The Little Book of Numbers

John M. Morrison

August 12, 2020

Contents

0 Numbers and their Representation **1**

1 Denominational Numbering Systems **2**

2 Base Numbering Systems **4**

3 Time for a Powerful Observation **6**

0 Numbers and their Representation

Suppose we have two finite collections of objects A and B. What does it mean for them to have "the same number of elements?" Can we determine this without counting them? The answer to this question is, "yes." The test works as follows. Pair one element from each set until you have no leftovers in one of the sets. If the other has no elements too, the sets have the same size. This is called creating a *one-to-one correspondence* or a *bijection* between the two sets. We say that such sets have the same number of elements.

We see that we can do this with any two finite sets of items. This is the basis for the idea of number. We declare two sets to be "of the same size" if they pass this test of having a one-to-one correspondence between them.

Separate from this is the process of *representing* numbers with symbols. Let us discuss several ways of representing numbers. The simplest is just with tally marks. You make one tally mark for each item present. To count the number of eggs in a standard carton, you would make the symbol

```
IIIIIIIIIII
```

These symbols have some advantages. Addition is easy. Just glue together (concatenate) the globs of `I`s. It is pretty easy to multiply too. If you think for a minute or two, you can devise a very simple scheme for doing this. However, the tally system rapidly becomes cumbersome if you want to keep track of hundreds of items. It is simple but bulky, error-prone and cumbersome. It would work for a pre-agricultural society that only keeps track of a few things at a time.

Project Write a short manual for doing arithmetic with tally marks. Be aware that there is no notion of zero and there are no negative numbers so an operation such as `II - IIII` is not defined.

Tell how to add, subtract (where allowed), and multiply numbers using tally marks.

Let us now add two new operations. The operation of `integer division` consists of dividing and discarding the remainder. We represent this with `//`. For example, in decimal numbers `5//2 = 1`.

The other operation is called *modulus* and it is represented with `%`. The expression `b % a` is evaluated by dividing `a` into `b` and computing the remainder. Example: `5 % 2 = 1`.

Tell how to do these in tally marks as well. Your explanation should pass the *Man on the Moon Test*: The man on the moon is reasonably intelligent but he does not know how do do your specific procedure. Explain it to him in such a way that he can read your little article and carry out the procedures in it without asking you questions.

This is the same standard you use for writing out procedure in a standard science lab report.

1 Denominational Numbering Systems

Why don't we use tally marks if the arithmetic is so simple? We could add zero and we could even add signed numbers. These would not cure the primary drawback of the system: representing large numbers is very cumbersome. Imagine representing the number 300 with tally marks. It looks like this

```
IIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIII
IIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIII
IIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIII
IIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIII
IIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIII
```

You cannot tell "at a glance" what number this is. But you can in our numbering system. So the next step is to create symbols for groups of tally

marks. The Romans did this; such a system is called a *denominational* number system. They had the alphabet {`I, V, X, L, C, D, M`} of symbols and gave them numerical values. These represent, respectively, the values 1, 5, 10, 50, 100, 500, and 1000 in our number system. For example, the number 300 is represented in Roman numerals by `CCC`. Each `C` is "worth" 100 tally marks.

In addition, They also used a borrowing convention. For example, to represent the number 9, they used IX (borrow 1 from 10). Arithmetic in this system is pretty clunky. With a little imagination, you can figure out how to add and multiply. If you think that the borrowing procedure you learned from Miss Wormwood is annoying, think about borrowing with Roman numerals.

The Mayans created a denominational number system based on the numbers 1, 5, and 20. The numbers 1–4 are represented by a row with that number of dots. A 5 is represented by a horizontal bar. The numbers 6–9 are represented by a bar (5) underneath an appropriate number of dots. Then 10 is two horizontal bars. Bars are stacked and dots are put on top to represent numbers less than 20. The Mayans had a zero too; it looked a little like a football. You can see more detail here [1].

Notice that modern currencies use a denominational number system, so you do not have to carry a Kansas City roll of bills to pay for things.

