Chapter 1

Fundamentals

Modern cryptography relies on mathematical structures and methods, and this chapter contains the mathematical background from discrete mathematics, computational complexity and probability theory. We recapitulate elementary structures like sets, relations, equivalence classes and functions in Section 1.1. Fundamental combinatorial facts are outlined in Section 1.2 and the asymptotic notation is explained. Section 1.3 discusses complexity and the Big-O notation. Section 1.4 then deals with basic probability theory. Random numbers and the birthday problem are addressed in Section 1.5.

For a general introduction to undergraduate mathematics the reader may, for example, refer to the textbook [**WJW$^+$14**]. Discrete mathematics and its applications are discussed in [**Ros12**].

1.1. Sets, Relations and Functions

The most elementary mathematical structure is sets.

Definition 1.1 (Cantor's definition of sets)**.** A *set* M is a well-defined collection of *distinct objects*. If the object x is a member of M, write $x \in M$ or otherwise $x \notin M$. One writes $N \subset M$ if N is a subset of M, i.e., if all elements of N are also elements of M.

Example 1.2. Basic examples of finite sets are $\{\ \} = \emptyset$ (the empty set), $\{0\}$ (a set with one element), $\{0, 1\}$ (the binary numbers) and $\{A, B, C, \dots, Z\}$ (the set of capital letters). The standard sets of numbers $\mathbb{N} \subset \mathbb{Z} \subset \mathbb{Q} \subset \mathbb{R} \subset \mathbb{C}$ (positive integers, integers, rational numbers, real and complex numbers) are examples of infinite sets.

Definition 1.3. The *cardinality* or *size* of a finite set X is the number of its elements and is denoted by $|X|$. ◊

Note that there are other notions of *size* (see Warning 1.34 below): the size of an integer is the *number of bits* needed to represent it and the size of a binary string is its *length.*

Sets can be defined *explicitly*, for example by enumeration or by intervals of real numbers, or *implicitly* by formulas.

Example 1.4. $M = \{x \in \mathbb{Z} \mid x^4 < 50\}$ implicitly describes the set of integers x for which $x^4 < 50$ holds. This set can also be described explicitly:

$$M = \{-2,\ -1,\ 0,\ 1,\ 2\}. \qquad \diamondsuit$$

There are several elementary set operations: $M \cup N$ (union), $M \cap N$ (intersection), $M \setminus N$ (set difference), $M \times N$ (Cartesian product), M^n (n-ary product) and $\mathcal{P}(M)$ (power set).

Example 1.5. $M = \{0,1\}^{128}$ is the set of binary strings of length 128. Elements in M can be written in the form $b_1 b_2 \dots b_{128}$ or $(b_1,\ b_2,\ \dots,\ b_{128})$ in vectorial notation. An element of M could, for example, represent one block of plaintext or ciphertext data. The cardinality of M is very large:

$$|M| = 2^{128} \approx 3.4 \cdot 10^{38}$$

```
sage: 2^128
340282366920938463463374607431768211456
```

Remark 1.6. It is useful to help understand the difference between small, big and inaccessible numbers in practical computations. For example, one can easily store one terabyte (10^{12} bytes, i.e., around 2^{43} bits) of data. On the other hand, a large amount of resources are required to store one exabyte (one million terabytes) or 2^{63} bits, and more than 2^{100} bits are out of reach.

The number of computing steps is also bounded: less than 2^{40} steps (say CPU clocks) are easily possible, 2^{60} operations require a lot of computing resources and take a significant amount of time, and more than 2^{100} operations are unfeasible. It is for example impossible to test 2^{128} different keys with conventional (non-quantum) computers.

Definition 1.7. A *function*, *mapping* or *map* $f : X \to Y$ consists of two sets (the *domain* X and the *codomain* Y) and a rule which assigns an output element (an *image*) $y = f(x) \in Y$ to each input element $x \in X$. The set of all $f(x)$ is a subset of Y called the *range* or the *image* $im(f)$. Any $x \in X$ with $f(x) = y$ is called a *preimage* of y. Let $B \subset Y$; then we say that $f^{-1}(B) = \{x \in X \mid f(x) \in B\}$ is the *preimage* or *inverse image* of B under f.

Example 1.8. Let $f : \{0,1\}^4 \to \{0,1\}$ be defined by

$$f(b_1, b_2, b_3, b_4) = b_1 \oplus b_2 \oplus (b_3 \cdot b_4) = b_1 \oplus b_2 \oplus b_3 b_4.$$

Refer to Table 1.1 for the definition of XOR ($\oplus$) and AND ($\cdot$). For example, $(1,1,1,1)$ is a preimage of 1 and $(0,1,0,0)$ is another preimage of 1. $(0,0,0,0)$ is a preimage of 0 and the image of f is $im(f) = \{0,1\}$. The function f is surjective, but not injective (see Definition 1.10 below).

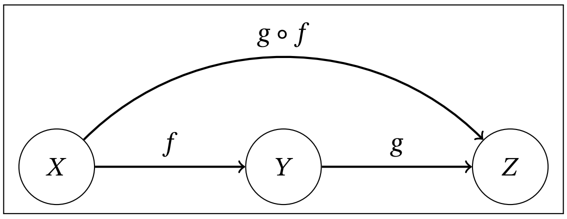
Table 1.1. XOR and AND operations.

$\oplus$	0	1
0	0	1
1	1	0

$\cdot$	0	1
0	0	0
1	0	1

Warning 1.9. A mathematical function is not the same as an *algorithm*. An algorithm is a step-by-step and possibly recursive procedure which produces output from given input. There can be different algorithms (or no known algorithm at all) for a given mathematical function. Furthermore, algorithms can also use random data as input and can have different behaviors on different runs. In computer science, a function is a piece of code in some programming language that performs a specific task.

Every set X has an *identity function* $id_X : X \to X$, which maps each $x \in X$ to itself. Functions can be *composed* if the range of the first function lies within the domain of the second function. Let $f : X \to Y$ and $g : Y \to Z$ be functions. Then there is a composite function $g \circ f : X \to Z$ with $(g \circ f)(x) = g(f(x))$ (see Figure 1.1).



$$\begin{tikzcd} X \arrow[r, "f"] \arrow[rr, bend left, "g \circ f"] & Y \arrow[r, "g"] & Z \end{tikzcd}$$

Figure 1.1. Composition of functions.

Definition 1.10. Let $f : X \to Y$ be a function.

- f is *injective* if different elements of the domain map to different elements of the range: for all $x_1,\ x_2 \in X$ with $x_1 \neq x_2$ we have $f(x_1) \neq f(x_2)$. Equivalently, f is injective if for all $x_1,\ x_2 \in X$:

$$f(x_1) = f(x_2) \Rightarrow x_1 = x_2.$$

- f is *surjective* or *onto* if every element of the codomain Y is contained in the image of f, i.e., for every $y \in Y$ there exists an $x \in X$ (a *preimage*) with $f(x) = y$. In other words, f is surjective if
$$im(f) = Y.$$
- f is *bijective* if it is both injective and surjective. Bijective functions possess an inverse map $f^{-1} : Y \to X$ such that $f^{-1} \circ f = id_X$ and $f \circ f^{-1} = id_Y$ (see Figure 1.2). Such functions are also called *invertible*.

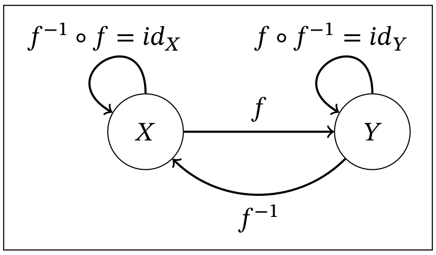


Figure 1.2. Inverse map and its properties.

Example 1.11. Let $f : \{0,1\}^2 \to \{0,1\}^2$ be defined by
$$f(b_1, b_2) = (b_1 \oplus b_2,\ b_1).$$
Since $f(0,0) = (0,0)$, $f(0,1) = (1,0)$, $f(1,0) = (1,1)$ and $f(1,1) = (0,1)$, the map is bijective. The inverse map $f^{-1} : \{0,1\}^2 \to \{0,1\}^2$ is given by
$$f^{-1}(b_1, b_2) = (b_2,\ b_1 \oplus b_2).$$
One has $f^{-1}(0,0) = (0,0)$, $f^{-1}(0,1) = (1,1)$, $f^{-1}(1,0) = (0,1)$ and $f^{-1}(1,1) = (1,0)$. ◊

Invertible (bijective) maps play an important role in ciphering. An encryption map E must have a corresponding decryption map D such that $D \circ E = id$ (the identity map on the plaintext space). Encryption maps are often bijective, although a slightly weaker property is sufficient (E has to be injective and D must be surjective). Ciphering is often defined as a composition of several operations. These operations are not necessarily bijective, even if their composition is ultimately bijective. On the other hand, a composition of bijective maps is of course bijective.

Lemma 1.12. *Let $f : X \to Y$ be a map between finite sets; then:*

(1) *If f is injective then $|X| \leq |Y|$.*
(2) *If f is surjective then $|X| \geq |Y|$.*
(3) *If f is bijective then $|X| = |Y|$.* ◊

Note that the above conditions are only necessary and not sufficient.

Remark 1.13. The contraposition of Lemma 1.12 (1) is called the *pigeonhole principle*: if $|X| > |Y|$ then f is not injective. Suppose X is a set of pigeons and Y a set of holes. If there are more pigeons than holes, then one hole has more than one pigeon.

Definition 1.14. A set S is said to be *countably infinite* if there is a bijective map

$$f : S \to \mathbb{N}$$

from S to the set of natural numbers. We say that a set S is *countable* if it is finite or countably infinite.

Example 1.15. $\mathbb{N}$ and $\mathbb{Z}$ are countably infinite sets. The sets $\mathbb{Z}^n$ (for $n \in \mathbb{N}$) and $\mathbb{Q}$ are also countable (see Exercise 3). However, a famous result of Cantor says that the set $\mathbb{R}$ of all real numbers is *uncountable*. ♢

The *floor*, *ceiling* and *rounding* functions are often used in numerical computations.

Definition 1.16. Let $x \in \mathbb{R}$.

(1) $\lfloor x \rfloor$ is the greatest integer less than or equal to x.
(2) $\lceil x \rceil$ is the least integer greater than or equal to x.
(3) $\lfloor x \rceil = \lfloor x + \frac{1}{2} \rfloor$ is rounding x to the nearest integer (round half up).

Example 1.17. $\lfloor 1.7 \rfloor = 1$, $\lceil -2.4 \rceil = -2$ and $\lfloor 1.5 \rceil = 2$. ♢

Functions from X to Y are particular cases of *relations between X and Y*:

Definition 1.18. A binary relation (or *relation*) between X and Y is a subset

$$R \subset X \times Y.$$

Hence a relation is a set of pairs (x, y) with $x \in X$ and $y \in Y$. If $X = Y$ then R is called a (binary) relation on X. ♢

A function $f : X \to Y$ defines a relation between X and Y:

$$R = \{(x, y) \in X \times Y \mid y = f(x)\}.$$

R contains all tuples $(x, f(x))$ with $x \in X$. This relation is also called the *graph* of f. However, a relation R between X and Y does not necessarily define a function. A relation R defines a function if and only if each $x \in X$ occurs exactly once as a first component in the relation R.

Equivalence relations on a set X are of special importance. An equivalence relation induces a segmentation of a set into disjoint subsets. The collection of these subsets defines a new set and elements in the same subset are said to be *equivalent*.

Definition 1.19. Let R be a relation on X. Then R is called an *equivalence relation* if it satisfies the following conditions:

(1) R is reflexive, i.e., $(x,x) \in R$ for all $x \in X$, and

(2) R is symmetric, i.e., if $(x,y) \in R$ then $(y,x) \in R$, and

(3) R is transitive, i.e., if $(x,y) \in R$ and $(y,z) \in R$ then $(x,z) \in R$. ◊

If R is an equivalence relation and $(x,y) \in R$, then x and y are called equivalent and we write $x \sim y$. For $x \in X$, the subset $\overline{x} = \{y \in X \mid x \sim y\} \subset X$ is called the *equivalence class* of x and all elements in $\overline{x}$ are representatives of that class. The above conditions of an equivalence relation ensure that the equivalence classes are disjoint and their union is X. The set of equivalence classes is said to be the *quotient set* $X/\sim$.

Note: the *elements* of $X/\sim$ are *sets*. By abuse of notation, the same x is sometimes viewed as an element of X and of $X/\sim$.

Example 1.20. We define an equivalence relation R_2 on $X = \mathbb{Z}$:

$$R_2 = \{(x,y) \in \mathbb{Z} \times \mathbb{Z} \mid x - y \in 2\mathbb{Z}\}.$$

Pairs $(x,y) \in R_2$ have the property that their difference $x - y$ is even, i.e., divisible by 2. For example, the tuples $(2,4)$, $(3,1)$ and $(-1,5)$ are all elements of R_2, but $(0,1) \notin R_2$ and $(-3,4) \notin R_2$. One can easily check that R_2 is an equivalence relation. The equivalence class of $x \in \mathbb{Z}$ is

$$\overline{x} = \{\ldots,\ x-4,\ x-2,\ x,\ x+2,\ x+4, \ldots\}.$$

There are only two *different* equivalence classes:

$$\overline{0} = \{\ldots, -4, -2, 0, 2, 4, \ldots\} \text{ and } \overline{1} = \{\ldots, -3, -1, 1, 3, 5, \ldots\}.$$

For example, one has $\overline{-2} = \overline{0} = \overline{2}$ and $\overline{-1} = \overline{1} = \overline{3}$. Hence the quotient set $\mathbb{Z}/\sim$ only has two elements. This set is denoted by $\mathbb{Z}_2$, $\mathbb{Z}/(2)$, $GF(2)$ or $\mathbb{F}_2$, and we call it the field of *residue classes modulo* 2. It can be identified with the binary set $\{0,1\}$. ◊

A generalization of the above example yields the *residue classes* modulo n:

Example 1.21. Let $n \in \mathbb{N}$ and $n \geq 2$. Consider the following equivalence relation R_n on $X = \mathbb{Z}$:

$$R_n = \{(x,y) \in \mathbb{Z} \times \mathbb{Z} \mid x - y \in n\mathbb{Z}\}.$$

Pairs $(x,y) \in R_n$ have the property that their difference $x-y$ is divisible by n, i.e., $x-y$ is a multiple of n. For example, let $n = 11$. Then the tuples $(2,13)$ and $(-1,10)$ are elements of R_{11}, but $(0,10) \notin R_{11}$ and $(-2,13) \notin R_{11}$.

Similar to the example above, R_n is an equivalence relation. The equivalence class of $x \in \mathbb{Z}$ is the set

$$\overline{x} = \{\ldots,\ x-2n,\ x-n,\ x,\ x+n,\ x+2n,\ \ldots\}.$$

Now we have n *different* equivalence classes and the quotient set $\mathbb{Z}/\sim$ has n elements. We call this set the *residue classes modulo n* or *integers modulo n* and denote it by $\mathbb{Z}_n$ or $\mathbb{Z}/(n)$. Each residue class has a *standard representative* in the set $\{0, 1, \dots, n-1\}$ and elements in the same residue class are called *congruent modulo n*. ♢

Two integers are congruent modulo n if they have the same remainder when they are divided by n. In many programming languages, the remainder of the integer division $a : n$ is computed by $a \;\%\; n$, but note that the result may be negative for $a < 0$, whereas the standard representative of a modulo n is non-negative.

