Code Jam 2016 - Round 1C

Analysis: Fashion Police

Fashion Police: Analysis

Small dataset

Let us denote the number of possible outfits by *W*; *W* = **J** * **P** * **S**. A viable brute force approach is to produce all 2^W subsets of the set of possible outfits, check them for violations, and take one of the largest sets with no violation. We can check a set of outfits without worrying about what order they are worn in; if some outfits in that set create a fashion violation, it will eventually happen no matter what order those outfits (and any other outfits) are worn in. Another helpful observation is that if **S** ≤ **K**, then the answer is trivial: simply return every possible outfit.

This approach works for most of the Small cases, but not all of them. For the cases `3 3 3 1` and `3 3 3 2`, there are 2^{27} possible sets of outfits, and that is over 100 million. One approach is to just solve these "problem cases" before downloading a dataset. You can notice a pattern in the other answers (more on this later) and infer that the maximum numbers of outfits you can wear are 9 and 18, respectively. It is more tractable to find a set of outfits of a particular size than to find the set of outfits of maximum size. If you are pretty sure that that size is 18, for instance, you can check all sets of size 18 until you find a working one, and maybe even check all sets of size 19 to confirm that none of them work. This is much faster than checking every size between 1 and 19, especially since 27 choose 18 is much smaller than 27 choose 13 and 27 choose 14, for example.

In fact, there are only 100 possible test cases that fall within the limits of the Small dataset, so you can compute the answer to any possible input before even downloading the dataset!

Large dataset

Again, if **S** ≤ **K**, we can return every possible outfit. Otherwise, a key insight is that that the pigeonhole principle tells us that a maximum solution can have no more than min(**J** × **P** × **K**, **P** × **S** × **K**, **J** × **S** × **K**) different outfits. Since **J** ≤ **P** ≤ **S**, this puts an upper bound of **J** × **P** × **K** on our solution. You can also infer this from the output of a brute force solution.

One pitfall to watch out for in this problem is that there exist sets of outfits that are *maximal* — that is, you cannot add any new outfit to such a set without going to jail — but are not the largest possible. That is, they are maxima*l* but not maxim*um*.

For instance, for an input of **J** = 1, **P** = 3, **S** = 3, **K** = 2, the following is a maximal set of outfits that is not maximum (the maximum size is 6): `1 1 1`, `1 1 2`, `1 2 2`, `1 2 1`, `1 3 3`.

So, if we just randomly choose outfits without violating the law, we are in danger of being trapped in a locally maximal set.

Fortunately, there are several greedy approaches to achieve a set of size **J** × **P** × **K** outfits without angering the Fashion Police. As argued above, we know that **J** × **P** × **K** is the maximum possible size, so if we can find a set of that size, we are done!

We cannot use a jacket-pants combination more than **K** times, so if we want to produce **J** × **P** × **K** outfits we are forced to use each combination exactly **K** times. To simplify the math, we use

jacket, pants and shirt numbers between 0 and the appropriate total minus 1. We just need to remember to add 1 to every identifier when we print it to the output.

Let us fix a combination with jacket number *j* and pants number *p*. Our proposal is assign to it shirts (*j* + *p*) % **S**, (*j* + *p* + 1) % **S**, ..., (*j* + *p* + **K** - 1) % **S**, where % stands for the modulo operation. Since **S** > **K**, these are all different, and by construction, the combination of jacket and pants is used exactly **K** times, as desired.

What about jacket-shirt and pants-shirt combinations? Let us fix a jacket number *j* and a shirt number *s*. If the outfit (*j*, *p*, *s*) appears in the constructed set, then *s* = (*j* + *p* + *d*) % **S** for some *d* between 0 and **K** - 1, inclusive. Then, by modular arithmetic, and noticing that *j* % **S** = *j*, *p* % **S** = *p*, and *s* % **S** = *s*, it turns out that *p* = (*s* - *j* - *d*) % **S**. Then, each choice of *d* uniquely determines *p*, so there cannot be more *p*s to go with a given combination of *j* and *s* than choices of *d*, and there are **K** choices of *d*, which means the combination of *j* and *s* does not exceed the maximum.

Since this is valid for any *j* and *s*, and a symmetrical proof is valid for each pants-shirt combination, we have proven that the proposed set of outfits does not break any of the rules of the Fashion Police.

Another way of thinking about it

The problem is equivalent to trying to select as many cells as possible in a 3-D **J** by **P** by **S** grid such that no line of three cells in the x, y, or z direction includes more than **K** selected cells. Each outfit corresponds to one such cell.

Here is an illustration of this approach for two cases. The left-right axis corresponds to shirts, the up-down axis corresponds to pants, and the layers (imagine them stacked up) correspond to jackets. Each `*` represents a selected outfit, and each `.` represents an unused outfit.

J = 2, **P** = 3, **S** = 4, **K** = 1:

```
*... .*..
.*.. ..*.
..*. ...*
```

Outfits: `1 1 1`, `1 2 2`, `1 3 3`, `2 1 2`, `2 2 3`, `2 3 4`.

J = 2, **P** = 3, **S** = 4, **K** = 2:

```
**.. .**.
.**. ..**
..** *..*
```

Outfits: `1 1 1`, `1 1 2`, `1 2 2`, `1 2 3`, `1 3 3`, `1 3 1`, `2 1 2`, `2 1 3`, `2 2 3`, `2 2 4`, `2 3 4`, `2 3 1`.

Observe that:

- In both cases, the second layer (jacket 2) is the first layer (jacket 1) shifted one unit to the right (and wrapping around).
- We can get the answer to the second case from the answer to the first case just by adding a `*` to the right of every existing `*` (and wrapping around).
- By construction, the first layer does not have more than **K** `*`s in any row or column. This is also true of each additional layer, since they are all rotations of the first layer. Moreover, no line of cells parallel to the jacket axis (that is, across layers) can possibly have more than

K *s; considering how the layers are constructed, that would directly imply more than **K** *s in one row of the first layer.

This construction works for any set of **J**, **P**, and **S** values. It takes advantage of the **J** ≤ **P** ≤ **S** condition in the problem statement. We put that condition in to save you the trouble of adding a bunch of case-based logic to try to figure out which dimension was largest; the fashion-conscious GCJ staff is well aware that some closets have more pants than shirts!