

Counting

Carlo Tomasi

To count means to determine the cardinality of a set, that is, the number of elements it contains. The cardinality of a set S is denoted by $\#(S)$.

This note shows two different counting methodologies. The first one establishes a calculus for the cardinalities of sets constructed from sets of known cardinalities through the following composition operators: (i) union of disjoint sets (“set addition”); (ii) set difference between a set and a subset of it (“set subtraction”); (iii) Cartesian product of sets. In contrast with other set operators such as union and intersection, the cardinalities of the sets obtained through these three operators are known functions of the cardinalities of the constituent sets.

The second method can be applied whenever a set can be defined by specifying a procedure for constructing its elements. The cardinality of the set in question is then determined by counting the options available for constructing a general member of the set.

1 Counting through Set Composition

Sets are often built by applying composition operators such as union, intersection, complement, Cartesian product, and so forth to other sets. It would be convenient if were possible to express the cardinality of such compositions as a function of the cardinalities of the constituent sets. This is possible for the cross product (or Cartesian product) of sets:

$$\#(A \times B) = \#(A)\#(B) . \quad (1)$$

When cross products are computed, the universes U_A and U_B in which A and B are defined are multiplied as well to yield $U = U_A \times U_B$ for $A \times B$.

Unfortunately, no counting rules are generally associated with the other set-composition operators. For instance, the cardinality of the union or intersection of two sets A and B cannot be determined from those of A and B . We only have bounds:

$$\max(\#(A), \#(B)) \leq \#(A \cup B) \leq \#(A) + \#(B)$$

where the lower bound is attained when one set is contained in the other, and the upper bound is attained when the two sets are disjoint. Similarly,

$$0 \leq \#(A \cap B) \leq \min(\#(A), \#(B)) .$$

Here, the lower bound is attained when the two sets are disjoint and the upper bound when one set is contained in the other.

When A and B are not disjoint, the following *inclusion-exclusion formula* holds:

$$\#(A \cup B) = \#(A) + \#(B) - \#(A \cap B)$$

because the elements in the intersection are counted once in $\#(A)$ and once again in $\#(B)$. Similar results can be obtained by *partitioning* the union of A and B , that is, by rewriting it as a union of disjoint sets:

$$A \cup B = A \cup (\bar{A} \cap B) = (A \cap \bar{B}) \cup B = (A \cap \bar{B}) \cup (\bar{A} \cap B) \cup (A \cap B).$$

This then yields the following three *partitioned counting* formulas:

$$\#(A \cup B) = \#(A) + \#(\bar{A} \cap B) = \#(A \cap \bar{B}) + \#(B) = \#(A \cap \bar{B}) + \#(\bar{A} \cap B) + \#(A \cap B). \quad (2)$$

These results are useful when the cardinalities of the various intersections that appear in them can be computed in some separate way. Fortunately, this is often the case.

Exercise. Draw Venn diagrams for all the formulas above.

A useful formal device for counting is to use a different symbol for the union of two sets when the two sets are disjoint:

$$A + B = \begin{cases} A \cup B & \text{if } A \cap B = \emptyset \\ \text{undefined} & \text{otherwise} \end{cases}$$

and to say that in that case A and B are *added* together. Set addition should not be viewed as a new set operator, but merely as the old union operator with emphasis added on the fact that A and B are disjoint. It is an error to write $A + B$ if A and B are not disjoint, just as it would be an error to write ' abc ' + 2 without having defined the meaning of '+' when applied to a string and a number.

If $C = A + B$, then we can write $A = C - B$ or $B = C - A$ as long as we interpret the set *subtraction* operator '-' as follows:

$$C - B = \begin{cases} C \setminus B & \text{if } B \subseteq C \\ \text{undefined} & \text{otherwise} \end{cases}$$

where '\setminus' is the set difference. Note in particular that if U is the universe, then the complement of a set A can be written as $\bar{A} = U - A$. This is useful whenever computing the cardinality of A is hard but computing those of U and \bar{A} is easier, or vice versa.

In summary, the elements of a set S can be counted as follows:

- Determine a collection of *basic sets* S_1, \dots, S_n whose cardinality is known.
- Write S as a combination of cross products and set addition and subtraction operations, starting from the basic sets.
- In the resulting combination, replace the basic sets with their cardinalities, set additions with sums, set subtractions with differences, and set cross products with products.
- Compute the value of the resulting formula.

2 Example

Consider a simplified password scheme for some security application. Valid characters for a password are the 26 letters of the alphabet (case insensitive, so 'A' and 'a' are the same letter) and the ten digits from 0 to 9. A password is a nonempty string of up to three characters, at least one of which must be a digit. If more

digits are present, they must be different. For instance, the strings '2', '15', '317', 'A6C', '2B' are valid. Examples of invalid passwords are 'C11' (digit repetition), 'ABC' (no digit), and 'BC36' (too long). How many passwords are possible in this scheme?

