NCNA 2018 Solution Sketches

November 13, 2018

Here are some brief outlines of solutions to the NCNA 2018 problem set. They are arranged in order of perceived judge difficulty. This document may be updated. Contact `bcsandlund@uwaterloo.ca` for corrections, alternative approaches, and any other comments.

Problem E - Euler's Number

Authors: Joshua Guerin and Kathleen Ericson. Solved/ Tries: 178/453.

This problem asks you to compute approximations to Euler's Number, e. The problem is straightforward except that n can be as large as 10 000, and you don't want to compute 10000! and get overflow, or run an $O(n^2)$ algorithm and TLE. You can either stop when you get to about $n = 16$, since at that point the approximations are within 10^{-12} of the exact value of Euler's constant, or iteratively divide instead of computing the exact factorial, and stop early or compute the reciprocal of all needed factorials in a single $O(n)$ time loop.

Problem J - Kaleidoscopic Palindromes

Author: Alexander Scheel. Solved/ Tries: 68/200.

You are asked to find numbers that are palindromic in many bases at once. Again this is straightforward except the input range can be as large as 2 000 000 and the number of bases as large as 100 000. To avoid TLE, you must observe that it is unlikely for a number to be palindromic in very many bases simultaneously. A number of size n must have half its digits agree with the other half, which, for small bases and randomly chosen n, occurs with probability about $1/\sqrt{n}$. Using this argument across k bases, if each base has independent probability, we get probability $1/n^{k/2}$. Rather than proving the independence, you can write a program to test it. It turns out there are only two non-trivial numbers palindromic in base 2 and 3 simultaneously in the input range: 6 643 and 1 422 733. There are no numbers in the input range simultaneously palindromic in bases 2, 3, and 4. Therefore a solution that just checks each base iteratively

and stops when a number is determined to not be palindromic in one of them runs fast enough.

This problem showed an interesting spread as to when it was solved. The naive solution gets accepted, but realizing this took thought, and so it appears some teams did not attempt the problem until late into the contest, while perhaps others simply programmed the naive approach which got accepted.

Problem F - Lipschitz Constant

Author: Volodymyr Lyubinets. Solved/ Tries: 28/316.

This problem asks you to find the Lipschitz constant associate with a function f defined on n integer inputs. A closer look at the definition shows this is the same as the maximum slope in absolute value. This can easily be computed in $O(n^2)$ time by checking all pairs of points. However, it suffices to sort points by x value and just check pairs of consecutive points. This can be proven as follows. Suppose there is a case where the max slope is not achieved by consecutive points. Let p_1 and p_2 be two points that achieve this maximum slope and further pick p_1 and p_2 to have minimal difference in x value amongst all valid choices. Then since p_1 and p_2 are not consecutive, there must be a point z with x value between them. If z lies either above or below the line passing through p_1 and p_2, then in fact z has a steeper slope with either p_1 or p_2 than p_1 and p_2 have with each other. If z lies on the line passing through p_1 and p_2 it has the same slope, but this contradicts our choice of p_1 and p_2 having maximum slope and minimal difference in x value.

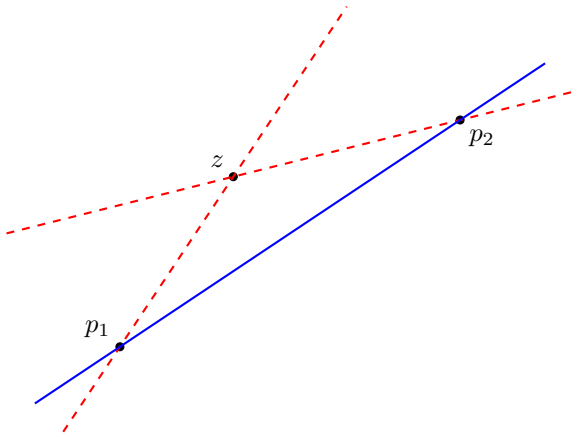


Figure 1: The red dashed line passing through p_1 and z has steeper slope than the blue line passing through p_1 and p_2.

It follows the maximum slope is formed by points consecutive in x value, and the $O(n \log n)$ solution is correct.

Problem D - Travel the Skies

Author: Alexander Scheel. Solved/ Tries: 20/61.

