The 2020 ICPC Asia Shenyang Regional Programming Contest

Solution Sketches

Disclaimer *This is an analysis of some possible ways to solve the problems of the 2020 ICPC Asia Shenyang Regional Programming Contest. They are not intended to give a complete solution, but rather to outline some approaches that can be used to solve the problems. If some of the terminology or algorithms mentioned below are not familiar to you, your favorite search engine should be able to help. Should you find any error, please send an e-mail to* `20194627@stu.neu.edu.cn` *about it.*

— Judges of the 2020 ICPC Shenyang

Summary

The contest started out with the team *Unpredictable* from Beijing Normal University submitted the first accepted solution in 2 minutes. Shortly afterward, the team *Inverted Cross* from Peking University became the leader by successively solving several problems. They led the scoreboard by 7 solved problems in the first hour, while the second place only solved 4. The margin had once enlarged to 9 versus 5 in the next hour, and the great advantage secured them the champions. Later, they slowed down due to the ascending problem difficulty, and ultimately ended up with 12 solved.

Congratulations to the team *Inverted Cross* from Peking University, the 2020 ICPC Shenyang Champions, for their outstanding performance during the contest!

The judges agreed that this problem set could be divided into a relatively easier part (A, C, D, E, F, G, H, I, K, M) and a tough part (B, J, L). There did be a wide gap between the two parts, but the judges expected that the easier part should be tractable for most of the teams.

In all, the contestants made 3139 effective submissions, 952 of which were accepted. Submissions to each problem are listed in the following table.

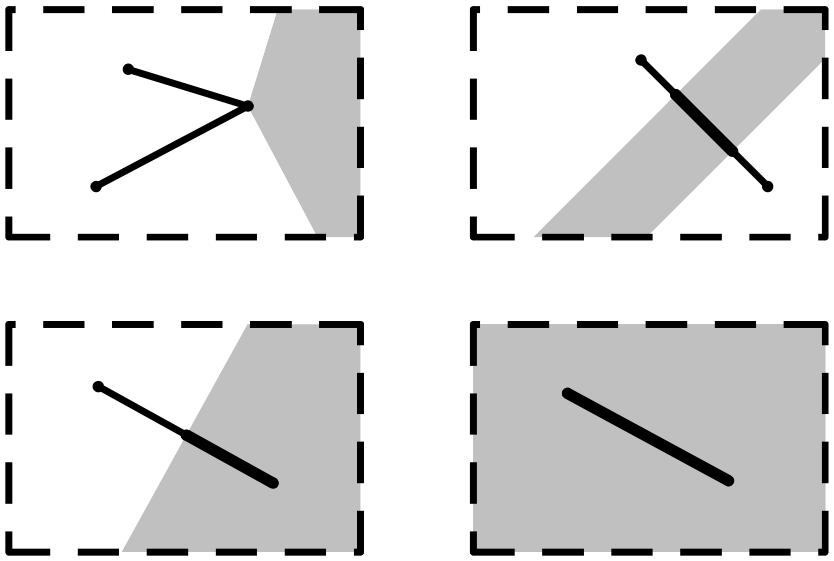
Problem	A	B	C	D	E	F	G	H	I	J	K	L	M
Solved	5	1	13	61	7	262	274	46	57	1	201	0	24
Submissions	55	12	60	292	36	732	290	126	619	51	457	18	391

A note about solution sizes: below the size of the smallest judge solutions for each problem is stated. It should be mentioned that these numbers are just there to give an indication of the order of magnitude. The judge solutions were not written to be minimal (though some of us may have a tendency to overcompactify our code) and it is trivial to make them shorter by removing spacing, renaming variables, and so on.

Problem A: The Grand Tournament

First solved after 139 minutes.
Shortest judge solution: 3689 bytes.

This is more of a case analysis, though the problem per se is geometric. What is not hard to find is, when the two segments don't share a common point, the answer must be zero. This can help us reduce the search space.



Actually, there are only four cases, as shown in the figure above, with a nonzero answer. For each case, the valid area is formed by the intersection of several half-planes. We can compute the area with some off-the-shelf half-plane intersection algorithm or simply derive the formulas on a per-case basis.

Problem B: Whispers of the Old Gods

First solved after 214 minutes.
Shortest judge solution: 5643 bytes.

