6

Algorithms

Computer Science is no more about computers than astronomy is about telescopes.

E.W. Dijkstra

Algorithms are often developed long before the machines they are supposed to run on. Classical algorithms predate classical computers by millennia, and similarly, there exist several quantum algorithms before any large-scale quantum computers have seen the light of day. These algorithms manipulate qubits to solve problems and, in general, they solve these tasks more efficiently than classical computers.

Rather than describing the quantum algorithms in the chronological order in which they were discovered, we choose to present them in order of increasing difficulty. The core ideas of each algorithm are based on previous ones. We start at tutorial pace, introducing new concepts in a thorough way. Section 6.1 describes Deutsch's algorithm that determines a property of functions from $\{0, 1\}$ to $\{0, 1\}$. In Section 6.2 we generalize this algorithm to the Deutsch–Jozsa algorithm, which deals with a similar property for functions from $\{0, 1\}^n$ to $\{0, 1\}$. Simon's periodicity algorithm is described in Section 6.3. Here we determine patterns of a function from $\{0, 1\}^n$ to $\{0, 1\}^n$. Section 6.4 goes through Grover's search algorithm that can search an unordered array of size n in $\sqrt{n}$ time as opposed to the usual n time. The chapter builds up to the ground-breaking Shor's factoring algorithm done in Section 6.5. This quantum algorithm can factor numbers in polynomial time. There are no known classical algorithms that can perform this feat in such time.

Reader Tip. This chapter may be a bit overwhelming on the first reading. After reading Section 6.1, the reader can move on to Section 6.2 or Section 6.4. Shor's algorithm can safely be read after Section 6.2. ♡

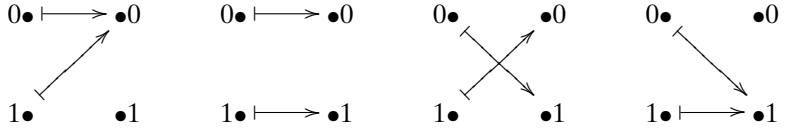
6.1 DEUTSCH'S ALGORITHM

All quantum algorithms work with the following basic framework:

- The system will start with the qubits in a particular classical state.
- From there the system is put into a superposition of many states.
- This is followed by acting on this superposition with several unitary operations.
- And finally, a measurement of the qubits.

Of course, there will be several variations of this theme. Nevertheless, it will be helpful to keep this general scheme in mind as we proceed.

The simplest quantum algorithm is Deutsch's algorithm, which is a nice algorithm that solves a slightly contrived problem. This algorithm is concerned with functions from the set $\{0, 1\}$ to the set $\{0, 1\}$. There are four such functions that might be visualized as

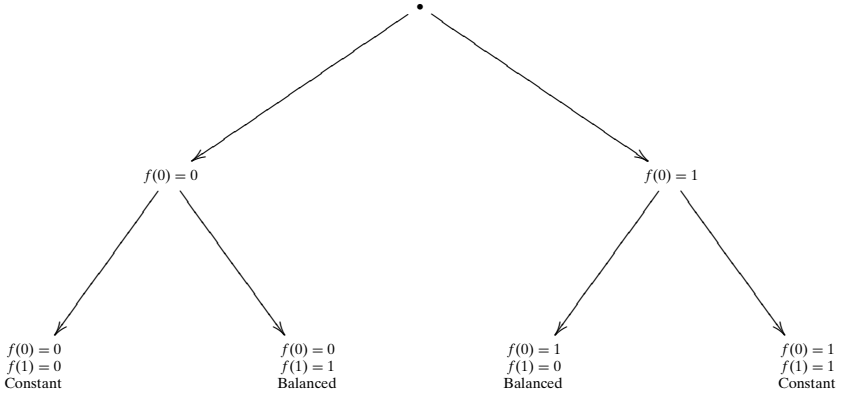


(6.1)

Call a function $f : \{0, 1\} \longrightarrow \{0, 1\}$ **balanced** if $f(0) \neq f(1)$, i.e., it is one to one. In contrast, call a function **constant** if $f(0) = f(1)$. Of the four functions, two are balanced and two are constant.

Deutsch's algorithm solves the following problem: Given a function $f : \{0, 1\} \longrightarrow \{0, 1\}$ as a black box, where one can evaluate an input, but cannot "look inside" and "see" how the function is defined, determine if the function is balanced or constant.

With a classical computer, one would have to first evaluate f on one input, then evaluate f on the second input, and finally, compare the outputs. The following decision tree shows what a classical computer must do:

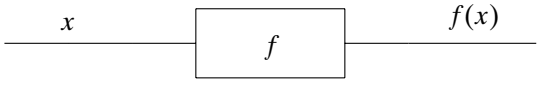


(6.2)

The point is that with a classical computer, f must be evaluated twice. Can we do better with a quantum computer?

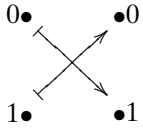
A quantum computer can be in a superposition of two basic states at the same time. We shall use this superposition of states to evaluate both inputs at one time.

In classical computing, evaluating a given function f corresponds to performing the following operation:



(6.3)

As we discussed in Chapter 5, such a function can be thought of as a matrix acting on the input. For instance, the function



(6.4)

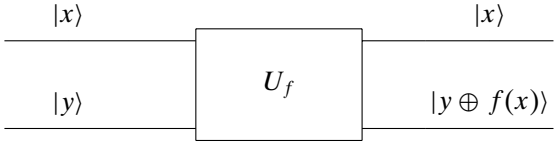
is equivalent to the matrix

$$\begin{array}{c} \\ \mathbf{0} \\ \mathbf{1} \end{array} \begin{array}{c} \begin{array}{cc} \mathbf{0} & \mathbf{1} \end{array} \\ \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \end{array}. \tag{6.5}$$

Multiplying state $|0\rangle$ on the right of this matrix would result in state $|1\rangle$, and multiplying state $|1\rangle$ on the right of this matrix would result in state $|0\rangle$. The column name is to be thought of as the input and the row name as the output.

Exercise 6.1.1 Describe the matrices for the other three functions from $\{0, 1\}$ to $\{0, 1\}$. ■

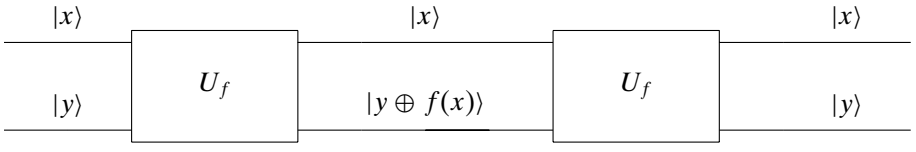
However, this will not be enough for a quantum system. Such a system demands a little something extra: every gate must be unitary (and thus reversible). Given the output, we must be able to find the input. If f is the name of the function, then the following black-box U_f will be the quantum gate that we shall employ to evaluate input:



(6.6)

The top input, $|x\rangle$, will be the qubit value that one wishes to evaluate and the bottom input, $|y\rangle$, controls the output. The top output will be the same as the input qubit $|x\rangle$ and the bottom output will be the qubit $|y \oplus f(x)\rangle$, where $\oplus$ is XOR, the exclusive-or operation (binary addition modulo 2.) We are going to write from left to right the top qubit first and then the bottom. So we say that this function takes the state $|x, y\rangle$ to the state $|x, y \oplus f(x)\rangle$. If $y = 0$, this simplifies $|x, 0\rangle$ to $|x, 0 \oplus f(x)\rangle = |x, f(x)\rangle$. This gate can be seen to be reversible as we may demonstrate by simply

looking at the following circuit:



(6.7)

State $|x, y\rangle$ goes to $|x, y \oplus f(x)\rangle$, which further goes to

$$|x, (y \oplus f(x)) \oplus f(x)\rangle = |x, y \oplus (f(x) \oplus f(x))\rangle = |x, y \oplus 0\rangle = |x, y\rangle, \quad (6.8)$$

where the first equality is due to the associativity of $\oplus$ and the second equality holds because $\oplus$ is idempotent. From this we see that U_f is its own inverse.