You are asked to determine if a given set of flight schedules can be filled by customers, who arrive at airports via commuting from their homes or through flights that you schedule. We can solve this problem through direct simulation. If you can't fill all the flights on the first day, the schedule is suboptimal. If you can fill all the flights on the first day, there is only one way to do it, so we do it this way and then we process the following day, accounting for the customers we just scheduled. This problem was originally intended to require a maximum flow computation on the graph given by the flight schedules, but it turned out to admit a simpler solution.

Problem I - Other Side

Author: Volodymyr Lyubinets. Solved/ Tries: 61/831.

This problem is a generalization of a classic puzzle. You must get some number of wolves, sheep, and cabbages to the other side of a body of water. You cannot leave wolves with sheep or sheep with cabbages on either bank.

This problem was the most attempted in the contest, but was the third-most solved. My suspicion is that teams may have been better off spending time trying to solve some of the problems listed above, but the simplicity of determining a few cases and getting an accepted solution was probably too enticing. First note that determining the cases mathematically is necessary, since the input size of 1 000 000 is too large to search the solution space.

My thought process in solving this problem went something like this. First consider if only two groups are represented. Clearly if it's wolves and cabbages, the transportation is possible. Now consider if sheep is one of the groups. Then one group must be size no more than K or else we can never leave the shore. If the smaller group has size less than K, we just leave that group on the boat (or think of taking them on every trip), and then we can send the other group to the other side one by one. An interesting case happens when the small group is size K. In this case, we can drop them off on the other side, return to the departure shore, move a full boat of the larger group to the other side, swap the cargo with the K we dropped off first, drop this K off at the original departure shore, move another full boat of the larger group to the other side, and then finally move the original K back over to the other side. Therefore if the smaller group is size K, the transportation is possible if and only if the larger group is size no more than $2 \cdot K$.

This gives a complete solution for two represented groups. Now consider if all three groups are represented. Then we must take either all the sheep or all of both the wolves and cabbages on the first trip, or else something is getting consumed when we're on the boat. Then if either the sheep or the wolves +

cabbages are size less than K, we again can shuttle the other group one by one as before. Further, if the sheep or wolves + cabbages are size equal to K, we can also perform the same shuffling strategy as before. At this point, we've covered all cases, since either both sheep and wolves + cabbages are size greater than K, or one of them is size less than or equal to K.

Implicit in the above solution is the observation that wolves and cabbage are interchangeable; they both cannot be left with sheep and will not consume one another when left alone. A correct case statement is shown below.

- If $\min(S, W + C) < K$, answer `YES`.
- If $\min(S, W + C) = K$ and $\max(S, W + C) \leq 2K$, answer `YES`.
- Else, answer `NO`.

Problem B - Maximum Subarrays

Author: Bryce Sandlund. Solved/ Tries: 8/57.

Hindsight I should've ranked this problem as harder than Problem C, but having developed it myself it was harder to gauge the difficulty. The problem is a generalization of the maximum subarray problem, where you are asked to find a single contiguous subarray whose sum is maximum. An $O(nk)$ solution gets accepted, but in fact $O(n)$ is indeed possible [1].

The $O(nk)$ solution is a simple dynamic program. First, recall the solution to the single subarray dynamic program. It defines the following states

$$\texttt{BestInclude}(i) := \text{The largest contiguous sum of } A[1:i] \text{ that includes index } i.$$
$$\texttt{Best}(i) := \text{The best solution in array } A[1:i].$$

and the following recurrence

$$\texttt{BestInclude}(i) := \max(0, \texttt{BestInclude}(i-1)) + A[i]$$
$$\texttt{Best}(i) := \max(\texttt{Best}(i-1), \texttt{BestInclude}(i)).$$

We can fairly easily parameterize the single subarray dynamic program into a multiple subarray dynamic program. Define

$$\texttt{BestInclude}(i, j) := \text{Largest sum of } j \text{ subarrays in } A[1:i], \text{ including index } i.$$
$$\texttt{Best}(i, j) := \text{Largest sum of } j \text{ subarrays in } A[1:i].$$

Then we can utilize the following recurrence

$$\texttt{BestInclude}(i, j) := \max(\texttt{BestInclude}(i-1, j), \texttt{Best}(i-1, j-1)) + A[i]$$
$$\texttt{Best}(i, j) := \max(\texttt{Best}(i-1, j), \texttt{BestInclude}(i, j)).$$

There are $O(nk)$ states and each state takes $O(1)$ time to compute, so we get $O(nk)$ time complexity. Put in the correct base cases and a bottom-up DP takes about 40 lines of code. Make sure to use 64-bit integers since the input bounds can overflow a 32-bit integer.