Example 1.22. Let $n = 11$. Then $\mathbb{Z}_{11} = \{\overline{0}, \overline{1}, \dots, \overline{10}\}$ has 11 elements. One has $\overline{-14} = \overline{8}$ since $-14-8 = -22$ is a multiple of 11. The integers 8 and -14 are congruent modulo 11 and one writes $-14 \equiv 8 \mod 11$. The standard representative of this residue class is 8, and $-3, -14, \dots$ as well as $19, 30, \dots$ are other representatives of the same residue class. Here is an example using SageMath:

```
sage: mod(-892342322327,11)
6
```

The discussion of residue classes will be continued in Section 3.2.

Definition 1.23. A map $f : \{0,1\}^n \to \{0,1\}$ is called an n-variable *Boolean function* and a map $f : \{0,1\}^n \to \{0,1\}^m$ is called an (n,m)-vectorial Boolean function. A vectorial Boolean function can be written as an m-tuple of n-variable Boolean functions: $f = (f_1, f_2, \dots, f_m)$. ♢

Boolean functions can be represented by their *truth table*. Since the table of an n-variable Boolean function has 2^n entries, this is only reasonable for small n. Another important representation is the *algebraic normal form* (ANF). This form uses XOR ($\oplus$) and AND ($\cdot$) combinations of the binary variables.

An n-variable Boolean function has a unique representation as a polynomial in n variables, say $x_1, x_2, \dots, x_n$:

$$f(x_1, x_2, \dots, x_n) = \bigoplus_{I \subset \{1,\dots,n\}} a_I \cdot \prod_{i \in I} x_i.$$

The coefficients a_I are either 0 or 1 and f is a sum (XOR) of products (AND) of the variables, for example

$$f(x_1, x_2, x_3) = x_1 \oplus x_1x_2 \oplus x_2x_3 \oplus x_1x_2x_3.$$

One can view f as a polynomial in n variables with coefficients in $\mathbb{Z}_2$ (residue classes modulo 2) and write $+$ instead of $\oplus$, e.g.,

$$f(x_1, x_2, x_3) = x_1 + x_1x_2 + x_2x_3 + x_1x_2x_3 \mod 2.$$

The *algebraic degree* of f is the maximal length of the products which appear (with nonzero coefficient) in the above representation. If f is a constant function, then the degree is 0. In the above example, the degree is 3.

We note that higher powers of a variable x_i are not needed since $x_i^k = x_i$ for all $k \geq 1$ and $x_i \in \{0,1\}$. Boolean functions of degree ≤ 1 are called *affine*. If the degree is ≤ 1 and the constant part is 0, then the function is *linear*. An n-variable linear Boolean function is a linear mapping from $GF(2)^n$ to $GF(2)$. Linear maps are discussed in Section 4.4.

The *degree* of a vectorial Boolean function is the maximal degree of its component functions. A vectorial Boolean function is called affine if all component functions are affine.

Example 1.24. (1) $f(x_1,x_2,x_3) = x_1x_2+x_2x_3+x_1+1 \mod 2$ is a 3-variable Boolean function of algebraic degree 2.

(2) $f(x_1,x_2,x_3,x_4) = (x_3,\ x_1+x_2,\ x_1+x_4) \mod 2$ is a $(4,3)$-vectorial Boolean function. The algebraic degree is 1 and the function is linear since all constants are 0.

(3) Let $f = f(x_0,x_1)$ be a 2-variable Boolean function given by the following table:

x_1	x_0	$f(x)$
0	0	1
0	1	1
1	0	0
1	1	1

We use SageMath to determine the algebraic normal form:

```
sage: B = BooleanFunction([1,1,0,1])
sage: B.algebraic_normal_form()
x0*x1 + x1 + 1
```

We find that $f(x_0,x_1) = x_0x_1 + x_1 + 1 \mod 2$. The algebraic degree is 2 and f is neither linear nor affine.

1.2. Combinatorics

Combinatorics investigates finite or countable discrete structures. We are interested in properties of finite sets like $\{1,2,\ldots,n\}$ or $\{0,1\}^n$ which often occur in cryptographic operations.

Definition 1.25. The *factorial* of a non-negative integer n is defined as follows:

$$n! = \begin{cases} 1 & \text{for } n = 0, \\ 1 \cdot 2 \cdots n & \text{for } n \geq 1. \end{cases}$$

The *binomial coefficients* have the following formula:

$$\binom{n}{k} = \frac{n!}{k!\,(n-k)!} \text{ for integers } 0 \leq k \leq n\,. \qquad \Diamond$$

The factorial increases very fast, for example

$$25! = 15511210043330985984000000 \text{ and } 50! \approx 3.04 \cdot 10^{64}.$$

An efficient way to compute the binomial coefficients is the reduced formula

$$\binom{n}{k} = \binom{n}{n-k} = \frac{n(n-1)\cdots(n-(k-1))}{k!}\,.$$

Example 1.26. One has $\binom{15}{8} = \binom{15}{7} = \frac{15\cdot14\cdot13\cdot12\cdot11\cdot10\cdot9}{7!} = \frac{32432400}{5040} = 6435$. The following SageMath code computes all binomials $\binom{15}{n}$ for $0 \leq n \leq 15$:

```
sage: for n in range(0,16):
            print binomial(15,n),
1 15 105 455 1365 3003 5005 6435 6435 5005 3003 1365 455 105 15 1
```

The binomial coefficients appear for example in the *number of subsets* of a given finite set and in expansions of terms like $(a + b)^n$.

Proposition 1.27. *Let X and Y be finite sets. Then*

(1) $|X \times Y| = |X| \cdot |Y|$ *and* $|X^k| = |X|^k$ *for* $k \in \mathbb{N}$.

(2) *Let* $n = |X|$ *be the cardinality of X and* $k \leq n$. *Then the number of subsets of X of cardinality k is* $\binom{n}{k}$. ◊

Note that there is a difference between k-tuples and subsets of cardinality k. In the first case, the order of elements is important and in the second case it is not.

Example 1.28. There are $\binom{128}{2} = \frac{128\cdot127}{2} = 8128$ different binary words of length 128 with exactly two ones and 126 zeros. Indeed, each subset of $X = \{1, 2, \ldots, 128\}$ with two elements gives the two positions where the digit is equal to 1. ◊

Permutations of finite sets are of great importance in cryptography.

Definition 1.29. Let S be a finite set. A *permutation* σ of S is a bijective map $\sigma : S \to S$. ◊

Permutations of a finite set S can be written using a two-line matrix notation. The first row lists the elements of S and the image of each element is given below in the second row. The first row can be omitted if S is given and the elements are naturally ordered.

Example 1.30. Let $S = \{1, 2, 3, 4, 5, 6, 7, 8\}$; then the following row describes a permutation of S:

$$(5\ 7\ 1\ 2\ 8\ 6\ 3\ 4).$$

Proposition 1.31. *Let S be a finite set and $|S| = n$; then there are $n!$ permutations of S.*

Proof. Write $S = \{x_1, x_2, \dots, x_n\}$ and let σ be a permutation of S. There are n different choices for $\sigma(x_1)$. Since σ is injective, there remain $n - 1$ possible images for $\sigma(x_2)$, then $n-2$ possibilities for $\sigma(x_3)$, etc. The last image $\sigma(x_n)$ is fixed by the other choices. Hence one has $n(n-1)(n-2)\cdots 1 = n!$ different permutations for $\sigma : S \to S$. □

Cryptographic operations are often based on permutations. Indeed, a randomly chosen family of permutations of a set like $M = \{0, 1\}^{128}$ would constitute an ideal block cipher. Unfortunately, it is impossible to write down or to store a general permutation since M has 2^{128} elements (see Example 1.5). Much simpler are *linear maps* (see Section 4.4) and in particular *bit permutations*, which permute only the *position* of the bits. For example, we might take the permutation $(5\ 7\ 1\ 2\ 8\ 6\ 3\ 4)$ of $S = \{1, 2, 3, 4, 5, 6, 7, 8\}$ in Example 1.30 to define a permutation on the set $X = \{0, 1\}^8$. A byte $(b_1, b_2, \dots, b_8) \in X$ is mapped to $(b_5, b_7, b_1, b_2, b_8, b_6, b_3, b_4) \in X$. Such bit permutations are used within ciphering operations, but one has to take into account the fact that bit permutations are (in contrast to general permutations) *linear* maps.

1.3. Computational Complexity

One often needs to analyze the computational complexity of algorithms, i.e., the necessary time and space relative to the input size. The precise values, for example the number of steps or CPU-clocks and the required memory in bytes, depend on the platform, the implementation and the chosen measurement units, but the *growth* of time or space (in terms of the input size) is often independent of the operational environment and the parameters chosen. The *Big-O* notation describes the growth of functions and is used to estimate the complexity of algorithms in terms of the input size.

Definition 1.32 (Big-O notation). Let $f, g : \mathbb{N} \to \mathbb{R}$ be two functions on $\mathbb{N}$. Then we say that g is an *asymptotic upper bound* for f if there exists a real number $C \in \mathbb{R}$ and an integer $n_0 \in \mathbb{N}$ such that

$$|f(n)| \leq C\,|g(n)| \text{ for all } n \geq n_0,$$

and one writes $f = O(g)$ or $f \in O(g)$. ◊

An asymptotic upper bound $g(n)$ is usually chosen to be as simple and as small as possible. We say that f has linear, quadratic, cubic or polynomial growth in n if $f = O(n)$, $f = O(n^2)$, $f = O(n^3)$ or $f = O(n^k)$ for some $k \in \mathbb{N}$, respectively.

Example 1.33. (1) Let $f(n) = 2n^3 + n^2 + 7n + 2$. Since $n^2 \leq n^3$, $n \leq n^3$ and $1 \leq n^3$ for $n \geq 1$, one has $f(n) \leq (2 + 1 + 7 + 2)n^3$. Set $C = 12$ and $n_0 = 1$. Thus $f = O(n^3)$ and f has cubic growth in n.

(2) Let $f(n) = 100 + \frac{20}{n+1}$. Set $C = 101$ and $n_0 = 19$. Since $\frac{20}{n+1} \leq 1$ for $n \geq 19$, we have $f = O(1)$. Hence f is asymptotically bounded by a constant.

(3) Let $f(n) = 5\sqrt{2^{n+3} + n^2 - 2n}$; then $f = O(2^{n/2})$ so that f grows exponentially in n. ♢

The Big-O notation is often used to assess the running time of an algorithm in a worst-case scenario. An asymptotic upper bound does not depend on the measuring unit or the platform, since the values would differ only by a multiplicative constant. If the running time function $f(n)$ has polynomial growth in the input length n, i.e., $f = O(n^k)$, then $f(n)$ is bounded by Cn^k for constants C, k and large n. For example, if n is doubled, then the upper bound is multiplied only by the constant 2^k.

Algorithms with *polynomial* running time in terms of the input size are considered to be *efficient* or *fast*. Many standard algorithms, for example adding or multiplying two numbers, are polynomial. On the other hand, an algorithm that loops over every instance of a set with 2^n elements is *exponential* in n. A problem is called *hard* if no efficient, i.e., polynomial-time, algorithm exists that solves the problem.

A *decision problem* has only two possible answers, yes or no. Decision problems for which a polynomial time algorithm exists are said to belong to the *complexity class* **P**. The class **NP** (nondeterministic polynomial) is the set of decision problems which can be *verified* in polynomial-time. Checking the correctness of a proof is apparently easier than solving a problem. Whether the class **NP** is strictly larger than **P** is a major unresolved problem in computer science.

In computer science, one is usually interested in the *worst-case* complexity of an algorithm which solves a certain problem. However, the worst-case complexity is hardly relevant for attacks against cryptographic systems. A cryptosystem is certainly insecure if attacks are inefficient in certain bad cases but efficient in other cases. Instead, the *average-case* complexity of algorithms that break a scheme should be large. A secure scheme should provide protection in *almost all cases* and the probability that a polynomial time algorithm breaks the cryptosystem should be very small.

Note that there is not only the time complexity but also the *space complexity* of an algorithm. The space complexity measures the memory or storage requirements in terms of the input size.

Warning 1.34. The complexity is measured in terms of the input *size*, not the input value! The following formula gives the relation between a positive integer n and its size:

$$\text{size}(n) = \lfloor \log_2(n) \rfloor + 1.$$

The size of an integer is the number of bits that is needed to represent it. For a given binary string m, the bit-length is also called size and denoted by $|m|$.

In cryptographic applications, the key size is usually bounded by several hundred or thousand bits, but the numbers represented can be very large.

Example 1.35. Let $n = 10^{24}$. Then $\text{size}(n) = \lfloor 24 \cdot \log_2(10) \rfloor + 1 = 80$. Hence 80 bits are needed for the binary representation of 10^{24}.

Example 1.36. Let $n \in \mathbb{N}$ and $1^n = 1 \ldots 1$ be the *unary string* of n ones. Obviously, $\text{size}(1^n) = n$. An algorithm that takes 1^n as input is polynomial if the running time is polynomial in n. Of course, 1^n is a very inefficient way to represent a positive integer n, but it allows us to set the input *size* to n. This will be used in Chapter 2, where algorithms are often expected to be polynomial in a security parameter n, for example the key length.

Example 1.37. Suppose we want to compute the prime divisors of an integer n. A related decision problem is to determine whether or not n is a prime number.

Our algorithm uses trial divisions by all odd positive integers $\leq \sqrt{n}$. The number of divisions (i.e., a multiple of the running time) is at most $\lfloor \frac{1}{2}\sqrt{n} \rfloor$. Since $\sqrt{n} = 2^{\frac{1}{2}\log_2 n}$, the worst-case running time function is $O(2^{\frac{1}{2}\,\text{size}(n)})$ and therefore *exponential*. On the other hand, our algorithm has polynomial space complexity, since we only need to store n and a small number of auxiliary variables. Note that the factor $\frac{1}{2}$ in the *exponent* of the running time function must not be omitted in the Big-O notation! $\diamondsuit$

One should understand the limitations of the asymptotic notation. First, the upper bound applies only if n is larger than some unknown initial value n_0. Furthermore, the multiplicative constant C can be large. For a given input value, an algorithm with polynomial running time may even be slower than an exponential-time algorithm! However, for large n the polynomial-time algorithm will eventually be faster.

Bounded functions are of type $O(1)$ and functions of type $O(\frac{1}{n})$ converge to 0. *Negligible* functions approach zero faster than $\frac{1}{n}$ and any other inverse polynomial:

Definition 1.38. Let $f : \mathbb{N} \to \mathbb{R}$ be a function. One says that f is *negligible* in n if $f = O(\frac{1}{q(n)})$ for all polynomials q, or equivalently if $f = O(\frac{1}{n^c})$ for all $c > 0$. $\diamondsuit$

Hence negligible functions are eventually smaller than any inverse polynomial. This means that $f(n)$ approaches zero faster than any of the functions $\frac{1}{n}, \frac{1}{n^2}, \frac{1}{n^3}$, etc.