In quantum systems, evaluating f is equivalent to multiplying a state by the unitary matrix U_f. For function (6.4), the corresponding unitary matrix, U_f, is

$$\begin{array}{c} \\ \mathbf{00} \\ \mathbf{01} \\ \mathbf{10} \\ \mathbf{11} \end{array} \begin{array}{c} \begin{array}{cccc} \mathbf{00} & \mathbf{01} & \mathbf{10} & \mathbf{11} \end{array} \\ \begin{bmatrix} 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \end{array}. \quad (6.9)$$

Remember that the top column name corresponds to the input $|x, y\rangle$ and the left-hand row name corresponds to the outputs $|x', y'\rangle$. A 1 in the xy column and the $x'y'$ row means that for input $|x, y\rangle$, the output will be $|x', y'\rangle$.

Exercise 6.1.2 What is the adjoint of the matrix given in Equation (6.9)? Show that this matrix is its own inverse. ■

Exercise 6.1.3 Give the unitary matrices that correspond to the other three functions from $\{0, 1\}$ to $\{0, 1\}$. Show that each of the matrices is its own adjoint and hence all are reversible and unitary. ■

Let us remind ourselves of the task at hand. We are given such a matrix that expresses a function but we cannot "look inside" the matrix to "see" how it is defined. We are asked to determine if the function is balanced or constant.

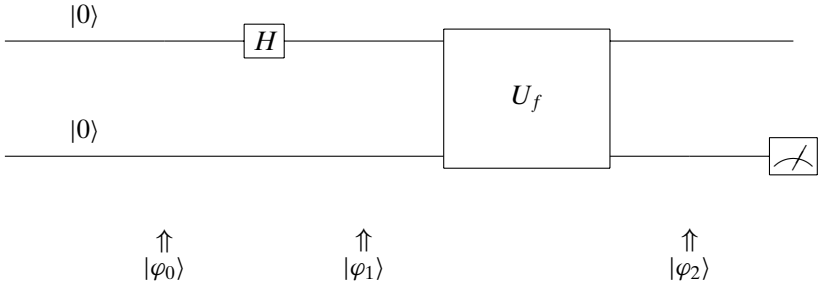
Let us take a first stab at a quantum algorithm to solve this problem. Rather than evaluating f twice, we shall try our trick of superposition of states. Instead of having the top input to be either in state $|0\rangle$ or in state $|1\rangle$, we shall put the top input in state

$$\frac{|0\rangle + |1\rangle}{\sqrt{2}}, \quad (6.10)$$

which is "half-way" $|0\rangle$ and "half-way" $|1\rangle$. The Hadamard matrix can place a qubit in such a state.

$$H|0\rangle = \begin{bmatrix} \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} & -\frac{1}{\sqrt{2}} \end{bmatrix} \begin{bmatrix} 1 \\ 0 \end{bmatrix} = \begin{bmatrix} \frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} \end{bmatrix} = \frac{|0\rangle + |1\rangle}{\sqrt{2}}. \quad (6.11)$$

The obvious (but not necessarily correct) state to put the bottom input into is state $|0\rangle$. Thus we have the following quantum circuit:



(6.12)

The $|\varphi_j\rangle$ at the bottom of the quantum circuit will be used to describe the state of the qubits at each time click.

In terms of matrices this circuit corresponds to

$$U_f(H \otimes I)(|0\rangle \otimes |0\rangle) = U_f(H \otimes I)(|0, 0\rangle). \tag{6.13}$$

The tensor product $|0, 0\rangle$ can be written as

$$\begin{matrix} \mathbf{00} \\ \mathbf{01} \\ \mathbf{10} \\ \mathbf{11} \end{matrix} \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \end{bmatrix} \tag{6.14}$$

and the entire circuit is then

$$U_f(H \otimes I) \begin{matrix} \mathbf{00} \\ \mathbf{01} \\ \mathbf{10} \\ \mathbf{11} \end{matrix} \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \end{bmatrix}. \tag{6.15}$$

We shall carefully examine the states of the system at every time click. The system starts in

$$|\varphi_0\rangle = |0\rangle \otimes |0\rangle = |0, 0\rangle. \tag{6.16}$$

We then apply the Hadamard matrix only to the top input – leaving the bottom input alone – to get

$$|\varphi_1\rangle = \left[\frac{|0\rangle + |1\rangle}{\sqrt{2}}\right]|0\rangle = \frac{|0, 0\rangle + |1, 0\rangle}{\sqrt{2}}. \tag{6.17}$$

After multiplying with U_f, we have

$$|\varphi_2\rangle = \frac{|0,\ f(0)\rangle + |1,\ f(1)\rangle}{\sqrt{2}}. \tag{6.18}$$

For function (6.4), the state $|\varphi_2\rangle$ would be

$$|\varphi_2\rangle = \begin{array}{c} \\ \mathbf{00} \\ \mathbf{01} \\ \mathbf{10} \\ \mathbf{11} \end{array} \begin{array}{c} \begin{array}{cccc} \mathbf{00} & \mathbf{01} & \mathbf{10} & \mathbf{11} \end{array} \\ \begin{bmatrix} 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \end{array} \begin{array}{c} \\ \mathbf{00} \\ \mathbf{01} \\ \mathbf{10} \\ \mathbf{11} \end{array} \begin{bmatrix} \frac{1}{\sqrt{2}} \\ 0 \\ \frac{1}{\sqrt{2}} \\ 0 \end{bmatrix} = \begin{array}{c} \\ \mathbf{00} \\ \mathbf{01} \\ \mathbf{10} \\ \mathbf{11} \end{array} \begin{bmatrix} 0 \\ \frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} \\ 0 \end{bmatrix} = \frac{|0,1\rangle + |1,0\rangle}{\sqrt{2}}. \tag{6.19}$$

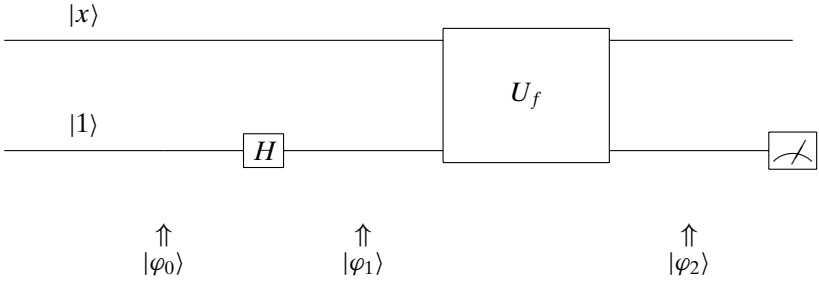
Exercise 6.1.4 Using the matrices calculated in Exercise 6.1.3, determine the state $|\varphi_2\rangle$ for the other three functions. ■

If we measure the top qubit, there will be a 50–50 chance of finding it in state $|0\rangle$ and a 50–50 chance of finding it in state $|1\rangle$. Similarly, there is no real information to be gotten by measuring the bottom qubit. So the obvious algorithm does not work. We need a better trick.

