Computational Thinking (CO2412): Tutorial – Calendar Week 43

Program Analysis

M. Horsch, O. Kerr, School of Psychology and Computer Science

1.3.1. Final digit enumeration problem

In the lecture, we discussed an iterative algorithm and its Python implementation, called `mod10_count_naive()` in the associated Jupyter Notebook,[1] for the following problem:

The input consists of two arguments, a list $\mathbf{x} = [x_0, x_1, \ldots, x_{n-1}]$ of $n =$ `len`($\mathbf{x}$) integer numbers, where multiple elements are allowed to have the same value, and a single-digit integer y with $0 \leq y \leq 9$. A list $[q_1, q_2, q_3]$ is returned where:

1. q_1 is the number of indices i such that x_i has y as its final digit. Differently expressed, it is the number of list elements such that $x_i \bmod 10 = y$, where mod stands for modulo, *i.e.*, remainder after division.[2] If the same number occurs multiple times in the list, it also counts multiple times, once for each index.
2. q_2 is the number of ordered pairs (i, j) of indices, with $i \neq j$, such that $x_i x_j \bmod 10 = y$; *i.e.*, y is the final digit of $x_i x_j$. The two different ways of arranging the indices, (i, j) and (j, i), both count separately – therefore, q_2 is always an even number. Note that the requirement is for i and j to be different, not x_i and x_j.
3. q_3 is the number of ordered triples (i, j, k) of indices, all different from each other ($i \neq j$, $i \neq k$, $j \neq k$), for which $x_i x_j x_k \bmod 10 = y$. As above, all the different permutations (*i.e.*, arrangements) of the three indices each count separately, of which there are six each time; accordingly, q_3 is always divisible by 6.

For example, if $\mathbf{x} = [24, 8, 19, 8, 2]$ and $y = 4$, the list $[1, 2, 24]$ needs to be returned.[3] The `mod10_count_naive()` code solves this problem, but it has $\mathsf{O}(n^3)$ time requirements, by which it does not perform very favourably for long lists.

a) Propose a more efficient algorithm and develop a more performant code.
b) Of what order is the time efficiency of your algorithm, using Landau notation (*i.e.*, "big O notation")? Provide a brief justification similar to those from the lecture.

---

[1] For the notebook, *cf.* `https://home.bawue.de/~horsch/teaching/co2412/material/iterative-algorithms.ipynb`.

[2] In Python, this condition is expressed by `x[i] % 10 == y`.

[3] $q_1 = 1$ for $x_0 = 24$, $q_2 = 2$ for $x_1 x_3 = x_3 x_1 = 64$, and $q_3 = 24$ for $x_1 x_2 x_4 = x_1 x_4 x_2 = x_2 x_1 x_4 = x_2 x_3 x_4 = x_2 x_4 x_1 = x_2 x_4 x_3 = x_3 x_2 x_4 = x_3 x_4 x_2 = x_4 x_1 x_2 = x_4 x_2 x_1 = x_4 x_2 x_3 = x_4 x_3 x_2 = 304$, in combination with $x_0 x_1 x_4 = x_0 x_3 x_4 = x_0 x_4 x_1 = x_0 x_4 x_3 = x_1 x_0 x_4 = x_1 x_4 x_0 = x_3 x_0 x_4 = x_3 x_4 x_0 = x_4 x_0 x_1 = x_4 x_0 x_3 = x_4 x_1 x_0 = x_4 x_3 x_0 = 384$.

c) Conduct performance measurements, including but not necessarily limited to the two demo lists `x200` and `x1000` from the notebook,[4] with $n = 200$ and 1000, respectively. What is the ratio between the two runtimes? For the naive implementation, which scales with $\mathsf{O}(n^3)$, it is close to $125 = (1000/200)^3$; for a code that has an asymptotic runtime in $\mathsf{O}(n^m)$, a ratio close to 5^m should be expected.

1.3.2. Number matching problem

The function `natmatch_iter()` takes two arguments: First, a list of k integer numbers $\mathbf{x} = [x_0, x_1, \ldots, x_{k-1}]$, and second, a natural number y; it determines whether there is a match, here defined by the existence of two list elements with $x_i + x_j = y$, where $x_i \neq x_j$.

In the present and the previous notebook, we were calling this function for a given value of k many times, where the k elements of the list $\mathbf{x}$ were assigned new random values each time, using a uniform random distribution[5] over all integers from 0 to $k^2 - 1$. The second argument was given by $y = k^2$. Statistics from these function calls make it apparent that for large values of k, a match is found in about 39% to 40% of the cases.

Determine the fraction of cases for which there is a match, in the case of large k (ideally, as k approaches infinity), as accurately as possible.[6]

Submission deadline: 13th November 2021; discussion planned for 25th November 2021. Group work by up to four people is welcome.

[4] For validation, the return value for $\mathbf{x} =$ `x200`, $y = 7$ should be `[28, 1528, 134610]`, and for $\mathbf{x} =$ `x1000`, $y = 7$ it should be `[105, 42660, 17483370]`.

[5] That is, each integer from 0 to $k^2 - 1$ had the same probability of being assigned to any of the list elements.

[6] The method suggested here is to run a large number of function calls with random input for a large value of k, by which a sufficient accuracy should be reached. With some mathematical knowledge, going beyond the scope of this module, is also possible to give an exact answer; note, however, that here you are not expected to do this (of course, any such solutions or attempts are nonetheless very welcome).