1 Sets and Lists

Sets and lists are fundamental concepts that arise in various contexts, including computer algorithms. We study basic counting problems in terms of these concepts.

Sorting. A common computational task is to rearrange elements in order. Given a linear array $A[1..n]$ of integers, rearrange them such that $A[i] \leq A[i+1]$ for $1 \leq i < n$.

```
for i = 1 to n − 1 do
  for j = i + 1 downto 2 do
    if A[j] > A[j − 1] then
      aux = A[j]; A[j] = A[j − 1]; A[j] = aux
    endif
  endfor
endfor.
```

We wish to count the number of comparisons made in this algorithm. For example, sorting an array of five elements uses 15 comparisons. In general, we make $1 + 2 + \cdots + (n-1) = \sum_{i=1}^{n-1} i$ comparisons.

Sums. We now derive a closed form for the above sum by adding it to itself. Arranging the second sum in reverse order and adding the terms in pairs, we get

$$[1 + (n-1)] + \ldots + [(n-1) + 1] \quad = \quad n(n-1).$$

Since each number of the original sum is added twice, we divide by two to obtain

$$\sum_{i=1}^{n-1} i \quad = \quad \frac{n(n-1)}{2}.$$

As with many mathematical proofs, this is not the only way to derive this sum. We can think of the sum as two sets of stairs that stack together, as in Figure 1. At the base, we have $n-1$ gray blocks and one white block. At each level, one more block changes from gray to white, until we have one gray block and $n-1$ white blocks. Together, the stairs form a rectangle divided into $n-1$ by n squares, with exactly half the squares gray and the other half white. Thus, $\sum_{i=1}^{n} i = \frac{n(n-1)}{2}$, same as before. Notice that this sum can appear in other forms, for example,

$$\begin{aligned} \sum_{i=1}^{n-1} i &= 1 + 2 + \ldots + (n-1) \\ &= (n-1) + (n-2) + \ldots + 1 \\ &= \sum_{i=1}^{n-1} (n-i). \end{aligned}$$

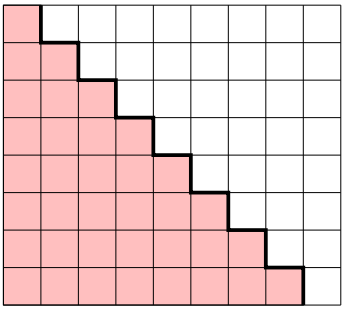

Figure 1: The number of squares in the grid is twice the sum from 1 to 8.

Sets. A *set* is an unordered collection of distinct elements. The *union* of two sets is the set of elements that are in one set or the other, that is, $A \cup B = \{x \mid x \in A \text{ or } x \in B\}$. The *intersection* of the same two sets is the set of elements that are in both, that is, $A \cap B = \{x \mid x \in A \text{ and } x \in B\}$. We say that A and B are disjoint if $A \cap B = \emptyset$. The *difference* is the set of elements that belong to the first but not to the second set, that is, $A - B = \{x \mid x \in A \text{ and } x \notin B\}$. The *symmetric difference* is the set of elements that belong to exactly one of the two sets, that is, $A \oplus B = (A-B) \cup (B-A) = (A \cup B) - (A \cap B)$. Look at Figure 2 for a visual description of the sets that

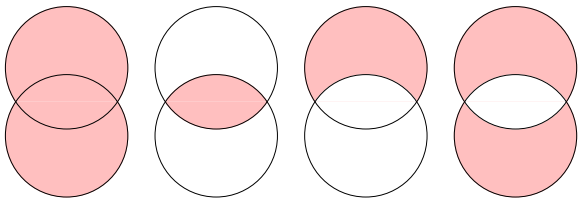

Figure 2: From left to right: the union, the intersection, the difference, and the symmetric difference of two sets represented as disks in the plane.

result from the four types of operations. The number of elements in a set A is denoted as $|A|$. It is referred to as the *size* or the *cardinality* of A. The number of elements in the union of two sets cannot be larger than the sum of the two sizes.

SUM PRINCIPLE 1. $|A \cup B| \leq |A| + |B|$ with equality if A and B are disjoint.

To generalize this observation to more than two sets, we call the sets $S_1, S_2, \ldots, S_m$ a *covering* of $S = S_1 \cup S_2 \cup \ldots \cup S_m$. If $S_i \cap S_j = \emptyset$ for all $i \neq j$, then the covering

is called a *partition*. To simplify the notation, we write $\bigcup_{i=1}^{m} S_i = S_1 \cup S_2 \cup \cdots \cup S_m$.

SUM PRINCIPLE 2. Let $S_1, S_2, \ldots, S_m$ be a covering of S. Then, $|S| \leq \sum_{i=1}^{m} |S_i|$, with equality if the covering is a partition.

