Code Jam 2012 - Round 3

Analysis: Quality Food

Multiple Solutions

Sometimes our problemsetters like to come up with big, complicated solutions to their problems, and sometimes it turns out that there are simpler alternatives. We'll present two solutions here, including the simple one that many problems solvers used, and the more complex one used by the author.

How much does a single delivery cost?

Let's look at the cost of a delivery containing **K** meals. It's easy to see that we should order the cheapest meal with a time-to-stale of at least 0, the cheapest meal with a time-to-stale of at least 1, the cheapest meal with a time-to-stale of at least 2, and so on until we order the cheapest meal with a time-to-stale of at least **K**. Since **K** could be very large, we'll solve this problem in O(**N** log(**N**)).

First we'll sort the kinds of food by price, and construct an array that contains [(0, 0), (d_1, c_1), (d_2, c_2), ...], where we should buy food that costs c_i for the meals between d_{i-1} and d_i after the delivery arrives. We can then process that into another array containing (d_i, C_i), where C_i is the total cost of a single delivery that lasts d_i days, and $(C_i-C_{i-1}) / (d_i-d_{i-1}) = c_i$.

We only had to do those above steps once. Now that we have that array, it's an O(log(N)) binary search (or a simpler linear search, since we'll find that we don't need the efficiency for either of our solutions) to find out how much it costs to have a delivery that lasts **K** days: find i such that $d_{i-1} \le \mathbf{K} < d_i$, and charge $C_{i-1} + (K-d_{i-1})*c_{i-1}$.

We'll call that function `SingleDeliveryCost(days)`, and for convenience later in this editorial we'll also define `SingleDayCost(day)` to be the cost to buy the cheapest meal with a time-to-stale of at least `day`. Observe that `SingleDayCost(a) ≤ SingleDayCost(b)` if `a ≤ b`, and that `SingleDeliveryCost(days+1) = SingleDeliveryCost(days) + SingleDayCost(days+1)`.

Deliveries should be (almost) the same size

Let's show that all of your deliveries should be almost the same size. Let's suppose that we have a solution that has a delivery, A, containing `a` meals, and another delivery, B, that contains `b` meals, with `b ≥ a+2`. Then the cost for those two deliveries is `SingleDeliveryCost(a) + SingleDeliveryCost(b)`. If instead we increased delivery A's size to `a+1` and decreased delivery B's size to `b-1`, we'd have the same total number of days, but the delivery cost would be:

```
SingleDeliveryCost(a+1) + SingleDeliveryCost(b-1)
= SingleDeliveryCost(a) + SingleDayCost(a+1) + SingleDeliveryCost(b) - SingleDayCost(b-1)
≤ SingleDeliveryCost(a) + SingleDeliveryCost(b).
```

This shows that all our deliveries should be of size `a` or `a+1` for some value of `a`.

Deliveries should be of an "interesting" size

Let's say that we want to buy a total of **D** days' worth of food, and we want to know how big our deliveries should be. We'll start by considering a number of deliveries `X` that we're going to show is "uninteresting": one such that if we look the size of the largest delivery, `ceil(D/X)`, then `ceil(D/(X-1)) = ceil(D/X) = ceil(D/(X+1))`. In such a case, adding a delivery changes the cost by `SingleDeliveryCost(ceil(D/X)) - ceil(D/X)*SingleDayCost(ceil(D/X))`. If we add a delivery, we'll change the cost by the negation of that amount. One of those two quantities has to be non-negative, which means that we never need to consider making `X` deliveries.

That means that we only need to look at numbers of deliveries such that if we changed the number of deliveries, we'd be changing `SingleDayCost(ceil(D/X))`. There are only 2**N** such delivery sizes, and trying all of them solves the problem in O(N log(N)).

The other approach

A competitor who couldn't convince himself or herself of the above still has a good chance of solving the problem. We'll start by presenting a trick ("Changing the Question") that you can use whether or not you have the above realization–it might make the math easier–and a totally different approach that, admittedly, requires some math of its own.

Changing the Question

An observation we can make is that instead of solving the problem that was asked, we can play a trick to let ourselves solve a simpler problem: "Can I eat quality food for **D** days?" We do this by **binary searching on the answer**. We know the answer is between 0 and **M**, so we can start by solving that problem O(log(M)) times.

How many deliveries should there be?

If the logic in the previous solution was too complicated, here's an alternative that probably isn't going to make you any happier. On the other hand, if you were thinking that there weren't enough logarithmic factors in the complexity analysis, or enough -ary searches, this could make you happy. This was our original solution for the problem, and it makes the I/O author perversely proud to have been part of making a problem to which a binary search in a ternary search in a binary search is a perfectly reasonable solution.

The argument here is that we can ternary search for the number of deliveries that minimizes the cost to buy quality food for **D** days. In order for a ternary search to be possible, the function has to be strictly decreasing, then strictly increasing (or vice versa). In this case, the domain of the function is from where the number of deliveries `X` is the minimum possible number at which we could order **D** meals (regardless of how expensive they are), to `X=`**`D`**. We'll show that the function decreases, then increases over this domain.

First let's take the `SingleDayCost` function and extend it over the real numbers. We already know what it is for the integers; let's just linearly interpolate for the real numbers in between, and call the result `G`. This has the nice result that the cost for `X` deliveries and **D** days, which we'll call `H(X)`, is `H(X) = G(D/X)*X + X*F`.

Now, we're about to start taking derivatives, and although `G` is continuous, it isn't differentiable. There are a couple of ways of getting around this, and we're going to take a physicists' route. We'll define a function `G''` to be a sum of delta functions such that the double-integral of `G''` over `X` is our original `G`. Then we'll define `G'` to be `G''`'s integral, and `G` to be `G'`'s integral. That lets us say that the first derivative of `G` is non-negative everywhere, and so is its second derivative. Don't try this at home, folks, especially if you live in a math class.

What we're really interested in doing here is proving that $H(X)$ is decreasing and then increasing, which we can do by proving that it has a positive second derivative:

$$H(X) = G(D/X)*X + X*F$$

$$H'(X) = G(D/X) - X*G'(D/X)*X^{-2} + F$$

$$H'(X) = G(D/X) - G'(D/X)/X + F$$

$$H''(X) = -X^{-2}G'(D/X) + X^{-2}G'(D/X) + G''(D/X)X^{-3}$$

$$H''(X) = G''(D/X)X^{-3} \geq 0$$

Therefore $H(X)$ is decreasing and then increasing (or just decreasing or increasing, which is also fine), and can be ternary searched on for the minimum cost of surviving **D** days. This algorithm's complexity works out to $O(N\log(N) + \log^2(M)\log(N))$.