Let us take another stab at solving our problem. Rather than leaving the bottom qubit in state $|0\rangle$, let us put it in the superposition state:

$$\frac{|0\rangle - |1\rangle}{\sqrt{2}} = \begin{bmatrix} \frac{1}{\sqrt{2}} \\ -\frac{1}{\sqrt{2}} \end{bmatrix}. \tag{6.20}$$

Notice the minus sign. Even though there is a negation, this state is also "half-way" in state $|0\rangle$ and "half-way" in state $|1\rangle$. This change of phase will help us get our desired results. We can get to this superposition of states by multiplying state $|1\rangle$ with the Hadamard matrix. We shall leave the top qubit as an ambiguous $|x\rangle$.



(6.21)

In terms of matrices, this becomes

$$U_f(I \otimes H)|x, 1\rangle. \tag{6.22}$$

Let us look carefully at how the states of the qubits change.

$$|\varphi_0\rangle = |x, 1\rangle. \tag{6.23}$$

After the Hadamard matrix, we have

$$|\varphi_1\rangle = |x\rangle \left[\frac{|0\rangle - |1\rangle}{\sqrt{2}}\right] = \frac{|x, 0\rangle - |x, 1\rangle}{\sqrt{2}}. \tag{6.24}$$

Applying U_f, we get

$$|\varphi_2\rangle = |x\rangle \left[\frac{|0 \oplus f(x)\rangle - |1 \oplus f(x)\rangle}{\sqrt{2}}\right] = |x\rangle \left[\frac{|f(x)\rangle - |\overline{f(x)}\rangle}{\sqrt{2}}\right], \tag{6.25}$$

where $\overline{f(x)}$ means the opposite of $f(x)$. Therefore, we have

$$|\varphi_2\rangle = \begin{cases} |x\rangle \left[\frac{|0\rangle - |1\rangle}{\sqrt{2}}\right], & \text{if } f(x) = 0, \\ |x\rangle \left[\frac{|1\rangle - |0\rangle}{\sqrt{2}}\right], & \text{if } f(x) = 1. \end{cases} \tag{6.26}$$

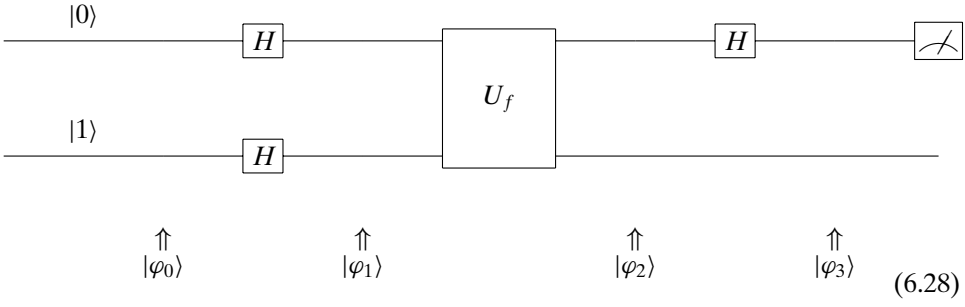
Remembering that $a - b = (-1)(b - a)$, we might write this as

$$|\varphi_2\rangle = (-1)^{f(x)}|x\rangle \left[\frac{|0\rangle - |1\rangle}{\sqrt{2}}\right]. \tag{6.27}$$

What would happen if we evaluate either the top or the bottom state? Again, this does not really help us. We do not gain any information if we measure the top qubit or the bottom qubit. The top qubit will be in state $|x\rangle$ and the bottom qubit will be either in state $|0\rangle$ or in state $|1\rangle$. We need something more.

Now let us combine both these attempts to actually give Deutsch's algorithm.

Deutsch's algorithm works by putting *both* the top and the bottom qubits into a superposition. We will also put the results of the top qubit through a Hadamard matrix.



(6.28)

In terms of matrices this becomes

$$(H \otimes I)U_f(H \otimes H)|0, 1\rangle \tag{6.29}$$

or

$$(H \otimes I)U_f(H \otimes H)\begin{matrix}\mathbf{00}\\ \mathbf{01}\\ \mathbf{10}\\ \mathbf{11}\end{matrix}\begin{bmatrix}0\\1\\0\\0\end{bmatrix}. \tag{6.30}$$

At each point of the algorithm the states are as follows:

$$|\varphi_0\rangle = |0, 1\rangle, \tag{6.31}$$

$$|\varphi_1\rangle = \left[\frac{|0\rangle + |1\rangle}{\sqrt{2}}\right]\left[\frac{|0\rangle - |1\rangle}{\sqrt{2}}\right] = \frac{+|0,0\rangle - |0,1\rangle + |1,0\rangle - |1,1\rangle}{2} = \begin{matrix}\mathbf{00}\\ \mathbf{01}\\ \mathbf{10}\\ \mathbf{11}\end{matrix}\begin{bmatrix}+\frac{1}{2}\\ -\frac{1}{2}\\ +\frac{1}{2}\\ -\frac{1}{2}\end{bmatrix}. \tag{6.32}$$

We saw from our last attempt at solving this problem that when we put the bottom qubit into a superposition and then multiply by U_f, we will be in the superposition

$$(-1)^{f(x)}|x\rangle\left[\frac{|0\rangle - |1\rangle}{\sqrt{2}}\right]. \tag{6.33}$$

Now, with $|x\rangle$ in a superposition, we have

$$|\varphi_2\rangle = \left[\frac{(-1)^{f(0)}|0\rangle + (-1)^{f(1)}|1\rangle}{\sqrt{2}}\right]\left[\frac{|0\rangle - |1\rangle}{\sqrt{2}}\right]. \tag{6.34}$$

For example, if $f(0) = 1$ and $f(1) = 0$, the top qubit becomes

$$\frac{(-1)|0\rangle + (+1)|1\rangle}{\sqrt{2}} = (-1)\left[\frac{|0\rangle - |1\rangle}{\sqrt{2}}\right]. \tag{6.35}$$

Exercise 6.1.5 For each of the other three functions from the set $\{0, 1\}$ to the set $\{0, 1\}$, describe what $|\varphi_2\rangle$ would be. ■

For a general function f, let us look carefully at

$$(-1)^{f(0)}|0\rangle + (-1)^{f(1)}|1\rangle. \tag{6.36}$$

If f is constant, this becomes either

$$+1(|0\rangle + |1\rangle) \text{ or } -1(|0\rangle + |1\rangle) \tag{6.37}$$

(depending on being constantly 0 or constantly 1).

If f is balanced, it becomes either

$$+1(|0\rangle - |1\rangle) \text{ or } -1(|0\rangle - |1\rangle) \tag{6.38}$$

(depending on which way it is balanced).

