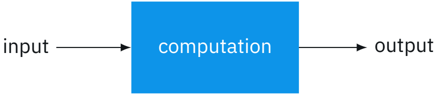
Overview

Contents

A standard picture of computation

A standard abstraction of computation looks like this:



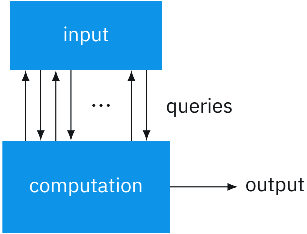
Different specific models of computation are studied, including *Turing machines* and *Boolean circuits.*

Key point

The *entire input* is provided to the computation — most typically as a string of bits — with nothing being hidden from the computation.

The query model of computation

In the query model of computation, the input is made available in the form of a *function,* which the computation accesses by making *queries.*



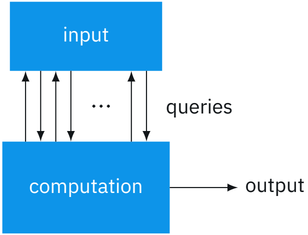
We often refer to the input as being provided by an *oracle* or *black box.*

The query model of computation

Throughout this lesson, the input to query problems is represented by a function

$$f : \Sigma^n \rightarrow \Sigma^m$$

where n and m are positive integers and $\Sigma = \{0, 1\}$.



Queries

To say that a computation ***makes a query*** means that it evaluates the function f once: $x \in \Sigma^n$ is selected, and the string $f(x) \in \Sigma^m$ is made available to the computation.

We measure the efficiency of query algorithms by counting the ***number of queries*** to the input they require.

Examples of query problems

Or

Input: $f : \Sigma^n \rightarrow \Sigma$

Output: 1 if there exists a string $x \in \Sigma^n$ for which $f(x) = 1$
0 if there is no such string

Parity

Input: $f : \Sigma^n \rightarrow \Sigma$

Output: 0 if $f(x) = 1$ for an even number of strings $x \in \Sigma^n$
1 if $f(x) = 1$ for an odd number of strings $x \in \Sigma^n$

Minimum

Input: $f : \Sigma^n \rightarrow \Sigma^m$

Output: The string $y \in \{f(x) : x \in \Sigma^n\}$ that comes first in the lexicographic ordering of Σ^m

Examples of query problems

Sometimes we also consider query problems where we have a *promise* on the input. Inputs that don't satisfy the promise are considered as "don't care" inputs.

Unique search

Input:	$f : \Sigma^n \rightarrow \Sigma$
Promise:	There is exactly one string $z \in \Sigma^n$ for which $f(z) = 1$, with $f(x) = 0$ for all strings $x \neq z$
Output:	The string z

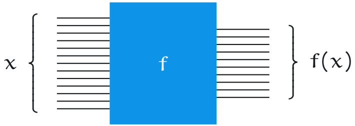
Or, Parity, Minimum, and Unique search are all very "natural" examples of query problems — but some query problems of interest aren't like this.

We sometimes consider very complicated and highly contrived problems, to look for extremes that reveal potential advantages of quantum computing.

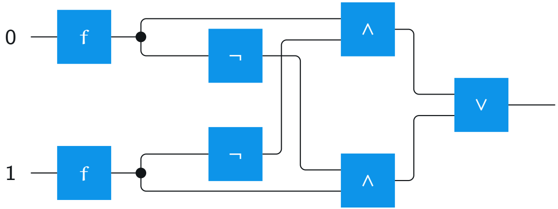
Query gates

For circuit models of computation, queries are made by *query gates.*

For Boolean circuits, query gates generally compute the input function f directly.



For example, the following circuit computes Parity for every $f : \Sigma \rightarrow \Sigma$.



Query gates

For the quantum circuit model, we choose a different definition for query gates that makes them *unitary* — allowing them to be applied to quantum states.

Definition

The *query gate* U_f for any function $f : \Sigma^n \rightarrow \Sigma^m$ is defined as

$$U_f(|y\rangle|x\rangle) = |y \oplus f(x)\rangle|x\rangle$$

for all $x \in \Sigma^n$ and $y \in \Sigma^m$.