In this case, basic sets are simple to define: The universe C for each character in the password is the set of alphanumeric characters, and $\#(C) = 36$. Digits and letters form the sets D and L with cardinality 10 and 26, respectively.

The set P of passwords can be written as the set addition of the sets of valid passwords with one, two, or three characters:

$$P = P_1 + P_2 + P_3$$

where

$$\begin{aligned} P_1 &= \{\text{passwords with one character}\} \\ P_2 &= \{\text{passwords with two characters}\} \\ P_3 &= \{\text{passwords with three characters}\} . \end{aligned}$$

These sets are disjoint, because a password cannot have two different lengths at the same time. A one-character password must be a digit, so we easily have

$$P_1 = D ,$$

a basic set. P_2 can be written as the set Q_2 of all two-character passwords that ignore the rule about distinct digits, minus the set V_2 of those that violate it (a subset of Q_2). Violations are most easily listed:

$$V_2 = \{00, 11, 22, 33, 44, 55, 66, 77, 88, 99\} ,$$

so this can be considered a basic set. The set Q_2 can be specified in different ways. One is to first start with a union

$$\begin{aligned} Q_2 &= \{\text{two-character passwords starting with a digit}\} \\ &\cup \{\text{two-character passwords ending with a digit}\} . \end{aligned}$$

[Note that both components of Q_2 contain all the repeated-digit pairs in V_2 .] These two sets are not disjoint, so they can be partitioned:

$$\begin{aligned} Q_2 &= \{\text{two-character passwords starting with a digit}\} \\ &+ \{\text{two-character passwords starting with a letter and ending with a digit}\} \end{aligned}$$

(without this partitioning, we would be counting passwords with two digits twice) and we can immediately write these two sets as cross products of basic sets:

$$Q_2 = (D \times C) + (L \times D) .$$

In summary,

$$P_2 = Q_2 - V_2 = ((D \times C) + (L \times D)) - V_2 .$$

Handling P_3 takes more work. First, we can write

$$P_3 = Q_3 - V_3$$

in analogy with a similar subtraction for P_2 . In this expression, Q_3 is the set of all three-character passwords that ignore the rule about distinct digits. The cases in V_3 (three-character violations of the distinct-digit rule) can be split based on what digit is being repeated:

$$V_3 = V_{30} + \dots + V_{39}$$

where V_{3d} is the set of three-letter passwords where the digit d is repeated. It is not possible to repeat two different digits in three characters, so the sets V_{3d} are disjoint.

Fortunately, we can reason about V_{3d} once, since the number of occurrences does not depend on which digit is being repeated. In other words,

$$\#(V_{30}) = \dots = \#(V_{39}) .$$

There are four types of violations of the distinct-digit rule, which correspond to four disjoint sets: V_{3d1} is the set of three-letter passwords in which digit d is repeated in positions 2, 3 and *not* in position 1. V_{3d2} and V_{3d3} are defined similarly. The set V_{3d0} is the singleton

$$V_{3d0} = \{ddd\} ,$$

a basic set. So

$$V_{3d} = V_{3d0} + V_{3d1} + V_{3d2} + V_{3d3} .$$

Since character permutations do non matter in our problem, the three sets V_{3dk} for $k = 1, 2, 3$ have the same cardinality. Specifically,

$$\#(V_{3dk}) = \#(C - \{d\}) = 36 - 1 = 35 ,$$

because passwords that repeat digit d in positions other than position k can be chosen by using any character other than the digit d in position k . The other two characters are fixed and equal to d .

Rather than writing V_3 as a function of all these sets, it is faster to write directly its cardinality (set for instance $d = 0$ and $k = 1$):

$$\#(V_3) = 10 \#(V_{30}) = 10(3 \#(V_{301}) + \#(V_{300})) = 10(3 \times 35 + 1) = 10 \times 106 = 1060 .$$

Finally, Q_3 is the set of all three-letter passwords that have at least one digit (including repeated digits). The simplest way to write this set is to first list all three-letter strings (a cross product of C with itself twice), and then to subtract the set of three-letter strings without any digit:

$$Q_3 = (C \times C \times C) - (L \times L \times L) .$$

We are done with the set combination. Now we just need to tally cardinalities:

$$\begin{aligned} \#(P_1) &= \#(D) = 10 \\ \#(P_2) &= \#(D)\#(C) + \#(L)\#(D) - \#(V_2) = 10 \times 36 + 26 \times 10 - 10 = 610 \\ \#(P_3) &= \#(Q_3) - \#(V_3) = \#(C)^3 - \#(L)^3 - \#(V_3) = 36^3 - 26^3 - 1060 = 28,020 \end{aligned}$$

and finally

$$\#(P) = \#(P_1) + \#(P_2) + \#(P_3) = 10 + 610 + 28,020 = 28,640 .$$

3 Counting Sets of Structured Elements

In the example in the previous Section, the elements of the target set S are made of parts: one element (password) of the set is a string, and a string is made of several characters. Elements that are made of parts are said to be *structured*.

