Math 222: Check Digits - an Application of Non-abelian Groups

Many of things we deal with in life (including ourselves!) are encoded with an identification number. For example, social security numbers, drivers license numbers, serial numbers on dollar bills, UPC codes on everything we buy, etc. Such identification numbers often contain a *check digit*. A check digit is a number (or sometimes numbers) added in some prescribed way to the end of an identification number. Indeed, most UPC codes have one.

Example: Suppose 3221457 was the identification number of something. A simple way to assign a check digit, c, is to simply let c be the sum of the digits modulo 10. e.g. $3+2+2+1+4+5+7 = 24 = 4 \bmod 10$. So, we would take $c = 4$, and the identification number would then be coded as 32214574. The point being that only the first seven digits are the "actual" ID number.

Question: What's the point?

There are at least two reasons for doing this. First, this is a way of detecting forgeries. If you tried to make up a random ID without knowing about the check digit, there's basically a 1 in 10 chance that you'll get a valid number. Note: if you used instead two check digits, this would be say 1 in 100 and so on. Of course, we don't want things to get too long. A second reason is simply to catch transcription errors. If the ID number is frequently copied, the check digit will hopefully catch a typing mistake made by someone. Can you think of any other reasons?

Question: What makes a "good" check digit scheme?

This depends on your perspective. If you're trying to detect counterfeiting, at some level, any system is as good as any other. Although, if the system is too simple, like the one above, it may be easy to figure it out - assuming the counterfeiters are aware of the check digit. On the other hand, if you're trying to avoid transcription errors, then you would like to have the system catch "common" mistakes. One such mistake is simply mistyping one of the numbers. e.g. you might have typed 32224574 for the above. Your computer could be programmed to beep at you because the sum of the first 7 digits is not 4 mod 10! Another common error is to transpose two adjacent digits, e.g. you might have typed 32215474. However, this is still a valid number and it wouldn't get caught.

With this in mind, the above system is good because it's easy to compute c and it will catch any single digit error. But, it's not so good because it could be easily figured out and it won't catch transposition errors.

Question: Can we do any better?

The answer is yes. Indeed, there are numerous strategies for assigning check digits, we are going to consider a slight modification of one developed by a German mathematician J. Verhoeff. His system was actually used for serial numbers on German money. The essence of his idea is very simple - to use the non-commutativity found in some groups to catch transposition errors.

In practice, we have to do a little work. First, since we're dealing with 10 digits, let's find a group of order 10. The dihedral group D_5 you suggest. Good idea. In fact, for various reasons, this is the only one that works. We write down the multiplication table for D_5:

D_5	R_0	R_1	R_2	R_3	R_4	F	R_1F	R_2F	R_3F	R_4F
R_0	R_0	R_1	R_2	R_3	R_4	F	R_1F	R_2F	R_3F	R_F
R_1	R_1	R_2	R_3	R_4	R_0	R_1F	R_2F	R_3F	R_4F	F
R_2	R_2	R_3	R_4	R_0	R_1	R_2F	R_3F	R_4F	F	R_1F
R_3	R_3	R_4	R_0	R_1	R_2	R_3F	R_4F	F	R_1F	R_2F
R_4	R_4	R_0	R_1	R_2	R_3	R_4F	F	R_1F	R_2F	R_3F
F	F	R_4F	R_3F	R_2F	R_1F	R_0	R_4	R_3	R_2	R_1
R_1F	R_1F	F	R_4F	R_3F	R_2F	R_1	R_0	R_4	R_3	R_2
R_2F	R_2F	R_1F	F	R_4F	R_3F	R_2	R_1	R_0	R_4	R_3
R_3F	R_3F	R_2F	R_1F	F	R_4F	R_3	R_2	R_1	R_0	R_4
R_4F	R_4F	R_3F	R_2F	R_1F	F	R_4	R_3	R_2	R_1	R_0

Now, to the elements in D_5, assign the digits 0 thru 9 in the order above so that we get the following multiplication table for the digits 0 thru 9:

D_5	0	1	2	3	4	5	6	7	8	9
0	0	1	2	3	4	5	6	7	8	9
1	1	2	3	4	0	6	7	8	9	5
2	2	3	4	0	1	7	8	9	5	6
3	3	4	0	1	2	8	9	5	6	7
4	4	0	1	2	3	9	5	6	7	8
5	5	9	8	7	6	0	4	3	2	1
6	6	5	9	8	7	1	0	4	3	2
7	7	6	5	9	8	2	1	0	4	3
8	8	7	6	5	9	3	2	1	0	4
9	9	8	7	6	5	4	3	2	1	0

The naive idea: For simplicity, let's assume we have a two digit code number, say ab. The same idea will work for any length. A naive way to assign the check digit c is to let $c = a \cdot b$, that is to let c be the product of a and b (or in general of all the digits in the ID number) as given by multiplication in D_5.

For example, we would get the following ID numbers with attached check digit: 123, 257, 443, 573, 784. But, sequences such as 864, 437, 124, ... would not be valid. Notice that any of the three digits a, b, and c are uniquely determined from the other two by the relationship $a \cdot b = c$. Hence, if we started with a valid code and then changed one of the three numbers, we would get an invalid code. e.g. 342 is valid, but 742, 312, and 348 are all invalid (among others).

Plus, this has an extra bonus. Some transposition errors will be caught. For example, 761 is a valid code, but 671 is not. Unfortunately, 716 is valid. The point is that some pairs of elements commute in D_5 and some don't.

Question: Can we fix this?

Yes. And this is really Verhoeff's idea. Notice that the multiplication table breaks up into four squares. His idea was to make use of this along with the non-commutativity of some elements to devise a scheme in which all transposition errors are caught.

The trick is to use a permutation of the 10 digits {0,1,2,3,4,5,6,7,8,9}. Specifically, we consider the permutation $\sigma = (07249851)(36)$. What's so good about σ? It has the following

two properties:

Fact: Suppose $a \neq b$. Then

(1) $\sigma(a) \neq \sigma(b)$
(2) $\sigma(a) \cdot b \neq \sigma(b) \cdot a$

The first fact is easy to see, the second is the one that makes things work and is much harder to show, but you might check it in a few cases.

The Assignment: To a code ab we assign the check digit c to be the number which satisfies

$$\sigma^2(a) \cdot \sigma(b) \cdot c = 0 \qquad \text{or} \qquad c = (\sigma(a)^2 \cdot \sigma(b))^{-1}.$$

For longer codes, you can extend this in the obvious way. e.g. to a code abc we would assign the check digit d to be the number which satisfies

$$\sigma^3(a) \cdot \sigma^2(b) \cdot \sigma(c) \cdot d = 0.$$

Example: Find the check digit for the code 13.

We compute $\sigma^2(1) = 7$ and $\sigma(3) = 6$. So c should satisfy $7 \cdot 6 \cdot c = 0$. Using the multiplication table, $7 \cdot 6 = 1$ and so we need $1 \cdot c = 0$. That is $c = 1^{-1} = 4$. Hence, this would be coded as 134.

Theorem: This method will detect all single digit errors and all adjacent transposition errors.

proof: See if you can prove this, using the two facts above.

In other words, for a two digit code, if abc is valid, bac and acb will be invalid (unless $a = b$ or $b = c$).

For example, we saw that 134 was a valid code. Let's just check that 314 and 143 are both invalid. In the first case, $\sigma^2(3) = 3$ and $\sigma(1) = 0$, so we would need $3 \cdot 0 \cdot 4 = 0$, but it equals 2. In the second case, $\sigma^2(1) = 7$ and $\sigma(4) = 9$, so we would need $7 \cdot 9 \cdot 3 = 0$, but it equals 1.

Problems:

1) Find the check digit for the following ID numbers:

(a) 27 (b) 355 (c) 8329 (d) 3221457

2) The following hypothetical ID numbers have the check digit included. Which of these are valid?

(a) 750 (b) 682 (c) 3333

Answers:

1) 7, 5, 3, 6

2) a, c