Problem C - Rational Ratio

Author: David Poplawski. Solved/ Tries: 40/454.

This problem asks you to find the fraction associated with a repeating decimal. What makes a decimal repeat? Do the long division of 1/7, and you see that after finding the first six decimal digits, .142857, you are left with a remainder of .000001 (10^{-6}). Then if you keep doing long division, clearly the next six digits will give you the quotient .142857142857, and this decimal will keep repeating. You can write an equation to describe the behavior. It is first easiest described with a fraction that has a single repeating digit. Suppose x/y has a single repeating digit in its decimal representation. Then x and y satisfy

$$10x - \left\lfloor \frac{10x}{y} \right\rfloor y = x. \tag{1}$$

We get this because, when we do long division, since $y > x$, we first find the largest factor of y that fits into $10x$, subtract this off, and then we are immediately left with a remainder equal to our original x. This causes the quotient to be a single repeating decimal digit.

We can generalize the formula to more than one repeating digit by increasing the power of 10 to however many digits repeat. For 6 repeating digits, we get

$$10^6 x - \left\lfloor \frac{10^6 x}{y} \right\rfloor y = x.$$

This is great progress, but does not solve the original problem. We need to supply our formula with the repeating decimal digits. What is $\left\lfloor \frac{10x}{y} \right\rfloor$ in (1)? It was the largest factor of y that fit into $10x$. This is the repeating digit that we write in the quotient. Take $1/3 = .\overline{3}$ as an example. Then (1) says

$$10x - 3y = x,$$

which we can solve to get $x/y = 3/9 = 1/3$. More generally, let d be the repeating decimal digits (so $d = 142857$ for 1/7), and n the number of digits that repeat. Then we get

$$10^n x - dy = x.$$

Rearranging, we get

$$\frac{x}{y} = \frac{d}{10^n - 1}.$$

This gives a complete solution when the input is any repeating decimal. However, the problem can have a whole number part of three digits, and some fractional part that does not repeat. We can just subtract this part off and multiply by a power of 10 to reduce to the above formula. This gives us a fraction, which we can then divide by the same power of 10 and add the subtracted part back in to get our final answer.

Make sure to reduce the fraction via a greatest common divisor subroutine, and be wary of $x/1$, which is possible with the repeating decimal $.\overline{9}$.

I was personally a bit surprised with how many attempts and accepted submissions there were to this problem, since it took a bit of math to figure out. On a sidenote, a couple days before the contest, it was discovered you could brute force the problem by trying all possible denominators counting up from 1, and solving for the numerator. The input was changed to require solutions to deal with 11 repeating digits, which makes the denominator brute-force TLE, but also requires any solution to use 64-bit integers.

Problem G - Tima goes to Xentopia

Author: Vaibhav Tulsyan. Solved/ Tries: 2/37

This problem asks you to find the shortest path in a graph using an exact number of colored edges. It is a great example of the general graph problem-solving paradigm: build graphs, not algorithms [3, p. 222]. In this case, you want to build a graph that incorporates the number of red and blue edges that have been visited thus far into your search space. The input constraint $k_1 \cdot k_2 \leq 800$ gives a hint as to how this can be done.

The idea is to duplicate the original set of vertices $(k_1+1)\cdot(k_2+1)$ times. You now have a layered graph, where each vertex is represented by a tuple (u, x, y), indicating you are at vertex u, and you have seen x red edges and y blue edges. A red edge $u-v$ takes you from (u, x, y) to $(v, x+1, y)$, a blue edge from (u, x, y) to $(v, x, y+1)$, and a white edge from (u, x, y) to (v, x, y). Adding the appropriate edges in all the layers, we get a new graph with $O(Mk_1k_2)$ edges. We then run dijkstra on this graph to determine the distance from $(S, 0, 0)$ to (T, k_1, k_2). The time complexity is $O(Mk_1k_2 \log(Mk_1k_2))$, which is at most about 17 million iterations.

I thought the solution to this problem was straightforward, so I was surprised only two teams managed to solve it. That being said, implementing it was perhaps tricky, and several of our judge solutions needed updates before agreeing.

Problem A - Pokegene

Author: Nalin Bhardwaj. Solved/ Tries: 0/108

This problem gives you a database of up to 200 000 strings and asks you to answer up to 200 000 queries on this database. Each query specifies a subset of the database and an integer L. You are to compute the number of strings that are prefixes of exactly L of the strings in the query subset. It is given that the total length of strings does not exceed 200 000 characters and the sum of K over all queries does not exceed 1 000 000.