Example 1.39. $f(n) = 10e^{-n}$ and $2^{-\sqrt{n}}$ are negligible in n, whereas $f(n) = \frac{1}{n^2+3n}$ is not negligible since $f(n) = O(\frac{1}{n^2})$, but $f \neq O(\frac{1}{n^3})$. ♢

Finally, we define the *soft-O* notation which ignores logarithmic factors.

Definition 1.40. Let $f, g : \mathbb{N} \to \mathbb{R}$ be functions. Then $f = \tilde{O}(g)$ if there exists $k \in \mathbb{N}$ such that $f = O(\ln(n)^k g(n))$.

Example 1.41. (1) Let $f_1(n) = \log_2(n)$. Then $f_1 = \tilde{O}(1)$.

(2) Let $f_2(n) = \log_{10}(n)^3 n^2$. Then $f_2 = \tilde{O}(n^2)$.

1.4. Discrete Probability

Probability theory is an important foundation of modern cryptography. We only consider *discrete probability spaces.*

Definition 1.42. Let Ω be a countable set, $\mathcal{S} = \mathcal{P}(\Omega)$ the power set of Ω and $Pr : \mathcal{S} \to [0, 1]$ a function with the following properties:

(1) $Pr[\Omega] = 1$.

(2) If $A_1, A_2, A_3, \ldots$ are pairwise disjoint subsets of A, then

$$Pr\left[\bigcup_i A_i\right] = \sum_i Pr[A_i].$$

The triple $(\Omega, \mathcal{S}, Pr)$ is called a *discrete probability space.* Ω is called the *sample space* and we say that Pr is a *discrete probability distribution* on Ω. The subsets $A \subset \Omega$ are said to be *events* and $Pr[A]$ is the *probability* of A. If Ω is finite, then $(\Omega, \mathcal{S}, Pr)$ is called a *finite probability space.* ♢

Note that the family of sets in (2) is either finite or countably infinite. Since all probabilities are non-negative and the sum is bounded by 1, the series converges and is also invariant under a reordering of terms.

Remark 1.43. In measure theory, a triple $(\Omega, \mathcal{S}, Pr)$, where Pr is a σ-additive function on a set $\mathcal{S} \subset \mathcal{P}(\Omega)$ of *measurable sets* such that $Pr[\Omega] = 1$, is called a *probability space.* We only consider the case of a *countable* sample space Ω and assume that all subsets (events) are measurable, i.e., $\mathcal{S} = \mathcal{P}(\Omega)$. A discrete probability distribution is fully determined by the values on the singletons $\{\omega\}$ (the elementary events). We define the function

$$p(\omega) = Pr[\{\omega\}]$$

and obtain for all events $A \subset \Omega$:

$$Pr[A] = \sum_{\omega \in A} p(\omega).$$

Hence a discrete probability distribution is fully determined by a function $p : \Omega \to [0,1]$ with $\sum_{\omega\in\Omega} p(\omega) = 1$.

Example 1.44. Let $\Omega = \mathbb{N}$ and $0 < p < 1$. Define a discrete probability distribution Pr on $\mathbb{N}$ by

$$p(k) = Pr[\{k\}] = (1-p)^{k-1}p.$$

The formula for the geometric series implies that $Pr[\mathbb{N}] = 1$:

$$\sum_{k=1}^{\infty} p(k) = p\sum_{k=1}^{\infty}(1-p)^{k-1} = p\,\frac{1}{1-(1-p)} = 1.$$

Pr is called a *geometric distribution*. Suppose the probability of success in an experiment is equal to p. Then the geometric distribution gives the probability that the first success requires k independent trials.

Definition 1.45. Let Pr be a probability distribution on a finite sample space Ω. Pr is a *uniform distribution* if all elementary events have equal probability, i.e., if

$$p(\omega) = Pr[\{\omega\}] = \frac{1}{|\Omega|} \text{ for all } \omega \in \Omega. \qquad \diamondsuit$$

Examples of uniform distributions include a fair coin or a dice. Random number generators should produce uniformly distributed numbers. A uniform distribution is sometimes assumed when other information is not available. Cryptographic schemes (see Chapter 2) often assume that a key is chosen *uniformly at random* so that all possible keys have the same probability.

Definition 1.46. Let A, B and $A_1, \dots, A_n$ be events in a probability space.

(1) A and B are said to be *independent* if the joint probability equals the product of probabilities:

$$Pr[A \cap B] = Pr[A] \cdot Pr[B].$$

(2) $A_1, A_2, \dots, A_n$ are *mutually independent* if for every non-empty subset of indices $I \subset \{1, \dots, n\}$, one has

$$Pr\left[\bigcap_{i\in I} A_i\right] = \prod_{i\in I} Pr[A_i]\,. \qquad \diamondsuit$$

Note that mutual independence of n events is stronger than pairwise independence of all pairs (see Example 1.53 below).

Definition 1.47. Let A and B be two events in a probability space and suppose that

$Pr[A] \neq 0$. Then the *conditional probability* $Pr[B|A]$ of B given A is defined by

$$Pr[B|A] = \frac{Pr[A \cap B]}{Pr[A]}. \qquad \Diamond$$

Obviously, $Pr[B|A] = Pr[B]$ holds if and only if A and B are independent.

The sample space Ω of a probability space is often mapped to a standard space and in particular to the real numbers. Such a map on Ω is called a *random variable*. For example, the outcome of rolling two dice can be mapped to $\{1, 2, \dots, 6\}^2$ or to $\{2, 3, \dots, 12\}$ if the numbers on the dice are added.

Definition 1.48. Let $Pr : \Omega \to [0, 1]$ be a discrete probability distribution; then a function $X : \Omega \to \mathbb{R}$ is called a *real random variable*. For any $x \in \mathbb{R}$, one obtains an event $X^{-1}(x) \subset \Omega$ and the probability $Pr[X = x]$ is defined by $Pr[X^{-1}(x)]$. The function $p_X : \mathbb{R} \to [0, 1]$ defined by $p_X(x) = Pr[X = x]$ is called the *probability mass function* (pmf) of X. Furthermore, the *cumulative distribution function* (cdf) $F : \mathbb{R} \to \mathbb{R}$ is defined by $F(x) = Pr[X \leq x]$. $\Diamond$

Note that X induces a discrete probability distribution Pr_X on the countable subset $X(\Omega) \subset \mathbb{R}$. The difference to the original distribution Pr is that the sample space of Pr_X is now a subset of $\mathbb{R}$. If the sample space Ω is already a subset of $\mathbb{R}$, then X is usually the inclusion map.

Example 1.49. Suppose two dice are rolled and the random variable X gives the sum of numbers on the dice. Then $X^{-1}(2) = \{(1, 1)\}$ and $X^{-1}(3) = \{(1, 2), (2, 1)\}$, so that $p_X(2) = P[X = 2] = \frac{1}{36}$, $p_X(3) = P[X = 3] = \frac{1}{36} + \frac{1}{36} = \frac{1}{18}$.

Furthermore, $F(x) = 0$ for $x < 2$, $F(2) = \frac{1}{36}$, $F(3) = \frac{1}{36} + \frac{1}{18} = \frac{1}{12}$, etc., and $F(x) = 1$ for $x \geq 12$.

Definition 1.50. Let Pr be a discrete probability distribution and $X : \Omega \to \mathbb{R}$ a random variable with countable range $X(\Omega) \subset \mathbb{R}$. One defines the *expected value* (also called *expectation*, *mean* or *average*) $E[X]$ and the *variance* $V[X]$ if the sums given below are either finite or the corresponding series converge absolutely:

$$E[X] = \sum_{x \in X(\Omega)} x \cdot Pr[X = x] = \sum_{x \in X(\Omega)} x \cdot p_X(x),$$

$$V[X] = E[(X - E[X])^2] = E[X^2] - (E[X])^2 = \left(\sum_{x \in X(\Omega)} x^2 \cdot p_X(x) \right) - (E[X])^2.$$

The square root $\sigma = \sqrt{V[X]}$ of the variance is called the *standard deviation*. It measures the quadratic deviation from the mean $E[X]$.

Example 1.51. (1) Let Pr be a uniform distribution on a finite set Ω. Assume that the random variable X maps Ω to the set $\{0,\ 1,\ \dots,\ n-1\}$. The pmf is $p_X(x) = \frac{1}{n}$

for $x = 0, 1, \dots, n-1$, and zero otherwise. The expectation is

$$E[X] = \sum_{k=0}^{n-1} k \cdot \frac{1}{n} = \frac{n(n-1)}{2} \cdot \frac{1}{n} = \frac{n-1}{2}.$$

The variance is

$$\begin{aligned} V[X] &= \left(\sum_{k=0}^{n-1} k^2 \cdot \frac{1}{n} \right) - \left(\frac{n-1}{2} \right)^2 \\ &= \frac{n(n-1)(2n-1)}{6} \cdot \frac{1}{n} - \frac{n^2-2n+1}{4} = \frac{n^2-1}{12}. \end{aligned}$$

(2) Consider the experiment of tossing a perfect coin and letting X be the associated random variable having values 0 and 1. It is hardly surprising that the expected value is $E[X] = \frac{1}{2}$. The variance is $V[X] = \frac{1}{4}$ and the standard deviation is $\sigma = \frac{1}{2}$.

Definition 1.52. Let $X_1, X_2, \dots, X_n$ be random variables on a discrete probability space with probability mass functions $p_{X_1}, p_{X_2}, \dots, p_{X_n}$. The random variables are called *mutually independent* if for any sequence $x_1, x_2, \dots, x_n$ of values

$$Pr[X_1 = x_1 \wedge X_2 = x_2 \wedge \cdots \wedge X_n = x_n] = p_{X_1}(x_1) \cdot p_{X_2}(x_2) \cdots p_{X_n}(x_n).$$

Example 1.53. Let X_1 and X_2 be two binary random variables (values 0 or 1) that are given by tossing two perfect coins so that X_1 and X_2 are independent. Now set $X_3 = X_1 \oplus X_2$. Then X_1, X_2, X_3 are pairwise independent and each of them has a uniform distribution, but they are not mutually independent. We have

$$Pr[X_1 = 1 \wedge X_2 = 1 \wedge X_3 = 1] = 0,$$

since X_3 must be zero if $X_1 = X_2 = 1$, but

$$Pr[X_1 = 1] \cdot Pr[X_2 = 1] \cdot Pr[X_3 = 1] = \left(\frac{1}{2}\right)^3 = \frac{1}{8}.$$

Example 1.54. Let $\Omega = \{0,1\}^8$ be a space of plaintext and ciphertexts. Suppose the plaintexts X are uniformly distributed. Let $\sigma : \Omega \to \Omega$ be a random *bit permutation* (see Section 1.2). Then the ciphertexts $Y = \sigma(X)$ are also uniformly distributed. X and Y are independent if

$$Pr[X = m \wedge Y = c] = Pr[X = m] \cdot Pr[Y = c]$$

holds for all plaintexts m and ciphertexts c. The right side of the equation gives $\frac{1}{2^8} \cdot \frac{1}{2^8} = \frac{1}{2^{16}}$ for all m and c. If m and c possess a different number of ones, then the left side is 0, because such a combination is impossible for a bit permutation. This shows that X and Y are not independent. Later we will see that bit permutations are not secure, since the ciphertext leaks information about the plaintext.

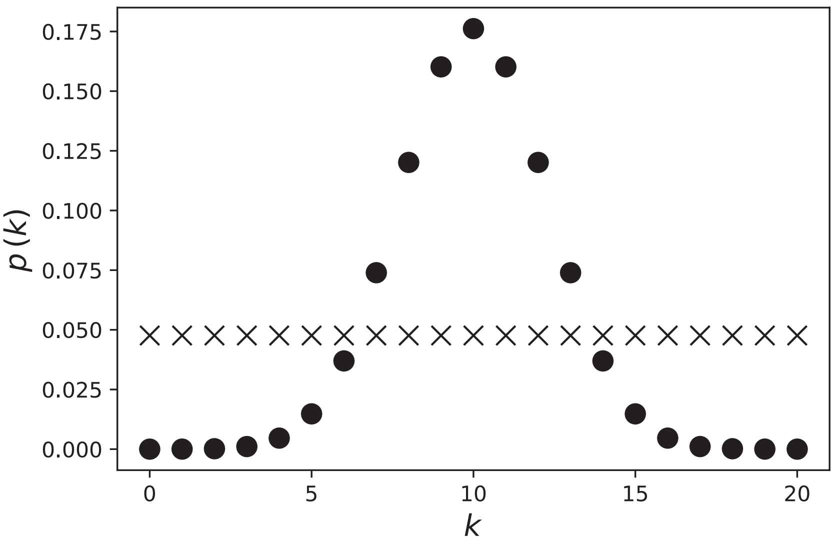


Figure 1.3. Probability mass functions of i) the binomial distribution $B(20, \frac{1}{2})$ (•) and ii) the uniform distribution (×) on $\{0, 1, 2, \dots, 20\}$.

Example 1.55. Let Pr be a probability distribution on a sample space Ω with two elements. Suppose $X : \Omega \to \{0, 1\}$ is a random variable with $Pr[X = 1] = p$ (success) and $Pr[X = 0] = 1 - p$ (failure). This is called a *Bernoulli trial*. Furthermore, let $X_1, \dots, X_n$ be n independent identical distributed (i.i.d.) random variables with $X_i = X$, and define

$$Y = X_1 + X_2 + \cdots + X_n.$$

The new random variable Y follows a *binomial distribution* $B(n, p)$ and gives the number of successes in n independent Bernoulli trials. For $k \in \{0, 1, \dots, n\}$ one has

$$Pr[Y = k] = \binom{n}{k} p^k (1-p)^{n-k},$$

since there are $\binom{n}{k}$ combinations of n trials with k successes and $n - k$ failures. The probability of each combination is $p^k(1-p)^{n-k}$. We have $E[X] = p$, $E[Y] = np$, $V[X] = p(1-p)$ and $V[Y] = np(1-p)$.

1.5. Random Numbers

Random numbers or random bits play an important role in cryptography. Keys are often chosen uniformly at random, and many cryptographic algorithms are probabilistic and require random input. One distinguishes between *true* random numbers in the sense of probability theory and *pseudorandom* numbers or bits. The latter are produced by deterministic algorithms, which take a short random input seed as input and generate a long output sequence that *appears* to be random. Pseudorandom generators are discussed in Section 2.8.

Definition 1.56. A random bit generator (RBG) is a mechanism or device which generates a sequence of random bits, such that the corresponding sequence of binary random variables $X_1, X_2, X_3, \dots$ has the following properties:

(1) $Pr[X_n = 0] = Pr[X_n = 1] = \frac{1}{2}$ for all $n \in \mathbb{N}$ (uniform distribution) and

(2) $X_1, X_2, \dots, X_n$ are mutually independent for all $n \in \mathbb{N}$.

Example 1.57. If at least one output bit is a combination of the other bits, for example if $X_3 = X_1 \oplus X_2$ (see Example 1.53), then this does not give a random bit sequence. This demonstrates that the obvious constructions to 'stretch' a given sequence cannot be used. ◊

Random bits or numbers can be produced manually (for example coin tossing, die rolling or mouse movements) or with *hardware random number generators*, which use physical phenomena like thermal noise, electrical noise or nuclear decay. Unfortunately, these mechanisms or devices tend to be slow, elaborate and/or costly. Fast all-digital random bit generators on current processor chips use thermal noise, but whether such generators can be trusted and do not have any weaknesses or even contain backdoors is disputed.