Notation

The string $y \oplus f(x)$ is the *bitwise XOR* of y and $f(x)$. For example:

$$001 \oplus 101 = 100$$

Query gates

For the quantum circuit model, we choose a different definition for query gates that makes them *unitary* — allowing them to be applied to quantum states.

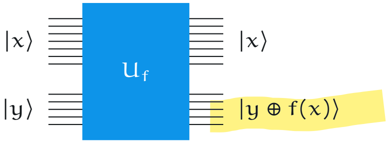
Definition

The *query gate* U_f for any function $f : \Sigma^n \rightarrow \Sigma^m$ is defined as

$$U_f(|y\rangle|x\rangle) = |y \oplus f(x)\rangle|x\rangle$$

for all $x \in \Sigma^n$ and $y \in \Sigma^m$.

In circuit diagrammatic form U_f operates like this:



will get permutation matrix

$|0^m\rangle$ = 0000... the amount of m

This gate is always unitary, for any choice of the function f.

Deutsch's problem

Deutsch's problem is very simple — it's the Parity problem for functions of the form $f : \Sigma \rightarrow \Sigma$.

There are four functions of the form $f : \Sigma \rightarrow \Sigma$:

a	$f_1(a)$
0	0
1	0

a	$f_2(a)$
0	0
1	1

a	$f_3(a)$
0	1
1	0

a	$f_4(a)$
0	1
1	1

The functions f_1 and f_4 are *constant* while f_2 and f_3 are *balanced.*

Deutsch's problem

Input: $f : \Sigma \rightarrow \Sigma$

Output: 0 if f is constant, 1 if f is balanced

Deutsch's problem

Deutsch's problem

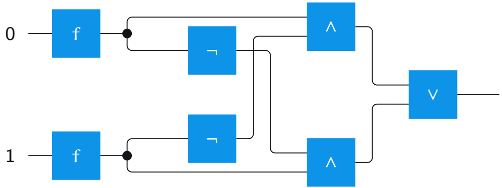
Input: $f : \Sigma \to \Sigma$

Output: 0 if f is constant, 1 if f is balanced

if we only know mapping 0 -> x; and we
don't know the mapping 1 -> ?
then we don't know the parity of the function

Every *classical* query algorithm must make 2 queries to f to solve this problem — learning just one of two bits provides no information about their parity.

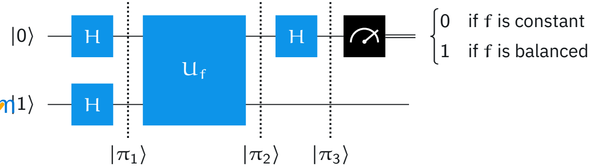
Our query algorithm from earlier is therefore optimal among classical query algorithms for this problem.



Deutsch's algorithm

Deutsch's algorithm solves Deutsch's problem using a single query.

read from right to left; top to bottom



distribute |->

$$|\pi_1\rangle = |-\rangle|+\rangle = \frac{1}{2}(|0\rangle - |1\rangle)|0\rangle + \frac{1}{2}(|0\rangle - |1\rangle)|1\rangle$$

$$\begin{aligned}|\pi_2\rangle &= \frac{1}{2}(|0 \oplus f(0)\rangle - |1 \oplus f(0)\rangle)|0\rangle + \frac{1}{2}(|0 \oplus f(1)\rangle - |1 \oplus f(1)\rangle)|1\rangle \\ &= \frac{1}{2}(-1)^{f(0)}(|0\rangle - |1\rangle)|0\rangle + \frac{1}{2}(-1)^{f(1)}(|0\rangle - |1\rangle)|1\rangle \\ &= |-\rangle\left(\frac{(-1)^{f(0)}|0\rangle + (-1)^{f(1)}|1\rangle}{\sqrt{2}}\right)\end{aligned}$$

