Password Counting

Clay Thomas

2013

1 Statement of the Problem

> Consider a password that uses an alphabet of a characters. Every character of the alphabet is used at least once. The password is of length l. How many possible passwords are there?

This problem would arise if you saw fingerprints on a characters of a keypad and you knew the password had length l.

2 Recursion

If all a characters did not have to be used, the problem would be fairly simple. You could have l choices of a options, for a^l options. Our approach is to begin with this value, and then subtract away the number of situations where one or more of the characters are not used. To do this, we index by the number of characters which are not used, and implement recursion.

Let $R_l(a)$ be the number of passwords of length l, using all of a characters. Then, there are $\binom{a}{1}R_l(a-1)$ ways to make a password that leaves out exactly one of the characters. Note that this value is $\binom{a}{1}$ (the number of ways to leave a character out of the set of used characters) times $R_l(a-1)$ (the number of ways to then make a password that uses all of the remaining characters). Similarly, there are $\binom{a}{2}R_l(a-2)$ ways to leave out exactly two of the characters, and $\binom{a}{3}R_l(a-3)$ ways to leave out exactly three.[1]

Extending the above logic, we see that

$$R_l(a) = a^l - \sum_{k=1}^{a-1} \binom{a}{k} R_l(a-k)$$

Our necessary starting conditions are that $R_l(1) = 1$, seeing as how there is only length l password using a single given character. Let us additionally

[1]All the preceding examples assume a is large enough to avoid calling R_l with a negative argument.

define $R_l(0) = 0$, because that seems like how it should be.[2] Additionally, we can adjust the sum (and use the fact that $\binom{a}{k} = \binom{a}{a-k}$) to reach

$$R_l(a) = a^l - \sum_{k=0}^{a-1} \binom{a}{k} R_l(k)$$

3 Induction

Our notation $R_l(a)$ (letting l be a subscript) is very suggestive of what we are about to do: fix a over certain values and let l very. Using our definition, we get the following sequence

$$R_l(1) = 1$$

$$R_l(2) = 2^l - 2$$

$$R_l(3) = 3^l - 3 \cdot 2^l + 3$$
$$R_l(4) = 4^l - 4 \cdot 3^l + 6 \cdot 2^l - 4$$

$$= 1 \cdot 4^l - 4 \cdot 3^l + 6 \cdot 2^l - 4 \cdot 1^l + 1 \cdot 0^l$$

If you can recognize binomial coefficients, you will notice a pattern beginning to emerge, as highlighted by the final line above. Define the function

$$C_l(a) = \sum_{j=0}^{a-1} (-1)^j \binom{a}{j} (a-j)^l$$

which extends the pattern recognized above. We now use induction to prove that $R_l(a) = C_l(a)$.

First, note that $R_l(1) = C_l(1)$. Next, assume that $R_l(m) = C_l(m)$ for $m = 1 \ldots n$. Then,

$$R_l(n+1) = (n+1)^l - \sum_{k=1}^{n} \binom{n+1}{k} C_l(k)$$

$$= (n+1)^l - \sum_{k=1}^{n} \left\{ \binom{n+1}{k} \sum_{j=0}^{k-1} \left[(-1)^j \binom{k}{j} (k-j)^l \right] \right\}$$

$$= (n+1)^l - \sum_{k=1}^{n} \sum_{j=0}^{k} \left[(-1)^j \binom{n+1}{k} \binom{k}{j} (k-j)^l \right]$$

[2] We could have started by defining $R_l(0) = 0$, and then deriving $R_l(1)$. However, it seems very unnatural to ground your answers on a password that uses an alphabet of zero characters.

Now we must change the order of integration so that we can group together where $(k-j)$ is constant. The double sum runs over all pairs (k, j) where $k = 1...n$ and correspondingly $j = 0...k-1$. This corresponds to a triangle of points with integer coordinates. We make the substitution $c = k - j$, and retain j as our second indexing variable. Then our sum runs over $c = 1...n$ and $j = 0...n-c$. Redoing the sum, making the substitution, and expanding the binomial coefficients, we obtain

$$R_l(n+1) = (n+1)^l - \sum_{c=1}^{n}\sum_{j=0}^{n-c}\left[(-1)^j \binom{n+1}{j+c}\binom{j+c}{j}(c)^l\right]$$

$$= (n+1)^l - \sum_{c=1}^{n}\left\{c^l \sum_{j=0}^{n-c}\left[(-1)^j \frac{(n+1)!}{(n+1-(j+c))!j!c!}\right]\right\}$$

$$= (n+1)^l - \sum_{c=1}^{n}\left\{c^l \frac{(n+1)!}{c!}\sum_{j=0}^{n-c}\left[(-1)^j \frac{1}{(n+1-(j+c))!j!}\right]\right\}$$

