Problem I: Secret Chamber at Mount Rushmore

Shortest judge solution: 405 bytes. Shortest team solution (during contest): 498 bytes.
Python solutions by the judges: both Pypy and CPython

This was one of the two easiest problems in the set. The set of translations forms a directed graph on the 26 letters of the alphabet. Given two query words $s_1 s_2 s_3 \ldots s_L$ and $t_1 t_2 t_3 \ldots t_L$ of equal length L (if the lengths are different, the answer is clearly `no` and we just answer that), we need to check whether for each $1 \leq i \leq L$, there is a path from s_i to t_i in the graph of letter translations.

One natural way of doing this is to precompute the transitive closure of the graph using the Floyd-Warshall algorithm, which allows you to check each pair (s_i, t_i) in constant time. But the bounds are quite small, so you can use pretty much any polynomial time algorithm for this (e.g. doing a DFS for every pair (s_i, t_i)).

Problem J: Son of Pipe Stream

Shortest judge solution: 1649 bytes. Shortest team solution (during contest): 5492 bytes.
Python solutions by the judges: only Pypy

Even though the problem does its best to avoid using the word "flow", it should be pretty clear that this is in fact a maximum flow problem, though it has a few twists to resolve. First, the viscosity parameter v is a red herring that is pretty much irrelevant to the problem and we will ignore it here.

A first small observation is that the desired flow will be *a* maximum flow from the water and Flubber sources to the destination. Let Z be that total maximum flow. Assume for the moment that it was possible to distribute this arbitrarily between Flubber and water, so that it was possible to route any amount F of Flubber between 0 and Z and $W = Z - F$ water. Then a bit of calculus (left as an exercise) shows that the maximum flow would pick $F = a \cdot Z$.

There are two different reasons why the assumption made above might not hold. One is a "trivial" one: the maximum amount of Flubber routable from Flubber source to destination might be less than $a \cdot Z$, or the amount of water routable might be less than $(1 - a) \cdot Z$. If this is the case, we should simply set the desired amount of Flubber F to value nearest to $a \cdot Z$ in the interval $[Z - W_{\max}, F_{\max}]$. Let us refer to the resulting potentially optimal values of F and W as F^* and $W^* = Z - F^*$.

The second reason is a bit more subtle – it is not clear whether it is the case that we can simultaneously achieve Flubber F^* and water W^*. Let $\vec{f_1}$ be a flow which routes $F_{\max}$ Flubber and $Z - F_{\max}$ water, and let $\vec{f_2}$ be a flow which routes $Z - W_{\max}$ Flubber and $W_{\max}$ water. Then for $\alpha \in [0, 1]$, $\alpha\vec{f_1} + (1 - \alpha)\vec{f_2}$ is a flow of fluids in which $\alpha F_{\max} + (1 - \alpha)(Z - W_{\max})$ of the fluid originates at the Flubber source, and we can set α appropriately to a constant so that this equals F^*. However, there is a snag with this: there might be pipes where $\vec{f_1}$ and $\vec{f_2}$ route fluids in opposite directions, so it is not clear that we can achieve this flow while satisfying the "water and Flubber must not go in opposite directions" constraint. Phrased a bit more abstractly, the "no opposing flows" constraint is not a convex constraint.

It turns out that this second reason is actually a non-reason – it is always possible to achieve Flubber F^* and water W^*. But we now have to construct the actual Flubber and water flows. One way of doing that is to take the mixed flow $\vec{f}^* := \alpha\vec{f}_1 + (1-\alpha)\vec{f}_2$ from above. This tells us how much fluid we would like to send (and in which direction) along each pipe, but it doesn't tell us how much of that fluid should be Flubber, and how much should be water. To figure that out, we make a new flow graph where we set the (directed) capacity of each edge to be the (directed) flow in $\vec{f}^*$ along that edge. We then compute the maximum flow from the Flubber source in the new graph. This gives the Flubber flow, and the unused capacity gives the water flow. An issue here is that the new graph has non-integer capacities, so one may have to be a bit careful when computing the maxflows.

Problem K: Tarot Sham Boast

Shortest judge solution: 473 bytes. Shortest team solution (during contest): 585 bytes.
Python solutions by the judges: both Pypy and CPython

This problem has a beautiful solution that is not impossible to get an intuition about, but hard to actually prove correct.

A naive way of computing the probability that a string X of length ℓ appears in a uniformly random string of length n is to use the principle of inclusion-exclusion:

$$p(X) = \sum_{I \subseteq [n-\ell+1]} (-1)^{|I|+1} \Pr[\text{there is an occurence of } X \text{ at all positions in } I]$$

For $I = \{i\}$ consisting of a single element, the probability in the sum is simple $3^{-\ell}$ – the probability that characters $i, i+1, \ldots, i+\ell-1$ match X. Thus the first-order contribution from index sets I of size 1 to $p(X)$ is simply $(n-\ell+1)/3^\ell$, regardless of what X looks like.

For the second-order terms $I = \{i, j\}$, things are a bit more interesting. If $j \geq i+\ell$ then the probability is simply $3^{-2\ell}$ since the two matches of X involve disjoint positions. But for $j < i+\ell$ however, the probability is 0 unless $j-i$ is an *overlap* of X, where we say that t is an overlap of X if the prefix of the first $\ell - t$ characters of X equals the suffix of the last $\ell - t$ characters of X. If t is an overlap of X, then the set $I = \{i, i+t\}$ contributes $-1/3^{2\ell-t}$ to the expression for $p(X)$ above.

This hints at the following intuition: strings X with more overlaps should have smaller values of $p(X)$. Among different overlap values t, higher values of t seems to result in smaller probabilities, because the subtracted terms $1/3^{2\ell-t}$ are larger. So a somewhat natural hypothesis is that if we write the overlaps of X in decreasing order, then strings with lexicographically smaller overlap sequences have higher likelihood.

However, this is just an intuition, and it is not at all clear what happens with higher-order terms – the degree-3 terms (having $|I| = 3$) give positive contributions to $p(X)$ and more overlaps will by a similar reasoning cause these to be larger. It turns out that the hypothesis above is correct, and that lexicographically smaller overlap sequences leads to larger probabilities. We only have a very long and non-intuitive proof of this, which we don't include here. But since several people have expressed an interest in it, it has been made available as a separate document here: `http://www.csc.kth.se/~austrin/icpc/tarotshamproof.pdf`.

As an example, consider the two strings $X =$ `RPSRPSRPS` and $Y =$ `RPRRPRRPR`. The overlaps of X are $Ov(X) = (6, 3, 0)$. The overlap sequence of Y is $Ov(Y) = (6, 3, 1, 0)$. This means that in general, X has a higher likelihood of appearing than Y (since $(6, 3, 0)$ is lexicographically smaller than $(6, 3, 1, 0)$).