Summing up, we have that

$$|\varphi_2\rangle = \begin{cases} (\pm 1)\left[\frac{|0\rangle+|1\rangle}{\sqrt{2}}\right]\left[\frac{|0\rangle-|1\rangle}{\sqrt{2}}\right], & \text{if } f \text{ is constant,} \\ (\pm 1)\left[\frac{|0\rangle-|1\rangle}{\sqrt{2}}\right]\left[\frac{|0\rangle-|1\rangle}{\sqrt{2}}\right], & \text{if } f \text{ is balanced.} \end{cases} \tag{6.39}$$

Remembering that the Hadamard matrix is its own inverse that takes $\frac{|0\rangle+|1\rangle}{\sqrt{2}}$ to $|0\rangle$ and takes $\frac{|0\rangle-|1\rangle}{\sqrt{2}}$ to $|1\rangle$, we apply the Hadamard matrix to the top qubit to get

$$|\varphi_3\rangle = \begin{cases} (\pm 1)|0\rangle\left[\frac{|0\rangle-|1\rangle}{\sqrt{2}}\right], & \text{if } f \text{ is constant,} \\ (\pm 1)|1\rangle\left[\frac{|0\rangle-|1\rangle}{\sqrt{2}}\right], & \text{if } f \text{ is balanced.} \end{cases} \tag{6.40}$$

For example, if $f(0) = 1$ and $f(1) = 0$, then we get

$$|\varphi_3\rangle = -1|0\rangle\left[\frac{|0\rangle - |1\rangle}{\sqrt{2}}\right]. \tag{6.41}$$

Exercise 6.1.6 For each of the other three functions from the set $\{0, 1\}$ to the set $\{0, 1\}$, calculate the value of $|\varphi_3\rangle$. ■

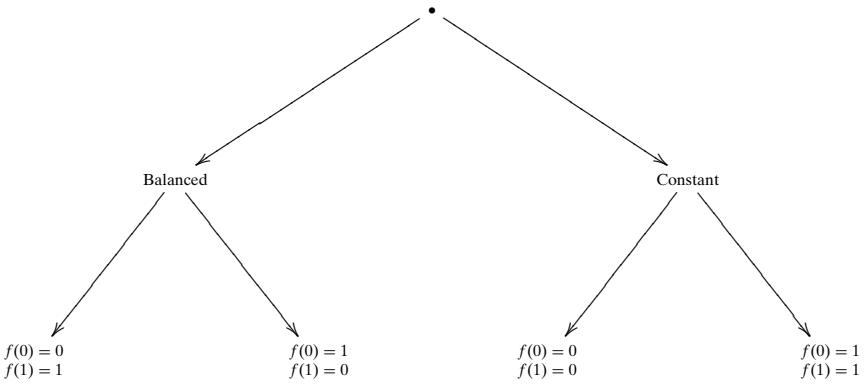
Now, we simply measure the top qubit. If it is in state $|0\rangle$, then we know that f is a constant function, otherwise it is a balanced function. This was all accomplished with only one function evaluation as opposed to the two evaluations that the classical algorithm demands.

Notice that although the ± 1 tells us even more information, namely, which of the two balanced functions or two constant functions we have, measurement will not grant us this information. Upon measuring, if the function is balanced, we will measure $|1\rangle$ regardless if the state was $(-1)|1\rangle$ or $(+1)|1\rangle$.

The reader might be bothered by the fact that the output of the top qubit of U_f should not change from being the same as the input. However, the inclusion of the Hadamard matrices changes things around, as we saw in Section 5.3. This is the essence of the fact that the top and the bottom qubits are entangled.

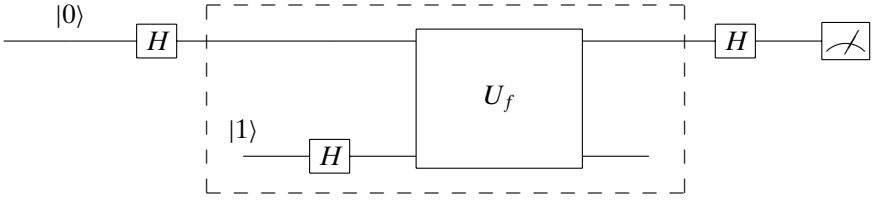
Did we perform a magic trick here? Did we gain information that was not there? Not really. There are four possible functions, and we saw in decision tree (6.2) that with a classical computer we needed two bits of information to determine which of the four functions we were given. What we are really doing here is *changing around the information*. We might determine which of the four functions is the case by asking the following two questions: "Is the function balanced or constant?" and "What

is the value of the function on 0?" The answers to these two questions uniquely describe each of the four functions, as described by the following decision tree:



(6.42)

The Hadamard matrices are changing the question that we are asking (change of basis). The intuition behind the Deutsch algorithm is that we are really just performing a change of basis problem as discussed at the end of Section 2.3. We might rewrite quantum circuit (6.28) as



(6.43)

We start in the canonical basis. The first Hadamard matrix is used as a change of basis matrix to go into a balanced superposition of basic states. While in this noncanonical basis, we evaluate f with the bottom qubit in a superposition. The last Hadamard matrix is used as a change of basis matrix to revert back to the canonical basis.

6.2 THE DEUTSCH–JOZSA ALGORITHM

Let us generalize the Deutsch algorithm to other functions. Rather than talking about functions $f : \{0, 1\} \longrightarrow \{0, 1\}$, let us talk about functions with a larger domain. Consider functions $f : \{0, 1\}^n \longrightarrow \{0, 1\}$, which accept a string of n 0's and 1's and outputs a zero or one. The domain might be thought of as any natural number from 0 to $2^n - 1$.

We shall call a function $f : \{0, 1\}^n \longrightarrow \{0, 1\}$ **balanced** if exactly half of the inputs go to 0 (and the other half go to 1). Call a function **constant** if *all* the inputs go to 0 or *all* the inputs go to 1.

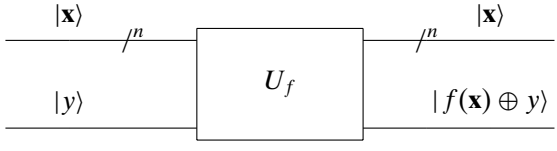
Exercise 6.2.1 How many functions are there from $\{0,1\}^n$ to $\{0,1\}$? How many of them are balanced? How many of them are constant? ■

The Deutsch–Jozsa algorithm solves the following problem: Suppose you are given a function from $\{0,1\}^n$ to $\{0,1\}$ which you can evaluate but cannot "see" the way it is defined. Suppose further that you are assured that the function is either balanced or constant. Determine if the function is balanced or constant. Notice that when $n = 1$, this is exactly the problem that the Deutsch algorithm solved.

Classically, this algorithm can be solved by evaluating the function on different inputs. The best case scenario is when the first two different inputs have different outputs, which assures us that the function is balanced. In contrast, to be sure that the function is constant, one must evaluate the function on more than half the possible inputs. So the worst case scenario requires $\frac{2^n}{2} + 1 = 2^{n-1} + 1$ function evaluations. Can we do better?

In the last section, we solved the problem by entering into a superposition of two possible input states. In this section, we solve the problem by entering a superposition of all 2^n possible input states.

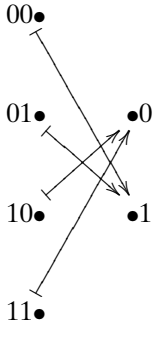
The function f will be given as a unitary matrix that we shall depict as



(6.44)

with n qubits (denoted as $\xrightarrow{\quad n \quad}$) as the top input and output. For the rest of this chapter, a binary string is denoted by a boldface letter. So we write the top input as $|\mathbf{x}\rangle = |x_0 x_1 \ldots x_{n-1}\rangle$. The bottom entering control qubit is $|y\rangle$. The top output is $|\mathbf{x}\rangle$ which will not be changed by U_f. The bottom output of U_f is the single qubit $|y \oplus f(\mathbf{x})\rangle$. Remember that although $\mathbf{x}$ is n bits, $f(\mathbf{x})$ is one bit and hence we can use the binary operation $\oplus$. It is not hard to see that U_f is its own inverse.