Matrix multiplication. Another common computational task is the multiplication of two matrices. Assuming the first matrix is stored in a two-dimensional array $A[1..p, 1..q]$ and the second matrix is stored in $B[1..q, 1..r]$, we match up rows of A with the columns of B and form the sum of products of corresponding elements. For example, multiplying

$$A = \begin{bmatrix} 1 & 3 & 2 \\ 0 & 2 & 4 \end{bmatrix}$$

with

$$B = \begin{bmatrix} 0 & 2 & 3 \\ 1 & 2 & 5 \\ 4 & 0 & 1 \end{bmatrix}$$

results in

$$C = \begin{bmatrix} 11 & 8 & 20 \\ 18 & 4 & 14 \end{bmatrix}.$$

The algorithm we use to get C from A and B is described in the following pseudo-code.

```
for i = 1 to p do
  for j = 1 to r do
    C[i, j] = 0;
    for k = 1 to q do
      C[i, j] = C[i, j] + A[i, k] · B[k, j]
    endfor
  endfor
endfor.
```

We are interested in counting how many multiplications the algorithm takes. In the example, each entry of the result uses three multiplications. Since there are six entries in C, there are a total of $6 \cdot 3 = 18$ multiplications. In general, there are q multiplications for each of pr entries of the result. Thus, there are pqr multiplications in total. We state this observation in terms of sets.

PRODUCT PRINCIPLE 1. Let $S = \bigcup_{i=1}^{m} S_i$. If the sets $S_1, S_2, \ldots, S_m$ form a partition and $|S_i| = n$ for each $1 \leq i \leq m$ then $|S| = nm$.

We can also encode each multiplication by a triplet of integers, the row number in A, the column number in A which is also the row number in B, and the column number in B. There are p possibilities for the first number, q for the second, and r for the third number. We generalize this method as follows.

PRODUCT PRINCIPLE 2. If S is a set of lists of length m with i_j possibilities for position j, for $1 \leq j \leq m$, then $|S| = i_1 \cdot i_2 \cdot \ldots \cdot i_m = \prod_{j=1}^{m} i_j$.

We can use this rule to count the number of cartoon characters that can be created from a book giving choices for head, body, and feet. If there are p choices for the head, q choices for the body, and r choices for the legs, then there are pqr different cartoon characters we can create.

Number of passwords. We apply these principles to count the passwords that satisfy some conditions. Suppose a valid password consists of eight characters, each a digit or a letter, and there must be at least two digits. To count the number of valid passwords, we first count the number of eight character passwords without the digit constraint: $(26+10)^8 = 36^8$. Now, we subtract the number of passwords that fail to meet the digit constraint, namely the passwords with one or no digit. There are 26^8 passwords without any digits. To count the passwords with exactly one digit, we note that there are 26^7 ways to choose an ordered set of 7 letters, 10 ways to choose one digit, and 8 places to put the digit in the list of letters. Therefore, there are $26^7 \cdot 10 \cdot 8$ passwords with only one digit. Thus, there are $36^8 - 26^8 - 26^7 \cdot 10 \cdot 8$ valid passwords.

Lists. A *list* is an ordered collection of elements which are not necessarily different from each other. We note two differences between lists and sets:

(1) a list is ordered, but a set is not;

(2) a list can have repeated elements, but a set can not.

Lists can be expressed in terms of another mathematical concept in which we map elements of one set to elements of another set. A *function* f from a *domain* D to a *range* R, denoted as $f : D \to R$, associates exactly one element in R to each element $x \in D$. A list of k elements is a function $\{1, 2, \ldots, k\} \to R$. For example, the function in Figure 3 corresponds to the list $a, b, c, b, z, 1, 3, 3$. We can use the Product Principle 2 to count the number of different functions from a finite domain, D, to a finite range, R.

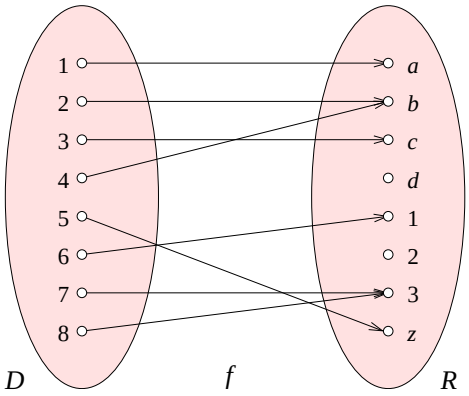

Figure 3: A function representing a list.

Specifically, we have a list of length $|D|$ with $|R|$ possibilities for each position. Hence, the number of different functions from D to R is $|R|^{|D|}$.

Bijections. The function $f : D \to R$ is *injective* or *one-to-one* if $f(x) \neq f(y)$ for all $x \neq y$. It is *surjective* or *onto* if for every $r \in R$, there exists some $x \in D$ with $f(x) = r$. The function is *bijective* or a *one-to-one correspondence* if it is both injective and surjective.

BIJECTION PRINCIPLE. Two sets D and R have the same size if and only if there exists a bijection $f : D \to R$.

Thus, asking how many bijections there are from D to R only makes sense if they have the same size. Suppose this size is finite, that is, $|D| = |R| = n$. Then being injective is the same as being bijective. To count the number of bijections, we assign elements of R to elements of D, in sequence. We have n choices for the first element in the domain, $n - 1$ choices for the second, $n - 2$ for the third, and so on. Hence the number of different bijections from D to R is $n \cdot (n - 1) \cdot \ldots \cdot 1 = n!$.

Summary. Today, we began with the building blocks of counting: sets and lists. We went through some examples using the sum and product principles: counting the number of times a loop is executed, the number of possible passwords, and the number of combinations. Finally, we talked about functions and bijections.