phase kickback, why we have |1> at the bottom

$|0\oplus a\rangle - |1\oplus a\rangle$

$= (-1)^a(|0\rangle - |1\rangle)$

If $a = 0$ $|0\oplus 0\rangle - |1\oplus 0\rangle$

$= (-1)^0 |0\rangle - |1\rangle$ ✓

If $a = 1$: $|0\oplus 1\rangle - |1\oplus 1\rangle$

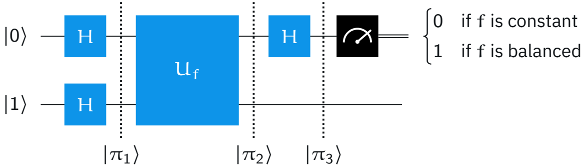
$= |1\rangle - |0\rangle$

$(-1)^1 (|0\rangle - |1\rangle)$

$-|0\rangle + |1\rangle$ ✓

Deutsch's algorithm

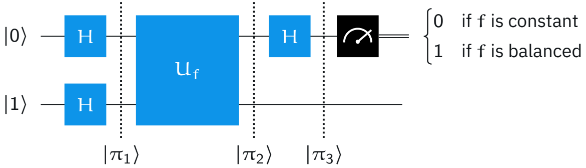
Deutsch's algorithm solves Deutsch's problem using a single query.



$$
\begin{aligned}
|\pi_2\rangle &= |-\rangle \left(\frac{(-1)^{f(0)}|0\rangle + (-1)^{f(1)}|1\rangle}{\sqrt{2}} \right) \\
&= (-1)^{f(0)}|-\rangle \left(\frac{|0\rangle + (-1)^{f(0)\oplus f(1)}|1\rangle}{\sqrt{2}} \right) \\
&= \begin{cases} (-1)^{f(0)}|-\rangle|+\rangle & f(0)\oplus f(1) = 0 \\ (-1)^{f(0)}|-\rangle|-\rangle & f(0)\oplus f(1) = 1 \end{cases}
\end{aligned}
$$

Deutsch's algorithm

Deutsch's algorithm solves Deutsch's problem using a single query.



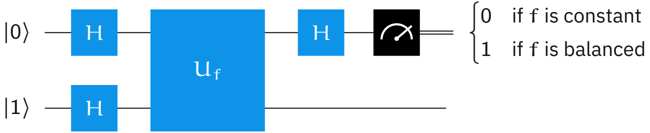
$$|\pi_2\rangle = \begin{cases} (-1)^{f(0)}|-\rangle|+\rangle & f(0) \oplus f(1) = 0 \\ (-1)^{f(0)}|-\rangle|-\rangle & f(0) \oplus f(1) = 1 \end{cases}$$

$$|\pi_3\rangle = \begin{cases} (-1)^{f(0)}|-\rangle|0\rangle & f(0) \oplus f(1) = 0 \\ (-1)^{f(0)}|-\rangle|1\rangle & f(0) \oplus f(1) = 1 \end{cases}$$

$$= (-1)^{f(0)}|-\rangle|f(0) \oplus f(1)\rangle$$

interference

Phase kickback



$$|b \oplus c\rangle = X^c|b\rangle$$

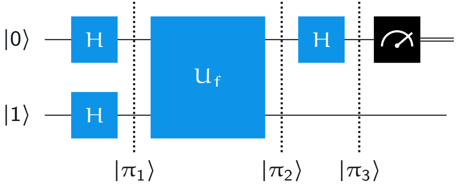
$$U_f(|b\rangle|a\rangle) = |b \oplus f(a)\rangle|a\rangle = (X^{f(a)}|b\rangle)|a\rangle$$

$$U_f(|-\rangle|a\rangle) = (X^{f(a)}|-\rangle)|a\rangle = (-1)^{f(a)}|-\rangle|a\rangle$$

$$U_f(|-\rangle|a\rangle) = (-1)^{f(a)}|-\rangle|a\rangle \quad \longleftarrow$$

phase kickback

Phase kickback



$$\begin{cases} 0 & \text{if } f \text{ is constant} \\ 1 & \text{if } f \text{ is balanced} \end{cases}$$

X |-> = - |->
negative 1 minus |-> state
minus state = an eigenvector of the X operation, with eigen value equal to negative 1.