Example 6.2.1 Consider the following balanced function from $\{0,1\}^2$ to $\{0,1\}$:



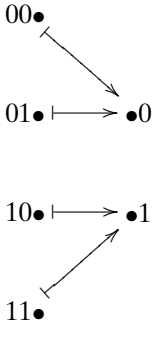
(6.45)

This function shall be represented by the following 8-by-8 unitary matrix:

$$\begin{array}{c} \\ \mathbf{00,0} \\ \mathbf{00,1} \\ \mathbf{01,0} \\ \mathbf{01,1} \\ \mathbf{10,0} \\ \mathbf{10,1} \\ \mathbf{11,0} \\ \mathbf{11,1} \end{array} \begin{array}{c} \begin{array}{cccccccc} \mathbf{00,0} & \mathbf{00,1} & \mathbf{01,0} & \mathbf{01,1} & \mathbf{10,0} & \mathbf{10,1} & \mathbf{11,0} & \mathbf{11,1} \end{array} \\ \begin{bmatrix} & 1 & & & & & & \\ 1 & & & & & & & \\ & & & 1 & & & & \\ & & 1 & & & & & \\ & & & & 1 & & & \\ & & & & & 1 & & \\ & & & & & & 1 & \\ & & & & & & & 1 \end{bmatrix} \end{array} \tag{6.46}$$

(the zeros are omitted for readability). □

Exercise 6.2.2 Consider the balanced function from $\{0,1\}^2$ to $\{0,1\}$:



(6.47)

Give the corresponding 8-by-8 unitary matrix. ■

Exercise 6.2.3 Consider the function from $\{0,1\}^2$ to $\{0,1\}$ that always outputs a 1. Give the corresponding 8-by-8 unitary matrix. ■

In order to place a single qubit in a superposition of $|0\rangle$ and $|1\rangle$, we used a single Hadamard matrix. To place n qubits in a superposition, we are going to use the tensor product of n Hadamard matrices. What does such a tensor product look like? It will be helpful to do some warm-up exercises. Let us calculate H, $H \otimes H$ which we may write as $H^{\otimes 2}$, and $H \otimes H \otimes H = H^{\otimes 3}$; and look for a pattern. Our goal will be to find a pattern for $H^{\otimes n}$.

Remember that the Hadamard matrix is defined as

$$H = \frac{1}{\sqrt{2}} \begin{array}{c} \\ \mathbf{0} \\ \mathbf{1} \end{array} \begin{array}{c} \begin{array}{cc} \mathbf{0} & \mathbf{1} \end{array} \\ \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \end{array}. \tag{6.48}$$

Notice that $H[i, j] = \frac{1}{\sqrt{2}}(-1)^{i \wedge j}$, where i and j are the row and column numbers in binary and $\wedge$ is the AND operation. We might then write the Hadamard matrix as

$$H = \frac{1}{\sqrt{2}} \begin{matrix} \\ \mathbf{0} \\ \mathbf{1} \end{matrix} \begin{matrix} \begin{matrix} \mathbf{0} & \mathbf{1} \end{matrix} \\ \begin{bmatrix} (-1)^{0\wedge 0} & (-1)^{0\wedge 1} \\ (-1)^{1\wedge 0} & (-1)^{1\wedge 1} \end{bmatrix} \end{matrix}. \tag{6.49}$$

Notice that we are thinking of 0 and 1 as both Boolean values and numbers that are exponents. (Remember: $(-1)^0 = 1$ and $(-1)^1 = -1$.) With this trick in mind we can then calculate

$$H^{\otimes 2} = H \otimes H = \frac{1}{\sqrt{2}} \begin{matrix} \\ \mathbf{0} \\ \mathbf{1} \end{matrix} \begin{matrix} \begin{matrix} \mathbf{0} & \mathbf{1} \end{matrix} \\ \begin{bmatrix} (-1)^{0\wedge 0} & (-1)^{0\wedge 1} \\ (-1)^{1\wedge 0} & (-1)^{1\wedge 1} \end{bmatrix} \end{matrix} \otimes \frac{1}{\sqrt{2}} \begin{matrix} \\ \mathbf{0} \\ \mathbf{1} \end{matrix} \begin{matrix} \begin{matrix} \mathbf{0} & \mathbf{1} \end{matrix} \\ \begin{bmatrix} (-1)^{0\wedge 0} & (-1)^{0\wedge 1} \\ (-1)^{1\wedge 0} & (-1)^{1\wedge 1} \end{bmatrix} \end{matrix}$$

$$= \frac{1}{\sqrt{2}} * \frac{1}{\sqrt{2}} \begin{matrix} \\ \mathbf{00} \\ \mathbf{01} \\ \mathbf{10} \\ \mathbf{11} \end{matrix} \begin{matrix} \begin{matrix} \mathbf{00} & \mathbf{01} & \mathbf{10} & \mathbf{11} \end{matrix} \\ \begin{bmatrix} (-1)^{0\wedge 0} * (-1)^{0\wedge 0} & (-1)^{0\wedge 0} * (-1)^{0\wedge 1} & (-1)^{0\wedge 1} * (-1)^{0\wedge 0} & (-1)^{0\wedge 1} * (-1)^{0\wedge 1} \\ (-1)^{0\wedge 0} * (-1)^{1\wedge 0} & (-1)^{0\wedge 0} * (-1)^{1\wedge 1} & (-1)^{0\wedge 1} * (-1)^{1\wedge 0} & (-1)^{0\wedge 1} * (-1)^{1\wedge 1} \\ (-1)^{1\wedge 0} * (-1)^{0\wedge 0} & (-1)^{1\wedge 0} * (-1)^{0\wedge 1} & (-1)^{1\wedge 1} * (-1)^{0\wedge 0} & (-1)^{1\wedge 1} * (-1)^{0\wedge 1} \\ (-1)^{1\wedge 0} * (-1)^{1\wedge 0} & (-1)^{1\wedge 0} * (-1)^{1\wedge 1} & (-1)^{1\wedge 1} * (-1)^{1\wedge 0} & (-1)^{1\wedge 1} * (-1)^{1\wedge 1} \end{bmatrix} \end{matrix} \tag{6.50}$$