Remark 1.58. The required *uniform distribution* of the output of a bit generator can be achieved by *de-skewing* a possibly biased generator (see Example 1.59 below), but the statistical *independence* of the output bits is hard to achieve and difficult to prove.

Example 1.59. Von Neumann proposed the following de-skewing technique (*von Neumann extractor*): group the output bits into pairs, then turn 01 into 0 and 10 into 1. If the bits are independent, then the pairs 01 and 10 must have the same probability. The pairs 00 and 11 are discarded. The derived generator is slower but unbiased. ◊

The generation of *random bits* is basically equivalent to the generation of *random integer numbers*. In cryptography, random data is needed quite often. Random bits are used to seed deterministic pseudorandom generators (see Definition 2.32). In practice, the generators are often based on hash functions, HMAC or block ciphers in counter mode (see [**BK15**]). These cryptographic primitives are explained in the following chapters. The output of such generators appears totally unpredictable to an observer who does not know the seed, although deterministic functions are used to derive the output bits.

Random bits are needed to *generate keys*. Another application are *probabilistic* algorithms which take random data as auxiliary input. This may seem odd, but suppose an algorithm sometimes gives an incorrect result. Repeated application with fresh random input then reduces the failure rate. Furthermore, probabilistic algorithms ensure that the same input can have a different output value, and this can be a desirable property for data encryption.

Random numbers match surprisingly often. This is known as the *Birthday Problem* or *Birthday Paradox*.

Example 1.60. Assume that 23 people are in a certain place. Then the probability that at least two of them have their birthday on the same day of the year is *above* 50%. Intuitively, one would expect that around $\frac{365}{2}$ people would be needed for a probable birthday match. $\diamondsuit$

The explanation of this 'paradox' is quite simple: the probability p that no collision occurs (i.e., all birthdays are different) decreases exponentially with the number n of persons. We assume that birthdays are uniformly distributed. For $n = 2$, one has $p = \frac{364}{365}$. For $n = 3$, one gets $p = \frac{364}{365} \cdot \frac{363}{365}$, and each increment of n yields another factor. We write a SageMath function and obtain $p \approx 0.493$ for $n = 23$. The complementary probability $1-p$ for a birthday collision with 23 people therefore lies above 0.5.

```
sage: def birthday(k,n):
          p=1
          for i in range(1,k):
               p=p*(1-i/n)
          print RR(p)
sage: birthday(23,365)
0.492702765676015
```

Proposition 1.61. *Let Pr be a uniform distribution on a set Ω of cardinality n. If we draw $k = \left\lceil \sqrt{2\ln(2)n} \right\rceil \approx 1.2\sqrt{n}$ independent samples from Ω, then the probability of a collision is around* 50%.

Proof. See Exercise 16. $\square$

Example 1.62. Consider collisions of binary random strings of length l. The above Proposition 1.61 shows that collisions are expected after around $\sqrt{2^l} = 2^{l/2}$ values. If the strings are sufficiently long and uniformly distributed, then random collisions should almost never occur. For example, for 256-bit hash values with a uniform distribution, about 2^{128} hash values are required for a collision. On the other hand, only around 100,000 32-bit strings will probably have a collision. $\diamondsuit$

The running time of finding a collision among binary strings of length l is $O(2^{l/2})$. Unfortunately, a large amount of space is also required, since all $O(2^{l/2})$ strings have to be stored to detect a collision.

An optimization is possible if the samples are defined recursively by a function:

$$x_i = f(x_{i-1}) \text{ for } i \geq 1,$$

where x_0 is some initial value. Now the problem is to find a cycle in a sequence of iterated function values. *Floyd's cycle-finding algorithm* uses very little memory and is based on the following observation:

Proposition 1.63. *Let $f : X \to X$ be a function on some set X, $x_0 \in X$ and $x_i = f(x_{i-1})$ for $i \geq 1$. Suppose there exist $i, j \in \mathbb{N}$ such that $i < j$ and $x_i = x_j$. Then there exists an integer $k < j$ such that*

$$x_k = x_{2k}\,.$$

Proof. Let $\Delta = j - i$; then $x_i = x_{i+\Delta}$ and hence $x_k = x_{k+\Delta} = x_{k+m\Delta}$ for all integers $k \geq i$ and $m \geq 1$. Now let $k = m\Delta$, where $m\Delta$ is the smallest multiple of Δ that is also greater than or equal to i. The sequence $i, i+1, \dots, j-1$ of Δ consecutive integers contains the required number $k = m\Delta$. Therefore, $x_k = x_{2k}$ and $k = m\Delta < j$. □

Note that a collision must exist if X is a finite set. The above Proposition 1.63 implies that a collision in $x_0, x_1, \dots, x_j$ yields a collision of the special form $x_k = x_{2k}$ for some $k < j$. The least period of the sequence divides k. It is therefore sufficient to compute the pairs (x_i, x_{2i}) for $i = 1, 2, \dots$ until a collision occurs. These values can be recursively calculated:

$$x_i = f(x_{i-1}) \text{ and } x_{2i} = f(f(x_{2(i-1)})).$$

Assuming that the sequence $x_0, x_1, \dots$ is uniformly distributed and $|X| = n$, the running time is still $O(\sqrt{n})$, but now it is sufficient to store only two values. This approach is used in birthday attacks against hash functions and in *Pollard's ρ algorithms* for factoring and discrete logarithms. The sequence $x_0, x_1, \dots$ can be depicted by an initial tail and a cycle so that it looks like the greek letter ρ.

Remark 1.64. We only consider the part of Floyd's algorithm which finds a collision. The algorithm can also compute the least period, i.e., the length of the shortest cycle, and find the beginning of the cycle.

Example 1.65. Let $X = \mathbb{Z}_{107}$ be the set of residue classes modulo 107 and let

$$f(x) = x^2 + 26 \mod 107.$$

Set $x_0 \equiv 1 \mod 107$ and let $x_i = f(x_{i-1})$ for $i \geq 1$. We want to find a collision within the sequence $x_0, x_1, x_2, \dots$ and implement Floyd's cycle finding algorithm:

```
sage: def f(x):
          return(x*x+26)
sage: x=mod(1,107)
sage: y=mod(1,107)
sage: x=f(x)
sage: y=f(f(y))
sage: k=1
sage: while x!=y:
          x=f(x)
```

```
            y=f(f(y))
            k=k+1
            print  "k =",k," x =",x
k = 9  x = 39
```

Hence $x_9 = x_{18} = 39$ is a collision. Let's compute the first few elements of the sequence and verify the result:

```
sage: x=mod(1,107)
sage: for i in range(46):
            x=f(x)
            print("{:2}".format(x)),
27 6  62 18 29 11 40 21 39 49 73 5  51 59 83 67 21 39 49 73 5  51
59 83 67 21 39 49 73 5  51 59 83 67 21 39 49 73 5  51 59 83 67 21
```

The first seven elements form the initial segment (tail) of the sequence. The beginning of the cycle is $x_8 = 21$, and the sequence $21, 39, 49, \ldots$ is cyclic of period 9.

1.6. Summary

- Sets, relations and functions are fundamental in mathematics and cryptography.
- Residue classes are defined by an equivalence relation on the integers. There are n different residue classes modulo n and the standard representatives are $0, 1, \ldots, n-1$.
- Permutations are bijective functions on a finite set.
- The Big-O notation is used to give an asymptotic upper bound of the growth of a function in one variable.
- Algorithms with polynomial running time in terms of the input size are considered as efficient. The decision problems that can be solved in polynomial time form the complexity class **P**.
- Negligible functions are eventually smaller than any inverse polynomial.
- A discrete probability space consists of a countable sample space Ω and a probability distribution Pr on Ω. A real random variable X is a mapping from Ω to $\mathbb{R}$ and gives a probability distribution on $\mathbb{R}$.
- A true random bit generator can be described by a sequence of mutually independent binary and uniformly distributed random variables.
- The birthday paradox states that collisions of uniformly distributed random numbers occur surprisingly often.

Exercises

1. Let $X = ([-1,1] \cap \mathbb{Z}) \times \{0,1\}$. Enumerate the elements of X and determine $|X|$. Let $Y = \{1, 2, \ldots, |X|\}$. Give a bijection from X to Y.
2. Which of the following maps are *injective*, *surjective* or *bijective*? Determine the image $im(f)$ and give the inverse map f^{-1}, if possible.
   (a) $f_1 : \mathbb{N} \to \mathbb{N}$, $f_1(n) = 2n + 1$.
   (b) $f_2 : \mathbb{Z} \to \mathbb{N}$, $f_2(k) = |k| + 1$.
   (c) $f_3 : \{0,1\}^8 \to \{0,1\}^8$, $f_3(b) = b \oplus (01101011)$.
   (d) $f_4 : \{0,1\}^8 \to \{0,1\}^8$, $f_4(b) = b$ AND (01101011).
3. Show that the following sets are countable:
   (a) $\mathbb{Z}$.
   (b) $\mathbb{Z}^2$.
   (c) $\mathbb{Q}$.
   Hint: It is sufficient to construct an injective function into $\mathbb{N}$.
4. Let $f : X \to Y$ be a map between finite sets and suppose that $|X| = |Y|$. Show the following equivalences:
$$f \text{ is injective} \iff f \text{ is surjective} \iff f \text{ is bijective.}$$
5. Let $f : X \to Y$ be a function.
   (a) Let $B \subset Y$. Show that $f(f^{-1}(B)) \subset B$ with equality occurring if f is surjective.
   (b) Let $A \subset X$. Show that $A \subset f^{-1}(f(A))$ with equality occurring if f is injective.
6. Enumerate the integers modulo 26. Find the standard representative of the following integers in $\mathbb{Z}_{26}$:
$$-1000,\ -30,\ -1,\ 15,\ 2001,\ 293829329302932398231.$$
7. Consider the following relation S on $\mathbb{R}$:
$$S = \{(x,y) \in \mathbb{R} \times \mathbb{R} \mid x - y \in \mathbb{Z}\}.$$
   Show that S is an equivalence relation on $\mathbb{R}$. Determine the equivalence classes $\overline{0}$, $\overline{-2}$ and $\overline{\frac{4}{3}}$. Can you give an interval I such that there is a bijection between I and the quotient set $\mathbb{R}/\sim$?
8. Find an asymptotic upper bound of the following functions in n. Which of them are polynomial and which are negligible?
   (a) $f_1 = 2n^3 - 3n^2 + n$.
   (b) $f_2 = 3 \cdot 2^n - 2n + 1$.
   (c) $f_3 = \sqrt{2n + 1}$.
   (d) $f_4 = \frac{2n}{2^{n/2}}$.
   (e) $f_5 = \frac{5n^2 - n}{2n^2 + 3n + 1}$.

(f) $f_6 = 2^{\frac{1}{3}n+3}$.
(g) $f_7 = \log_2(n)^2 n$.

9. Let $f = f(x_0, x_1, x_2)$ be a 3-variable Boolean function with the following truth table:

x_2	x_1	x_0	$f(x)$
0	0	0	1
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	0
1	0	1	0
1	1	0	1
1	1	1	1

Determine the algebraic normal form and the algebraic degree of f.

10. Suppose the number of operations of the most efficient attack against a cipher is

$$f(n) = e^{(2n^{1/3})},$$

where n is the key length. Why is $f(n)$ sub-exponential, but not polynomial in n? Compute *effective key length* $\log_2(f(n))$ for $n = 128$, $n = 1024$ and $n = 2048$.

11. Let $\Omega = \{0,1\}^8$ be a probability space with a uniform probability distribution. Compute the probability that a randomly chosen byte is balanced, i.e., contains four zeros and four ones.

12. Verify the expected values and variances in Example 1.55.

13. Two perfect dice are rolled and the random variables which give the numbers on the dice are called X and Y. Compute

$$Pr[X + Y \geq 10] \text{ and } Pr[X + Y \geq 10 \mid X - Y = 0].$$

Are the random variables $X + Y$ and $X - Y$ independent? Determine the expectation, the variance and the standard deviation of $X + Y$ and $X - Y$.

14. Suppose that 100 bits are generated uniformly at random and that additional bits are produced by XORing the preceding 100 bits. Are the new bits uniformly distributed? Does this construction give a random bit generator?

15. Let $X = \mathbb{Z}_{56807}$ and define a sequence by $x_0 \equiv 1$ and

$$x_i = x_{i-1}^2 + 1 \mod 56807$$

for $i \geq 1$. Use SageMath to find a collision and determine the least period of the sequence.

16. (Birthday Paradox) Let *Pr* be a uniform distribution on the sample space Ω with $|\Omega| = n$. If $k \le n$ samples are independently chosen, then the probability p that all k values are different (i.e., no collision occurs) is

$$p = \prod_{i=1}^{k-1} \left(1 - \frac{i}{n}\right).$$

   (a) Show that the probability $1 - p$ of a collision satisfies

$$1 - p \ge 1 - e^{-\frac{k(k-1)}{2n}}.$$

   (b) Determine the smallest number k such that $p \approx \frac{1}{2}$.

   Hint: Use the inequality $1 - x \le e^{-x}$ for $0 \le x \le 1$ and replace the factors $1 - \frac{i}{n}$ by $e^{-\frac{i}{n}}$. Compute the product and obtain a sum in the exponent. Use the formula $\sum_{i=1}^{k-1} i = \frac{k(k-1)}{2}$. For part (b), set $p = \frac{1}{2}$ and determine k using the quadratic formula. You may also approximate $k(k-1)$ by k^2. This gives the approximate number of samples needed for a probable collision.

Chapter 13

Quantum Computing

This chapter provides an introduction to *quantum computing*. We focus on quantum computation and cryptographic applications, not on physical realizations of quantum computers, and we do not require any previous knowledge in quantum mechanics. Quantum computers have amazing features: they can process a very large number of input values simultaneously and solve certain hard problems. At the time of this writing, quantum computers are not yet large and stable enough to break any real-world cryptosystems, but this might change within a decade. Today's quantum systems are noisy and quantum error correction is an important topic. There is now a broad range of research into post-quantum cryptography, and new schemes should be secure in the presence of quantum computers. Chapters 14 and 15 give an introduction to several post-quantum schemes.

The basic information unit of quantum computing is a quantum bit (qubit), which can assume not only two, but infinitely many states. However, only one classical bit can be extracted (or measured) from a qubit and the output is probabilistic. In Section 13.1, qubits and operations on single qubits are introduced. Systems with multiple qubits are dealt with in Section 13.2. Analogous to classical computing, quantum algorithms are realized with quantum gates. Since the output is probabilistic, quantum algorithms need to be carefully designed in order to obtain useful results. The Deutsch-Josza algorithm is explained in Section 13.3 and demonstrates the possibilities of quantum computing.

A major algorithm is the Quantum Fourier Transform (Section 13.4), which can efficiently find periods in a large (exponential) set of numbers. In Section 13.5, we show that Shor's algorithm can efficiently factorize integers and break the RSA cryptosystem.

