Code Jam 2009 - Round 1A

Analysis: Multi-base happiness

How can you tell if a number is unhappy? You can tell if it enters a cycle when you apply the process described in the problem. Here is one example, in base 10:

```
42 → 20 → 4 → 16 → 37 → 58 → 89 → 145 → 42 repeated
```

Naturally, x → y denotes that y is the next number if we apply the process on x. Starting from any number, if you apply the process, eventually you will reach 1 or enter one of those cycles.

But is that really true? A careful reader might ask: Couldn't the process keep hitting new numbers, and never enter a cycle? Look at the cycle for 42: Before you hit 42 again, the numbers just jump around without a clear pattern.

It turns out that this can never happen: that there are only finitely many such cycles, and they're all finite in length. In fact, in base 10, this is the only one! All the numbers involved in such a cycle must be reasonably small. Indeed, when you start a number that is big (think about 99999..9999), applying the process will lead to numbers that become smaller and smaller very rapidly. One can easily prove that, in any base B, there is a threshold H of $O(B^3)$ such that any number larger than H will have a smaller successor.

Another question is: Given a set of bases, does there exist a number that is happy in all of them? We don't know the answer to that question, but based on our computation we know such numbers exist for all bases up to 10. On the other hand, if you intuit that the property of a number being happy is *somehow* random, and *somehow* independent across different bases, then you can believe that multi-base happiness is rare, and that the density of such numbers decreases exponentially with the number of bases. In our problem. The smallest happy number for the input (2, 3, 4, 5, 6, 7, 8, 9, 10) is 11814485; just barely affordable with a brute-force search.

In the computation, one obvious trick is to cache, for a pair (x, B), whether x is happy in base B, so we can avoid following the jumping sequence or the cycles every time. We only need to do this for small values of x -- for example all x ≤ 1000 would be more than enough -- since any 32-bit integer bigger than 10000 becomes smaller than 1000 in one step.
Since the maximum base is 10, one may realize that there are only $2^9 - 10 = 502$ possible distinct inputs. So why don't we just calculate them all? This can actually be faster than solving the input cases one by one, if we solve the smaller sets first. For a set S of bases, we do not need to start the search from 2; we can start from the answer for any $S' \subset S$, since a number happy in all bases from S is at least happy in all bases from S'.

If you think your implementation is still not fast enough. Then run your program while you are solving other problems. There are only 502 possible input cases. Solve all of them, produce the list of answers, and then start the submission process; just don't forget you also need to submit the slow program that produced the list. This is why we had a special note at the bottom of the problem statement.

More Information

Wikipedia article: Happy Numbers