This is a formal language problem and is one of the most challenging problems in the set. There is a standard algorithm to match a string S exactly with a regular expression R in $O(|S||R|)$ time: build the nondeterministic finite automaton of R, which contains $O(|R|)$ states and transitions, then simulate the execution of the NFA in $O(|S||R|)$ time.

The algorithm can be adapted to compute the "edit distance" between a string S and a regular expression R. Let Q be the NFA states, Σ be the character set, $\Delta : Q \times \Sigma \to 2^Q$ be the transitions, we may build a directed weighted graph with vertices $\{0, 1, \cdots, |S|\} \times Q$ and edges:

- $(i, q) \to (i + 1, q)$ with weight 1 (to remove the character S_i);
- $(i, q) \to (i, q')$ where $q' \in \Delta(q, c)$ for some $c \in \Sigma$, with weight 1 (to insert the character c);
- $(i, q) \to (i+1, q')$ where $q' \in \Delta(q, c)$ for some $c \in \Sigma$ with weight 1 (to replace the character S_i with c);
- $(i, q) \to (i + 1, q')$ where $q' \in \Delta(q, S_i)$ with weight 0 (to match the character S_i).

The answer is the shortest path from $(0, s)$ to any $(|S|, t)$, where s is the initial state and t is an accept state.

It might be easier to directly use the ϵ-NFA by Thompson's construction than converting ϵ-NFA to NFA or using Glushkov's construction. Still, some slight modifications should be made to the abovementioned algorithm to handle the ϵ-transitions.

Problem C: Mean Streets of Gadgetzan

First solved after 56 minutes.
Shortest judge solution: 2348 bytes.

This problem is called Horn-SAT, which is one of the two major subclasses of Boolean satisfiability problems solvable in polynomial time (the other being 2-SAT).

We will maintain a set of "known facts", the truth assignments to some propositions that we deduce. Initially, we add all unconditional testimonies to the set. When a new fact is deduced:

- if some proposition gets both confirmed and denied, the testimonies must have conflict.
- otherwise, if we deduce some proposition is true, we have to inspect all conditional testimonies containing the proposition as an antecedent. If a conditional testimony has all its antecedents confirmed, we shall deduce its consequent and add it to the set of known facts.

We can iterate this process until no more facts can be deduced.

Having figured out all facts that we can deduce, some conditional testimonies might still not be used because they have false or undetermined antecedents. Setting all undetermined propositions to false will not violate them, and we thus get an assignment respecting all testimonies.

To implement the entire algorithm in linear time, we have to

- compute, for each proposition, a list of conditional testimonies containing the proposition as an antecedent; and
- bookkeep the number of unconfirmed antecedents for each conditional testimony.

When a proposition gets confirmed for the first time, we iterate over the list of testimonies, decrement its count, and if the count touches zero, add its consequent to the set of known facts.

Problem D: Journey to Un'Goro

First solved after 26 minutes.
Shortest judge solution: 559 bytes.

Let p_i be the number of red steps in the first i steps, the number of red steps from the ith step to the jth step is odd means that p_{i-1} and p_j are of different parity. To maximize the number of "satisfied" pairs, the odd and the even numbers in $p_0(=0), p_1, p_2, \ldots, p_n$ should differ by at most one, thus the maximum number of "satisfied" pairs is $\lfloor \frac{n+1}{2} \rfloor \lceil \frac{n+1}{2} \rceil$, and the lexicographically smallest 100 colorings can be easily generated by depth first search with pruning.

Problem E: Knights of the Frozen Throne

First solved after 86 minutes.
Shortest judge solution: 1648 bytes.

Let $t_i^{(j)}$ be the jth request (in nondecreasing order) from ith player. The candidate set of optimal values of X is $\left\{ t_i^{(j)} - t_i^{(j-1)} + 1 \middle| 1 \le i \le m, 2 \le j \le k_i \right\}$; any other value must have the same cache hit as some candidate but strictly cost more in memory. There are at most 500 000 candidates, so we can calculate the CPU and memory cost for each candidate in increasing order.

For CPU cost, initially, ZSC has to load data for every $t_i^{(j)}$. With the candidate answer X increasing, the cache hit rate increases, and ZSC don't have to load data for request $t_i^{(j)}$ as long as $t_i^{(j)} - t_i^{(j-1)} < X$. We can maintain the number of data loads in every second as X increases.