Quantum bits can also be used for a secure key distribution, and we explain the BB84 quantum cryptographic protocol in Section 13.6.

Two recommended textbooks for further reading on quantum computing are [**NC00**] and [**RP11**].

13.1. Quantum Bits

A classical computer processes the input data sequentially in order to solve a problem. Turing machines or logical circuits can be used to model classical computation. A Boolean circuit is made up of elementary gates and performs a computational task, where input bits are transformed into output bits. In each step, the system has a fixed state and, depending on the complexity of an algorithm and the size of the input data, the computation can be inefficient or infeasible: suppose that f is a vectorial Boolean function and an algorithm iterates over all $x \in \{0,1\}^n$ until $f(x)$ is equal to a fixed value y. The worst-case running time is exponential in n and the algorithm is inefficient on conventional computers, even if the individual computations of $f(x)$ run very fast.

Now, quantum computers can compute *all* 2^n *values* $f(x)$ *simultaneously*. This sounds fantastic, but there is a catch: the computation requires a system of n qubits, the internal state is inaccessible and only a single output value can be extracted from a quantum system. Nevertheless, we will see that quantum algorithms can efficiently solve many hard problems.

Quantum computing uses *quantum circuits* instead of Boolean circuits, and quantum algorithms take *qubits* (quantum bits) instead of classical bits as input. A qubit is a two-level quantum-mechanical system and can have different physical implementations, for example the polarization of a photon, the spin of an electron or the state of an ion. The physical realization of quantum systems is an important field of research and development, but it is beyond the scope of this book.

A qubit can assume infinitely many states between 0 or 1; the state is represented by a normalized vector in the vector space $\mathbb{C}^2$. We fix an orthonormal basis of $\mathbb{C}^2$, for example the standard basis $e_1 = (1,0)$ and $e_2 = (0,1)$, and denote the basis states by $|0\rangle$ and $|1\rangle$. The *Dirac* or *ket* notation of a vector ψ in the state space is $|\psi\rangle$. The state of a qubit $|\psi\rangle$ is a *superposition* (linear combination) of the basis states $|0\rangle$ and $|1\rangle$:

$$|\psi\rangle = a\,|0\rangle + b\,|1\rangle\,,\text{ where } a,\ b \in \mathbb{C} \text{ and } |a|^2 + |b|^2 = 1.$$

The coefficients of $|0\rangle$ and $|1\rangle$ can be interpreted as probabilities, and a qubit as a random variable. A measurement (observation) changes the state of a qubit and yields a classical bit, i.e., either 0 or 1. The state of a qubit determines the probability of the result of a measurement: the probability of 0 is $|a|^2$ and the probability of 1 is $|b|^2$. The original state of a qubit, and hence the amplitudes a and b, are lost after the measurement. Thus the quantum information (a and b) is hidden and cannot be directly extracted. However, the quantum information can be used in quantum circuits that transform the state of qubits.

Example 13.1. The measurement of a qubit having the state $|0\rangle = 1 \cdot |0\rangle + 0 \cdot |1\rangle$ always gives 0, and the measurement of the state $|1\rangle$ always gives 1. On the other hand, if a qubit has the state

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

then the probability of both 0 and 1 is $\frac{1}{2}$, and a measurement outputs a uniform random bit. This above state is quite useful and we denote it by $|+\rangle$. ◊

A geometric representation of the state of a single qubit is given by the *Bloch sphere* (see Figure 13.1). The points on a two-dimensional unit sphere in the three-dimensional space $\mathbb{R}^3$ are given by two angles, the polar angle $\theta \in [0, \pi]$ and the azimuth $\varphi \in]-\pi, \pi]$. In geography, the polar angle and the azimuth are called *colatitude* and *longitude*, respectively.

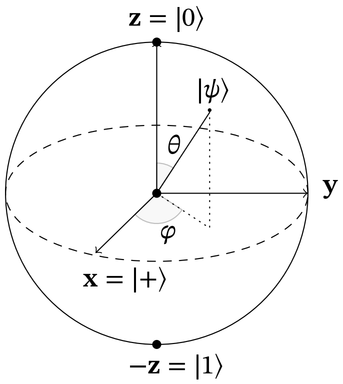


Figure 13.1. Bloch sphere.

How can we represent a state $|\psi\rangle = a|0\rangle + b|1\rangle$ by a point on the Bloch sphere? We use the fact that the state of a qubit does not depend on a global phase γ:

$$|\psi\rangle \sim e^{i\gamma}|\psi\rangle.$$

We may thus assume that the *phase* of the first coefficient a is zero, so that a is a non-negative real number. We express the second coefficient $b \in \mathbb{C}$ in polar form:

$$b = re^{i\varphi},$$

where $r \geq 0$ and $\varphi \in]-\pi, \pi]$. The condition $|a|^2 + |b|^2 = 1$ implies $a^2 + r^2 = 1$, since $|e^{i\varphi}| = 1$. The non-negative parameters a and r thus lie on a unit circle in the first quadrant and we can write

$$a = \cos\left(\frac{\theta}{2}\right) \text{ and } r = \sin\left(\frac{\theta}{2}\right),$$

where $\theta \in [0, \pi]$. We can therefore rewrite $|\psi\rangle = a|0\rangle + b|1\rangle$ as

$$|\psi\rangle = \cos\left(\frac{\theta}{2}\right)|0\rangle + e^{i\varphi}\sin\left(\frac{\theta}{2}\right)|1\rangle,$$

where $\theta \in [0, \pi]$ and $\varphi \in]-\pi, \pi]$.

Example 13.2. Here are the representations of several states:

$$\begin{aligned}
|0\rangle &= \cos(0)|0\rangle + e^{i\cdot 0}\sin(0)|1\rangle & \theta &= 0 & \varphi &= 0,\\
|1\rangle &= \cos\left(\frac{\pi}{2}\right)|0\rangle + e^{i\cdot 0}\sin\left(\frac{\pi}{2}\right)|1\rangle & \theta &= \pi & \varphi &= 0,\\
|+\rangle &= \cos\left(\frac{\pi}{4}\right)|0\rangle + e^{i\cdot 0}\sin\left(\frac{\pi}{4}\right)|1\rangle & \theta &= \frac{\pi}{2} & \varphi &= 0,\\
|-\rangle &= \cos\left(\frac{\pi}{4}\right)|0\rangle + e^{i\cdot \pi}\sin\left(\frac{\pi}{4}\right)|1\rangle & \theta &= \frac{\pi}{2} & \varphi &= \pi.
\end{aligned}$$

Note that a global phase does not change the state, e.g., $i|+\rangle \sim |+\rangle$, but the relative phase φ is important! For example, $|+\rangle$ and $|-\rangle$ are different states. ◊

Next, we study operations on a single qubit by a *quantum gate*.

Definition 13.3. A *quantum gate* U_f with a single input and output qubit is described by a *unitary* 2×2 matrix

$$U = \begin{pmatrix} c_{11} & c_{12} \\ c_{21} & c_{22} \end{pmatrix}.$$

A state $|\psi\rangle = a|0\rangle + b|1\rangle$ is transformed into

$$U_f|\psi\rangle = U_f(a|0\rangle + b|1\rangle) = (c_{11}a + c_{12}b)|0\rangle + (c_{21}a + c_{22}b)|1\rangle. \quad ◊$$

Surprisingly, there is more than one nontrivial operation on a single qubit.

Definition 13.4. (1) The quantum analogue of the classical `NOT` gate transforms the state $|\psi\rangle = a|0\rangle + b|1\rangle$ into $|\overline{\psi}\rangle = b|0\rangle + a|1\rangle$. The swapping of coefficients is given by the *Pauli-X* matrix

$$X = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}.$$

(2) The *Hadamard* gate is described by the unitary matrix

$$H = \frac{1}{\sqrt{2}}\begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix}. \quad ◊$$

Figure 13.2 depicts the Pauli-X and Hadamard gates.

Since $H \cdot \begin{pmatrix} 1 \\ 0 \end{pmatrix} = \frac{1}{\sqrt{2}}\begin{pmatrix} 1 \\ 1 \end{pmatrix}$, the state $|0\rangle$ is transformed into $|+\rangle = \frac{1}{\sqrt{2}}|0\rangle + \frac{1}{\sqrt{2}}|1\rangle$. This is very useful in order to produce a balanced superposition of the basis states. Loosely speaking, a 0-bit is turned into a qubit that is simultaneously 0 and 1. Measuring $H|0\rangle$ gives a uniform random bit (see Figure 13.3).

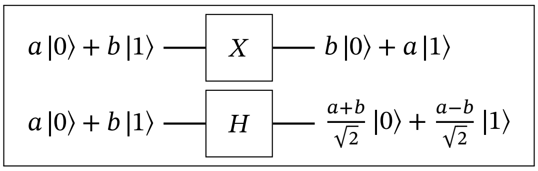

$a\,|0\rangle + b\,|1\rangle$
X
$b\,|0\rangle + a\,|1\rangle$
$a\,|0\rangle + b\,|1\rangle$
H
$\frac{a+b}{\sqrt{2}}\,|0\rangle + \frac{a-b}{\sqrt{2}}\,|1\rangle$

Figure 13.2. Pauli-X and Hadamard gates.

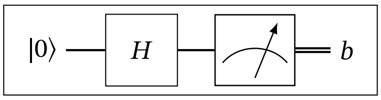


Figure 13.3. Measuring $H\,|0\rangle$ gives a uniform random bit b.

Definition 13.5. Other commonly used single qubit gates are:

(1) *Pauli-Y gate* with matrix $Y = \begin{pmatrix} 0 & -i \\ i & 0 \end{pmatrix}$.

(2) *Pauli-Z gate* with matrix $Z = \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix}$.

(3) *Phase gate* with matrix $S = \begin{pmatrix} 1 & 0 \\ 0 & i \end{pmatrix}$.

(4) $\frac{\pi}{8}$ *gate* with matrix $T = \begin{pmatrix} 1 & 0 \\ 0 & e^{i\pi/4} \end{pmatrix}$. ◊

Since $Y = -iZX$, Pauli-Y is a composition of Pauli-X and Pauli-Z. The gates Pauli-Z, Phase and $\frac{\pi}{8}$ change the relative phase by π, $\frac{\pi}{2}$ and $\frac{\pi}{4}$, respectively. On the Bloch sphere, these gates give rotations of π, $\frac{\pi}{2}$ and $\frac{\pi}{4}$ around the z-axis. For example, one has

$$Z\,|\psi\rangle = Z\left(\cos\left(\frac{\theta}{2}\right)|0\rangle + e^{i\varphi}\sin\left(\frac{\theta}{2}\right)|1\rangle\right) = \cos\left(\frac{\theta}{2}\right)|0\rangle - e^{i\varphi}\sin\left(\frac{\theta}{2}\right)|1\rangle.$$

This state can be written as

$$Z\,|\psi\rangle = \cos\left(\frac{\theta}{2}\right)|0\rangle + e^{i(\varphi+\pi)}\sin\left(\frac{\theta}{2}\right)|1\rangle.$$

Hence the Z gate adds π to the azimuth φ on the Bloch sphere.

The reason for the historical name $\frac{\pi}{8}$ *gate* is the fact that T is (up to an unimportant global phase) equal to a matrix with $\pm\frac{\pi}{8}$ on its diagonals:

$$T = e^{i\pi/8}\begin{pmatrix} e^{-i\pi/8} & 0 \\ 0 & e^{i\pi/8} \end{pmatrix}.$$

13.2. Multiple Qubit Systems

More interesting quantum operations require systems of *multiple qubits*. A composite system of n qubits can represent 2^n states simultaneously. The basis states are

$$|x_1 x_2 \dots x_n\rangle,$$

where $x_i \in \{0,1\}$. The general state of an n-qubit system is a superposition of the 2^n basis states. Such a system is not the same as n individual qubits!

Remark 13.6. The state of system of n qubits is represented by the n-fold *tensor product* of $\mathbb{C}^2$:

$$\mathbb{C}^2 \otimes \cdots \otimes \mathbb{C}^2 = (\mathbb{C}^2)^{\otimes n}.$$

The general construction of tensor products is beyond our scope, but in this case $(\mathbb{C}^2)^{\otimes n}$ is a $\mathbb{C}$-vector space of dimension 2^n. The elements in $(\mathbb{C}^2)^{\otimes n}$ are linear combinations of vectors $v_1 \otimes v_2 \otimes \cdots \otimes v_n = |v_1, v_2, \dots, v_n\rangle$, where $v_i \in \mathbb{C}^2$. The tensor product is linear in each component. The standard basis states are given by

$$x_1 \otimes x_2 \otimes \cdots \otimes x_n = |x_1 x_2 \dots x_n\rangle,$$

where $x_i = |0\rangle$ or $|1\rangle$. A general vector in $(\mathbb{C}^2)^{\otimes n}$ can be written as

$$v = \sum_{x \in \{0,1\}^n} a_x |x_1 x_2 \dots x_n\rangle.$$

The norm of v is the non-negative real number

$$||v|| = \sum_{x \in \{0,1\}^n} |a_x|^2.$$

The normalization condition for multiple qubit systems requires that $||v|| = 1$. Unitary operators $U_{f_1}, \dots, U_{f_n}$ on $\mathbb{C}^2$ induce the unitary operator

$$U_f = U_{f_1} \otimes \cdots \otimes U_{f_n}$$

on $(\mathbb{C}^2)^{\otimes n}$, but there are additional operators that are not of this type.

As with single qubits, a multiple qubit state does not depend on a global phase and

$$|\psi\rangle \sim e^{i\gamma} |\psi\rangle.$$

A two qubit system is represented by a state in $\mathbb{C}^2 \otimes \mathbb{C}^2$. The four basis states are $|00\rangle$, $|01\rangle$, $|10\rangle$, $|11\rangle$, and a general state is a superposition of the four basis states:

$$|\psi\rangle = a_{00}|00\rangle + a_{01}|01\rangle + a_{10}|10\rangle + a_{11}|11\rangle.$$

The normalization condition requires $|a_{00}|^2 + |a_{01}|^2 + |a_{10}|^2 + |a_{11}|^2 = 1$. Measuring a two qubit system gives 00 with probability $|a_{00}|^2$, 01 with probability $|a_{01}|^2$, 10 with probability $|a_{10}|^2$ and 11 with probability $|a_{11}|^2$.

Example 13.7. (1) Consider a two qubit system in the state

$$\begin{aligned}|\psi\rangle &= \left(\frac{1}{\sqrt{2}}|0\rangle + \frac{1}{\sqrt{2}}|1\rangle\right) \otimes \left(\frac{1}{\sqrt{2}}|0\rangle + \frac{1}{\sqrt{2}}|1\rangle\right) \\ &= \frac{1}{2}|00\rangle + \frac{1}{2}|01\rangle + \frac{1}{2}|10\rangle + \frac{1}{2}|11\rangle .\end{aligned}$$

All four combinations have the same probability $(\frac{1}{2})^2 = \frac{1}{4}$.

(2) One of the *Bell states* is

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

After a measurement, the state is either 00 or 11. The combinations 01 and 10 cannot occur; a measurement of the two qubits must give the same result. The qubits are *entangled*. In quantum mechanics, this is called the EPR (Einstein, Podolsky, Rosen) paradox, where two particles are mysteriously correlated. The other three Bell states are given in Exercise 2. ♢

One can show that the Bell states are not the product of any two single qubit states (see Exercise 1). In fact, two entangled qubits behave differently from two single qubits.

