USACO OCT09 Problem 'diet' Analysis

by Fatih Gelgi and Rob Kolstad

This problem is known as a 'knapsack problem'.

While the standard solution is known as 'dynamic programming', the explication gains little from that particular name.

The standard solution defines an array (named 'weight' below) whose contents are '1' only when it is possible to achieve the weight of that particular index (thus weight[12] means there is *some* (unspecified) way to achieve a weight of 12 units.

There's always a way to achieve 0 units, so we mark weight[0] as 1.

Then, for each bale weight that we encounter, we look at all the possible weights we have so far and add the new weight to each of them, marking the new locations with 1 (whether or not they are already 1).

Consider the first bale weight 2:

```
Original possible weights  1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

Adding in bale weight 2    1 0 * 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
```

If the next bale weighs 5, it can create two additional possible weights:

```
Adding in bale weight 5    1 0 1 0 0 * 0 * 0 0 0 0 0 0 0 0 0 0 0 0 0
```

Here's what happens when the next bale weighs 4:

```
Adding in bale weight 4    1 0 1 0 * 1 * 1 0 * 0 * 0 0 0 0 0 0 0 0 0
```

But it's tricky to add in a bale of weight 2 at this point:

```
Adding in bale weight 4    1 * 1 * 1 * * * * 1 * 1 * 0 0 0 0 0 0 0 0
```

Note that some 1's were overwritten (presumably with 1's). This is just one way of seeing that one must work *backwards* through the array when processing or one will accidentally propagate a 1 through the end of the array.

Fatih's program below implements this clever O(N*h) algorithm.

```
#include <fstream>

#define MAXH 45000
#define MAXN 500

using namespace std;
```

```
int h,n,bales[MAXN],weight[MAXH];

// Knapsack
int solve() {
    int eat;
    // add each haybale one by one
    for (int i=0; i<n; i++) {
        for (int j = h; j > bales[i]+1; j--)
            if (weight[j-bales[i]]) weight[j] = 1;
        weight[bales[i]] = 1;
    }
       // find best weight
    for (eat = h; eat > 0 && !weight[eat]; eat--);
    return eat;
}

int main() {
    ifstream f("diet.in");
    f >> h >> n;
    for (int i=0; i<n; i++) f >> bales[i];
    f.close();

    ofstream f("diet.out");
    f << solve() << "\n";
    f.close();
}
```