When we multiply $(-1)^x$ by $(-1)^y$, we are not interested in $(-1)^{x+y}$. Rather, we are interested in the parity of x and y. So we shall not add x and y but take their exclusive-or ($\oplus$). This leaves us with

$$H^{\otimes 2} = \frac{1}{2} \begin{matrix} \\ \mathbf{00} \\ \mathbf{01} \\ \mathbf{10} \\ \mathbf{11} \end{matrix} \begin{matrix} \begin{matrix} \mathbf{00} & \mathbf{01} & \mathbf{10} & \mathbf{11} \end{matrix} \\ \begin{bmatrix} (-1)^{0\wedge 0\oplus 0\wedge 0} & (-1)^{0\wedge 0\oplus 0\wedge 1} & (-1)^{0\wedge 1\oplus 0\wedge 0} & (-1)^{0\wedge 1\oplus 0\wedge 1} \\ (-1)^{0\wedge 0\oplus 1\wedge 0} & (-1)^{0\wedge 0\oplus 1\wedge 1} & (-1)^{0\wedge 1\oplus 1\wedge 0} & (-1)^{0\wedge 1\oplus 1\wedge 1} \\ (-1)^{1\wedge 0\oplus 0\wedge 0} & (-1)^{1\wedge 0\oplus 0\wedge 1} & (-1)^{1\wedge 1\oplus 0\wedge 0} & (-1)^{1\wedge 1\oplus 0\wedge 1} \\ (-1)^{1\wedge 0\oplus 1\wedge 0} & (-1)^{1\wedge 0\oplus 1\wedge 1} & (-1)^{1\wedge 1\oplus 1\wedge 0} & (-1)^{1\wedge 1\oplus 1\wedge 1} \end{bmatrix} \end{matrix}$$

$$= \frac{1}{2} \begin{matrix} \\ \mathbf{00} \\ \mathbf{01} \\ \mathbf{10} \\ \mathbf{11} \end{matrix} \begin{matrix} \begin{matrix} \mathbf{00} & \mathbf{01} & \mathbf{10} & \mathbf{11} \end{matrix} \\ \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & -1 & 1 & -1 \\ 1 & 1 & -1 & -1 \\ 1 & -1 & -1 & 1 \end{bmatrix} \end{matrix}. \tag{6.51}$$

Exercise 6.2.4 Prove by induction that the scalar coefficient of $H^{\otimes n}$ is

$$\frac{1}{\sqrt{2^n}} = 2^{-\frac{n}{2}}. \tag{6.52}$$

■

Thus, we have reduced the problem to determining if the exponent of (-1) is odd or even. The only time that this exponent should change is when the (-1) is in the lower-right-hand corner of a matrix. When we calculate $H^{\otimes 3}$ we will again multiply each entry of $H^{\otimes 2}$ by the appropriate element of H. If we are in the lower-right-hand corner, i.e., the (1, 1) position, then we should toggle the exponent of (-1).

The following operation will be helpful. We define

$$\langle\ ,\ \rangle : \{0,1\}^n \times \{0,1\}^n \longrightarrow \{0,1\} \tag{6.53}$$

as follows: Given two binary strings of length n, $\mathbf{x} = x_0x_1x_2\ldots x_{n-1}$ and $\mathbf{y} = y_0y_1y_2\ldots y_{n-1}$, we say

$$\begin{aligned}\langle \mathbf{x}, \mathbf{y}\rangle &= \langle x_0x_1x_2\ldots x_{n-1},\, y_0y_1y_2\ldots y_{n-1}\rangle \\ &= (x_0 \wedge y_0) \oplus (x_1 \wedge y_1) \oplus \cdots \oplus (x_{n-1} \wedge y_{n-1}).\end{aligned} \tag{6.54}$$

Basically, this gives the parity of the number of times that both bits are at 1.[1]

If $\mathbf{x}$ and $\mathbf{y}$ are binary strings of length n, then $\mathbf{x} \oplus \mathbf{y}$ is the pointwise (bitwise) exclusive-or operation, i.e.,

$$\mathbf{x} \oplus \mathbf{y} = x_1 \oplus y_1, x_2 \oplus y_2, \ldots, x_n \oplus y_n. \tag{6.55}$$

The function $\langle\ ,\ \rangle : \{0,1\}^n \times \{0,1\}^n \longrightarrow \{0,1\}$ satisfies the following properties:

(i)

$$\langle \mathbf{x} \oplus \mathbf{x}', \mathbf{y}\rangle = \langle \mathbf{x}, \mathbf{y}\rangle \oplus \langle \mathbf{x}', \mathbf{y}\rangle, \tag{6.56}$$

$$\langle \mathbf{x}, \mathbf{y} \oplus \mathbf{y}'\rangle = \langle \mathbf{x}, \mathbf{y}\rangle \oplus \langle \mathbf{x}, \mathbf{y}'\rangle. \tag{6.57}$$

(ii)

$$\langle 0 \cdot \mathbf{x}, \mathbf{y}\rangle = \langle 0^n, \mathbf{y}\rangle = 0, \tag{6.58}$$

$$\langle \mathbf{x}, 0 \cdot \mathbf{y}\rangle = \langle \mathbf{x}, 0^n\rangle = 0. \tag{6.59}$$

With this notation, it is easy to write $H^{\otimes 3}$ as

$$\frac{1}{2\sqrt{2}}\begin{array}{c} \\ \mathbf{000} \\ \mathbf{001} \\ \mathbf{010} \\ \mathbf{011} \\ \mathbf{100} \\ \mathbf{101} \\ \mathbf{110} \\ \mathbf{111} \end{array}\begin{array}{c}\begin{array}{cccccccc}\mathbf{000} & \mathbf{001} & \mathbf{010} & \mathbf{011} & \mathbf{100} & \mathbf{101} & \mathbf{110} & \mathbf{111}\end{array} \\ \begin{bmatrix} (-1)^{\langle 000,000\rangle} & (-1)^{\langle 000,001\rangle} & (-1)^{\langle 000,010\rangle} & (-1)^{\langle 000,011\rangle} & (-1)^{\langle 000,100\rangle} & (-1)^{\langle 000,101\rangle} & (-1)^{\langle 000,110\rangle} & (-1)^{\langle 000,111\rangle} \\ (-1)^{\langle 001,000\rangle} & (-1)^{\langle 001,001\rangle} & (-1)^{\langle 001,010\rangle} & (-1)^{\langle 001,011\rangle} & (-1)^{\langle 001,100\rangle} & (-1)^{\langle 001,101\rangle} & (-1)^{\langle 001,110\rangle} & (-1)^{\langle 001,111\rangle} \\ (-1)^{\langle 010,000\rangle} & (-1)^{\langle 010,001\rangle} & (-1)^{\langle 010,010\rangle} & (-1)^{\langle 010,011\rangle} & (-1)^{\langle 010,100\rangle} & (-1)^{\langle 010,101\rangle} & (-1)^{\langle 010,110\rangle} & (-1)^{\langle 010,111\rangle} \\ (-1)^{\langle 011,000\rangle} & (-1)^{\langle 011,001\rangle} & (-1)^{\langle 011,010\rangle} & (-1)^{\langle 011,011\rangle} & (-1)^{\langle 011,100\rangle} & (-1)^{\langle 011,101\rangle} & (-1)^{\langle 011,110\rangle} & (-1)^{\langle 011,111\rangle} \\ (-1)^{\langle 100,000\rangle} & (-1)^{\langle 100,001\rangle} & (-1)^{\langle 100,010\rangle} & (-1)^{\langle 100,011\rangle} & (-1)^{\langle 100,100\rangle} & (-1)^{\langle 100,101\rangle} & (-1)^{\langle 100,110\rangle} & (-1)^{\langle 100,111\rangle} \\ (-1)^{\langle 101,000\rangle} & (-1)^{\langle 101,001\rangle} & (-1)^{\langle 101,010\rangle} & (-1)^{\langle 101,011\rangle} & (-1)^{\langle 101,100\rangle} & (-1)^{\langle 101,101\rangle} & (-1)^{\langle 101,110\rangle} & (-1)^{\langle 101,111\rangle} \\ (-1)^{\langle 110,000\rangle} & (-1)^{\langle 110,001\rangle} & (-1)^{\langle 110,010\rangle} & (-1)^{\langle 110,011\rangle} & (-1)^{\langle 110,100\rangle} & (-1)^{\langle 110,101\rangle} & (-1)^{\langle 110,110\rangle} & (-1)^{\langle 110,111\rangle} \\ (-1)^{\langle 111,000\rangle} & (-1)^{\langle 111,001\rangle} & (-1)^{\langle 111,010\rangle} & (-1)^{\langle 111,011\rangle} & (-1)^{\langle 111,100\rangle} & (-1)^{\langle 111,101\rangle} & (-1)^{\langle 111,110\rangle} & (-1)^{\langle 111,111\rangle} \end{bmatrix}\end{array}$$