The basis states of a system of n qubits are $|x_1 x_2 \ldots x_n\rangle$ where $x_i \in \{0,1\}$. A general state is a superposition

$$|\psi\rangle = \sum_{x\in\{0,1\}^n} a_x |x\rangle \text{ with } \sum_{x\in\{0,1\}^n} |a_x|^2 = 1.$$

A basis state $|x_1 x_2 \ldots x_n\rangle$ can also be written as $|x\rangle$, where

$$x = x_1 2^{n-1} + x_2 2^{n-2} + \cdots + x_n \in \{0,\ 1,\ \ldots,\ 2^n - 1\}.$$

Then, a general state of an n-qubit system is

$$|\psi\rangle = \sum_{x=0}^{2^n-1} a_x |x\rangle \text{ with } \sum_{x=0}^{2^n-1} |a_x|^2 = 1.$$

An obvious generalization of the Bloch sphere for multiple qubits is not known. Note that the full state of a system of n qubits involves 2^n complex amplitudes (and $2^n - 1$ degrees of freedom, since the amplitude vector is normalized and multiplication by a global phase is unimportant). This is a huge number, say for $n > 100$. We emphasize that a measurement outputs only *one binary word* x of length n and the probability of x being measured is $|a_x|^2$.

13.3. Quantum Algorithms

Classical Boolean circuits transform input bits into output bits. Not surprisingly, *quantum circuits* process qubits. The basic building blocks of quantum circuits are quantum logic gates, which implement unitary (and therefore reversible) transformations on qubits. This is a major difference to classical circuits, which are often not reversible, for example the elementary AND or OR gates. However, this does not pose a serious restriction, since there are invertible analogues of the classical gates. It turns out that every classical circuit has a quantum analogue, giving the same output on the basis states, but also processing superpositions of the basis states.

Definition 13.8. The *controlled-NOT* operation *CNOT* on two input qubits with basis states $|x\rangle$ and $|y\rangle$ is given by

$$CNOT\,|x, y\rangle = |x, x \oplus y\rangle$$

(see Figure 13.4). This transformation acts on the basis states as follows: the first bit (the *control bit*) is unchanged and the second bit (the *target bit*) is flipped if the control bit is 1. The states $|00\rangle$ and $|01\rangle$ are thus unchanged, $|10\rangle$ is mapped to $|11\rangle$ and $|11\rangle$ to $|10\rangle$. The CNOT gate is represented by the unitary matrix

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

since the first two basis states remain unchanged, while the third and the fourth basis state are swapped. ◊

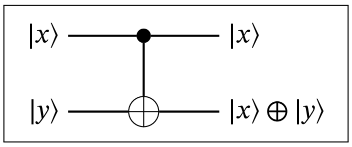


Figure 13.4. Controlled NOT gate.

CNOT can produce entangled states and this two-bit gate is not the tensor product of two single-qubit gates. For example, two qubits are transformed into the Bell state:

$$CNOT\left(\frac{1}{\sqrt{2}}(|00\rangle + |10\rangle)\right) = \frac{1}{\sqrt{2}}\,(|00\rangle + |11\rangle)\,.$$

CNOT is an example of a two qubit *controlled* gate (see Figure 13.5): if the first (control) qubit is $|1\rangle$, then a single qubit operation Q is performed on the second (target) qubit. If the control qubit is $|0\rangle$, then the target qubit is unchanged (see Figure 13.5). The controlled Q gate is represented by a 4×4 matrix of four 2×2 blocks, where I_2 is the 2×2 identity matrix:

$$\begin{pmatrix} I_2 & 0 \\ 0 & Q \end{pmatrix}.$$

CNOT is a controlled Q gate with $Q = X$ (Pauli-X). The CNOT gate is especially important, since single qubit and CNOT gates are *universal*.

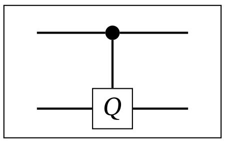


Figure 13.5. Controlled Q gate.

Theorem 13.9. *Single qubit gates and the CNOT gate are sufficient to implement an arbitrary unitary operation on n qubits.*

Proof. We refer to [**NC00**] for a proof. Firstly, one shows that an arbitrary unitary matrix can be decomposed into a product of *two-level* unitary matrices, where at most two coordinates are changed. Secondly, circuits from two-level unitary matrices are built from single qubit and CNOT gates. ☐

Remark 13.10. At the time of writing, IBM offers quantum computers and simulators for public use (`https://quantum-computing.ibm.com`). You can create and run your own algorithms on quantum devices using a graphical composer or a Quantum Assembly Language Code (QASM) editor. Circuits are built from a set of elementary gates (X, Y, Z, H, S, T, CNOT, ...), and combinations of them can express more complicated unitary transformations. Also look at the open source framework Qiskit (`https://qiskit.org`) for working with noisy quantum computers.

Many quantum algorithms take as input a superposition of the basis states. The single qubit Hadamard gate H (see Section 13.1) transforms the basis state $|0\rangle$ into the superposition

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

The *Walsh-Hadamard transformation* generalizes this to a system of n qubits. It can be implemented by n parallel Hadamard gates.

Definition 13.11. The Walsh-Hadamard transformation W acts on a system of n qubits and is defined by

$$W = H \otimes H \otimes \cdots \otimes H = H^{\otimes n}. \qquad \Diamond$$

The Walsh-Hadamard transformation W transforms the zero state into a balanced superposition of all 2^n basis states (see Figure 13.6):

$$\begin{aligned} H^{\otimes n}\,|00\ldots0\rangle &= \frac{1}{\sqrt{2}}\left(|0\rangle + |1\rangle\right) \otimes \frac{1}{\sqrt{2}}\left(|0\rangle + |1\rangle\right) \otimes \cdots \otimes \frac{1}{\sqrt{2}}\left(|0\rangle + |1\rangle\right) \\ &= \frac{1}{\sqrt{2^n}}\left(|00\ldots0\rangle + |00\ldots1\rangle + \cdots + |11\ldots1\rangle\right). \end{aligned}$$

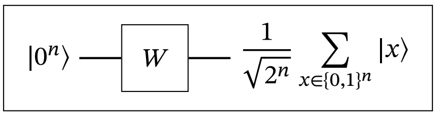


Figure 13.6. Walsh-Hadarmard transformation.

The second equation follows from the fact that the tensor product is linear in each component.

Quantum algorithms can use this superposition to simultaneously compute *all values* of a vectorial Boolean function $f : \{0,1\}^n \to \{0,1\}^m$. Since f may not be invertible (which is always the case if $n \neq m$), but quantum circuits must be invertible, f has to be tweaked. Given f, we define $F : \{0,1\}^{n+m} \to \{0,1\}^{n+m}$ by

$$F(x,y) = (x, y \oplus f(x)).$$

Obviously, F is invertible and $F^{-1} = F$. The corresponding unitary transformation on a system of $n + m$ qubits is given by

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

(see Figure 13.7). Such a transformation can be efficiently implemented by combining elementary gates.

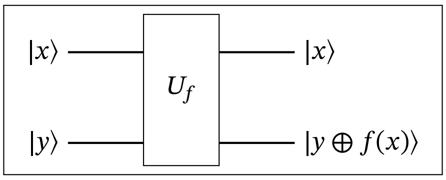


Figure 13.7. Unitary transformation associated to a Boolean function f.

Example 13.12. Let $f(x_1, x_2) = x_1x_2$ be the classical AND operation on two input bits (see Table 1.1). The corresponding invertible function is $F : \{0,1\}^3 \to \{0,1\}^3$, where $F(x_1, x_2, y) = (x_1, x_2, y \oplus x_1x_2)$. The quantum transformation is called a *Toffoli gate* and is given by

$$U_f(|x_1, x_2, y\rangle) = |x_1, x_2, y \oplus x_1x_2\rangle$$

on three input qubits with the basis states $|x_1\rangle$, $|x_2\rangle$ and $|y\rangle$. ◊

One can leverage F to compute f by setting the second input component y to the zero string 0^m. Then $F(x, 0^m) = (x, f(x))$ and thus $U_f\,|x, 0^m\rangle = |x, f(x)\rangle$. Furthermore, U_f maps a superposition $\sum_x a_x\,|x, 0^m\rangle$ to

$$\sum_{x \in \{0,1\}^n} a_x\,|x, f(x)\rangle\,.$$

We combine the Walsh-Hadarmard transformation W (on the first n qubits) and U_f. This transforms the zero state into a superposition of all values of f (see Figure 13.8):

$$U_f(W\,|0^n\rangle\,, 0^m) = U_f\left(\frac{1}{\sqrt{2^n}}\sum_{x\in\{0,1\}^n}|x,0^m\rangle\right) = \frac{1}{\sqrt{2^n}}\sum_{x\in\{0,1\}^n}|x,f(x)\rangle\,.$$

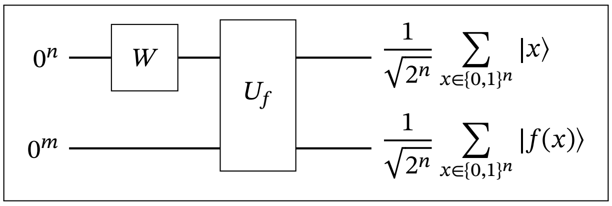

0^n
W
U_f
0^m
$\frac{1}{\sqrt{2^n}}\sum_{x\in\{0,1\}^n}|x\rangle$
$\frac{1}{\sqrt{2^n}}\sum_{x\in\{0,1\}^n}|f(x)\rangle$

Figure 13.8. Input of the circuit is the zero state and output is the superposition of all values of f.

The simultaneous computation of *all* values of a function looks miraculous. However, the state is not directly accessible and a measurement only extracts a single value. Furthermore, the output is probabilistic, and values with a very small coefficient, i.e., a small probability, almost never occur. One therefore has to construct a clever quantum algorithm, which makes use of the superposition and outputs the requested value with a significant probability. One of the most important algorithms is the *Quantum Fourier Transform*, which we study in the following section.

Next, we explain the *Deutsch-Josza Algorithm* (see Figure 13.9). It illustrates the capabilities of quantum algorithms, although the underlying problem is of limited interest.

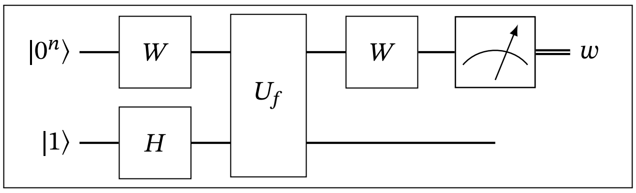


Figure 13.9. Deutsch-Josza algorithm. The measurement produces a classical binary string w of length n. If $w = 0^n$ is measured then f must be constant, otherwise, f is balanced.

Suppose a Boolean function $f : \{0,1\}^n \to \{0,1\}$ is either *constant or balanced*, i.e., the number of input strings having the output 0 and 1 is equal. A classical non-probabilistic algorithm requires (in the worst case) $2^{n-1}+1$ function calls in order to determine whether or not f is balanced. This is only feasible for small n. In contrast, the Deutsch-Josza quantum algorithm is polynomial in n. The first step of the Deutsch-Josza algorithm is to apply the Walsh-Hadamard transformation $H^{\otimes(n+1)}$ to the input state $|0^n, 1\rangle$. Since $H|1\rangle = \frac{1}{\sqrt{2}}(|0\rangle - |1\rangle)$, we obtain the state

$$\begin{aligned} |\psi\rangle &= (H^{\otimes n} \otimes H)(|0\rangle \otimes \cdots \otimes |0\rangle \otimes |1\rangle) \\ &= H^{\otimes n}|0^n\rangle \otimes H|1\rangle \\ &= \frac{1}{\sqrt{2^{n+1}}} \sum_{x \in \{0,1\}^n} |x\rangle \otimes (|0\rangle - |1\rangle). \end{aligned}$$

Next, $U_f(|x,y\rangle) = |x, y \oplus f(x)\rangle$ is applied to the state ψ:

$$U_f|\psi\rangle = \frac{1}{\sqrt{2^{n+1}}} \sum_{x \in \{0,1\}^n} |x\rangle \otimes ((|0\rangle - |1\rangle) \oplus f(x)).$$

Suppose $|x\rangle$ is a basis state $|x_1 \ldots x_n\rangle$. If $f(x) = 0$ then $|0\rangle - |1\rangle$ remains unchanged. If $f(x) = 1$ then $|0\rangle - |1\rangle$ is mapped to $|1\rangle - |0\rangle = -(|0\rangle - |1\rangle)$. In both cases, we have

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

and thus obtain

$$U_f|\psi\rangle = \frac{1}{\sqrt{2^{n+1}}} \sum_{x \in \{0,1\}^n} (-1)^{f(x)}|x\rangle \otimes (|0\rangle - |1\rangle).$$

In the next step, $H^{\otimes n} \otimes Id$ is applied to $U_f|\psi\rangle$, so that $|x\rangle$ is mapped to $H^{\otimes n}|x\rangle$. One can check the expansion

$$H^{\otimes n}|x\rangle = \frac{1}{\sqrt{2^n}} \sum_{z \in \{0,1\}^n} (-1)^{x \cdot z}|z\rangle$$

(see Exercise 8), where $x \cdot z$ is the dot product modulo 2. This yields

$$\left(H^{\otimes n} \otimes Id\right)(U_f|\psi\rangle) = \sum_{z \in \{0,1\}^n} \sum_{x \in \{0,1\}^n} \frac{1}{2^n}(-1)^{f(x)+x\cdot z}|z\rangle \otimes \frac{1}{\sqrt{2}}(|0\rangle - |1\rangle).$$

Finally, we measure the first n qubits. The coefficient of the basis state $|0^n\rangle$ is

$$\sum_{x \in \{0,1\}^n} \frac{1}{2^n}(-1)^{f(x)},$$

since $z = 0^n$ obviously yields $x \cdot z = 0$ for all $x \in \{0,1\}^n$. If f is constant, then the above coefficient is either 1 or -1. Hence the probability of measuring 0^n is equal to 1 and any measurement must give 0^n. On the other hand, if f is balanced then positive and negative terms cancel and the probability of measuring 0^n is 0. The Deutsch-Josza algorithm outputs *f is constant*, if a measurement gives 0^n, and otherwise *f is balanced*. This result is always correct and the quantum algorithm runs in polynomial time.

13.4. Quantum Fourier Transform

The Deutsch-Josza algorithm demonstrates that measuring a quantum state can provide useful information. The *Quantum Fourier Transform* is a key algorithm, which efficiently finds a period of a large input vector. In Section 13.5 below we will see how this can be leveraged to solve the factoring problem.

The classical *Discrete Fourier Transform* (DFT) is a bijective $\mathbb{C}$-linear map on $\mathbb{C}^N$. A sequence of N complex numbers (in the discrete time domain) is mapped to the frequency domain. The resulting vector of complex Fourier coefficients reveals the periodic structure of the input data.

