If we denote by $W(k,y)$ the number of distributions where at least k islanders have at least y gems, then $V(k,y) = W(k,y) - W(k+1,y)$. And $W(k,y)$ can be calculated from the inclusion-exclusion principle as

$$W(k,y) = \sum_{\ell \geq k} (-1)^{\ell-k} \binom{\ell-1}{k-1} \binom{n}{\ell} \binom{n+d-\ell(y-1)-1}{d-\ell(y-1)}.$$

The proofs of these equalities, as well as of the fact that the resulting algorithm is quadratic, are left as an exercise to the reader.

Problem E: Getting a Jump on Crime

Shortest judge solution: 1945 bytes. Shortest team solution (during contest): 2738 bytes.

This was one of the last problems solved, and the judges were a bit surprised by how few teams solved it. While the problem requires a bit of work, the hard part of that work is pen and paper calculations without the computer, which is a nice type of work for a three person team with one computer.

The problem has two parts – figuring out which jumps are possible, and then computing the shortest jump sequence from the starting position. The second part is easily solved with a breadth first search so let us focus on the first part. Given a horizontal jump distance d, and a height difference Δh between the takeoff and landing positions, we need to figure out how to split the total speed v into horizontal speed v_d and vertical speed v_h such that when making a jump with these initial horizontal and vertical speeds, we end up at a height difference of exactly Δh after having travelled d meters horizontally. From the equations in the problem statement we see that the time t it takes us to travel d meters horizontally is d/v_d, and that after this time our vertical position (relative to where we started) will be $v_h t - \frac{gt^2}{2}$ (follows by integrating the identity for the velocity from 0 to t). Plugging in $t = d/v_d$ and $v_h = \sqrt{v^2 - v_d^2}$, we thus need to solve the equation

$$\Delta h = \frac{d\sqrt{v^2 - v_d^2}}{v_d} - \frac{g^2 d^2}{2v_d^2}$$

for v_d. This may look like pretty nasty, but it can easily be transformed into a quadratic equation in v_d^2: move the $\frac{g^2d^2}{2v_d^2}$ term to the left side, multiply both sides by v_d^2, and then square both sides (this may potentially introduce spurious solutions since it throws away sign differences). The expression for the solution gets a bit messy (a tip to make things a bit cleaner is to parameterize $v_d^2 = xv^2$ and $v_h^2 = (1-x)v^2$ for an unknown $0 \leq x \leq 1$ and then solve for x instead) but can be found using only pen, paper, and patience; we do not want to spoil the joy of doing the math so leave the details of this to you. Solving this gives (up to) two solutions, one or both of which may be spurious due to the squaring trick above. We observe that if there are two real solutions, it is always better to take the one with higher value of v_h, because that corresponds to making a higher jump. It also turns out that the solution with higher value of v_h is never spurious so we can in fact take it without even checking if it is spurious.

With the calculus out of the way: suppose we now want to check whether we can make a jump from the building at position (x_1, y_1), with height h_1 to the one at position (x_2, y_2), with height h_2. This corresponds to making a jump with height difference $\Delta h = h_2 - h_1$ and horizontal length $d = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$, so we use what we figured out above to determine the jump parameters v_d and v_h (or we figure out that such a jump is impossible). Now that we have the jump parameters we know the exact parabola of the jump trajectory, we also need to check all buildings that lie between (x_1, y_1) and (x_2, y_2) to make sure that the jump trajectory goes above the building. To do this, it is sufficient to check the first and last times where we are over the building in question (because the parabola is concave). Finding the buildings to check can be done in $O(d_x + d_y)$ time but it was OK (at least if the checks were done reasonably efficiently) to do it in $O(d_x \cdot d_y)$ time by simply iterating over all buildings and checking which ones the trajectory passes by. Overall, this leads to an $O(d_x^2 d_y^2 (d_x + d_y))$ or $O(d_x^3 d_y^3)$ running time for building the graph (because we have to do this for all $O(d_x^2 d_y^2)$ pairs of buildings).

Problem F: Go with the Flow

Shortest judge solution: 872 bytes. Shortest team solution (during contest): 869 bytes.

This was one of the easiest problems in the set. The most straightforward approach is to try all line lengths from the shortest possible and upwards, until there is only one line. To try a given line length, we essentially just simulate – place word by word, and keep track of where in the previous line there was a river and how long it was.

A priori, the running time of this sounds dangerous – there are potentially $\Theta(nL)$ line lengths to try (where $L = 80$ is the max word length) and trying a line length naively takes $\Theta(nL)$ time for a total of $\Theta(n^2L^2)$ time which sounds pretty dangerous for $n = 2500$, $L = 80$. As far as we know it was not possible to get this to pass, but there are several easy optimizations that can be made, and doing any one of them was sufficient to pass:

1. Instead of going all the way up until we just have one line, we can stop when the longest river found is at least as long as the current number of lines.
2. Instead of checking a line length in $\Omega(nL)$ time we can do it in $O(n)$ time.
3. Instead of checking every possible line length, we can check “what’s the next line length that will cause the words to be wrapped differently?” and jump directly to that one.

Optimizations 1 and 3 are very hard to analyze exactly how well they perform, and in general it is very hard to craft test data for this problem. To do well against optimization 1, it is useful to have cases with 2500 words where the longest river is as short as possible. The best such case we have (which was found only during the night before the contest!) has a longest river of 3 (but its average word length is not very high, which would be needed to really make optimization 1 work poorly). However, we don’t even know whether it is possible to make such a case with maximum river length 1 or not (we suspect that this is impossible with word lengths bounded by 80, and if anyone finds a proof we would be very interested in seeing it)!

Problem G: Panda Preserve

Shortest judge solution: 4669 bytes. Shortest team solution (during contest): 4372 bytes.

This was the only computational geometry problem in the set. We are given a polygon, and asked to find the point inside (or on the boundary of) the polygon that is furthest away from any vertex of the polygon (or rather, we are only asked to find this furthest distance, but we