Lab: Generating and Counting Bags  
CSC/MAT 208, “Discrete Structures”  
Department of Computer Science  
Grinnell College  
October 31, 2018

In this lab, we’ll generate and count bags that satisfy specific conditions.
Suppose, first, that we have the three familiar color values, red, green, and blue. From this collection of values, we could build eight different classes, by the rule of the product: we’d make a two-way decision, in or out, for each value. But how many bags can we build?

Once repetitions are allowed, we have more than two options for each of the values in the collection: We could decide to put zero reds into a bag that we’re constructing, or one red, or two, or three, or . . . well, any number of them. So the question won’t make any sense unless we constrain the size of the bags that we want to construct.

If the size is 0, then there’s just one choice for each color: We have to choose zero reds, zero greens, and zero blues. There’s only one way to build an empty bag from these values.

If the size is 1, we obviously get three possibilities: [red], [green], or [blue]. (As in the reading, we’ll use this “white-bracket” notation when specifying bags by listing their members.)

If the size is 2, the choices start getting interesting. There are six different bags, since we could choose two of the same color (in three ways) or one each of two different colors (again in three ways, since order doesn’t count): [red, red], [red, green], [red, blue], [green, green], [green, blue], and [blue, blue].

1. How many bags of size 3 can be constructed from these three values? List them.

We’d like a general rule or formula for computing the number of bags of size \( m \) that can be constructed from \( n \) values, for any natural numbers \( m \) and \( n \). Let’s call this quantity \( B(n, m) \). For instance, we saw in the preceding section that \( B(3, 0) = 1 \), \( B(3, 1) = 3 \), and \( B(3, 2) = 6 \). In the previous exercise, you computed \( B(3, 3) \).

It’s tempting to think that we could apply the Division Rule in this case, computing the number of finite sequences of length \( m \), which would be \( n^m \), and then dividing by \( m! \) to compensate for the fact that bags are unordered. The flaw in this idea is that not every bag corresponds to \( m! \) different sequences. For instance, when \( n = 3 \) and \( m = 4 \), the four-member bag \([\text{red, red, red, green}]\) corresponds to only the four sequences

\[
\langle \text{red, red, red, green} \rangle \\
\langle \text{red, red, green, red} \rangle \\
\langle \text{red, green, red, red} \rangle \\
\langle \text{green, red, red, red} \rangle
\]

not to 24 (which is 4!) different sequences. Thus the precondition for applying the Division Rule isn’t met, because the divisor isn’t fixed, but instead depends on how much repetition there is within the bag. Also, of course, the incorrect formula

\[
B(n, m) = \frac{n^m}{m!}
\]

doesn’t match our observations above: \( B(3, 2) \) is 6, not \( \frac{9^2}{2!} \) (which is 9/2 — obviously, that can’t be the correct count).

Let’s look instead at how one might proceed systematically to construct all possible bags of size \( m \) from \( n \) distinct values. One way to prepare for this problem is to arrange the \( n \) values in a
particular order. Often this order is arbitrary, and of course it won’t carry over to the values once they are in the bags, but it’s a useful thing to do just as a matter of bookkeeping as we construct the bags. In the example above, we might arrange the color values in the order with red first, then green, then blue.

Then we can imagine two kinds of bags: those that contain the first of our \( n \) values at least once, and those that leave it out entirely. In the former case, we’ll still have to choose the \( m - 1 \) other members of the bag, but we can use any of our \( n \) values in the process, so it should be possible to do this in any of \( B(n, m - 1) \) different ways. In the latter case, where we leave out the first value entirely, all \( m \) members of the bag remain to be selected, but we have only \( n - 1 \) values to choose them from, so there will be only \( B(n - 1, m) \) ways to do this. By the Addition Rule, then,

\[
B(n, m) = B(n, m - 1) + B(n - 1, m)
\]

provided that \( n \) and \( m \) are both positive.

Fortunately, the cases at the base of this recursive definition are pretty obvious: \([ \, ]\) is the one and only way to get a bag of size 0 no matter what values are available, so \( B(n, 0) = 1 \) for every natural number \( n \); and there is no way at all to get a bag of positive size if there are no values available, so \( B(0, m) = 0 \) for any positive integer \( m \).

2. In Scheme, write a recursive procedure `bag-count` that computes \( B(n, m) \) given any natural numbers \( n \) and \( m \).

3. Build a table of the values that `bag-count` returns for small values of \( n \) and \( m \), say from \( B(0,0) \) to \( B(6,6) \). (You can write out the table on paper, invoking the `bag-count` procedure to work out the values of the entries.)

When you constructed the table in the previous exercise, the values you computed may have looked familiar. One traditional way of arranging the binomial coefficients \( \binom{n}{k} \) is in the form of a triangular table (“Pascal’s triangle”) with each row corresponding to a value of \( n \) and each column to a value of \( k \) such that \( k \leq n \):

\[
\begin{array}{ccccccc}
  & k = 0 & k = 1 & k = 2 & k = 3 & k = 4 & k = 5 & k = 6 \\
 n = 0 & 1 &   &   &   &   &   &   \\
 n = 1 & 1 & 1 &   &   &   &   &   \\
 n = 2 & 1 & 2 & 1 &   &   &   &   \\
 n = 3 & 1 & 3 & 3 & 1 &   &   &   \\
 n = 4 & 1 & 4 & 6 & 4 & 1 &   &   \\
 n = 5 & 1 & 5 & 10 & 10 & 5 & 1 &   \\
 n = 6 & 1 & 6 & 15 & 20 & 15 & 6 & 1 \\
\end{array}
\]

4. Use this observation (and the table you constructed in the preceding exercise) to express \( B(n, m) \) as a binomial coefficient (under the precondition that \( n \) is positive).

5. Prove the relationship between binomial coefficients and recursively computed values of \( B(n, m) \) that you stated in exercise 3. The proof can be either algebraic or combinatorial, as you prefer.

6. We noted above that, because of repetitions, the members of a bag of cardinality \( n \) might form fewer than \( n! \) sequences of length \( n \). The actual number of sequences depends on the multiplicities of the members of the bag. Suggest a run for computing the number of sequences that can be formed from the members of a bag, given their multiplicities.

7. Write a Scheme procedure that implements the formula you discovered in the preceding exercise, taking as its argument a list of the multiplicities of the members of a bag and returning the number of sequences that can be formed from them.
I am indebted to Kim Spasaro 2014 for pointing out an error in a previous version of this document.

Copyright © 2013, 2014, 2016, 2018  John David Stone

This work is licensed under the Creative Commons Attribution-ShareAlike 4.0 International License. To view a copy of this license, visit

http://creativecommons.org/licenses/by-sa/4.0/deed.en

or send a letter to Creative Commons, 543 Howard Street, 5th Floor, San Francisco, California, 94105, USA.