$$U_f(|-\rangle|a\rangle) = (-1)^{f(a)}|-\rangle|a\rangle \quad \longleftarrow$$ phase kickback

$$|\pi_1\rangle = |-\rangle|+\rangle$$

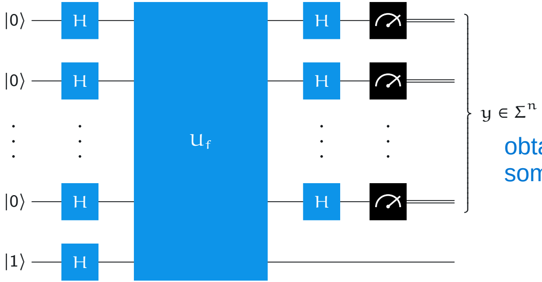
$$|\pi_2\rangle = U_f(|-\rangle|+\rangle) = \frac{1}{\sqrt{2}}U_f(|-\rangle|0\rangle) + \frac{1}{\sqrt{2}}U_f(|-\rangle|1\rangle)$$

$$= |-\rangle\left(\frac{(-1)^{f(0)}|0\rangle + (-1)^{f(1)}|1\rangle}{\sqrt{2}}\right)$$

The Deutsch-Jozsa circuit

The Deutsch-Jozsa algorithm extends Deutsch's algorithm to input functions of the form $f : \Sigma^n \rightarrow \Sigma$ for any $n \geq 1$.

The quantum circuit for the Deutsch-Jozsa algorithm looks like this:



obtain some n bit-string y, that tells us something about the function f

We can, in fact, use this circuit to solve multiple problems.

gain some information about f, that can potentially help to solve query problems where f is the input

The Deutsch-Jozsa problem

The Deutsch-Jozsa problem generalizes Deutsch's problem: for an input function $f : \Sigma^n \rightarrow \Sigma$, the task is to output 0 if f is constant and 1 if f is balanced.

When $n \geq 2$, some functions $f : \Sigma^n \rightarrow \Sigma$ are neither constant nor balanced.

Example

This function is neither constant nor balanced:

x	$f(x)$
00	0
01	0
10	0
11	1

Input functions that are neither constant nor balanced are "don't care" inputs.

The Deutsch-Jozsa problem

The Deutsch-Jozsa problem generalizes Deutsch's problem: for an input function $f : \Sigma^n \rightarrow \Sigma$, the task is to output 0 if f is constant and 1 if f is balanced.

Deutsch-Jozsa problem

Input:	$f : \Sigma^n \rightarrow \Sigma$
Promise:	f is either constant or balanced
Output:	0 if f is constant, 1 if f is balanced

we're promised that the input function f is either constant or balanced, and the goal is to figure out which one it is.

we're not responsible for what happens if the input doesn't meet the promise.

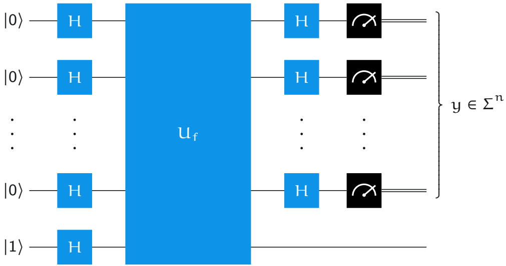
The Deutsch-Jozsa problem

Deutsch-Jozsa problem

Input: $f : \Sigma^n \rightarrow \Sigma$

Promise: f is either constant or balanced

Output: 0 if f is constant, 1 if f is balanced



Output: 0 if $y = 0^n$ and 1 otherwise.

if y = all zero string

Deutsch-Jozsa analysis

The Hadamard operation works like this on standard basis states:

$$H|0\rangle = \frac{1}{\sqrt{2}}|0\rangle + \frac{1}{\sqrt{2}}|1\rangle$$

$$H|1\rangle = \frac{1}{\sqrt{2}}|0\rangle - \frac{1}{\sqrt{2}}|1\rangle$$

We can express these two equations as one:

$$H|a\rangle = \frac{1}{\sqrt{2}}|0\rangle + \frac{1}{\sqrt{2}}(-1)^{a}|1\rangle = \frac{1}{\sqrt{2}}\sum_{b\in\{0,1\}}(-1)^{ab}|b\rangle$$

Deutsch-Jozsa analysis

The Hadamard operation works like this on standard basis states:

$$H|a\rangle = \frac{1}{\sqrt{2}} \sum_{b\in\{0,1\}} (-1)^{ab} |b\rangle$$

Now suppose we perform a Hadamard operation on each of n qubits:

^ tensor n = n-fold tensor product of H with itself = n copies of H all tensor together

$$\begin{aligned}
&H^{\otimes n}|x_{n-1}\cdots x_1 x_0\rangle \\
&\quad = \big(H|x_{n-1}\rangle\big) \otimes \cdots \otimes \big(H|x_0\rangle\big)
\end{aligned}$$

n-fold of Hadamard gate applied to each qubit

$$\begin{aligned}
&\quad = \left(\frac{1}{\sqrt{2}} \sum_{y_{n-1}\in\Sigma} (-1)^{x_{n-1}y_{n-1}} |y_{n-1}\rangle\right) \otimes \cdots \otimes \left(\frac{1}{\sqrt{2}} \sum_{y_0\in\Sigma} (-1)^{x_0 y_0} |y_0\rangle\right) \\
&\quad = \frac{1}{\sqrt{2^n}} \sum_{y_{n-1}\cdots y_0\in\Sigma^n} (-1)^{x_{n-1}y_{n-1}+\cdots+x_0 y_0} |y_{n-1}\cdots y_0\rangle
\end{aligned}$$

Deutsch-Jozsa analysis

$$H^{\otimes n}|x_{n-1}\cdots x_1 x_0\rangle = \frac{1}{\sqrt{2^n}} \sum_{y_{n-1}\cdots y_0\in\Sigma^n} (-1)^{x_{n-1}y_{n-1}+\cdots+x_0y_0}|y_{n-1}\cdots y_0\rangle$$

$$H^{\otimes n}|x\rangle = \frac{1}{\sqrt{2^n}} \sum_{y\in\Sigma^n} (-1)^{x\cdot y}|y\rangle$$

Binary dot product

For binary strings $x = x_{n-1}\cdots x_0$ and $y = y_{n-1}\cdots y_0$ we define

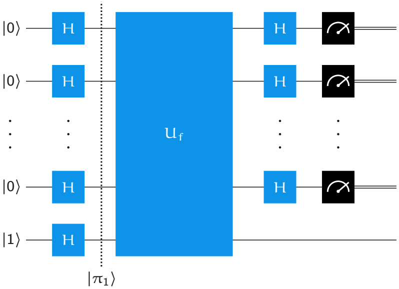
dot product = parity of individual bits / XOR of the logical and of the individual bits

$$x \cdot y = x_{n-1}y_{n-1} \oplus \cdots \oplus x_0y_0$$

$$= \begin{cases} 1 & \text{if } x_{n-1}y_{n-1} + \cdots + x_0y_0 \text{ is odd} \\ 0 & \text{if } x_{n-1}y_{n-1} + \cdots + x_0y_0 \text{ is even} \end{cases}$$

Deutsch-Jozsa analysis

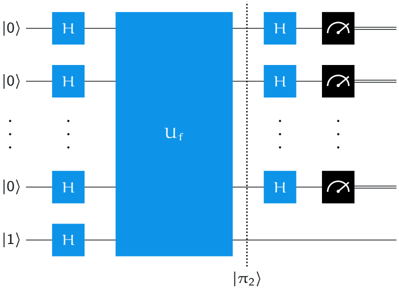
$$H^{\otimes n}|x\rangle = \frac{1}{\sqrt{2^n}} \sum_{y \in \Sigma^n} (-1)^{x \cdot y} |y\rangle$$



$$|\pi_1\rangle = |-\rangle \otimes \frac{1}{\sqrt{2^n}} \sum_{x \in \Sigma^n} |x\rangle$$

Deutsch-Jozsa analysis

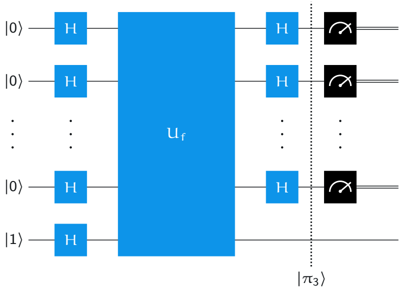
$$H^{\otimes n}|x\rangle = \frac{1}{\sqrt{2^n}} \sum_{y \in \Sigma^n} (-1)^{x \cdot y} |y\rangle$$



$$|\pi_2\rangle = |-\rangle \otimes \frac{1}{\sqrt{2^n}} \sum_{x \in \Sigma^n} (-1)^{f(x)} |x\rangle$$

