Code Jam 2015 - Round 1C

Analysis: Less Money, More Problems

We will incrementally build a set S of denominations that solves the problem using the minimal number of additional denominations, by restricting ourselves to adding denominations from smallest to largest.

As we add denominations to S, we also maintain an integer N, which is the largest value such that we can produce each value up to and including N. In fact, after all of our choices, S will be able to produce exactly the set of values from 0 to N, and no others.

When we add a new denomination X to S, the new set of values we could produce include each of the values we could produce with the existing set S plus between 0 and C of the new denomination X. If X is at most N+1, then this new set of values will be the set of all values from 0 to N+X*C, so we can update N to N+X*C.

So, we initialize S to the empty set, and N to 0.

Then while N is less than V, we do the following:

- Identify the smallest value we cannot produce: N+1.
- If there is still a pre-existing denomination which we haven't used, let the minimum such denomination be X. If X is less than or equal to N+1, we add it to S, and update N to N+X*C.
- Otherwise, we have no way yet to produce N+1 using the denominations we have, so we must add to S a new denomination X between 1 and N+1. This will increase N to N+X*C. We use X=N+1. No other choice for X could lead to a better solution, since for X=N+1, the set of values the new S will be able to produce is a superset of the values S would be able to produce with any other choice.

Finally, when we have a set S which can produce all values up to V, we output the number of new denominations we had to add.

In the above algorithm, the first option — using a pre-existing denomination — can only occur D times. When the second option is chosen, N increases to (C+1)N+C. Since we stop when N reaches V, this will occur O(log V) times. So the overall time complexity is O(D+log(V)).

Sample implementation in Java:

```
import java.util.*;

public class C {
  public static void main(String[] args) {
    Scanner scan = new Scanner(System.in);
    int T = scan.nextInt();
    for (int TC = 1; TC <= T; TC++) {
      int C = scan.nextInt();
      int D = scan.nextInt();
      int V = scan.nextInt();
      Queue<Integer> Q = new ArrayDeque<>();
      for (int i = 0; i < D; i++) {
        Q.add(scan.nextInt());
      }
```

```
      long N = 0;
      int add = 0;
      while (N < V) {
        // X = The smallest value we cannot produce.
        long X = N + 1;
        if (!Q.isEmpty() && Q.peek() <= X) {
          // Use pre-existing denomination we haven't used.
          X = Q.poll();
        } else {
          // No way to produce N+1, add a new denomination.
          add++;
        }
        N += X * C;
      }
      System.out.printf("Case #%d: %d\n", TC, add);
    }
  }
}
```

Vitaliy's solution in C, which you can download from the scoreboard, is another good example of this approach.