Problem 71. Find the neighbouring fraction

Our task is to find the largest (reduced) fraction less than $t = \frac{a}{b}, 0 < t < 1$, whose denominator does not exceed N.

The most naïve method to achieve that goal is to generate all fractions $\frac{p}{q} < t$ with $q \leqslant N$, discard those which aren't reduced and scan the remaining for the maximum. But the number of fractions one would have to compare using this method is huge. There are $\frac{N(N-1)}{2}$ proper fractions whose denominator doesn't exceed N, approximately $0.3 \cdot t \cdot N^2$ of them are reduced and smaller than t. For the problem, where $t = \frac{3}{7}$ and $N = 1,000,000$, one would have to check about 2.1×10^{11} fractions for reducedness and scan roughly 1.3×10^{11} for size, that would take considerably longer than one minute.

However, a moment's thought frees us from most of that work. Clearly, for each denominator q, we need only consider one fraction, the one with the largest numerator p for which $\frac{p}{q} < \frac{a}{b}$ holds. That numerator can easily be determined:

$$\frac{p}{q} < \frac{a}{b} \iff p \cdot b < q \cdot a \iff p \cdot b \leqslant q \cdot a - 1 \iff p \leqslant \left\lfloor \frac{q \cdot a - 1}{b} \right\rfloor, \tag{71.1}$$

so if we set $\nu(q) := \left\lfloor \frac{qa - 1}{b} \right\rfloor$ for $q \leqslant N$, then $\frac{\nu(q)}{q}$ is the largest fraction with denominator q below t.

Now we can step through the admissible denominators, remembering the largest fraction found so far. In pseudocode:

```
bestNum   := 0
bestDenom := 1
for currDenom := 2 to N
    currNum   := (a*currDenom - 1) div b
    if bestNum*currDenom < currNum*bestDenom then
        bestNum   := currNum
        bestDenom := currDenom
    end if
end for
output bestNum, bestDenom
```

As the code stands, it is imaginable that the maximum isn't reported in reduced form, so unless one proves that it always is, one should reduce the reported fraction.

Author: daniel.is.fischer

This way, we only consider $N - 1$ fractions and we can find the maximum for denominators up to several million within a second.

Still, since fractions with large denominators are spaced more closely than fractions with small denominators, we can expect the maximal fraction to have a fairly large denominator, so we should find it faster if we only consider the fractions with the largest denominators, going downwards. But that will only speed up our programme if we can soon determine that we have already found the maximum and then exit the loop, otherwise we would have to go the whole way down and gained nothing.

How can we determine that the search is over? We want to break out of the loop as early as possible, so raise the lower end of the loop as high as we can.
Given any fraction $\frac{r}{s} < \frac{a}{b}$, its distance from the target is

$$\frac{a}{b} - \frac{r}{s} = \frac{a \cdot s - b \cdot r}{b \cdot s} \geqslant \frac{1}{b \cdot s}. \tag{71.2}$$

If the maximal fraction found so far is $\frac{p}{q}$, a necessary condition for $\frac{r}{s}$ to be closer to the target is that $\frac{1}{b \cdot s} < \frac{a \cdot q - b \cdot p}{b \cdot q}$, or equivalently $s > \frac{q}{a \cdot q - b \cdot p}$.

So every maximum gives a lower bound for the denominators to consider, in particular if $\delta := a \cdot q - b \cdot p = 1$, every fraction closer to the target must have a denominator larger than q and we can stop immediately, because all denominators from q to N have already been examined. If $\delta > 1$, we have the lower bound $\frac{q}{\delta}$ for the denominator, which still is good if small values of δ are reached soon, but hardly an improvement if δ always is large. Fortunately, the latter does not happen. If you try this algorithm

```
bestNum   := 0
bestDenom := 1
currDenom := N
minDenom  := 1
while currDenom ⩾ minDenom
    currNum := (a*currDenom - 1) div b
    if bestNum*currDenom < currNum*bestDenom then
        bestNum   := currNum
        bestDenom := currDenom
        delta   := a*currDenom - b*currNum
        minDenom  := currDenom div delta + 1
    end if
    currDenom := currDenom - 1
end while
output bestNum, bestDenom
```


Author: daniel.is.fischer

for a number of inputs $\frac{a}{b}$ and $N \geqslant b$, you will notice that it always quickly finds a maximal fraction with $\delta = 1$. This is not a lucky coincidence, if we look a bit more closely at δ, we can prove how fast it *must* find the solution.

As we saw above, for any denominator q the maximal numerator is given by $p = \nu(q) = \left\lfloor \frac{a \cdot q - 1}{b} \right\rfloor$, so $p \leqslant \frac{a \cdot q - 1}{b} < (p + 1)$ and we find

$$1 \leqslant \delta(q) = a \cdot q - b \cdot p < b + 1. \tag{71.3}$$

Now let us look at $\delta(q - 1)$. We have $\delta(q - 1) = a \cdot (q - 1) - b \cdot \nu(q - 1)$ and $\nu(q - 1) = \left\lfloor \frac{a \cdot q - 1 - a}{b} \right\rfloor$ is either $\nu(q) - 1$ or $\nu(q)$, depending on whether the remainder of $a \cdot q - 1$ modulo b is less than a or at least a. If $\nu(q - 1) = \nu(q)$, then $\delta(q - 1) = \delta(q) - a$, otherwise $\delta(q - 1) = \delta(q) + (b - a)$. In both cases $\delta(q - 1) \equiv \delta(q) - a \pmod{b}$ and inductively

$$\delta(N - k) \equiv \delta(N) - k \cdot a \pmod{b} \text{ for all } k \geqslant 0. \tag{71.4}$$

Since a and b are coprime (tacit assumption not used until now) and $N \geqslant b$, there is a k_1, $0 \leqslant k_1 < b$ so that $\delta(N - k_1) \equiv 1 \pmod{b}$. We saw above that $1 \leqslant \delta(q) \leqslant b$, hence indeed $\delta(N - k_1) = 1$.

It remains to show that $\frac{\nu(N - k_1)}{N - k_1}$ is the sought fraction. That is easy to see if $N \geqslant 2b$, because the distance of any fraction with denominator $N - k_1 < q \leqslant N$ to the target is at least $\frac{2}{N \cdot b}$ and then $N - k_1 > N - b \geqslant N/2$. It is harder for $N < 2b$, but remains true[1].

So we have found an algorithm whose running time is basically independent of N (it depends on the remainder of N and a modulo b, but not on the size of N). On average it needs $\left(\frac{b + 1}{2}\right)$ iterations of the loop, b iterations in the worst case and 1 in the best case.

From the fact that the maximal fraction is always $\frac{\nu(N - k_1)}{N - k_1}$ one can derive an even faster algorithm which runs in $O(\log b)$ time. Some pertinent theory will be presented in the overview for problem 73.

[1] However, if $N < b$, it can happen that $\delta(q) > 1$ for all $q \leqslant N$.