Suppose the input vector is the complex vector $(a_0, a_1, \dots, a_{N-1})$. Then the DFT is $(y_0, y_1, \dots, y_{N-1})$ and the Fourier coefficients are defined by

$$y_k = \frac{1}{\sqrt{N}} \sum_{x=0}^{N-1} a_x e^{-2\pi i x k/N}, \text{ where } k = 0, 1, \dots, N-1.$$

We note that the above unitary DFT differs from other variants by a normalization factor. The computation can be accelerated using the Fast Fourier Transform (FFT). The DFT is given by the following unitary $N \times N$ matrix:

$$U = \frac{1}{\sqrt{N}} \begin{pmatrix} 1 & 1 & 1 & \dots & 1 \\ 1 & \omega & \omega^2 & \dots & \omega^{N-1} \\ 1 & \omega^2 & \omega^4 & \dots & \omega^{2(N-1)} \\ & & & \dots & \\ 1 & \omega^{N-1} & \omega^{2(N-1)} & \dots & \omega^{(N-1)(N-1)} \end{pmatrix},$$

where $\omega = e^{-2\pi i/N}$ is a primitive N-th root of unity.

The original data can be recovered from the Fourier coefficients using the *inverse DFT*:

$$a_x = \frac{1}{\sqrt{N}} \sum_{k=0}^{N-1} y_k e^{2\pi i x k/N}, \text{ where } x = 0, 1, \dots, N-1.$$

The corresponding matrix of the inverse DFT is $U^{-1} = \overline{U}^T$, the conjugate transpose matrix.

The DFT computes the discrete spectrum of the input data. If the data of length N is r-periodic and N is divisible by r, then the Fourier coefficients y_k are nonzero only for multiples of $\frac{N}{r}$. In general, a Fourier amplitude $|y_k| \gg 0$ indicates that $\frac{N}{k}$ is an *approximate multiple* of the period.

Example 13.13. Let $a = (1, 2, 1, 2)^T$ be a data vector of length $N = 4$. The data is $r = 2$-periodic. The unitary 4×4 matrix that describes the DFT is

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

We compute the Fourier coefficients $y = Ua = (3, 0, -1, 0)^T$. Only the coefficients y_0 and y_2 are nonzero and hence a is $\frac{4}{2} = 2$-periodic. The input vector a can be recovered from the Fourier coefficients by computing $a = U^{-1}y = \overline{U}^T y$. ♢

In a quantum setting, we want to compute the DFT of a large input vector $(a_0, a_1, \dots, a_{N-1})$ of length $N = 2^s$ and find a *hidden period* of the data. The basic idea is that the indices k with Fourier coefficients $|y_k|^2 \gg 0$ reveal the period. The measurement of a state vector of Fourier amplitudes will give such indices k with a significant probability.

The Quantum Fourier Transform (QFT) does a DFT on the amplitudes of the quantum state and outputs a superposition of Fourier coefficients:

$$|\psi\rangle = C \sum_{x=0}^{N-1} a_x |x\rangle \mapsto U|\psi\rangle = C \sum_{k=0}^{N-1} y_k |k\rangle .$$

The scaling factor C ensures that the coefficient vector of $|\psi\rangle$ is normalized. The Fourier coefficients are

$$y_k = \frac{1}{\sqrt{N}} \sum_{x=0}^{N-1} a_x e^{2\pi i x k/N}.$$

Note that the above QFT uses the primitive N-th root of unity $\omega = e^{2\pi i/N}$ and not the conjugate value $e^{-2\pi i/N}$. One can show that the QFT has an efficient circuit and runs in time $O(\text{size}(N)^2)$. In the simplest case ($N = 2$), the QFT is given by a single Hadamard gate.

Now, suppose that the input vector $(a_0, a_1, \dots, a_{N-1})$ is r-periodic, where r is an unknown period between 1 and N. One prepares the state $|\psi\rangle = C\sum_{x=0}^{N-1} a_x |x\rangle$ of an n-qubit system, applies the QFT and measures the state $U|\psi\rangle = C\sum_{k=0}^{N-1} y_k |k\rangle$. With a high probability, the measured output k is an approximate multiple of $\frac{N}{r}$.

13.5. Shor's Factoring Algorithm

The assumption that factoring is a hard problem plays an important role in public-key cryptography, and especially for RSA encryption and signature schemes. The most efficient classical factoring algorithm is the *Number Field Sieve*, which has subexponential complexity (see Section 9.7). In 1994, Peter Shor found a quantum algorithm that runs in polynomial time ([**Sho94**]). This algorithm is a major application of quantum computing. It has been successfully implemented for toy examples (like factoring $21 = 7 \cdot 3$) and will certainly be applied to real-world problems as soon as quantum computers with thousands of qubits become available.

Shor's algorithm finds a *hidden period* of a function and is based on the Quantum Fourier Transform.

Firstly, we explain how to derive the unknown factors of a composite number n from the multiplicative order of an element $a \in \mathbb{Z}_n^*$. Note that the multiplicative order of a is also the least period of the function $f(x) = a^x \mod n$.

Suppose that $n = pq$, where n is known and p, q are unknown. Choose a uniform random integer a with $1 < a < n$. If $\gcd(a, n) \neq 1$, then $\gcd(a, n)$ is either p or q and the unknown factors are found. However, the probability of $\gcd(a, n) \neq 1$ is very small if the prime factors are large and a is randomly chosen.

Now assume that $\gcd(a, n) = 1$; then $a \mod n \in \mathbb{Z}_n^*$ and

$$r = \operatorname{ord}(a) \mid \operatorname{ord}(\mathbb{Z}_n^*) = (p-1)(q-1)$$

(see Corollary 4.14). By definition, $a^r \equiv 1 \mod n$. If r is even, then

$$a^r - 1 = (a^{r/2} - 1)(a^{r/2} + 1) \equiv 0 \mod n,$$

and thus $n \mid (a^{r/2} - 1)(a^{r/2} + 1)$. Since $\operatorname{ord}(a) \neq \frac{r}{2}$, we have $n \nmid (a^{r/2} - 1)$ and there are two possibilities:

(1) p divides one of the two factors and q divides the other. In this case $\gcd(a^{r/2}+1, n)$ gives p or q.

(2) $n \mid (a^{r/2} + 1)$, then the algorithm fails and one has to choose another base a.

The algorithm is successful if r is even and $n \nmid (a^{r/2} + 1)$. Closer analysis shows that the probability of success is at least 50% (compare [**NC00**]): let r_p and r_q be the order of a in $\mathbb{Z}_p^*$ and $\mathbb{Z}_q^*$, respectively. Then r is odd if and only if r_p and r_q are odd (see Exercise 10). Now suppose that r is even and 2^d is the maximal power of 2 that divides r. We have $a^{r/2} + 1 \equiv 0 \mod n$ if and only if $a^{r/2} \equiv -1 \mod p$ and $a^{r/2} \equiv -1 \mod q$. This requires $r_p \nmid \frac{r}{2}$ and $r_q \nmid \frac{r}{2}$. Since $r_p \mid r$ and $r_q \mid r$, we obtain $2^d \mid r_p$ and $2^d \mid r_q$.

Summarizing, the algorithm fails if either $2 \nmid r_p$ and $2 \nmid r_q$ or $2^d \mid r_p$ and $2^d \mid r_q$. If a is chosen uniformly at random, then the probability for this to happen is at most 50%.

Example 13.14. Let $n = 77$ and $a = 3$; then $\gcd(a, n) = 1$. Suppose we know the order of $a \mod n$ in the multiplicative group $\mathbb{Z}_n^*$: the order is $r = \operatorname{ord}(3 \mod 77) = 30$ and this is an even number. We obtain

$$a^{r/2} + 1 = 3^{15} + 1 \equiv 35 \mod 77.$$

We compute $\gcd(a^{r/2} + 1, n) = \gcd(35, 77) = 7$ and obtain one of prime factors of $n = 77$. Note that $\gcd(a^{r/2} - 1, n) = \gcd(33, 77) = 11$ gives the other factor of n.

In fact, we succeeded in this example since $r_p = \text{ord}(3 \mod 7) = 6$ is even and $r_q = \text{ord}(3 \mod 11) = 5$ is odd. ◊

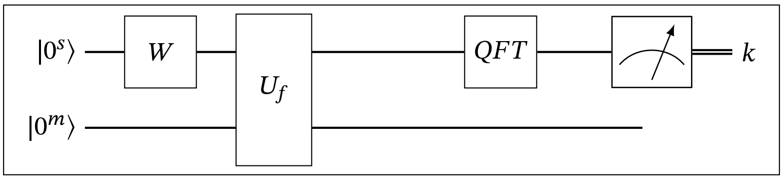


Figure 13.10. Shor's algorithm uses the Quantum Fourier Transform and finds a hidden period of a function f.

Now, the quantum part of Shor's algorithm is to compute the unknown order r of a given residue class $a \in \mathbb{Z}_n^*$. For that purpose, one prepares a superposition of input values $x = 0, 1, \dots, N-1$ and simultaneously computes all $a^x \mod n$. The values are r-periodic, i.e., $a^x \equiv a^{x+r}$. The QFT is applied to the state and we will see that a measurement reveals the period with high probability. The sequence must be significantly longer than the period and it turns out that $N = 2^s$ with $n^2 \leq N \leq 2n^2$ is a reasonable choice.

We view $f(x) = a^x \mod n$ as a Boolean function. The corresponding unitary transformation on quantum bits is $U_f\,|x, y\rangle = |x, y \oplus f(x)\rangle$. The first register has s qubits and the second has $m = \text{size}(n)$ qubits.

The Walsh-Hadamard transformation maps $|x\rangle = |0^s\rangle$ to a superposition of all basis states. We set $|y\rangle = |0^m\rangle$, apply U_f and obtain the state

$$|\psi\rangle = U_f(W\,|0^s\rangle, |0^m\rangle) = \frac{1}{\sqrt{N}} \sum_{x=0}^{N-1} |x, f(x)\rangle .$$

Next, the QFT operator U is applied to the first register while the second remains unchanged:

$$(U \otimes Id)\,|\psi\rangle = \frac{1}{N} \sum_{x=0}^{N-1} \sum_{k=0}^{N-1} e^{2\pi i x k/N}\,|k, f(x)\rangle .$$

Finally, the first register is measured (see Figure 13.10). The second register takes a random value $u \in im(f)$ and the probability of measuring $|k, u\rangle$ is

$$\left| \frac{1}{N} \sum_{x:\ f(x)=u} e^{2\pi i x k/N} \right|^2 .$$

There is a high probability that k is an approximate multiple of $\frac{N}{r}$ and the inequalities

$$\left| k - j\frac{N}{r} \right| < \frac{1}{2} \Rightarrow \left| \frac{k}{N} - \frac{j}{r} \right| < \frac{1}{2N} \Rightarrow \left| \frac{k}{N} - \frac{j}{r} \right| < \frac{1}{2n^2}$$

hold for some j. One can show that at most one fraction $\frac{j}{r}$ with $0 < j < n$ and $0 < r < n$ can satisfy this inequality. The fraction and the requested period r can be efficiently determined using the *continued fraction expansion* (see Example 13.16 below).

The reader might be surprised that the QFT of the *first register* gives anything interesting, since the amplitudes of the first register of $|\psi\rangle$ are constant. However, the amplitudes are partitioned by the second register, i.e., by different values of $u = f(x) = a^x \mod n$. We can rewrite $|\psi\rangle$ as

$$|\psi\rangle = \frac{1}{\sqrt{N}} \sum_{u \in im(f)} \sum_{f(x)=u} |x, u\rangle .$$

The amplitudes with different u in the second register do not interfere with each other when the QFT is applied to the first register. Now, for a fixed second register u, the first register is r-periodic and applying the QFT gives peaks at multiples of $\frac{N}{r}$. If N is divisible by r, the Fourier amplitudes are zero outside multiples of $\frac{N}{r}$.

Remark 13.15. Shor's algorithm requires around $3\,\text{size}(n)$ qubits and uses $O(\text{size}(n)^3)$ operations, and optimizations are known.

Example 13.16. We continue Example 13.14 and apply Shor's algorithm to $n = 77$ and $a = 3 \in \mathbb{Z}_n^*$. We want to compute $\text{ord}(a)$. First, we have to define $N = 2^s$. Since $n^2 = 5929$ and $2n^2 = 11858$, we choose $s = 13$ and $N = 8192$. We create a superposition of the basis states $|x, 3^x \mod 77\rangle$ for $x = 0, 1, \dots, 8191$ and obtain

$$|\psi\rangle = \frac{1}{\sqrt{8192}} \sum_{x=0}^{8191} |x,\, 3^x \mod 77\rangle .$$

Note that we need 13 qubits for the first register and 7 qubits for the second. We can reorder the terms with respect to the second register $u = 3^x \mod 77$. Suppose for example that $u = 59$; then the corresponding terms in $|\psi\rangle$ are

$$\frac{1}{\sqrt{8192}} \left(|19, 59\rangle + |49, 59\rangle + |79, 59\rangle + \cdots + |8179, 59\rangle\right).$$

The expansions for other values u in the second register look similar and the period $r = 30$ is clearly visible, but the state is not directly accessible to an observer. Instead, we apply the QFT to the first register and measure it. The second register takes a random value $u \in im(f)$, for example $u = 59$. The amplitudes $|y_k|$ of $|k, u\rangle$ for $k = 0, 1, \dots, 8191$ and $u = 59$ are shown in Figure 13.11. The squares $|y_k|^2$ give the probability that $|k, 59\rangle$ is measured. The probabilities for any other value $u \in im(f)$ are identical. Closer inspection shows peaks at all multiples of $\frac{N}{r} \approx 273$. Table 13.1 lists some amplitudes around $k = 273$.

Again, we remark that these amplitudes are not accessible to an observer, but the measured value k is likely to be a multiple of $\frac{N}{r}$. Suppose that $k = 7100$. We expect

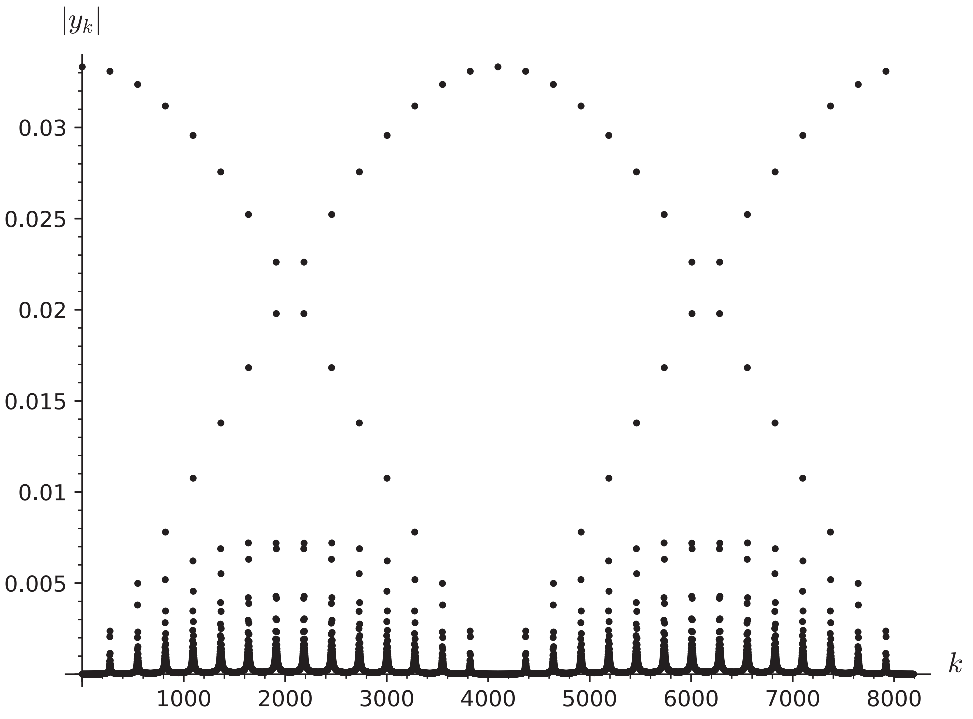


Figure 13.11. Fourier amplitudes of $|k, 59\rangle$. The spectrum has peaks at multiples of $\frac{8192}{30} \approx 273$.