The numbers in this problem are intimidating. Many of the wrong submissions got TLE. You must devise an algorithm that is near-linear (about $O(n \log n)$) in the sum of K over all queries and the sum of the lengths of all the strings. This means you cannot even read all the characters of each string in a query subset. Therefore we must do some preprocessing on the original database.

The judges found three different solutions to this problem. The first two begin with the same framework. First sort the strings in the database. For a given query subset, you can get its sorted order by mapping to the indices of the sorted strings. Then, if L strings share a prefix, these strings will be in order in the sorted subset. You can now turn the problem into finding the longest common prefix (LCP) between strings i, $i + L - 1$, and $i + L$ in the ordered query subset. Specifically, any prefix that is shared between exactly L of the strings in the query subset is a prefix of a unique string i of length larger than $LCP(i, i + L)$ and less than or equal to $LCP(i, i + L - 1)$. We can sum the difference $LCP(i, i + L - 1) - LCP(i, i + L)$ (when positive) over all i to answer the query. Thus we have reduced each query to $O(K)$ LCP queries on any two strings in the database.

We cannot spend time proportional to the length of the strings to answer an LCP query, since there are too many LCP queries that need to be answered. One way to avoid this is to use hashing. Compute a rolling hash for all prefixes of each string in the original database. This takes time linear in the total length of all strings, and allows us to compare prefix hashes to determine if a prefix matches in constant time. Then, to determine the length of the longest common prefix of strings i and j, we do binary search with the hashes. Let M be the sum of all the lengths of all strings in the original database. The time complexity of this approach is upper bounded by $O(\sum K \log M + M \log M)$. However, it is probabilistic. If the hashes of two prefixes match, we cannot afford to spend linear time to check the strings, so we must assume they are equal. You can use multiple hash functions to decrease this probability, but this was not needed in our judge solutions.

For a deterministic approach, you can answer the LCP queries by reducing to lowest common ancestor (LCA) queries. Make a trie of the original database of strings. Then, $LCP(i, j)$ is the same as the depth of $LCA(i, j)$ in the trie. This solution is as fast or faster than the hashing solution, but depending on your familiarity with LCA computation, might require more thought or more code.

The final solution uses a more general technique to perform computation on a subtree of a tree and was adapted from [2, problem D]. The first observation is that in the trie of strings for a particular query, the answer is the number of nodes with exactly L strings that end in the subtree associated with this node. It is too slow to construct the trie for each query, but instead, we can construct the trie for the entire database and then construct an "auxiliary tree" that contains only the $O(K)$ nodes important to the query. These are the nodes where a query string ends, and the pairwise LCA's of all of these nodes. Intuitively, this is the same as removing the nodes not in the query and then compressing all

paths. We can compute all pairwise lowest common ancestors by performing LCA queries on nodes consecutive in the DFS order of the full trie. We can then perform a modified count on the auxiliary tree for a given query. The time complexity of this approach is either $O(\sum K \log M + M)$ or $O(\sum K + M)$, depending on your LCA algorithm. The original sort on the entire database of strings is essentially replaced by a trie-based radix sort [4].

There was one submission to this problem that got accepted by Matthew Schallenkamp, a previous World Finalist from SDSMT, using still a different algorithm. His solution was to construct the full trie, but compress paths as is done in the auxiliary tree solution. He counts subtrees containing L nodes in the full trie by querying each string in the compressed trie. I believe this solution has worst case time complexity $O(\sum K\sqrt{M})$. This is because the time complexity to query the trie is dependent on the longest compressed path. Every step taken corresponds to a string in the database of length at least as long as the depth we have traversed thus far. Thus the maximum depth in the compressed trie cannot exceed $O(\sqrt{M})$. We intended to kill a solution with time complexity dependent on $\sqrt{\sum K}$ (viewable in the "time_limit_exceeded" folder of pokegene submissions), but did not see this approach. His c++ implementation runs in 6.5 seconds, while our intended c++ solutions run in about half a second. The time limit of 8 seconds for the problem was set as three times the slowest judge solution, which was written in Java.

We thought more teams would solve this problem, but it seems LCA plus the thought process to get to this point may have been out of reach for this year's teams. This would be something to make sure to master for the World Finals and next year's Regional.