Sets of structured elements lend themselves to a counting method that is different than the compositional method of Section 1. The new method first defines a procedure for constructing an element of the target set S from its parts. If there are p parts to an element, the procedure constructs that element in p steps. Different elements in the set may have different numbers of parts, so a general procedure for constructing an element has the following form, where P is the greatest number of parts an element can have:

```
for  $p = 1, \dots, P$  do  
  Select part  $p$   
  if done then break  
end for
```

The choices available when selecting part p may depend on what choices were made for parts $q < p$. This suggests drawing a *selection tree* with P levels, each level corresponding to a part. A branch in the tree corresponds to a group of equivalent choices for a particular part of the element, in the sense that choices in the same group constrain further choices in the same way. A node at level p in the tree represents a group of elements with p parts. If such an element is complete (that is, it is a member of the target set S), then that node is a *terminal node*. Terminal nodes are not necessarily leaves of the tree, because additional elements can be obtained by adding parts to a complete element. For instance, $p1$ is a complete password, but so is $p1c$. Choices that cannot lead to complete elements are not represented in the tree.

Here is how to determine the cardinality of a set S of structured elements:

- Build a selection tree for set S as follows:
 - Build a tree made of a single node. This represents the empty set.
 - For each leaf of the current tree, let p be the depth of the leaf (its distance from the root), and do the following:
 - * If the leaf is complete, mark it as such.
 - * If $p < P$ and part $p + 1$ can be added to elements in the leaf without violating the definition of S , group the possible options for part $p + 1$ into groups that constrain further choices in the same way. Add a branch to the leaf for each group, and label each branch with the number of choices it represents.
- Label the nodes of the selection tree with cardinalities as follows:
 - Label the root with 0, and its children with the label of the branch that connects each of them to the root.
 - While visiting the tree in some order starting from the root, label each unlabelled node with the product of the label of its parent and the label of the branch that connects the node with its parent.
- The cardinality of S is the sum of the labels for all terminal nodes.

4 Example Revisited

In the example of Section 2, the parts of an element (password) of the target set S are its characters. A possible procedure for constructing a password is to (i) select the first character, then either stop or (ii) select the second character, then either stop or (iii) select the third character. Each selection may constrain the remaining choices: if a digit is selected, that digit cannot be selected again, because digits in a password must be different. If no digit has been selected so far, a digit is needed for at least one of the future characters.

Figure 1 shows part of the selection tree for this problem. Terminal nodes are drawn as squares, and non-terminal nodes as circles. Nodes can be labelled after the tree is complete, or while building the tree, as done in the figure. Children of the root are labelled differently from the other nodes: they just inherit the number on the branch that links them to the root. Each child at a deeper level is labelled with the product of the label on the branch that leads to it and the label of the parent. For instance, the bottom leaves in Figure 1 have labels of $26 \times 26 = 676$ and $10 \times 26 = 260$, respectively. This last leaf is complete, because a letter followed by a digit is a valid password. So is a single digit, corresponding to the rightmost node in Figure 1, which is also terminal, and therefore drawn as a square. The other nodes in the partial tree in Figure 1 represent strings with just letters, which are not valid passwords (no digits). These nodes are therefore not terminal, and are drawn as circles.

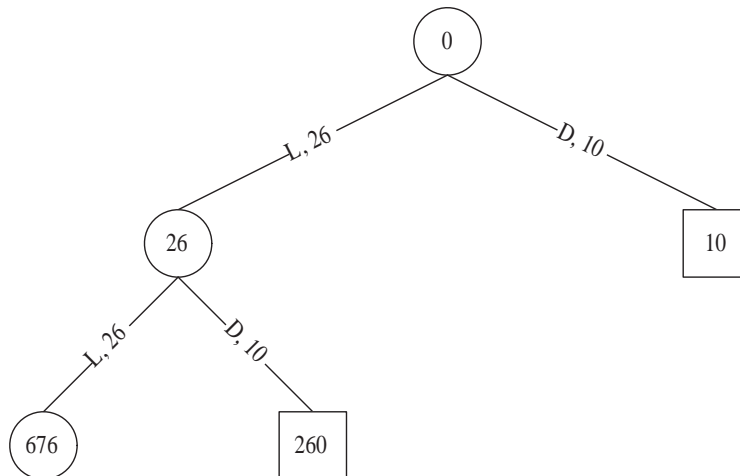


Figure 1: First part of a selection tree for the password problem. 'D' means 'Digit' and 'L' means 'Letter'.