Deutsch-Jozsa analysis

$$H^{\otimes n}|x\rangle = \frac{1}{\sqrt{2^n}} \sum_{y\in\Sigma^n} (-1)^{x\cdot y}|y\rangle$$



If we see that every measurement outcome is 0, the function is constant.
If any one of the measurements is 1, the function is balanced.

$$|\pi_3\rangle = |-\rangle \otimes \frac{1}{2^n} \sum_{y\in\Sigma^n} \sum_{x\in\Sigma^n} (-1)^{f(x)+x\cdot y}|y\rangle$$

Deutsch-Jozsa analysis

The probability for the measurements to give $y = 0^n$ is

$$p(0^n) = \left| \frac{1}{2^n} \sum_{x \in \Sigma^n} (-1)^{f(x)} \right|^2 = \begin{cases} 1 & \text{if } f \text{ is constant} \\ 0 & \text{if } f \text{ is balanced} \end{cases}$$

The Deutsch-Jozsa algorithm therefore solves the Deutsch-Jozsa problem without error with a single query.

Any *deterministic* algorithm for the Deutsch-Jozsa problem must at least $2^{n-1} + 1$ queries. because we have to query the function on more of the half of inputs to be sure of the solution.

A *probabilistic* algorithm can, however, solve the Deutsch-Jozsa problem using just a few queries:

1. Choose k input strings $x^1, \ldots, x^k \in \Sigma^n$ uniformly at random.
2. If $f(x^1) = \cdots = f(x^k)$, then answer 0 (constant), else answer 1 (balanced).

If f is constant, this algorithm is correct with probability 1.

If f is balanced, this algorithm is correct with probability $1 - 2^{-k+1}$.

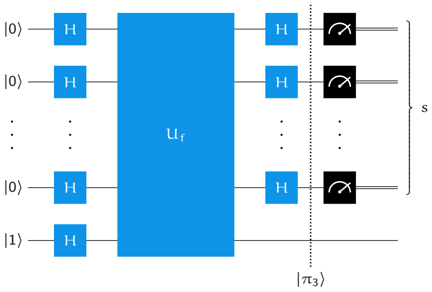
The Bernstein-Vazirani problem

Bernstein-Vazirani problem

Input: $f : \Sigma^n \rightarrow \Sigma$

Promise: there exists a binary string $s = s_{n-1} \cdots s_0$ for which $f(x) = s \cdot x$ for all $x \in \Sigma^n$

Output: the string s



The Bernstein-Vazirani problem

$$
\begin{aligned}
|\pi_3\rangle &= |-\rangle \otimes \frac{1}{2^n} \sum_{y\in\Sigma^n} \sum_{x\in\Sigma^n} (-1)^{f(x)+x\cdot y} |y\rangle \\
&= |-\rangle \otimes \frac{1}{2^n} \sum_{y\in\Sigma^n} \sum_{x\in\Sigma^n} (-1)^{s\cdot x+y\cdot x} |y\rangle \\
&= |-\rangle \otimes \frac{1}{2^n} \sum_{y\in\Sigma^n} \sum_{x\in\Sigma^n} (-1)^{(s\oplus y)\cdot x} |y\rangle \\
&= |-\rangle \otimes |s\rangle
\end{aligned}
$$

The Deutsch-Jozsa circuit therefore solves the Bernstein-Vazirani problem with a single query.

Any probabilistic algorithm must make at least n queries to find s.

Simon's problem

Simon's problem

Input: A function $f : \Sigma^n \to \Sigma^m$

the input: f maps n bit string to m bit string, where m is some positive integer that isn't equal to 1 in general

Promise: There exists a string $s \in \Sigma^n$ such that

hidden string s

$$[f(x) = f(y)] \Leftrightarrow [(x = y) \text{ or } (x \oplus s = y)]$$

for all $x, y \in \Sigma^n$

there exists string s having length equal to n, such that a particular "if" and "only if" condition is true for all choices of all n-bit string x and y

Output: The string s

f of x is equal to f of y, iff. one of two possibilities holds:
1. x = y
2. x with hidden string s = y