[1] This is reminiscent of the definition of an inner product. In fact, it *is* an inner product, but on an interesting vector space. The vector space is not a complex vector space, nor a real vector space. It is a vector space over the field with exactly two elements $\{0,1\}$. This field is denoted $\mathbb{Z}_2$ or $\mathbb{F}_2$. The set of elements of the vector space is $\{0,1\}^n$, the set of bit strings of length n, and the addition is pointwise $\oplus$. The zero element is the string of n zeros. Scalar multiplication is obvious. We shall not list all the properties of this inner product space but we strongly recommend that you do so. Meditate on a basis and on the notions of orthogonality, dimension, etc.

$$= \frac{1}{2\sqrt{2}} \begin{array}{c} \\ \mathbf{000} \\ \mathbf{001} \\ \mathbf{010} \\ \mathbf{011} \\ \mathbf{100} \\ \mathbf{101} \\ \mathbf{110} \\ \mathbf{111} \end{array} \begin{array}{c} \begin{array}{cccccccc} \mathbf{000} & \mathbf{001} & \mathbf{010} & \mathbf{011} & \mathbf{100} & \mathbf{101} & \mathbf{110} & \mathbf{111} \end{array} \\ \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & -1 & 1 & -1 & 1 & -1 & 1 & -1 \\ 1 & 1 & -1 & -1 & 1 & 1 & -1 & -1 \\ 1 & -1 & -1 & 1 & 1 & -1 & -1 & 1 \\ 1 & 1 & 1 & 1 & -1 & -1 & -1 & -1 \\ 1 & -1 & 1 & -1 & -1 & 1 & -1 & 1 \\ 1 & 1 & -1 & -1 & -1 & -1 & 1 & 1 \\ 1 & -1 & -1 & 1 & -1 & 1 & 1 & -1 \end{bmatrix} \end{array}. \tag{6.60}$$

From this, we can write a general formula for $H^{\otimes n}$ as

$$H^{\otimes n}[\mathbf{i}, \mathbf{j}] = \frac{1}{\sqrt{2^n}}(-1)^{\langle \mathbf{i}, \mathbf{j} \rangle}, \tag{6.61}$$

where **i** and **j** are the row and column numbers in binary.

What happens if we multiply a state with this matrix? Notice that all the elements of the leftmost column of $H^{\otimes n}$ are $+1$. So if we multiply $H^{\otimes n}$ with the state

$$|\mathbf{0}\rangle = |00\ldots0\rangle = \begin{array}{l} 00000000 \\ 00000001 \\ 00000010 \\ \vdots \\ 11111110 \\ 11111111 \end{array} \begin{bmatrix} 1 \\ 0 \\ 0 \\ \vdots \\ 0 \\ 0 \end{bmatrix}, \tag{6.62}$$

we see that this will equal the leftmost column of $H^{\otimes n}$:

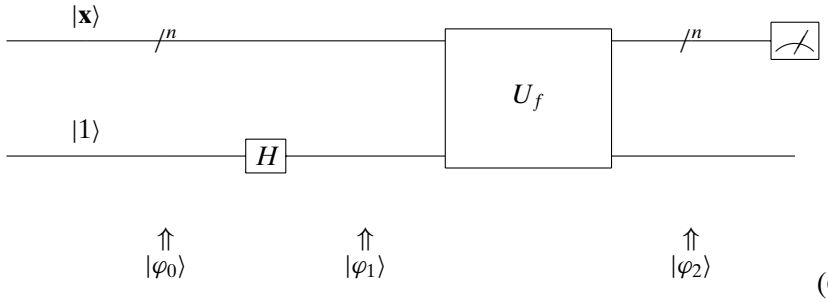
$$H^{\otimes n}|\mathbf{0}\rangle = H^{\otimes n}[-, \mathbf{0}] = \frac{1}{\sqrt{2^n}} \begin{array}{l} 00000000 \\ 00000001 \\ 00000010 \\ \vdots \\ 11111110 \\ 11111111 \end{array} \begin{bmatrix} 1 \\ 1 \\ 1 \\ \vdots \\ 1 \\ 1 \end{bmatrix} = \frac{1}{\sqrt{2^n}} \sum_{\mathbf{x} \in \{0,1\}^n} |\mathbf{x}\rangle. \tag{6.63}$$

For an arbitrary basic state $|\mathbf{y}\rangle$, which can be represented by a column vector with a single 1 in position **y** and 0's everywhere else, we will be extracting the **y**th column of $H^{\otimes n}$:

$$H^{\otimes n}|\mathbf{y}\rangle = H^{\otimes n}[-, \mathbf{y}] = \frac{1}{\sqrt{2^n}} \sum_{\mathbf{x} \in \{0,1\}^n} (-1)^{\langle \mathbf{x}, \mathbf{y} \rangle} |\mathbf{x}\rangle. \tag{6.64}$$

Let us return to the problem at hand. We are trying to tell whether the given function is balanced or constant. In the last section, we were successful by placing

the bottom control qubit in a superposition. Let us see what would happen if we did the same thing here.



(6.65)

In terms of matrices this amounts to

$$U_f(I \otimes H)|\mathbf{x}, 1\rangle. \tag{6.66}$$

For an arbitrary $\mathbf{x} = x_0x_1x_2 \ldots x_{n-1}$ as an input in the top n qubits, we will have the following states:

$$|\varphi_0\rangle = |\mathbf{x}, 1\rangle. \tag{6.67}$$

After the bottom Hadamard matrix, we have

$$|\varphi_1\rangle = |\mathbf{x}\rangle \left[\frac{|0\rangle - |1\rangle}{\sqrt{2}}\right] = \left[\frac{|\mathbf{x}, 0\rangle - |\mathbf{x}, 1\rangle}{\sqrt{2}}\right]. \tag{6.68}$$

Applying U_f we get

$$|\varphi_2\rangle = |\mathbf{x}\rangle \left[\frac{|f(\mathbf{x}) \oplus 0\rangle - |f(\mathbf{x}) \oplus 1\rangle}{\sqrt{2}}\right] = |\mathbf{x}\rangle \left[\frac{|f(\mathbf{x})\rangle - |\overline{f(\mathbf{x})}\rangle}{\sqrt{2}}\right], \tag{6.69}$$

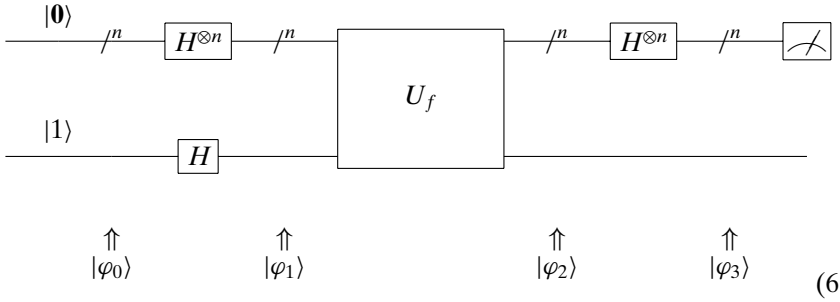
where $\overline{f(\mathbf{x})}$ means the opposite of $f(\mathbf{x})$.

$$|\varphi_2\rangle = \begin{cases} |\mathbf{x}\rangle \left[\frac{|0\rangle - |1\rangle}{\sqrt{2}}\right], & \text{if } f(\mathbf{x}) = 0 \\ \\ |\mathbf{x}\rangle \left[\frac{|1\rangle - |0\rangle}{\sqrt{2}}\right], & \text{if } f(\mathbf{x}) = 1 \end{cases} = (-1)^{f(\mathbf{x})}|\mathbf{x}\rangle \left[\frac{|0\rangle - |1\rangle}{\sqrt{2}}\right]. \tag{6.70}$$

This is almost exactly like Equation (6.27) in the last section. Unfortunately, it is just as unhelpful.