Exercises

1. Convert the Roman Numeral `MCMLVII` to a decimal number.
2. Convert the number 4096 to a Roman Numeral.
3. Convert your birth year and this year into Roman Numerals. See if you can subtract them directly using only Roman numerals and get your correct age.
4. Write a procedure to decide if a string of characters is a valid Roman numeral.
5. Write down an procedure that tells how to count out n dollars using as few bills as possible. Remember US currency comes in these denominations: 1,2,5, 10, 20, 50, and 100. See if you can describe it in everyday English in a very simple way.
6. The method you just created would be described as "greedy." Why?
7. **This one is very important!** Imagine you are in the land of Binaria where currency is called a bit and it comes in an infinite number of denominations that are powers of two. What is a recipe for counting out n bits using as few bills as possible? Do you ever use more than one bill of any given denomination?

Notes

1. The article [2] discusses the history of our modern numbering system.
2. The article [3] discusses the history of numeration systems and gives a nice bibliography of further references.

2 Base Numbering Systems

First, it's time for a trip back to grade school to remember how we *parse*, or impose meaning upon, our familiar decimal numbering system. When you see the number 256, you do not read it as a sequence of characters 2, 5, 6; it is something more. The decimal numbering system is based on *place value*; a

digit's place in a number determines its contribution to the whole number. The number 256 is a *token* because it is an atom of meaning: If we break it into smaller pieces, such as 25 and 6, it loses its original meaning.

When we see 256, we learned in grade school that the 6 is in the "ones place", the 5 is in the "tens place" and the 2 is in the "hundreds place." To wit,

$$256 = 6 + 5 * 10 + 2 * 100.$$

When reading a number, the last digit has place value one. As you move to the left, the place value of a given digit's value goes up by a factor of 10. This is the familiar *decimal* system of numbers.

Why 10? The answer is simple: humans have 10 fingers so the number 10 is a basic unit of counting. The number 10 is called the *base* of our numbering system. The *alphabet* of our numbering system consists of the digits 0-9. You can learn about the history and origins of this alphabet of symbols in this article [2].

What is interesting to note is that there is nothing special about 10. You could use any positive integer larger than 1 as a base and arithmetic works in a manner entirely similar to that of decimal arithmetic.

Money! A base numbering system is actually a denominational numbering system that has an infinity of denominations: these are just powers of the base. Let us see how this works. Suppose we want to convert the number 257 to base 3.

Our "currency" comes in denominations 1, 3, 9 27, 81, 243, 729,... What we do is get greedy and choose the largest bill we can use first and use as many of them as we can. In that case, it's 243 and we can use one. Subtracting, we have 14 left to count out. This means that 81 and 27 are no good, so we go to 9 and use one of those. That leaves us 4 to count out. We can do that with a 3 and two 1s. Now we make this little table that shows us how many of each bill we used.

243	1
81	0
27	0
9	1
3	1
1	2

Read down the second column and you can see the base 3 representation for the number 257. So $257 = 100112_3$. Observe that at any step, you never use more than two of these "bills."

There is nothing special about 3; this greedy method works for any base. This procedure is greedy in this sense: Start using the largest bill possible and use as many of them as you can. Keep doing this until you have counted out the amount.

The greedy method is an example of a precise procedure for carrying out a computational task. Such procedures are called *algorithms*. You can think of an algorithm as a precise computational recipe. Algorithms need to have these three characteristics to work correctly.

- **Repeatability** The instructions you furnish must be precise and the user must be able to carry them out without your intervention. Note that this user might be an unthinking machine! You must also tell what the allowable inputs and expected outputs are.
- **Finite Resource Restriction** You may only use a finite amount of memory (which can be paper) to store data. Each datum only has a finite size.
- **Termination Condition** The algorithm must halt after a finite number of steps.

Even with its very simple formulation the greedy algorithm meets these three requirements. The greedy algorithm for base conversion works for any base.

