Warm-up

In how many ways can you draw a first card and then a second card from a deck of 52 cards if you *do not* put the drawn cards back into the deck?

In how many ways can you draw a first card and then a second card from a deck of 52 cards if you *do* put the drawn cards back into the deck?

Lists, Permutations, Subsets

Example 1. A password for a certain computer system is supposed to be between four and eight characters long and composed of lowercase and/or uppercase letters, and the digits 0-9. How many passwords are possible? What counting principles did you use? Estimate the percentage of the possible passwords that have exactly four letters.

Lists, Permutations, Subsets

Example 1. A password for a certain computer system is supposed to be between four and eight characters long and composed of lowercase and/or uppercase letters, and the digits 0-9. How many passwords are possible? What counting principles did you use? Estimate the percentage of the possible passwords that have exactly four letters.

First, let's consider whether we can or should use either the Sum or the Product Principle. We know that we can make a password containing four, five, six, seven, or eight characters. Let P_i for $i \in \{4,5,6,7,8\}$ be the set of all passwords with i characters. Then for $i \neq j$, we have that P_i and P_j are disjoint, and if P is the set of all passwords that satisfy the problem, then by the Sum Principle we have

$$|P| = \left|\bigcup_{i=4}^{8} P_i\right| = \sum_{i=4}^{8} |P_i|.$$

We now need to find $|P_i|$. For an i-character password, there are 26 choices of both lower and uppercase letters, and ten digits, for a total of 62 choices of character. By the Product Principle, we have $|P_i| = 62^i$. Therefore the total number of passwords is given by

$$|P| = \sum_{i=4}^{8} |P_i| = 62^4 + 62^5 + 62^6 + 62^7 + 62^8 = 221,919,451,335,856.$$

Of these, 62^4 of them contain four characters, so the percentage of passwords with four characters is given by

$$\frac{62^4}{62^4 + 62^5 + 62^6 + 62^7 + 62^8} \times 100 \approx 0.0000067\%$$

Suppose that a computer can test one password every microsecond. Then it would take

$$62^4 * 0.000001 \approx 14.78s$$

to test every four-character password, but it would take

$$62^8 * 0.000001 \approx 218,340,105.58s$$

to test every eight-character password, or around 7 years! If we allowed passwords of 20 characters, then it would take this computer about 22 sextillion years (22×10^{22}) to try every 20-character password! For reference, the best science estimates the age of the univserse to be about 13 billion years, some 12 orders of magnitude difference! Lesson: use longer passwords!

Notice that we used the Product Principle without writing a union of sets of equal size. We could have, but it would have been a little clumsy-looking. Instead we can use the following, second version of the Product Principle, which is easily derived using mathematical induction, which we will discuss in detail in a couple of weeks.

Proposition 1. (Product Principle - Version 2) If S is a set of lists of length m having the properties

1. there are i_1 different first elements of lists in S, and
2. for each $k > 1$ and each choice of the first $k - 1$ elements of a list in S, there are i_k choices of elements in position k of those lists,

then there are

$$\prod_{k=1}^{m} i_k = i_1 i_2 i_3 \cdots i_m$$

lists in S.

Let's apply this version of the product principle to compute the number of m-character passwords. Because an m-character password is simply a list of m characters and because there are 62 different first elements of the password and 62 choices for each other position of the password, we have that $i_1 = 62$, $i_2 = 62$, $\ldots$, $i_m = 62$. Thus, this version of the Product Principle tells us immediately that the number of passwords of length m is

$$\prod_{k=1}^{m} i_k = i_1 i_2 i_3 \cdots i_m = 62^m.$$

We very conveniently used the word *list* above without ever defining it. Let's clear this up now, but first we will define a function and some associated terms so that our definition will be precise.

A *function*, f, from a set S to a set T, written

$$f : S \to T$$

is a mapping that associates exactly one element of T to each element of S. The set S is called the *domain* of f and the set T is called its *range* or *codomain*. We often use the notation $f(x)$ so indicate that the function f is acting on some element $x \in S$. Sometimes we refer to this mapping, emphasizing that it is associating an element $x \in S$ to and element $y \in T$ using the notation

$$x \mapsto y$$

It is often handy to know that the *image* of a function $f : S \to T$, denoted $\operatorname{Im} f$, is the set of all values $f(x)$ for $x \in S$, *i.e.*,

$$\operatorname{Im} f = \{f(x) \mid x \in S\}\,.$$

In this course we often explore functions over finite sets, in which case it is sometimes possible to simply explicitly list every association between our domain and codomain; *e.g.*

$$f(\alpha) = \text{Tim}, f(\beta) = \text{🌳}, f(\gamma) = 7 \tag{1}$$

defines a function

$$f : \{\alpha, \beta, \gamma\} \to \left\{\text{Tim}, \text{🌳}, 7\right\}.$$

We could also write the function as a set of *ordered pairs*; *i.e.*,

$$\left\{(\alpha, \text{Tim}), \left(\beta, \text{🌳}\right), (\gamma, 7)\right\}.$$

Now we can give a precise definition of a *list*.