Table 13.1. Absolute values of selected amplitudes.

k	270	271	272	273	274	275
$\lvert y_k \rvert$	0.00071	0.00106	0.002060	0.033082	0.002372	0.001149

that $\frac{k}{N} = \frac{7100}{8192}$ is close to a fraction $\frac{j}{r}$ where j and r are integers less than $n = 77$. In this toy example, we could simply try out the possible values for j and r, but in general, the efficient method of *continued fractions expansions* is used.

The idea is to approximate a real number x by continued fractions of integer numbers. The number is split into its integer part $\lfloor x \rfloor$ and its fractional part ϵ_0:

$$x = \lfloor x \rfloor + \epsilon_0 = \lfloor x \rfloor + \frac{1}{\left(\frac{1}{\epsilon_0}\right)}.$$

Next, $\frac{1}{\epsilon_0}$ is split into an integer and a fractional part and we obtain

$$x = \lfloor x \rfloor + \epsilon_0 = \lfloor x \rfloor + \frac{1}{\lfloor \frac{1}{\epsilon_0} \rfloor + \epsilon_1}.$$

We continue in the same fashion, write $\epsilon_1 = \frac{1}{\left(\frac{1}{\epsilon_1}\right)}$ and split $\frac{1}{\epsilon_1}$ into an integer and a fractional part. The method terminates after a finite number of steps if x is a rational number, and otherwise approximates x.

For our example, we let SageMath compute the sequence of integer parts:

```
sage: (7100/8192).continued_fraction()
[0; 1, 6, 1, 1, 136]
```

Hence the complete continued fractions expansion is

$$\frac{7100}{8192} = 0 + \cfrac{1}{1 + \cfrac{1}{6 + \cfrac{1}{1 + \cfrac{1}{1 + \frac{1}{136}}}}}.$$

However, we need an approximation with a denominator less than $n = 77$ and so we discard the last fraction $\frac{1}{136}$. Then

$$\frac{7100}{8192} \approx \cfrac{1}{1 + \cfrac{1}{6 + \cfrac{1}{1 + \frac{1}{1}}}} = \frac{13}{15}.$$

We obtain $\frac{j}{r} = \frac{13}{15}$ and therefore the period must be a multiple of 15 less than $n = 77$.

One checks that $3^{15} \equiv 34 \not\equiv 1 \mod 77$ and $3^{30} \equiv 1 \mod 77$. We conclude that $r = \operatorname{ord}(3) = 30$. Finally, we verify that $\frac{N}{r} = \frac{8192}{30} \approx 273.07$, which explains the peaks of the spectrum at multiples of this number. $\diamondsuit$

We have seen above that Shor's algorithm can efficiently solve the factoring problem. The other major cryptographic problem, the *discrete logarithm problem*, can also be solved with a period-finding algorithm. This can be applied to the multiplicative group of integers modulo a prime number p and also to the group of points on an elliptic curve over a finite field.

Suppose that $A = g^a$ holds in a cyclic group $G = \langle g \rangle$ of order n and a is unknown. Consider the map

$$f : \mathbb{Z} \times \mathbb{Z} \to G, \ f(x, y) = A^x g^{-y}.$$

This map is $(1, a)$-periodic since

$$f(x + 1, y + a) = A^{x+1} g^{-y-a} = g^{ax+a} g^{-y-a} = A^x g^{-y} = f(x, y).$$

Any period (r, s) is a multiple of $(1, a) \mod n$, since

$$f(x+r,\, y+s) = f(x,y) \Longrightarrow A^{x+r} g^{-y-s} = A^x g^{-y}.$$

This implies $a(x+r) - y - s \equiv ax - y \mod n$ and hence $ar \equiv s \mod n$. If we find an approximate period (r, s) using an efficient quantum algorithm, then we can also compute the discrete logarithm a.

This means that *classical public-key cryptography is broken* once sufficiently large and stable quantum computers become available!

On the other hand, it is expected that symmetric algorithms are less severely affected by the capabilities of quantum computers. *Grover's algorithm* [**Gro96**] provides a speedup from n to $\sqrt{n}$ evaluations, which effectively halves the security level. As a consequence, AES with 128-bit keys provides only 64 bits of post-quantum security, but attacking 256-bit AES with 128-bit post-quantum security is still unfeasible.

13.6. Quantum Key Distribution

A remarkable cryptographic application of quantum mechanics is *key distribution* over an insecure channel. There are popular, well-known key establishment protocols, for example the Diffie-Hellman key exchange (see Section 10.3), but once large-scale quantum computing becomes available, the conventional public-key protocols are broken.

Quantum Key Distribution (QKD) is using *qubits* to transfer a secret key. Unlike the quantum algorithms discussed above, QKD does not require an n-qubit system, but uses a sequence of single qubits or pairs of entangled qubits. This is much easier to realize and QKD has already been successfully implemented in practice, where polarized photons were sent over large distances.

In this section, we present the BB84 (Bennett and Brassard) quantum key distribution protocol [**BB84**]. Other protocols (in particular Ekert's E91 protocol [**Eke91**]) are also used in practical implementations.

The idea of QKD is that quantum bits cannot be measured without changing the state. Furthermore, an unknown quantum state cannot be cloned, since the internal state is unknown to an observer who does not know how the system was initialized.

Suppose that Alice and Bob want to exchange a key of length n and they have access to a public communication channel and a quantum channel of single polarized photons. They use two different bases B_0 and B_1 (polarizations) of the single-qubit space $\mathbb{C}^2$. Assume that the standard basis is

$$B_0 = \{|0\rangle, |1\rangle\}.$$

Applying the Hadamard transformation yields the Hadamard basis B_1 which is diagonal to B_0:

$$B_1 = HB_0 = \left\{ \frac{1}{\sqrt{2}} |0\rangle + \frac{1}{\sqrt{2}} |1\rangle, \frac{1}{\sqrt{2}} |0\rangle - \frac{1}{\sqrt{2}} |1\rangle \right\} = \{|+\rangle, |-\rangle\}.$$

Let δ be a positive integer constant which determines the probability of success of the protocol. Alice chooses a uniform random key string k of length $4n + \delta$ bits. Furthermore, Alice and Bob choose $4n + \delta$ bits that will determine whether B_0 or B_1 is used:

$$k \overset{\$}{\leftarrow} \{0,1\}^{4n+\delta},\ b_A \overset{\$}{\leftarrow} \{0,1\}^{4n+\delta},\ b_B \overset{\$}{\leftarrow} \{0,1\}^{4n+\delta}.$$

Now Alice sends the key k as a sequence of single qubits to Bob. She uses the basis B_0 if the corresponding bit of b_A is 0, or otherwise B_1.

Bob receives the sequence of qubits and measures them with respect to B_0 or B_1, depending on the corresponding bit of b_B. If the i-th bit of b_A and b_B are equal, then Bob measures the correct i-th bit of k. Otherwise, the probability of a correct key bit is only 50%. For example, suppose that Alice is using B_0 and sending the bit 1; then the transmitted qubit is $|1\rangle$. If Bob chooses the basis B_1, then

$$|1\rangle = \frac{1}{\sqrt{2}}\,|+\rangle - \frac{1}{\sqrt{2}}\,|-\rangle\,.$$

Hence the probability of measuring $|-\rangle$, which corresponds to the bit 1, is only 50%.

After Bob has received and measured the key bits, both partners exchange their bases b_A and b_B via the conventional public channel. They discard the key bits that were measured in a different basis and restart the key exchange if less than $2n$ bits remain. Obviously, the constant δ determines the probability of a successful exchange. They keep the first $2n$ key bits. The following step aims to reveal the interference of an adversary. Alice chooses a subset of n key bits and sends Bob the selected positions. They exchange the associated key bits via the public channel. They compare the bits and abort the protocol if the number of errors is higher than expected. The remaining n bits are used as a secret key, which needs to be further transformed (*information reconciliation* and *privacy amplification*), in order to reduce the effects of errors and undetected interference by adversaries.

We are now discussing the security of the BB84 protocol. Firstly, the non-quantum communication channel between Alice and Bob can be public, but integrity is important. Furthermore, Alice has to generate and transmit *single* qubits, for example single polarized photons, since an eavesdropper could otherwise use any extra particles with the same state for a measurement.

It may seem surprising that the key can be sent without any protection. However, an eavesdropper has to measure the qubits in order to get any information. This requires choosing a basis, i.e., B_0 or B_1, which is incorrect in about half of the cases. Remember that the correct basis is only known to Alice during the transmission of the qubits. At first, Alice and Bob are not aware of an interception, but the error rate of the check bits will increase significantly. In fact, Bob will measure around 25% incorrect check bits, since the error rate is 50% if an eavesdropper used the wrong basis. Therefore, Alice and Bob can detect an adversary who has intercepted a sufficient number of quantum bits. Assuming that Alice and Bob accept a maximum error rate of 2.5%, an adversary can only eavesdrop around 10% of the bits if they want to remain undetected. Privacy amplification methods reduce an adversary's partial information on the key by producing a new, shorter key.

Table 13.2. Quantum key distribution example.

Position	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
Alice's key	0	1	0	0	1	1	0	1	0	0	1	0	1	0	1	1	0
Alice's basis	1	0	1	1	0	0	0	1	1	1	0	0	0	1	1	1	0
Alice sends	+	1	+	+	1	1	0	−	+	+	1	0	1	+	−	−	0
Bob's basis	1	0	0	0	0	1	0	1	0	1	1	1	1	1	1	0	1
Bob measures	+	1	1	0	1	−	0	−	1	+	−	+	+	+	−	0	+
Same basis	✓	✓			✓		✓	✓		✓				✓	✓		
Shared key	0	1			1		0	1		0				0	1		
Check bits		1			1					0				0			
Key bits	0						0	1							1		

Example 13.17. (See Table 13.2) Suppose $n = 4$ and $\delta = 1$. Alice generates the random key

$$k = 0100\ 1101\ 0010\ 1011\ 0$$

of length 17. Alice and Bob's choice of bases is given by the binary strings

$$b_A = 1011\ 0001\ 1100\ 0111\ 0,$$
$$b_B = 1000\ 0101\ 0111\ 1110\ 1,$$

where 0 represents the basis $B_0 = \{|0\rangle, |1\rangle\}$ and 1 the basis $B_1 = \{|+\rangle, |-\rangle\}$. Alice sends the following qubits:

$$+1 + +110 - + + 101 + - - 0.$$

Alice and Bob exchange b_A and b_B. The following positions coincide: 1, 2, 5, 7, 8, 10, 14, 15. Hence Alice and Bob used the same basis for these positions and 8 shared key bits remain. Alice chooses positions 2, 5, 10, 14 to check for eavesdropping, which would probably change at least one bit. They exchange the check bits and verify them. If the check bits match, then they accept the key exchange. The resulting key 0011 is defined by the remaining four positions 1, 7, 8 and 15. ◊

13.7. Summary

- Quantum computing relies on quantum mechanics and uses quantum bits (qubits) instead of classical bits.
- The state of a qubit is a superposition of the basis states $|0\rangle$ and $|1\rangle$.
- A single qubit state can be represented by a point on the Bloch sphere.
- The measurement of a qubit gives a classical bit. The probability of measuring 0 or 1 is given by the square of the amplitude of $|0\rangle$ and $|1\rangle$, respectively.
- A system of n qubits has 2^n basis states and a measurement gives n classical bits.
- A quantum gate has n input qubits and n output qubits and the transformation is described by a unitary matrix. Quantum algorithms use quantum gates and measurements.
- The Quantum Fourier Transform (QFT) maps an input state to a superposition of Fourier coefficients.
- Shor's factoring algorithm leverages the QFT to solve the factoring problem in polynomial time. Quantum computers can break conventional public-key cryptography, but sufficiently large and stable systems are not yet available.
- The Quantum Key Distribution (QKD) protocol BB84 uses a sequence of single qubits for key distribution. Since the internal state of a qubit is changed by a measurement, the interference by an adversary can be detected.

Exercises

1. Show that the Bell state $|\psi\rangle = \frac{1}{\sqrt{2}}|00\rangle + \frac{1}{\sqrt{2}}|11\rangle$ cannot be written as the product of two single qubit states $(a_1|0\rangle + b_1|1\rangle) \otimes (a_2|0\rangle + b_2|1\rangle)$.
2. Show that applying the CNOT gate to $H|0\rangle \otimes |0\rangle$ gives the above Bell state. Depict the corresponding circuit. Give the other three Bell states $H|0\rangle \otimes |1\rangle$, $H|1\rangle \otimes |0\rangle$ and $H|1\rangle \otimes |1\rangle$.
3. Describe the transformations of Pauli-X, Y, Z, S (Phase), T $(\frac{\pi}{8})$ and H gates on the Bloch sphere.
4. Prove that there is a bijection between the state space for a single-qubit system and the complex projective space $\mathbb{P}^1(\mathbb{C})$.
5. Give the matrix of the Walsh-Hadamard transformation of two qubits.
6. Determine the matrix of the Toffoli gate on three input qubits.
7. Show that the Toffoli gate gives a quantum analogue of the classical NAND operation.
8. Let $x \in \{0,1\}^n$. Prove the formula

$$H^{\otimes n}\,|x\rangle = \frac{1}{\sqrt{2^n}} \sum_{z\in\{0,1\}^n} (-1)^{x\cdot z}\,|z\rangle\,.$$

9. Give an explicit description of the Quantum Fourier Transform for $N = 2$ and $N = 4$.
10. Suppose that $n = pq$, where p and q are prime numbers. Let $a \in \mathbb{Z}_n^*$ and let r, r_p, r_q be the order of a in $\mathbb{Z}_n^*$, $\mathbb{Z}_p^*$ and $\mathbb{Z}_q^*$, respectively. Show that r is the least common multiple of r_p and r_q.
11. Let $n = 47053$ and $a = 11 \in \mathbb{Z}_n^*$. Suppose that $\operatorname{ord}(a) = 7770$ is already known. Find the factors p and q of n.
12. Alice and Bob use the BB84 protocol and Mallory intercepts m key bits. Determine the expected number of faulty bits, i.e., bits that differ from Alice's original key.
13. Suppose $n = 2048$ and $\delta = 128$ in the BB84 protocol. Mallory intercepts 64 key bits. How many of the check bits are likely to differ if no other transmission errors occur?