3 Time for a Powerful Observation

One disadvantage to binary numbers is that they get big fast. For example the number of feet in a mile is 5280. In binary this is `0b1010010100000`. Now a computer does not care about this clunkiness but humans do. The decimal representation has the advantage of being shorter. The fly in the proverbial ointment is that it takes some work to convert back and forth. So, here is the lurking question: *Can we find a way to compactly represent binary numbers so it is easy to convert them back to binary?*

That is what we set out to do here. We are going to take a look at three number bases, 4, 8, and 16. Let us take the number 1225 and convert it to each base. One problem we have is that we only have the alphabet of digits 0-9 and, if we wish to use 16 as a base, we must extend this to an alphabet of 16 symbols. We take the cheap way out; the alphabet for base 16 is {`0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F`}. We let `A` be the symbol for 10, `B` be the symbol for 11, etc. Let us begin with base 2. We organize this into a little table. Column 1 is the denominations, Column 2 is how many you use of the denominations, Column 3 is what's left to count out.

bill	number	left
1024	1	201
512	0	201
256	0	201
128	1	71
64	1	7
32	0	7
16	0	7
8	0	7
4	1	3
2	1	1
1	1	0

We have $1225 =$ `0b10011001001`.

Next, let's try this with 4

bill	number	left
1024	1	201
256	0	201
64	3	9
16	0	9
4	2	1
1	1	0

We have $1225 = 103021_4$. Look at the base-4 digits and their binary representations

0	1	2	3
00	01	10	11

There are four base-4 digits and four possible combinations of two bits. This is no accident. Each base-4 digit translates into a block of two bits. Now check this out. Translate each digit of the base-4 representation of 1225 into a two-bit block.

```
 1   0   3   0   2   1
01  00  11  00  10  01
```

Now glue those blocks together and trim off any lead 0s.

```
 1   0   3   0   2   1
01  00  11  00  10  01
010011001001 -> 10011001001
```

BAM! This representation is half as long as its binary sibling. Now let's try this with 8. First calculate the base-8 representation of 1225 directly using the greedy algorithm.

bill	number	left
512	2	201
64	3	9
8	1	1
1	1	0

We have $1225 = 2311_8$, where the subscript indicates this is a base-8 number. Before we go any further, let us make a side-by-side comparison. Now let us attempt to turn the same crank as we did for base 4. To do this, realize that every base-8 digit turns into a block of *three* binary digits.

0	1	2	3	4	5	6	7
000	001	010	011	100	101	110	111

We forge ahead and try the same trick.

```
   2    3    1    1
 001  011  001  001   -> 1011001001
```

Et voila! It all works becasue $2^2 = 4$ and $2^3 = 8$.

This is the the *packing-unpacking* algorithm. Happily, since $2^4 = 16$, every base-16 digit unpacks into four bits. This table tells all.

0	1	2	3	4	5	6	7
0000	0001	0010	0011	0100	0101	0110	0111
8	9	A	B	C	D	E	F
1000	1001	1010	1011	1100	1101	1110	1111

We use the term *hexadecimal* for base 16 numbers, or "hex" for short. We use the term *octal* for base-8 numbers. Octal numbers are designated on computers with the prefix `0o` and hex numbers are designated by `0x`.

Hex numbers are extremly common in computing. You will see the when you learn about colors and how they are represeneted in a computer. You will also see them if you are discussing memory addresses.

Exercises

1. See if you can complete this table. If you can, you have this business well at hand.

binary	base4	octal	decimal	hex
0b11101101				
	21230_4			
		0o1236		
			233	
				0xA51

2. Write down an algorithm for translationg a hex number to decimal. Is there more than one way?
3. Fill in the blanks. Each question mark stands for a single digit.

$$0x?4 = 124?_5 5.$$

4. How can you use `%` to obtain the last digit of a number in any base?
5. 4. **A Balanced Ternary Expansion** It is possible to uniquely write any integer in an expansion of the form

$$e_0 + 3e_1 + 3^2 e_2 + \cdots + 3^n e_n,$$

where each of the e_k are 0, 1 or -1. Find such an expansion for each of the folllowing numbers: 5, 13, 37 and 79. Can you describe an algorithm for doing this in general?

References

[1] Luke Mastin. The story of mathamatics, 2010. `http://www.storyofmathematics.com/mayan.html`.

[2] `http://en.wikipedia.org/wiki/Arabic_Numerals`. Arabic numerals. *Wikipedia*, 2013.

[3] `http://en.wikipedia.org/wiki/Numeral_system`. Numeral system. *Wikipedia*, 2013.