CS29002 Algorithms Laboratory

Warm-Up Assignment (Not for Evaluation)

Last date of submission: 20–July-2016

This programming exercise demonstrates the philosophy: *Think of the algorithm before writing the code.* A PDS-style program may run for hours (or even for your entire lifetime). Only if you carefully look into the problem, understand its inner details, devise a suitable algorithm, and carefully structure your data, you can come up with a program which finishes in a decent time. So here you go.

Part 1

Let $N = n_0 n_1 n_2 \ldots n_9$ be a ten-digit decimal number with each digit $n_i \in \{0,1,2,\ldots,9\}$. The number N has the property that n_0 is equal to the number of zero digits in N, n_1 is equal to the number of one's in N, n_2 is equal to the number of two's in N, …, and n_9 is equal to the number of nine's in N. Let us call such an N a *digit-counting number*. Your task is to find *all* digit-counting numbers.

A PDS-style code would make an exhaustive search over all ten-digit numbers ($0,000,000,000$ through $9,999,999,999$). There are 10^{10} of them, and many of them do not fit in a 32-bit integer even if unsigned (2^{32} is slightly larger than four billion), so you need to use 64-bit integer arithmetic. The resulting program would take several hours to several days depending on how efficiently you can code this naïve algorithm.

You need to think about the problem to see how the search space can be reduced. First, notice that n_0 cannot be zero, because $n_0 = 0$ implies that there is no zero in N, whereas you have already put a 0 as n_0. So you may search among the numbers $1,000,000,000$ through $9,999,999,999$, leading to a reduction of the search space by 10%. But that gain is marginal. You need to do more.

Since the total number of digits in N is ten, you must have $n_0 + n_1 + n_2 + \cdots + n_9 = 10$. Numbers not satisfying this property are not eligible to be digit-counting. For example, once you are done with $1,234,000,000$, the next numbers you should try are $1,240,000,003$, $1,240,000,012$, $1,240,000,021$, $1,240,000,030$, and so on. This is a significant saving of time.

But then, how do you code this? This calls for a careful structuring of the data. You actually do not need N. You need its digits only. So maintain an array of ten digits. Finally, how do you step through the eligible numbers? A good solution is to use backtracking that can be implemented recursively. It is not mandatory to generate the eligible numbers in the increasing order of values. It only suffices that no eligible number is missed out.

Now, you go ahead, and write your code. A good program using these observations is expected to solve your problem in a few seconds.

Part 2

Generalize Part 1 to an arbitrary base. That is, take a base b, and express N as a b-digit number in base b:

$$N = (n_0 n_1 n_2 \ldots n_{b-1})_b,$$

where each n_i is in $\{0,1,2,\ldots,b-1\}$ and counts the number of times the digit i occurs in the base-b representation of N. Your program should be able to handle bases as large as $b = 16$.

[redacted]