For memory cost, we can consider the lifetime of data in memory as intervals, with $[l, r]$ representing that one player's data is loaded at l and dropped at time r, and the total memory cost is $a \times \sum(r - l)$. Initially, when $X = 1$, there are about $\sum_{i=1}^{m} k_i$ intervals of length 1, excluding those requests sent by the same user in the same second. With the candidate answer X increasing, the right end of each interval will extend, and two adjacent intervals of a player will merge into one if they touch. This can also be done easily by maintaining the total length and the number of intervals.

The overall time complexity is $O(K \log K)$, where $K = \sum_{i=1}^{m} k_i \leq 500\,000$.

Problem F: Kobolds and Catacombs

First solved after 7 minutes.
Shortest judge solution: 443 bytes.

We can split at some point if and only if the maximum before the split point is less than or equal to the minimum after the point. Hence we can compute the prefix maximum and suffix minimum of the array, and decide if we can split at each point.

Problem G: The Witchwood

First solved after 2 minutes.
Shortest judge solution: 299 bytes.

This is the easiest one in the problem set. Just sum up the k greatest pleasures.

Problem H: The Boomsday Project

First solved after 36 minutes.
Shortest judge solution: 872 bytes.

This is a fairly easy dynamic programming problem. Note that although the input rental records are given in some kind of "compacted form", the total number of rents is no more than 300 000, and we can materialize all rental records in ascending date order and store them in an array.

Let $f(r)$ be the minimum cost to complete the first r rental records. For the ith kind of discount card, we have to find the minimum $g(i, r)$, such that the rental card can cover the rents $g(i, r) + 1, g(i, r) + 2, \cdots, r$. The state transition can thus be written

$$f(r) = \min_{1 \leq i \leq n} f(g(i, r)) + c_i.$$

The value of $g(i, r)$ should simultaneously satisfy the following conditions:

- $r - g(i, r) \leq k_i$;
- the rental time of the $(g(i, r) + 1)$th rent is later than d_i days before the rth rent.

While it is possible to binary search for $g(i, r)$, we can prove that $g(i, r) \geq g(i, r - 1)$ and just maintain n monotonic pointers, which saves us a log factor.

Problem I: Rise of Shadows

First solved after 13 minutes.
Shortest judge solution: 229 bytes.

Consider the relative motion between the two hands, the minute hand goes around the full circle with length HM at the constant speed $H-1$ per minute relative to the hour hand. The problem then comes to count the number of integral moments $t \in [0, HM)$, such that

$$t(H-1) \bmod HM \le A \tag{1}$$

or

$$t(H-1) \bmod HM \ge HM - A. \tag{2}$$

A simple observation, which follows directly from basic number theory, says that the function $f(t) = t(H-1) \bmod HM$, repeats its values on intervals of HM/G for $G = \gcd(H-1, HM)$ times. Dividing out G on both sides of (1) and (2), we have

$$t(H-1)/G \bmod HM/G \le A/G \tag{3}$$

or

$$t(H-1)/G \bmod HM/G \ge (HM-A)/G \tag{4}$$

and the left-hand side of either inequality yields every integer in $\mathbb{Z}_{HM/G}$ exactly once in every interval of length HM/G. There are $\lfloor A/G \rfloor + 1$ integer solutions for (3) and $\lfloor A/G \rfloor$ integer solutions for (4). Hence the answer is $G(2\lfloor A/G \rfloor + 1)$.

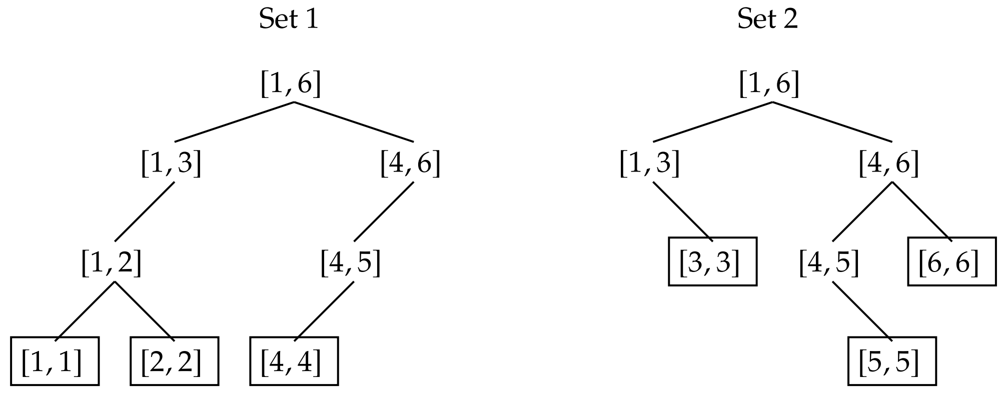
There is a corner case when $A = HM/2$, whose answer is simply HM, since the angle between the hour hand and the minute hand is always less than or equal to π.