Let us take another stab at the problem and present the Deutsch–Jozsa algorithm. This time, we shall put $|\mathbf{x}\rangle = |x_0x_1 \cdots x_{n-1}\rangle$ into a superposition in which all

2^n possible strings have equal probability. We saw that we can get such a superposition by multiplying $H^{\otimes n}$ by $|\mathbf{0}\rangle = |000\cdots0\rangle$. Thus, we have



(6.71)

In terms of matrices this amounts to

$$(H^{\otimes n} \otimes I)U_f(H^{\otimes n} \otimes H)|\mathbf{0}, 1\rangle. \tag{6.72}$$

Each state can be written as

$$|\varphi_0\rangle = |\mathbf{0}, 1\rangle, \tag{6.73}$$

$$|\varphi_1\rangle = \left[\frac{\sum_{\mathbf{x}\in\{0,1\}^n} |\mathbf{x}\rangle}{\sqrt{2^n}}\right]\left[\frac{|0\rangle - |1\rangle}{\sqrt{2}}\right] \tag{6.74}$$

(as in Equation (6.63)). After applying the U_f unitary matrix, we have

$$|\varphi_2\rangle = \left[\frac{\sum_{\mathbf{x}\in\{0,1\}^n} (-1)^{f(\mathbf{x})}|\mathbf{x}\rangle}{\sqrt{2^n}}\right]\left[\frac{|0\rangle - |1\rangle}{\sqrt{2}}\right]. \tag{6.75}$$

Finally, we apply $H^{\otimes n}$ to the top qubits that are already in a superposition of different $\mathbf{x}$ states to get a superposition of a superposition

$$|\varphi_3\rangle = \left[\frac{\sum_{\mathbf{x}\in\{0,1\}^n} (-1)^{f(\mathbf{x})} \sum_{\mathbf{z}\in\{0,1\}^n} (-1)^{\langle \mathbf{z},\mathbf{x}\rangle}|\mathbf{z}\rangle}{2^n}\right]\left[\frac{|0\rangle - |1\rangle}{\sqrt{2}}\right] \tag{6.76}$$

from Equation (6.64). We can combine parts and "add" exponents to get

$$\begin{aligned} |\varphi_3\rangle &= \left[\frac{\sum_{\mathbf{x}\in\{0,1\}^n} \sum_{\mathbf{z}\in\{0,1\}^n} (-1)^{f(\mathbf{x})}(-1)^{\langle \mathbf{z},\mathbf{x}\rangle}|\mathbf{z}\rangle}{2^n}\right]\left[\frac{|0\rangle - |1\rangle}{\sqrt{2}}\right] \\ &= \left[\frac{\sum_{\mathbf{x}\in\{0,1\}^n} \sum_{\mathbf{z}\in\{0,1\}^n} (-1)^{f(\mathbf{x})\oplus\langle \mathbf{z},\mathbf{x}\rangle}|\mathbf{z}\rangle}{2^n}\right]\left[\frac{|0\rangle - |1\rangle}{\sqrt{2}}\right]. \end{aligned} \tag{6.77}$$

Now the top qubits of state $|\varphi_3\rangle$ are measured. Rather than figuring out what we will get after measuring the top qubit, let us ask the following question: What is the probability that the top qubits of $|\varphi_3\rangle$ will collapse to the state $|\mathbf{0}\rangle$? We can answer this by setting $\mathbf{z} = \mathbf{0}$ and realizing that $\langle \mathbf{z}, \mathbf{x} \rangle = \langle \mathbf{0}, \mathbf{x} \rangle = 0$ for all $\mathbf{x}$. In this case, we have reduced $|\varphi_3\rangle$ to

$$\left[\frac{\sum_{\mathbf{x} \in \{0,1\}^n} (-1)^{f(\mathbf{x})} |\mathbf{0}\rangle}{2^n} \right] \left[\frac{|0\rangle - |1\rangle}{\sqrt{2}} \right]. \tag{6.78}$$

So, the probability of collapsing to $|\mathbf{0}\rangle$ is totally dependent on $f(\mathbf{x})$. If $f(\mathbf{x})$ is constant at 1, the top qubits become

$$\frac{\sum_{\mathbf{x} \in \{0,1\}^n} (-1) |\mathbf{0}\rangle}{2^n} = \frac{-(2^n)|\mathbf{0}\rangle}{2^n} = -1|\mathbf{0}\rangle. \tag{6.79}$$

If $f(\mathbf{x})$ is constant at 0, the top qubits become

$$\frac{\sum_{\mathbf{x} \in \{0,1\}^n} 1 |\mathbf{0}\rangle}{2^n} = \frac{2^n|\mathbf{0}\rangle}{2^n} = +1|\mathbf{0}\rangle. \tag{6.80}$$

And finally, if f is balanced, then half of the $\mathbf{x}$'s will cancel the other half and the top qubits will become

$$\frac{\sum_{\mathbf{x} \in \{0,1\}^n} (-1)^{f(\mathbf{x})} |\mathbf{0}\rangle}{2^n} = \frac{0|\mathbf{0}\rangle}{2^n} = 0|\mathbf{0}\rangle. \tag{6.81}$$

When measuring the top qubits of $|\varphi_3\rangle$, we will only get $|\mathbf{0}\rangle$ if the function is constant. If anything else is found after being measured, then the function is balanced.

In conclusion, we have solved the – admittedly contrived – problem in one function evaluation as opposed to the $2^{n-1} + 1$ function evaluations needed in classical computations. That is an exponential speedup!

Exercise 6.2.5 What would happen if we were tricked and the given function was neither balanced nor constant? What would our algorithm produce? ■

6.3 SIMON'S PERIODICITY ALGORITHM

Simon's algorithm is about finding patterns in functions. We will use methods that we already learned in previous sections, but we will also employ other ideas. This algorithm is a combination of quantum procedures as well as classical procedures.

Suppose that we are given a function $f : \{0, 1\}^n \longrightarrow \{0, 1\}^n$ that we can evaluate but it is given to us as a black box. We are further assured that there exists a secret (hidden) binary string $\mathbf{c} = c_0 c_1 c_2 \cdots c_{n-1}$, such that for all strings $\mathbf{x}, \mathbf{y} \in \{0, 1\}^n$, we have

$$f(\mathbf{x}) = f(\mathbf{y}) \quad \text{if and only if } \mathbf{x} = \mathbf{y} \oplus \mathbf{c}, \tag{6.82}$$