Solution Description - Chika Wants to Cheat

Author: Vlad-Alexandru Gavrila-Ionescu

In a nutshell, the task asks to find a bijection between numbers from 1 to `N` and sets of segments (or patterns) that follow certain rules.

Subtask 1 (2 points) N <= 2

This subtask was intended to reward competitors who can perform the two-stage interaction correctly. To solve this subtask, simply encode `N = 1` as a pattern containing one segment, and `N = 2` as a pattern containing two segments.

Subtask 2 (9 points): N <= 25

First of all, we need to define what segments we should use to describe our patterns. We observe that a segment `{{x, y}, {x', y'}}` is valid if and only if `gcd(|x - x'|, |y - y'|) = 1`. For a card of size `2 x 2`, there are 28 such segments.

Therefore, the solution to this subtask is to simply code `N` as a set of `N` different segments. Likewise, the decoding is trivial.

Subtask 3 (15 points): N <= 1000, no rotations are performed

So far, we have encoded `N` as a set of `N` segments. To improve upon that, let us first order the valid segments we can draw by some rule that is common between the `BuildPattern` and `GetCardNumber` functions. Then, use each segment to encode individual bits of `N`: drawing a segment will mean that the corresponding bit is 1, and not drawing it will mean the corresponding bit is 0. Likewise, by knowing that the grader will not rotate the cards, we can easily find each card number by looking at which segments we received in the pattern to decode and setting the corresponding bits in our answer.

This subtask can be solved by finding 10 valid segments (for example, the ones of length 1 parallel to the coordinate axes) and using them to build binary encodings.

Subtask 4 (3 points): N <= $1.6 * 10^7$, no rotations are performed

The same solution applies as for the previous subtask, but we need to use all 28 valid segments.

Subtask 5 (24 points): N <= 1.6 * 10^7

To progress with finding a solution, we need to find a way of establishing “which way is up” for a given pattern; that is, to find some way of determining whether a certain rotation of a pattern is the one we initially produced as output for the `BuildPattern` function.

We observe that, when we rotate a pattern, a given segment can transpose to exactly 3 other segments. We will partition the segments into groups of 4, where each segment in a group can transpose to any other in the same group via rotations.

We assign labels 0, 1, 2 and 3 to each segment in a group such that segments 1, 2, and 3 are obtained by rotating segment 0 by 90, 180 and 270 degrees counter-clockwise, respectively. We call the “drawing configuration” for a group a string `C` of four 0s and 1s: `C[i]` is 1 if and only if we choose to draw the segment with label `i` from the group.

We will select one such group of segments, and call it the “self-righting” group - by choosing to draw only some segments in this group, we can uniquely determine the correct rotation for the whole pattern.

For instance, let’s choose our self righting group as being comprised of the following segments:

- Segment 0: `{{0, 0}, {0, 1}}`
- Segment 1: `{{0, 2}, {1, 2}}`
- Segment 2: `{{2, 2}, {2, 1}}`
- Segment 3: `{{2, 0}, {1, 0}}`

We choose to set “0111” as the drawing configuration for this group, meaning that we draw all segments except segment 0. Therefore, when we receive a pattern that we need to decode, we know there is a unique rotation which produces this drawing configuration for this group: the one for which segment `{{0, 0}, {0, 1}}` is not drawn.

This uses up 4 out of our 28 segments. We can treat the remaining 24 segments as bits to encode `N`; drawing a certain segment will signify a bit of 1, while not drawing it will result in a bit of 0. This leaves us able to encode numbers up to 2^{24} with this algorithm, which is enough to solve this subtask.

Subtask 6 (18 points): N <= 4 * 10^7

We improve upon the previous subtask by noticing that we can actually use three configurations for the self-righting group: “0001”, “0011”, and “0111”. This leaves us able to encode numbers up to $3 * 2^{24}$.

Subtask 7 (29 points): N <= $6.7 * 10^7$

To solve this subtask, we must consider what happens if we don't use any of the above configurations for the self-righting group. Naturally, we would have to use another group as the self-righting one. Therefore, we get the following idea: we will consider the `K = 28 / 4` groups of segments in a certain order and use the following recurrence to count how many patterns we can use to encode numbers:

`DP[i]` - the number of self-righting patterns (i.e. that contain one self-righting group) we can produce using the first `i` groups of segments.

If we use the i^{th} group as the self-righting one, it means we are free to use the segments in the previous `i-1` groups as bits, so we add the number of numbers we can encode with those bits ($2^{4(i-1)}$) times the number of self-righting configurations we can assign to our current group (`3`).

If we don't use the i^{th} group as the self-righting one, it means that we must have used a previous group instead. Therefore, the current group must not also be self-righting. There are `4` remaining non-self-righting configurations for the current group: "0000", "0101", "1010" and "1111". We therefore add this number (`4`) times the number of self-righting patterns we can produce with `i-1` groups (`DP[i-1]`).

Therefore, the final formula is `DP[i] = 3 * 2^4(i-1) + 4 * DP[i-1]` (which turns out to equal $2^{4i-2} - 2^{2i-2}$, but this is not at all relevant for building our solution).

The value for `DP[K]` is slightly above $6.7 * 10^7$, therefore we can successfully solve this subtask. To actually produce the correct pattern for a given `N`, we perform the standard approach of traversing the recurrence values from `K` towards 0 and observing which of the two terms of the sum is needed to produce `N`, selecting the correct configurations for each group of segments as we go. The `GetCardNumber` function is implemented in a similar manner, building up `N` depending on the segment configurations we encounter in the correctly rotated pattern.

Inspiration for the problem

We will conclude the editorial by presenting the two sources of inspiration for this problem. The first one is the problem "parrots" from IOI 2011, which the author had to solve during the live contest and thought was the highlight of IOI 2011 day 2. The task also involved finding a bijection to convert some data into another format and back again, but the bijection was between ordered arrays and unordered sets.

The second source of inspiration for the task concerns the main character of the task, Chika Fujiwara, who is a main character in the Kaguya-sama: Love is War manga and anime. A

certain chapter of the manga / episode of the show also involves Chika using a marked deck of cards to cheat in a memory game (and being humorously caught in the process), which the author thought served nicely as the premise of a competitive programming task.