Now, to simplify the innermost sum, consider the binomial expansion

$$0 = (1-1)^{n-c+1} = \sum_{j=0}^{n-c+1}(-1)^j \binom{n+1-c}{j}$$

$$= \sum_{j=0}^{n-c+1}(-1)^j \frac{(n-c+1)!}{(n+1-(c+j))!j!}$$

By separating the last term from the sum, then solving for the rest of the sum, we see

$$\sum_{j=0}^{n-c}(-1)^j \frac{(n-c+1)!}{(n+1-(c+j))!j!} = -(-1)^{n-c}\frac{(n-c+1)!}{(0)!(n-c+1)!} = (-1)^{n-c+1}$$

Now we return to $R_l(n+1)$, and multiply in a factor of $(n-c+1)!$ to the innermost summand. We then simply, using binomial coefficients and the sum above, to obtain

$$R_l(n+1) = (n+1)^l - \sum_{c=1}^{n}\left\{c^l \frac{(n+1)!}{c!(n+1-c)!}\sum_{j=0}^{n-c}\left[(-1)^j \frac{(n-c+1)!}{(n+1-(j+c))!j!}\right]\right\}$$

$$= (n+1)^l - \sum_{c=1}^{n}\left\{c^l \binom{n+1}{c}\cdot(-1)^{n-c+1}\right\}$$

$$= \sum_{c=1}^{n+1}(-1)^{n-c}\binom{n+1}{c}c^l$$

$$= \sum_{i=0}^{n+1} (-1)^i \binom{n+1}{i} (n+1-i)^l = C_l(n+1)$$

Thus we arrive via induction upon the result that $R_l(n) = C_l(n)$ for all $n \geq 1$.

4 Brute Force Verification

The following code segment in C produces a function that runs through all possible passwords and counts the number of those that use every available character. We don't really feel like documenting this code right now, but you can check it if you want. I guess the comments are pretty good. Anyway, it agreed $R_l(a)$ for many values of l and a.

```
int combsBrute( int a, int l) {
        //tests all password and counts up the ones that use every character
        //passwords are represented by an array of integers in the set {0...a-1}
        //thus having a characters
        int ar[l];
        for( int i=0; i<l; i++)  ar[i]=0;
        int numb = 0;
        while ( increment(ar,a,l) ){
                numb+= arrayContainsTo( ar, l, a);
        }
        return numb;
}
int arrayContainsTo( int ar[], int len, int base){
        //test if ar contains all the numbers in 0...base-1
        for(int i=0; i<base && i<base; i++)
                if ( !arrayContains(ar, len, i) )
                        return 0;
        return 1;
}
int arrayContains( int ar[], int len, int test){
        //Test if ar contains a particular number test
        int i;
        for( i=0; i<len; i++)
                if( ar[i] == test)0
                        return 1;
        return 0;
}
int increment( int ar[], int base, int len ){
        //treat ar as a little endian number in the specified base
        //increment it and return 0 if the number overflows to more than len places
        int i=0;
        while( ar[i]==base-1 ){
```

```
                        ar[i]=0;
                        i++;
                        if( i==len )
                                return 0;
                }
                ar[i]++;
                return 1;
        }
```

5 Numerical Values

Here is a table of a few values, as agreed upon by both the above brute force code and $C_l(a)$, with values of a on the left and l on the right. If you are a hacker who is somehow in the extremely ridiculous situation described, this is what you came for I guess.

	1	2	3	4	5	6	7	8	9	10
1	1	1	1	1	1	1	1	1	1	1
2		2	6	14	30	62	126	254	510	1022
3			6	36	150	540	1806	5796	18150	55980
4				24	240	1560	8400	40824	186480	818520
5					120	1800	16800	126000	834120	5103000
6						720	15120	191520	1905120	16435440
7							5040	141120	2328480	29635200
8								40320	1451520	142558344
9									362880	1103271435
10										213161150

Before you leave, be sure to appreciate how the values in this table are $a!$ along the diagonal. It would be pretty interesting to know what the maximizing value of a is for a given l, I guess. There doesn't seem to be a super regular pattern for that.