Code Jam 2011 - Round 3

Analysis: Mystery Square

Overview

This problem may have a simple statement, but make no mistake: it is really hard. Unless you have a small army of computers working in parallel, there is no way to try all 2^{40} possible values. You need a way to limit your search.

The key observation is that for the most part, a perfect square is uniquely determined by *both* the first half of its digits, *and* by the second half. Either one suffices. So the high-level idea is to pick the half that has fewer question marks in it, try all possible ways of filling the question marks, deduce what the rest of the number has to be, and then see if it works. Before we get into the details though, let's talk about one nasty little implementation detail that you can't help but notice.

Dealing with big integers

If a binary number has 120 digits, there is obviously no way to fit it into a standard 64-bit integer! And that means arithmetic can be a nuisance. Here are a few ways you can deal with this extra complication:

- Use Java and take advantage of the BigInteger class.
- In g++, you can use the little known __uint128_t type.
- Roll your own BigInteger functions for 128 bits. In practice you only need Square and SquareRoot. The first is doable with grade-school long multiplication formulas and some care. The second can be done with a binary search.
- Use an external library such as GMP.

Hopefully you remembered your Fair Warning from last year, and were prepared! Even so, we would have loved to let you work on only 64-bit integers if we could, but it turns out computers are so fast today that you can simply loop over ALL 64-bit perfect squares in a few seconds. The problem becomes pretty boring in that case.

Filling in a perfect square top-down

All right, so with that detail of the way, let's get down to the solution. If you know the first half of the digits in a perfect square, how can you easily figure out the rest? For example, let's look at 10110????? (in binary).

Notice the square root has to be at least sqrt(1011000000) and at most sqrt(1011011111). In fact, there is only one integer that is between these two, and it is 11011! So we can just see if 11011^2 matches 10110?????, and then we're done. And this always works. If a number X has 2t digits, then its square root Y has t digits, and $(Y+1)^2 = Y^2 + 2Y + 1$, which already differs from Y^2 in more than the last t digits.

So in summary: once we know the first half of the digits in **N**, we can just replace the ? characters with 1, take the square root, round down, and that is the only possible option.

Filling in a perfect square bottom-up

The other half of the solution is not much harder conceptually, but the devil is in the details. If you know the second half of the digits in a perfect square, how can you easily figure out the rest? For example, let's look at ???011001.

Getting started is actually tricky, but let's suppose we have figured out the last two binary digits of the square root are 01.

- The square root must then be 4A + 1 for some integer A. Its square is $16A^2 + 8A + 1 = 8A + 1 \bmod 16$. However, we know that the square is 9 mod 16, and hence A must be odd. Therefore, the square root must end in 101.
- We now know the square root must be 8B + 5 for some integer B. Its square is then $64B^2 + 80B + 25 = 16B + 25 \bmod 32$. However, we know that the square is 25 mod 32, and hence B must be even. Therefore, the square root must end in 0101.
- Continuing in this way, we can use the last k+1 digits of **N** to calculate the last k digits of its square root. If you know just over half of the digits in **N**, this is enough to completely determine the square root. As above, we can now just check if it works, and then we're done.

This technique always works, subject to two condition: (a) **N** must be odd, and (b) you must already know the last two digits of the square root. You might enjoy writing down the formula in both failure cases, and seeing what goes wrong.

Let's think about (b) first. If **N** is odd, then the square root must also be odd, and so the last digit must be 1. There is no easy way to determine what the second last digit has to be in advance, but who cares? Just try both of them, and see which one works!

Next let's suppose **N** is even. Since it is a perfect square, it must actually be a multiple of 4, and **N**/4 is also a perfect square. So we can just cut the last two digits off of **N**, repeat until **N** becomes odd, and then solve as above. In fact, this trick is pretty much required. If **N** is odd, then the last k digits are enough to find k-1 digits in the square root. If **N** is even though, then the last k digits may only be enough to find k/2 digits in the square root.

Of course, we might not know whether **N** is even or odd in advance. If the last digit is a '?' character, then we just try both possibilities and see what happens.

Remark: This whole approach works only because 2 is prime. If we were working base-10, then an even number that is not a multiple of 10 would be pretty nasty to deal with!

Putting it all together

Here is the full solution:

- Assume that **N** is odd, if possible.
- If **N** has more question marks in its bottom half than in its top half, then iterate over all possible ways of filling in the top half question marks, deduce the whole number, and see if it works.
- If **N** has more question marks in its top half than in its bottom half, then iterate over all possible ways of filling in the bottom half question marks, deduce the whole number, and see if it works.
- Now assume that **N** is even, if possible. Fill in the last two digits as zero, and repeat from the very beginning (potentially solving either top-down or bottom-up) for **N**/4.

The parenthetical comment in the last part is actually quite important! Consider the following input for example:
"10010000010000011100000110110010001???????????????????????????????????
000000000??000000000000000?0000000000000000000000??00". Most of the '?' characters are in the first half, so it is tempting to just start from the back and never re-evaluate your decision. However, there are only '0' characters back there, and they will not give you much information. To run fast enough in this case, you need to start from the front after eliminating some of the 0's.