Problem H - New Salaries

Author: Volodymyr Lyubinets. Solved/ Tries: 0/51

This was, in my opinion, by far the hardest problem on the set. Let X_i denote a random variable corresponding to salary i, uniformly distributed between L_i and R_i. The problem asks you to compute

$$\mathbb{E}\left[|X_i - X_j|\right]$$

for all pairs i, j.

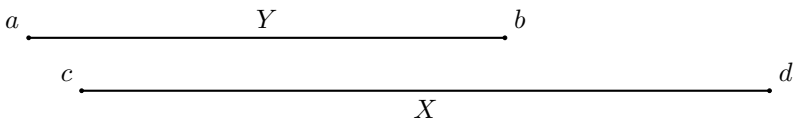
First consider the two salary case. For simplicity, let's give the larger salary range (salary 2) random variable X, and the smaller salary range (salary 1), random variable Y, without the indexing. We can break the absolute value into the following expectations

$$\begin{aligned}\mathbb{E}\left[|X - Y|\right] = \Pr\left[X > Y\right] \cdot \mathbb{E}\left[X - Y \;\middle|\; X > Y\right] \\ + \Pr\left[X < Y\right] \cdot \mathbb{E}\left[Y - X \;\middle|\; X < Y\right].\end{aligned}$$

If the intervals for the two salaries do not overlap, then $\Pr[X < Y] = 0$, and the expression becomes

$$\mathbb{E}[|X - Y|] = \mathbb{E}[X - Y] = \mathbb{E}[X] - \mathbb{E}[Y],$$

where in the last step we use linearity of expectation. This gives the case where you just subtract their expected values. Now consider the overlapping case, pictured below.



When I first solved the problem, I conditioned on the value of salary 2 (random variable X), and eventually reduced to the case both salary 1 and 2 fall in the range $[c, b]$. You can get the correct values using this method, but the algebra gets very cumbersome. If we manipulate the random variables X and Y in the correct way, it turns out we can leverage the disjoint interval case to make the math much easier going forward.

Observe

$$\mathbb{E}[X - Y] = \Pr[X > Y]\mathbb{E}\left[X - Y \mid X > Y\right] + \Pr[X < Y]\mathbb{E}\left[X - Y \mid X < Y\right].$$

This is quite similar to $\mathbb{E}[|X - Y|]$. If we apply linearity of expectation again, we can get

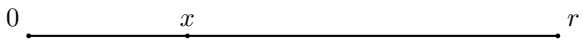
$$\mathbb{E}[X - Y] = \Pr[X > Y]\mathbb{E}\left[X - Y \mid X > Y\right] - \Pr[X < Y]\mathbb{E}\left[Y - X \mid X < Y\right].$$

Now we can substitute to get

$$\mathbb{E}[|X - Y|] = \mathbb{E}[X - Y] + 2\Pr[X < Y] \cdot \mathbb{E}\left[Y - X \mid X < Y\right].$$

We know $\mathbb{E}[X - Y] = \mathbb{E}[X] - \mathbb{E}[Y]$, and $2\Pr[X < Y] \cdot \mathbb{E}\left[Y - X \mid X < Y\right]$ is actually also easy to calculate. Random variable X is only less than Y when both are in range $[c, b]$ (here, we know $b < d$ and $c > a$ due to the monotonicity condition on the salaries). Then $2 \cdot \mathbb{E}[Y - X \mid X < Y]$ is the same as the expected difference of two random variables in range $[c, b]$. This is the same case we would have reduced to with conditioning.

We must find the expected difference of two uniform random variables in some range $[l, r]$. Since we are computing a difference, we may as well normalize to $l = 0$, since any offset will get subtracted away. We can solve this with an integral. Consider when one of the variables has value x.



There is an x/r probability the other variable falls to the left of x, with expected payout $x - (x + 0)/2 = x/2$. There is an $(r - x)/r$ probability the other

variable falls to the right of x, with expected payout $(x+r)/2 - x$. Integrating over all x and normalizing, we get

$$\frac{1}{r}\int_0^r \left(\frac{x}{r}\left(\frac{x}{2}\right) + \frac{r-x}{r}\left(\frac{x+r}{2} - x\right)\right) dx$$

which solves to

$$\frac{r}{3}.$$

Thus the expected difference of two uniform random variables in range $[l, r]$ is $\frac{1}{3}(r-l)$. We should expect a simple expression like this, with dependence only on the size of the range. For comparison, I knew before solving the integral, the expected maximum of two uniformly random variables in range $[0, r]$ is $\frac{2}{3}r$.