Case 1: $s = 0^n$

The condition in the promise simplifies to

$$[f(x) = f(y)] \Leftrightarrow [x = y]$$

This is equivalent to f being *one-to-one.*

Simon's problem

Case 2: $s \neq 0^n$

The function f must be *two-to-one* to satisfy the promise:

$$f(x) = f(x \oplus s)$$

with *distinct output strings* for each pair.

x	$f(x)$
000	10011
001	00101
010	00101
011	10011
100	11010
101	00001
110	00001
111	11010

$$s = 011$$

$$f(000) = f(011) = 10011$$

$$f(001) = f(010) = 00101$$

$$f(100) = f(111) = 11010$$

$$f(101) = f(110) = 00001$$

$000 \oplus 011 = 011$

$011 \oplus 011 = 000$

$001 \oplus 011 = 010$

$010 \oplus 011 = 001$

bitwise input

$000 \oplus 011$

$011 = s$

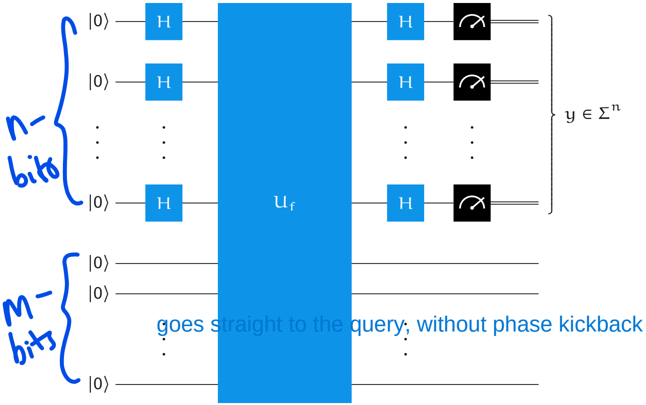
001
010

$011 = s$

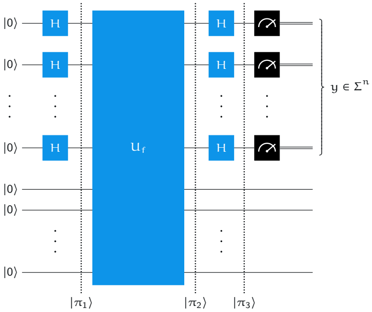
only one s, satisfy this problem

Simon's algorithm

Simon's algorithm consists of running the following circuit several times, followed by a post-processing step.



Simon's algorithm analysis



$$|\pi_1\rangle = \frac{1}{\sqrt{2^n}} \sum_{x\in\Sigma^n} |0^m\rangle|x\rangle$$

$$|\pi_2\rangle = \frac{1}{\sqrt{2^n}} \sum_{x\in\Sigma^n} |f(x)\rangle|x\rangle$$

$$|\pi_3\rangle = \frac{1}{\sqrt{2^n}} \sum_{x\in\Sigma^n} |f(x)\rangle \otimes \left(\frac{1}{\sqrt{2^n}} \sum_{y\in\Sigma^n} (-1)^{x\cdot y}|y\rangle\right) = \frac{1}{2^n} \sum_{y\in\Sigma^n} \sum_{x\in\Sigma^n} (-1)^{x\cdot y}|f(x)\rangle|y\rangle$$

Simon's algorithm analysis

1:02:37

$$\frac{1}{2^n}\sum_{y\in\Sigma^n}\sum_{x\in\Sigma^n}(-1)^{x\cdot y}|f(x)\rangle|y\rangle$$

range(f) = { f(x) : x ∈ Σⁿ }

f⁻¹({z}) = { x ∈ Σⁿ : f(x) = z }

$$p(y) = \left\| \frac{1}{2^n}\sum_{x\in\Sigma^n}(-1)^{x\cdot y}|f(x)\rangle \right\|^2$$

$$= \left\| \frac{1}{2^n}\sum_{z\in\text{range}(f)}\left(\sum_{x\in f^{-1}(z)}(-1)^{x\cdot y}\right)|z\rangle \right\|^2$$

$$= \frac{1}{2^{2n}}\sum_{z\in\text{range}(f)}\left|\sum_{x\in f^{-1}(\{z\})}(-1)^{x\cdot y}\right|^2$$

$$\text{range}(f) = \{f(x) : x\in\Sigma^n\}$$

$$f^{-1}(\{z\}) = \{x\in\Sigma^n : f(x) = z\}$$

Simon's algorithm analysis

$$p(y) = \frac{1}{2^{2n}} \sum_{z \in \mathrm{range}(f)} \left| \sum_{x \in f^{-1}(\{z\})} (-1)^{x \cdot y} \right|^2$$

