Code Jam 2010 - World Finals

Analysis: Candy Store

Summary

At first this problem seems intimidating. There are a huge number of ways customers can order, and even once you've decided which boxes to order, verifying that all possible scenarios are covered is non-trivial.

But as it turns out, a greedy algorithm for choosing the boxes, and a greedy approach to handing them out, ends up working! And as with most greedy problems, the difficulty lies only in convincing yourself that it works. It's actually the easiest problem in the set to implement. The only thing you must overcome is self-doubt!

Proof

Suppose that part of our order is a set of boxes with total weight N grams, with no box larger than X = floor(N/k)+1. Furthermore, suppose that we know these boxes have the property that if the k customers' orders don't total more than N grams, these boxes can be used greedily to satisfy their orders. "Greedily" means that when a customer comes to the store, you simply find the largest box that is less than or equal to his order, and give it to him, repeating until you've filled his order exactly. This is our inductive hypothesis: a guarantee that this strategy works.

Now, suppose we order another box with weight X (ie, floor(N/k)+1) grams, making a new set of boxes with total weight N+X grams (and note that no box is larger than floor((N+X)/k)+1, trivially). Suppose the k customers order no more than N+X grams. What happens if we apply the greedy strategy? Well, as soon as we see an order of ≥ X cents, we will use this new X-gram box on it immediately. If we now pretend that the customer instead ordered X fewer cents, then we know the greedy strategy with the rest of the boxes works. The choices made by the strategy are unchanged aside from the use of the new box. If it turns out that there is NO order of ≥ X cents, then the total is at most k*(X-1) ≤ k*(N/k) = N grams. The new box can't ever be used, so the greedy strategy does the same thing for these orders that it did before, and thus it works.

So using induction, we now have what I'll call "The Algorithm", which constructs sets of boxes that work greedily. We start with 0 boxes, and keep adding new boxes of size floor("total sum so far"/k)+1. eg, first we'll add k boxes of size 1 (which we obviously need), then a box of size 2, and so on.

Ok, now how do we prove that this is the best we can do? Suppose we've ordered some set S of boxes that works. Sort the boxes from smallest to largest, and consider the first box that is larger than would be chosen by "The Algorithm" based on all the boxes smaller than it. In other words, if this box is of size Y, and N is the total of all the boxes smaller than it, Y > X = floor(N/k)+1. (Note that X ≤ C since N < k*C.) What happens if the customers all order X cents? None of the boxes of size Y or greater can be used, and the total is k*X > N, so all the boxes smaller than Y don't add up to enough to handle them. This can't possibly work, regardless of strategy.

Thus, every box in S, considered in sequence, is no larger than would be added by "The Algorithm" based on the previous boxes. But if any box chosen is STRICTLY smaller, then "The Algorithm"'s later selections will all be reduced accordingly (it is monotonic in that sense). Since S must have sum at least k*C, if we ignore S and instead order an equal number of boxes using

"The Algorithm", their sum will be at least k*C as well. This proves that "The Algorithm" is the best we can do.

Implementation

"The Algorithm" is really, really simple to code. In fact, this is a complete implementation of a Candy Store solution:

```
long long T, k, C, prob = 1;
for (cin >> T; T--;) {
  cin >> k >> C;
  long long sum = 0, num_boxes = 0;
  while (sum < k*C) {
    num_boxes++;
    sum += (sum / k) + 1;
  }
  cout << "Case #" << prob++ << ": " << num_boxes << endl;
}
```

Other Cool Stuff

The observant among you may have noticed that this solution is very general, and remains unchanged even if we vary the problem a little. For instance:

- The bound C is effectively useless. We can throw it out and let customers order any amount, as long as the TOTAL order is no more than k*C.

- We can do no better even if we're told all the customers' orders in advance (at the beginning of the day). The "online" solution is just as good as an "offline" one.

- We can do no better even if we constrain the customers to all order the same amount as each other. This was actually the original proposed form of the problem.