Code Jam 2010 - Round 3

Analysis: Fence

The basic scenario here is very similar to the traditional change-making problem, except that the input can be (and actually is guaranteed to be) very large. A condition on the minimum size of the input is very unusual for a programming contest problem, and we didn't add it just for fun. Our solution really, truly does require that the fence length be at least 10^10.

Small Input

Before getting into the real solution though, let's discuss a simpler approach that at least solves the small input. Let's suppose the longest board is of length $A \leq 100$. The key idea is that we should never use more than A boards of any size less than A. If we did, we could replace A of those boards with a smaller number of length-A boards. And that would of course be a better solution.

In particular, this means the total length of all shorter boards is at most $N * A * A \leq 1000000$. Using a breadth-first search, we can find the optimal way of choosing these boards to get each length in that range. The cost of completing the fence using length-A boards can then be computed with a simple division.

By the way, you can actually replace $N * A * A$ with just $A * A$ in the above solution. Hopefully you will see why after reading the rest of the solution!

Large Input

The small-input solution does not actually take advantage of the minimum length of the fence. So the big question is: how could we possibly do that?

Well, the previous solution offers a bit of a hint. For a really long fence, it makes sense that in the end, we are going to want to make heavy use of the longest board just to cover up as much length as possible. So let's suppose the longest board has length A, and that **L** is equal to $p*A + q$ for integers p, q with $q < A$. (Note that the problem statement guarantees $p \geq A$.) Then we need to do one of the following things:

- Use a number $T_{0,q}$ of shorter boards to create a fence of length $0*A + q$, then use p boards of length A to cover the rest.
- Use a number $T_{1,q}$ of shorter boards to create a fence of length $1*A + q$, then use $p-1$ boards of length A to cover the rest.
- ...
- Use a number $T_{p,q}$ of shorter boards to create a fence of length $p*A + q$, then use 0 boards of length A to cover the rest.

So we need to calculate $p + S_{p,q}$ where $S_{p,q}$ is defined to be $\min(T_{0,q} - 0, T_{1,q} - 1, \ldots, T_{p,q} - p)$. Intuitively, $S_{p,q}$ can be thought of as measuring the minimal number of boards required to get a fence length of q mod A, subject to two modifications:

- Every time the length increases by A, it means one less max-length board in the future, so you can subtract one from the total count.
- The total length is not allowed to go over $p*A + q$.

And now, we can make concrete how the condition that **L** is very large simplifies things:

Lemma: The second condition in the definition of $S_{p,q}$ is unnecessary.

Proof: We claim that $T_{i,q} - i$ is minimized when $i \le p$, which will prove the lemma. So let's consider a minimal $T_{i,q} - i$. Then we have a set of boards $b_1, b_2, \ldots, b_m$ making a length of $i*A + q$. If $i > p$, then $m > p \ge A$. But then, the set $\{b_1, b_1 + b_2, \ldots, b_1 + b_2 + \ldots b_m\}$ contains at least $A+1$ numbers, so two of these numbers are congruent modulo A. Subtracting them, we can find a non-empty subset of $\{b_1, b_2, \ldots, b_m\}$ whose sum is a multiple of A. Therefore, we can replace that subset with boards of length A to get a strictly better solution, implying $T_{i,q} - i$ could not have been optimal in the first place!

Okay, that's all very nice, but what's the solution? Well the previous lemma implies we need to calculate the minimum number of boards required to get a fence of length q mod A, subject to the fact that each time the total goes up by A, we will need one fewer board in the future. (For shorter fences, this approach just does not work. Our algorithm would make a very long fence with length correct modulo A, and then try to subtract length-A boards, which of course is not allowed!)

Anyway, once the problem has been reduced in this way, it can be done pretty straightforwardly with a breadth-first search. Our graph has one vertex for each residue modulo A. From each vertex, we add an edge for each possible board length. If adding that board involves wrapping past A, then it has weight 0. Otherwise, it has weight 1. So the final algorithm is: calculate the minimum distance in this graph to vertex q to get $S_{p,q}$, and finally add p.