Case 1: $s = 0^n$

Because f is a one-to-one, there a single element $x \in f^{-1}(\{z\})$ for every $z \in \mathrm{range}(f)$:

$$\left| \sum_{x \in f^{-1}(\{z\})} (-1)^{x \cdot y} \right|^2 = 1$$

There are 2^n elements in range(f), so

because f one-to-one

$$p(y) = \frac{1}{2^{2n}} \cdot 2^n = \frac{1}{2^n}$$

(for every $y \in \Sigma^n$).

Simon's algorithm analysis

$$p(y) = \frac{1}{2^{2n}} \sum_{z \in \text{range}(f)} \left| \sum_{x \in f^{-1}(\{z\})} (-1)^{x \cdot y} \right|^2$$

Case 2: $s \neq 0^n$

There are two strings in the set $f^{-1}(\{z\})$ for each $z \in \text{range}(f)$; if $w \in f^{-1}(\{z\})$ either one of them, then $w \oplus s$ is the other.

$$\left| \sum_{x \in f^{-1}(\{z\})} (-1)^{x \cdot y} \right|^2 = \left| (-1)^{w \cdot y} + (-1)^{(w \oplus s) \cdot y} \right|^2 = \left| 1 + (-1)^{s \cdot y} \right|^2 = \begin{cases} 4 & s \cdot y = 0 \\ 0 & s \cdot y = 1 \end{cases}$$

There are 2^{n-1} elements in $\text{range}(f)$, so

$$p(y) = \frac{1}{2^{2n}} \sum_{z \in \text{range}(f)} \left| \sum_{x \in f^{-1}(\{z\})} (-1)^{x \cdot y} \right|^2 = \begin{cases} \frac{1}{2^{n-1}} & s \cdot y = 0 \\ 0 & s \cdot y = 1 \end{cases}$$

Classical post-processing

Running the circuit from Simon's algorithm one time gives us a random string $y \in \Sigma^n$.

Case 1: $s = 0^n$

$$p(y) = \frac{1}{2^n}$$

hidden string (s) is all-zero string
we obtain a completely random string from the measurements, where each end bit string y is equally likely.

Case 2: $s \neq 0^n$

$$p(y) = \begin{cases} \frac{1}{2^{n-1}} & s \cdot y = 0 \\ 0 & y \cdot s = 1 \end{cases}$$

the function f satisfy for a different string s (not equal to the all-zero string), then we get random string y having binary dot product equal to 0 with s, each is equally likely.

Suppose we run the circuit independently $k = n + r$ times, obtaining strings $y^1, \ldots, y^k$.

$$\begin{aligned} y^1 &= y^1_{n-1} \cdots y^1_0 \\ y^2 &= y^2_{n-1} \cdots y^2_0 \\ &\vdots \\ y^k &= y^k_{n-1} \cdots y^k_0 \end{aligned} \qquad M = \begin{pmatrix} y^1_{n-1} & \cdots & y^1_0 \\ y^2_{n-1} & \cdots & y^2_0 \\ \vdots & \ddots & \vdots \\ y^k_{n-1} & \cdots & y^k_0 \end{pmatrix} \qquad M \begin{pmatrix} s_{n-1} \\ \vdots \\ s_0 \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ \vdots \\ 0 \end{pmatrix}$$

Using *Gaussian elimination* we can efficiently compute the *null space* (modulo 2) of M.
With probability greater than $1 - 2^{-r}$ it will be $\{0^n, s\}$.

?

Classical difficulty

Any probabilistic algorithm making fewer than $2^{n/2-1} - 1$ queries will fail to solve Simon's problem with probability at least $1/2$.

- Simon's algorithm solves Simon's problem with a *linear* number of queries.
- Every classical algorithm for Simon's problem requires an *exponential* number of queries.