Problem J: Descent of Dragons

First solved after 250 minutes.
Shortest judge solution: 2239 bytes.

This problem can be solved in various ways, and we only describe one run in $O(q \log^2 n)$ time here, though some good implementations with higher complexities may pass.

We maintain n sets, the ith of which contains the indices of dragons with level i, with n parallel space-partition trees of the same partition rule. For example, if the levels of dragons are `[1,1,2,1,2,2]`, the trees look like as below.



To support the "changing all x in $[l, r]$ to $x+1$" operation, recall that every interval $[l, r]$ can be decomposed into at most $O(\log n)$ disjoint subintervals in a space-partition tree. Such decomposition can be done in $O(\log n)$ time. What we need now is to transplant the $O(\log n)$ subtrees in the xth tree to the same positions in the $(x+1)$th tree. During transplantation, some nodes may already exist in the new tree, which may be dealt with by a Merge procedure.

Algorithm 1 Merge(from, to)

```
1: if both from and to are NIL then
2:     return NIL
3: end if
4: if one of from and to is NIL then
5:     return the other
6: end if
7: return newNode(Merge(from.left, to.left), Merge(from.right, to.right))
```

Note that the above procedure only visits the intersection of the two subtrees, and only one copy is preserved after merging. Hence, the procedure is free of charge, which follows from some amortized analysis by defining the potential function as the total number of nodes in all trees. In all, the change operation can be done in amortized $O(\log n)$ time.

To efficiently support the range maximum query, for each interval in the space partition, we additionally maintain a pointer to the node representing that interval in the maximally-numbered set. For example, in the configuration above, $[1,3]$ and $[4,5]$ point to the ones in Set 2, but $[1,2]$ and $[4,4]$ point to the ones in Set 1. These pointers should be modified accordingly during transplantation.

Given an interval $[l,r]$, we first break it down into $O(\log n)$ subintervals. For each subinterval, we find the corresponding node in the maximally-numbered set and climb up via parent pointers to the root to identify which set the node is located in. Each subinterval requires $O(\log n)$ time to find its root, so the total time complexity is $O(\log^2 n)$.

Problem K: Scholomance Academy

First solved after 28 minutes.
Shortest judge solution: 702 bytes.

At first glance, this problem appears to be overwhelming, but the teams will find it not hard after reading the statement carefully.

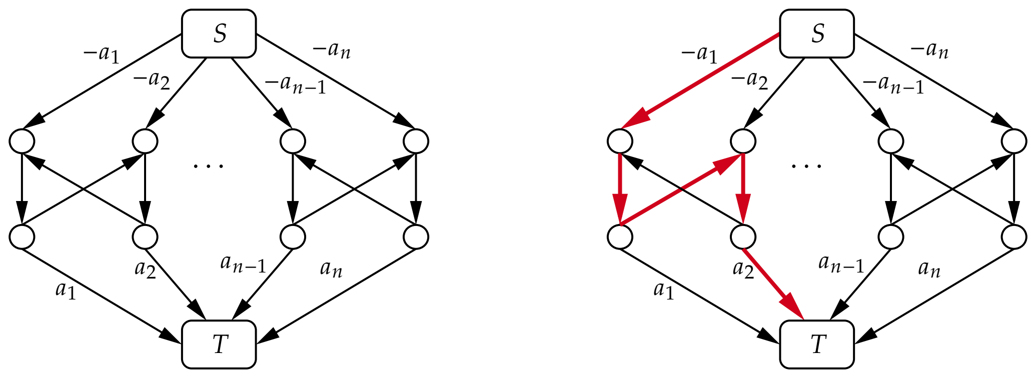
First, we group all instances by their scores. Then, we lift the value of θ. The classifier changes only when θ reaches the score of some group. We update the TPR and FPR in such case. The ROC can be written as a piecewise constant function, and the AUC can be computed accordingly.

Problem L: Forged in the Barrens

First solved after N/A minutes.
Shortest judge solution: 2524 bytes.

This is probably the hardest one in the problem set...