Now, using our math from before, the expected difference in the two overlapping salary case is

$$\mathbb{E}\left[|X-Y|\right] = \mathbb{E}\left[X\right] - \mathbb{E}\left[Y\right] + \Pr\left[X, Y \in [c,d]\right] \cdot \mathbb{E}\left[|X-Y| \;\middle|\; X, Y \in [c,d]\right] \quad (2)$$

$$= \frac{c+d}{2} - \frac{a+b}{2} + \frac{(b-c)^3}{3(b-a)(d-c)}. \quad (3)$$

We could run this formula on all pairs of intervals and get the correct result. I believe several solutions in the contest got to this point. But with $n = 100\,000$, this solution gets TLE. We must find a way to take (3) and apply it in a sweeping fashion, simultaneously calculating the corresponding values for many salary pairs in each step.

Let's convert (3) back to the case with a general number of salaries. We get

$$\mathbb{E}\left[|X_i - X_j|\right] = \frac{L_i + R_i}{2} - \frac{L_j + R_j}{2} + \frac{(R_j - L_i)^3}{3(R_j - L_j)(R_i - L_i)}$$

if the salary ranges for i and j intersect, and

$$\mathbb{E}\left[|X_i - X_j|\right] = \frac{L_i + R_i}{2} - \frac{L_j + R_j}{2}$$

otherwise. Let d denote the smallest index that intersects salary i. Then the contribution of adding salary i to the final sum is

$$\sum_{j<i} \mathbb{E}\left[|X_i - X_j|\right] = i\frac{L_i + R_i}{2} - \sum_{j<i} \frac{L_j + R_j}{2} + \sum_{d \le j < i} \frac{(R_j - L_i)^3}{3(R_j - L_j)(R_i - L_i)}. \quad (4)$$

We can maintain $\sum_{j<i} \frac{L_j+R_j}{2}$ as a rolling sum, but the last sum has dependencies on both salaries i and j and so is problematic. Expand the cube in the numerator.

$$(R_j - L_i)^3 = R_j^3 - 3R_j^2 L_i + 3R_j L_i^2 - L_i^3.$$

Rewrite (4) to use this expansion, factoring out all dependencies on i.

$$\sum_{j<i} \mathbb{E}\left[|X_i - X_j|\right] = i\frac{L_i + R_i}{2} - \sum_{j<i} \frac{L_j + R_j}{2} \tag{5}$$

$$+ \frac{1}{R_i - L_i} \sum_{d \le j < i} \frac{R_j^3}{3(R_j - L_j)} \tag{6}$$

$$- \frac{L_i}{R_i - L_i} \sum_{d \le j < i} \frac{R_j^2}{R_j - L_j} \tag{7}$$

$$+ \frac{L_i^2}{R_i - L_i} \sum_{d \le j < i} \frac{R_j}{R_j - L_j} \tag{8}$$

$$- \frac{L_i^3}{R_i - L_i} \sum_{d \le j < i} \frac{1}{R_j - L_j}. \tag{9}$$

By maintaining the rolling sums of (5), (6), (7), (8), and (9) with corresponding smallest intersecting index d, the problem can be solved in linear time. Make sure to special case intervals of length 0, and divide by n^2 throughout to reduce loss of precision.

Acknowledgements

A big thanks to the problem developers. They spent a lot of time developing test cases and making sure they were all Kattis-compliant. Several also wrote solutions to other developers' problems. Also, a big thank you to Menghui Wang, Antonio Molina, Nick Wu, and Aditya Sarode for additional problem solutions and test cases. Additionally, thank you very much to the Kattis team! Greg Hamerly reviewed the problem set, correcting important ambiguities, and also made us aware of discrepancies in what our input validators checked for and what our problem specifications gave. They also coordinate the contest itself, dealing with some login complications and slight mishaps on our end. They do this all free of charge for ICPC Regionals. Thank you Kattis!

References

[1] Bengtsson, F., Chen, J. *Computing maximum-scoring segments optimally.* Research Report, Luleå University of Technology (2007). `http://ltu.diva-portal.org/smash/get/diva2:995901/FULLTEXT01.pdf`

[2] Codeforces. `https://codeforces.com/blog/entry/22832`.

[3] Skiena, S. S. *The Algorithm Design Manual.* Second Edition. 2008. Springer-Verlag London.

[4] Wikipedia. `https://en.wikipedia.org/wiki/Radix_sort#trie-based_radix_sort`.