The two branches leaving the root represent two different groups of choices for the first character of the password. If a letter is chosen, it does not matter *which* letter it is, so all these options are gathered into a group. On the other hand, choosing a digit for the first character has different implications on later choices than choosing a letter has. If a letter is chosen, a digit must be selected at some later stage. If a digit is chosen instead, the only constraint is that that digit is not repeated. The counting method would still work if the root had more branches, corresponding to a finer division into groups. For instance, one could have 36 branches off the root, one for each alternative for character 1. This finer distinction, however, is not useful, and would produce a much bigger tree for no advantage.

The rightmost node in Figure 1 represents all one-digit passwords (of which there are ten). This choice rules out using that same digit for subsequent characters (if any). Strictly speaking, the nature of this implication for subsequent choices does depend on *which* digit is selected for the first character. For instance, if a 1 is selected for the first character, then subsequent digits must be different from 1, while if a 5 is selected,

subsequent digits must be different from 5. However, the implications on counting are the same in either case: a subsequent digit must be different from the current digit, no matter what the current digit is. So it is safe to draw a single branch off of the rightmost node in Figure 1, as shown in Figure 2. Subscripts are added to the explanatory notes on each branch, to distinguish between digits added at different stages. Note that D_1 is used in different parts of the tree for different digits. This is fine: We only need different subscripts for different characters along any path from the root to a leaf, and only when the subscript is needed to state a constraint (such as $D_2 \neq D_1$).

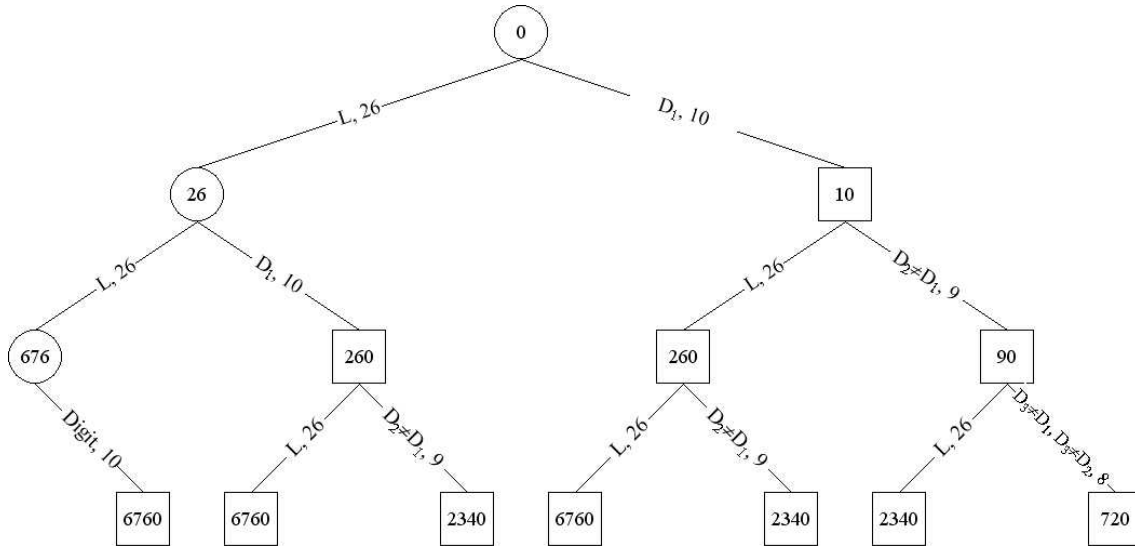


Figure 2: The complete, labelled selection tree for the password problem.

Note the missing leftmost leaf at the bottom of Figure 2: There is no password with only letters. Also note the double constraint on the rightmost branch at the bottom: the third digit must be different from both the first and the second.

Since the tree in Figure 2 is already labelled, we just need to tally the cardinalities in the terminal nodes (rectangles). These come conveniently in a small number of alternatives, so the sum is $10 + 90 + 2 \times 260 + 720 + 3 \times (2340 + 6760) = 28,640$. This is the same cardinality obtained in Section 2.

Which counting method to choose depends on the type of problem. A set whose elements are not structured cannot be counted with the selection method. Problems with many constraints on the parts of an element may be easier to solve with the selection method. Familiarity and comfort with one method over the other is also part of the choice criterion. However, the best option is to use *both* methods when possible: it is relatively easy to make mistakes when counting, and deriving the same result in two different ways is a useful sanity check.