The problem can be rephrased as follows: for every k, find k disjoint intervals $[l_1,r_1]$, $[l_2,r_2]$, $\cdots$, $[l_k,r_k]$, such that $\sum_{i=1}^{k} |a_{l_i} - a_{r_i}|$ is maximized. This can be solved by max cost flows, as shown in the figure below. However, the network is too large (around 400 000 nodes) to fit in the time limit if we use the naïve successive longest paths algorithm. That said, the network flow scheme will still give us some hints about the problem.



Let us consider a simpler algorithm. Let $f_s(m, x)$ be the maximum sum for the first m beacon towers with x pairs of towers selected, and there's a standalone $+a_i$ included at the end if $s = 1$, nothing else included if $s = 0$, or a standalone $-a_i$ included at the end if $s = -1$. This gives us an $O(n^2)$ dynamic program, still too slow to pass.

Now let's speed up the algorithm. We may consider a subarray of the beacon towers, rather than just a prefix. Let $g_{s_l,s_r}(l, r, x)$ be the maximum sum if we have already selected x pairs of towers therein, where s_l, s_r denote whether any standalone tower is included in the left or right of the subarray. Also, $g_{s_l,s_r}(l, r, x)$ can be computed via max-plus convolution: for every $m \in [l, r)$,

$$\begin{aligned} g_{s_l,s_r}(l,r,x) = \max\{ &\max_{x_1+x_2=x} g_{s_l,0}(l,m,x_1) + g_{0,s_r}(m+1,r,x_2),\\ &\max_{x_1+x_2+1=x} g_{s_l,1}(l,m,x_1) + g_{-1,s_r}(m+1,r,x_2),\\ &\max_{x_1+x_2+1=x} g_{s_l,-1}(l,m,x_1) + g_{1,s_r}(m+1,r,x_2),\\ &g_{s_l,s_r}(l,m,x),\quad g_{s_l,s_r}(m+1,r,x)\}. \end{aligned}$$

One key observation is, the function $g_{s_l,s_r}(l, r, x)$ is convex upward with respect to x. This can be derived from the aforementioned max cost flows.

Now the problem boils down to find the max-plus convolution, a.k.a. Minkowski sum, of two convex functions. This is a standard task solvable in linear time, similar to the one merging two ordered sequences.

To put all things together, we have to compute $g_{-,-}(l, r, \cdot)$, by taking $m = (l + r)/2$, and

- compute $g_{-,-}(l, m, \cdot)$ and $g_{-,-}(m + 1, r, \cdot)$ recursively;
- compute $g_{-,-}(l, r, \cdot)$ by computing the Minkowski sum of $g_{-,-}(l, m, \cdot)$ an d $g_{-,-}(m+1, r, \cdot)$ in linear time.

This runs in $O(n \log n)$ from a standard argument.

It is also possible to directly compute the max cost flows, but it requires some data structure work to accelerate finding the augmenting paths.

Problem M: United in Stormwind

First solved after 17 minutes.
Shortest judge solution: 1008 bytes.

We may use a set of A-answered questions, x_i, to represent the ith result. A set S is discriminative if and only there exists no less than k pairs of i, j $(1 \le i < j \le n)$, such that $(x_i \triangle x_j) \cap S \neq \emptyset$, where $A \triangle B = (A \setminus B) \cup (B \setminus A)$. This can be essentially broken down into two parts.

Let N_x denote the number of results equaling x. Firstly, for each question set S', we want to compute the number of ordered pairs i, j $(1 \le i, j \le n)$ such that $x_i \triangle x_j = S$, denoted by $F(S')$. Then,

$$F(S') = \sum_{S' = x \triangle y} N_x N_y.$$

This is a dyadic (XOR) convolution, which can be computed in $O(m2^m)$ time with the help of the fast Walsh-Hadamard transform.

Secondly, for each set of questions S, compute

$$G(S) = \sum_{S' : S' \cap S \neq \varnothing} F(S')$$

which is the actual number of ordered pairs of results that S can tell the difference. Note that

$$G(S) = \sum_{S'} F(S') - \sum_{S' : S' \cap S = \varnothing} F(S') = n^2 - \sum_{S' : S' \subseteq U \setminus S} F(S')$$

where the last term, known as "sum over subsets" or "multidimensional prefix sum" problem, can be solved by some dynamic programming algorithm.

Having computed $G(\cdot)$, we just count the number of nonempty sets S such that $G(S) \ge 2k$.