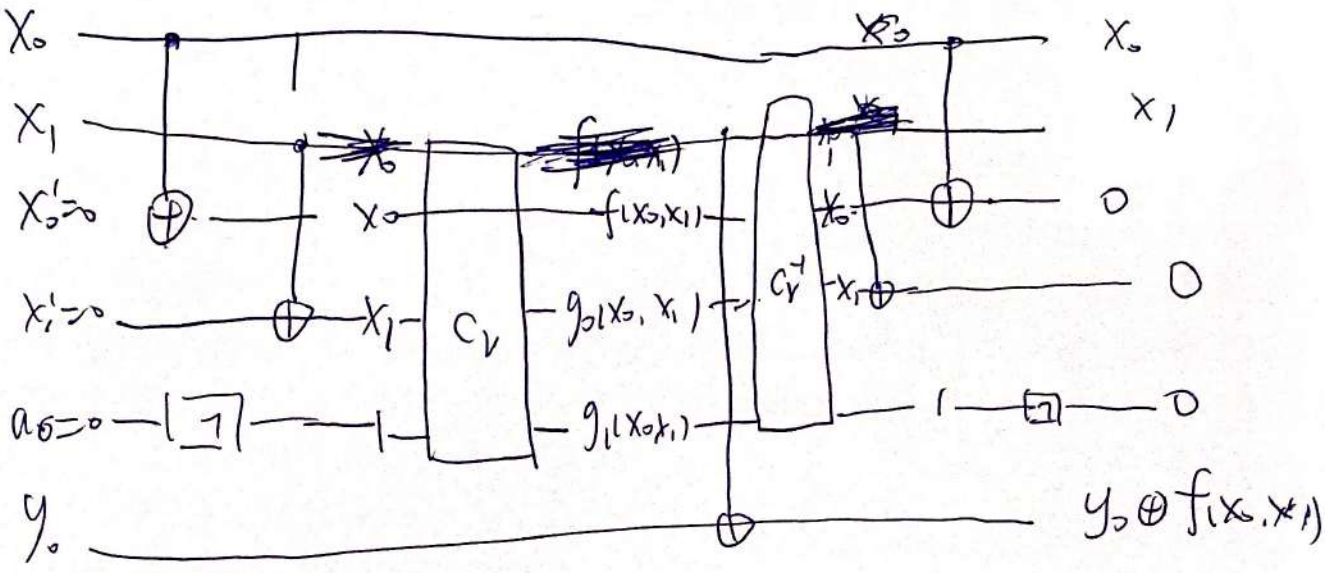
When we use construction from Th. 1.7.10., the garbage may be problematic.

uncompute trick

Theorem 1.7.12. For all Boolean function $f: \{0,1\}^n \to \{0,1\}^m$, there is $p \in \mathbb{N}_0$, $p \leq 2|f|_F$, a reversible Pf with $p_{r,1} = O(|f|_F)$ that use only Toffoli, NOT. and CNOT, such that Pf implement a function

(1.7.9) $h: \{0,1\}^n \times \{0,1\}^{n+p} \times \{0,1\}^m \to \{0,1\}^n \times \{0,1\}^{n+p} \times \{0,1\}^m$

with $h(\vec{x}, 0, \vec{y}) = (\vec{x}, \vec{0}, \vec{y} \oplus f(\vec{x}))$.

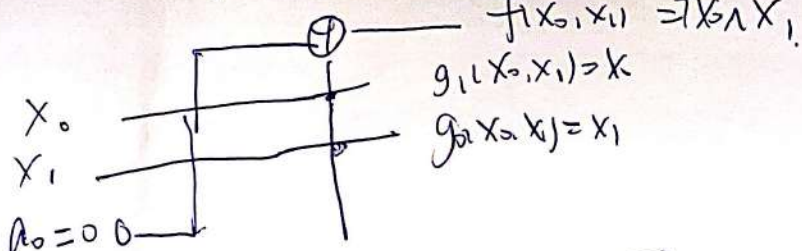


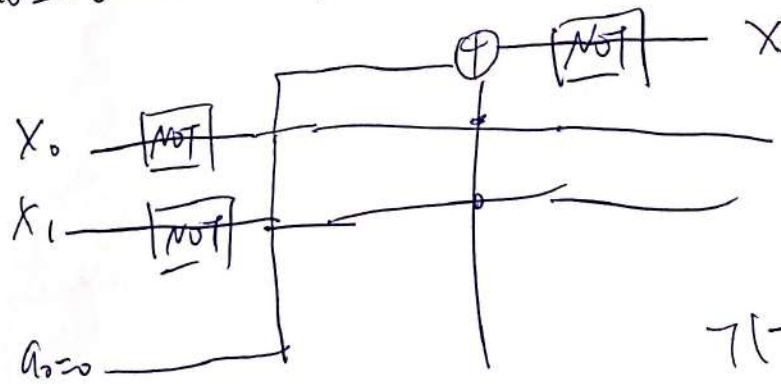
Note that $g \longrightarrow 0$ garbage (uncompute trick)

$g_1 \longrightarrow \varepsilon$

Exercise. $f(b_0, b_1) = b_0 \downarrow b_1$.

C_V





$\neg(\neg x_0 \wedge \neg x_1) = \neg(x_0 \vee x_1)$

$= x_0 \downarrow x_1$