A *list* of k elements from a set T is a function

$$\{1, 2, \ldots, k\} \to T.$$

Example 2. How many functions are there from a two-element set to a three-element set? How many functions are there from a three-element set to a two-element set?

A function f is *one-to-one*, or *injective*, or an *injection*, if $f(x) \neq f(y)$ when $x \neq y$. In symbols, f is an *injection* if

$$x \neq y \quad \Rightarrow \quad f(x) \neq f(y).$$

Note that this is equivalent to

$$f(x) = f(y) \quad \Rightarrow \quad x = y$$

by *contraposition*, the veracity of which is proved by *modus tollens* or *DeMorgan's laws*. Note also that a function is injective if and only if it passes the horizontal line test.

A function, $f : S \longrightarrow T$ is called onto, or *surjective*, or a *surjection*, if for all $y \in T$ there is an $x \in S$ such that $f(x) = y$. Note that, for example, $x \longmapsto x^2$ is not surjective as a function $\mathbb{R} \longrightarrow \mathbb{R}$, but it is a surjection as a function $\mathbb{R} \longrightarrow [0, \infty)$. Indeed, every function is a surjection onto its image; *i.e.*,

$$f : S \longrightarrow \operatorname{Im} f$$

is always surjective.

A function that is both injective and surjective is called *bijective*, or is a *bijection*, or a *one-to-one correspondence*. A bijection from a set to itself is called a *permutation*. We also have the *Bijection Principle*:

Proposition 2. (Bijection Principle) If $f : S \longrightarrow T$ is a bijection, then $|S| = |T|$.

Example 3. Using two- or three-element sets as domains and ranges, find an example of a one-to-one function that is not onto. Using two- or three-element sets as domains and ranges, find an example of an onto function that is not one-to-one.

Example 4. Among all iterations of line 5 of the pseudocode, what is the total number of times this line checks three points to see if they are collinear?

Example 4 Code

```
1    trianglecount = 0
2    for i = 1 to n-1
3        for j = i+1 to n
4            for k = j+1 to n
5                if i, j, and k are not collinear
6                    trianglecount += 1
```

Note that, since $i < j < k$, the code examines each such increasing triple (i, j, k) exactly once. If $n = 4$, then we would count the increasing triples $(1, 2, 3)$, $(1, 2, 4)$, $(1, 3, 4)$, and $(2, 3, 4)$. We could count all such triples, but note instead that this is precisely the same as the number of 3-element subsets of $\{1, 2, \ldots, n\}$. Because of the Bijection Principle, we can count such subsets instead to find `trianglecount`.

Since sets do not have repeating elements, if we want to choose k elements from a set of cardinality n, with $k \leq n$, then we have n choices for the first element, $n - 1$ for the second, up to $n - k + 1$ choices for the kth element. This value, called the kth *falling factorial power of n* is important enough that Donald E. Knuth (a name you ought to learn now) suggested it have its own symbol, $n^{\underline{k}}$. We have

$$n^{\underline{k}} = \prod_{i=0}^{k-1}(n - i) = n(n - 1) \cdots (n - k + 1) = \frac{n!}{(n - k)!}.$$

We still have a problem! The subsets we have generated have repeats! For instance, if $n = 4$ using the above logic to choose 3-element subsets would yield $\{1, 2, 3\}$, $\{2, 3, 1\}$, $\{3, 1, 2\}$, $\{3, 2, 1\}$, $\{2, 1, 3\}$, and $\{1, 3, 2\}$, even though these are all the same set; that is, we get 6 different lists, which each represent the same set. Indeed, there are 6 such lists because there are three different numbers in each and $3 \cdot 1 \cdot 1 = 6$ ways to choose them. This argument gives us our general solution. If we wish to choose k unique elements from an n-element set where order does not matter, then we have

$$\binom{n}{k} = \frac{n^{\underline{k}}}{k!} = \frac{n!}{k!(n - k)!}.$$

The numbers $\binom{n}{k}$, read *n choose k*, are called *binomial coefficients*.