It's been some time since my last blog post, so this is the first one of the new year. Although I didn't participate in Round 102, I worked through several of the problems, and this post is motivated by [Problem D](http://codeforces.com/contest/142/problem/D). It is about a variation on our favorite algorithmic game, Nim. There is already an editorial for the round, but I intend to use this post as future reference in case I come upon such problems again. If you don't see the reduction from the problem to the Nim variation, please read the editorial.

Nim

The normal many-pile version of Nim is played as follows. There are *n* piles of stones, and two players take turns removing stones. On any turn, a player may remove as many stones as desired from a single pile. The player who cannot make a move loses.

A very nice and classical result is that a position *a*1, *a*2, ..., *an* is a losing position if and only if http://espresso.codeforces.com/a83aa3ec56033259314bd4dc40f8d610a6c31aad.png, where http://codeforces.com/predownloaded/7b/ea/7beade55e90846d70020a3d03521d3458b66751b.png is the XOR operator. To prove this, we simply realize that from any losing position, if we change the value of one pile, the XOR will no longer be zero (thus a winning position). And from any winning position, we can look at the highest order bit which does not XOR to zero, pick a pile in which that bit is 1, change it to0, and adjust the rest of the bits in that pile to make the XOR zero.

Nim Variation

The variation in Problem D lies in the fact that a player may remove stones from up to *k* different piles on each turn rather than just a single one. As such, the winning and losing positions change. We begin by defining the operation *bi*(*x*) which equals the *i*-th bit of *x* (always 0 or 1). Then define the *p*-XOR of the values *a*1, *a*2, ..., *an* to be

http://codeforces.com/predownloaded/60/81/608103a9d5ff97c95493aa96d7cdb471249f941e.png.

That is, for each digit in the binary expansion, we add up all the values and take the result modulo *p*. If we choose *p* = 2, for example, we get the sequence of bits resulting from the normal XOR of the values.

It turns out that in this variation of Nim, a position *a*1, *a*2, ..., *an* is a losing position if and only if http://codeforces.com/predownloaded/74/02/74027e843cc438dc91d579b859d2d9218be55e10.png is all zeros. The proof of this result is a little less nice, so I will just outline the idea. From any losing position, consider the highest order bit affected among any of the piles. This bit always goes from 1 to 0(or else the pile is getting bigger). Since at most *k* piles are affected, the (*k* + 1)-XOR of that bit can no longer be 0, leaving us in a winning position.

From any winning position, we again look at the highest order bit which does not (*k* + 1)-XOR to 0. Let's say those bits add up to *m* modulo *k* + 1. Choose *m* piles which have that bit set to 1 and change them to 0, as well as zeroing out all of the lower order bits in those piles. Then move on to the next bit of lower order. If we can fix the (*k* + 1)-XOR of that bit using just the piles we have selected so far (by changing some 0's to 1's), then do so and move on. Otherwise, choose as many additional piles as necessary to change, noting that this will be at most *k* - *m*. If we continue this process we can set the (*k* + 1)-XOR of all bits to be 0 while choosing at most *k* piles.

Conclusion

It is useful to understand Nim and its variations very well because game problems often reduce to them. If you know what you're trying to reduce to, it can often be a lot easier to find the reduction. Then knowing how to solve a simple algorithmic game is all that's left!