USACO NOV10 Problem 'candy' Analysis

by Nathan Pinsker

The key to this problem is to realize that one of the following two things must happen:

- Bessie can eat an infinite amount of candy.
- Bessie will only be able to have each "favorite number" in the basket at most once during her candy-eating spree.

An infinite candy party occurs if Bessie were able to have a favorite number in the basket twice in a row. If that happens she could duplicate her candy-eating procedure to achieve that number an infinite number of times.

This problem therefore lends itself to the knapsack method: for each possible amount of candy left in the basket, calculate the maximum amount of candy Bessie could have eaten to get to that state. It's easiest to start with the state where there are N candies and work downwards. However, there is a twist -- whenever we encounter a "favorite number" K, we add M to the basket and continue working from our new state K+M.

If Bessie is unable to eat candy forever, then this process is guaranteed to terminate -- each state can only be considered at most F times throughout the entire process. The knapsack method will give us an O((N + F*F*M)*N_opt) runtime.

(It's also possible to solve this problem even faster with a topological sort, although that was not required.) My code is below:

```
#include <fstream>
#include <iostream>
#include <cstring>

#define N 40500

using namespace std;

FILE *in = fopen ("candy.in", "r"), *out = fopen ("candy.out", "w");

int a, b, c, add, t;
int options[50], most[N], isFavorite[N], checked[N];

int main() {
    fscanf (in, "%d%d%d%d", &a, &b, &c, &add);
    memset (most, -1, sizeof (most));
    most[a] = 0;

    for (int i = 0; i < b; i++) fscanf (in, "%d", &options[i]);
    for (int i = 0, j; i < c; i++) {
        fscanf (in, "%d", &j);
        isFavorite[j] = 1;
    }
```

```
    isFavorite[a] = 0;
    for (int i = a; i>0; i--) {
        if (most[i] == -1) continue;

        if (checked[i] > c+1) {
            fprintf (out, "-1\n");
            return 0;
        }
        if (isFavorite[i]) {
            checked[i]++;
            if (most[i] > most[i + add]) {
                most[i + add] = most[i];
                i += add+1;
            }
                continue;
        }
        for (int j = 0; j < b; j++) {
            if (i - options[j] < 0) continue;
            most[i - options[j]] = max (most[i - options[j]], most[i] +
options[j]);
        }
    }
    for (int i = 0; i < a; i++) t = max (t, most[i]);
    fprintf (out, "%d\n", t);
    return 0;
}
```