Problem Tutorial: "Takeover"

Problem author and developer: Ivan Smirnov

Let's track the (multi)set of possible perimeter increases if we add each possible of the remaining facilities. For each facility the corresponding number may only decrease after we perform each new step.

It means that if we want to check if the actual answer is x, and if we can capture some facility in cost of no more than x, then it is definitely not bad to capture it.

It immediately yields a solution with binary search, but binary search is indeed not needed: simply capture a facility that provides the minimum perimeter increase on each step. You may need priority queue to track increases for the facilities located above the campus, to the right of the campus and to the right and above comparing by y, x or $x + y$ respectively.

Problem Tutorial: "Unfair Card Deck"

Problem author: Dmitry Zhukov, **problem developer**: Andrey Khalyavin

There are two approaches to this problem.

The first approach is to calculate the ratio between the weights. Let x and y be two different card types. Let us consider probability that the first x card comes before the first y card. Consider the first moment where either x or y card comes up. Whatever cards come up before that, the probability that the next card is x is $\frac{c_xW_x}{c_yW_y}$ greater than the probability that the next card is y. That means that when we sum up all probabilities, we get that probability of x coming before y is $\frac{c_xW_x}{c_xW_x+c_yW_y}$. We can use game log to easily find out approximations to these probabilities and restore W_i. Note that the smaller the ratio, the less reliable its estimate. So we should sort the cards by c_xW_x first.

We can determine the card with greatest c_xW_x by collecting statistics of the first card in the games. And we can determine next card by choosing the card with greatest ratio if this ratio is large or use the same trick as for the first card if all ratios are small.

The second approach is to find the point with maximum a posteriori probability (MAP). I.e. weights that give maximum probability of the given log occurring. It is easy to calculate logarithm of probability of the game given W_i. Derivative of logarithm is easy to calculate. And we can use a few tricks with accumulation to calculate gradient for the whole game in $O(l)$ time instead of $O(l^2)$ where $l = 30$ is the length of the game. It takes too long to calculate the gradient for all games so we can use stochastic gradient descent instead. Choose only one game and add gradient multiplied by some learning rate. Stochastic gradient descent error converges as $O(1/\sqrt{n})$ where n is the number of steps but it can be speed up. All we need to do is average W_i over all steps. This improves convergence to $O(1/n)$ which is good enough.

Some implementation notes. We need to ensure that our gradients do not blow up and that our problem has good condition number. This can be achieved by multiplying i-th gradient coordinate by W_i^2. The reason is that logarithm derivative is of order $1/W_i$, so if we multiply by W_i^2, it becomes of order W_i. This is exactly what we need such we can have W_i which are very different order of magnitude and this way all coefficients converge at similar speeds. Also we need to restrict gradients so that W_i can't became smaller than zero. Just use $\max(-cW_i, \min(cW_i, grad_i))$ for some constant c. Another problem is that $\log(W_i)$ might not be bounded. So we need to periodically check for large gaps of magnitude and multiply smaller values to keep them in reasonable limits. This process should have some hysteresis otherwise it affects convergence. Naturally, the final averaging should use geometric average rather than arithmetic average to deal with large changes in magnitude.

Problem Tutorial: "Diverse Singing"

Problem author: Maxim Babenko, **problem developer**: Ivan Smirnov

If we formulate the problem in terms of graph theory, we need to find the edge cover of the edge-colored bipartite graph such that in each color the matching is taken.

This problem can be reduced to constrained flow problem (LR-flow). The network will consist of 6 layers:

- The first layer consists of source s;
- The second layer corresponds to people;
- The third layer corresponds to pairs (person, language);
- The fourth layer corresponds to pairs (song, language);
- The fifth layer corresponds to songs;
- The sixth layer consists of source t;

Edges from layer 1 to layer 2 and from layer 5 to layer 6 have lower capacity 1, edges from layer 2 to layer 3 and from layer 4 to layer 5 have upper capacity 1 and edges from layer 3 to layer 4 have no restriction on capacity.

Problem Tutorial: "Pick Your Own Nim"

Problem author: Ivan Smirnov and Maxim Akhmedov, **problem developer**: Ivan Smirnov

We get rid of Alice's heaps. For each her heap we make an extra box for Bob containing only that heap. He has to take exactly one heap from each box, so all Alice's heaps will be necessarily taken.

We treat n-bit numbers as vectors over $\mathbb{Z}_2^n$. For Bob to win, he should pick such heaps that the set of vectors formed by his heaps and Alice's heaps is linearly independent.

The problem reduces to matroid intersection. The domain is the set of all heaps. The first matroid is a matroid of linear independence, the second matroid is a matroid of choice (no two items are taken from the same box).

If the base of the intersection of these matroids has the size of the number of boxes then it is the answer, otherwise there is no solution.

Problem Tutorial: "Permutasino"

Problem author and developer: Maxim Akhmedov

There are different solutions for this problem, we will describe the most general approach.

Reformulate the problem as follows: you are given $n!$ points in R^n and point x, express x as a convex combination of these $n!$ points (i.e. linear combination with coefficient total weight 1 and non-negative coefficients). It is easy to see that the desired combination exists if and only if x belongs to the convex hull of these points and the combination is a certificate of that fact. Such convex hull is called a permutahedron of numbers $1, 2, \ldots, n$.

Let's try to come up with the procedure of checking if x belongs to the convex hull. There is a necessary condition: the total sum of k minimum components of x should be at least $1 + 2 + \ldots + k$ and the total sum of all components should be equal to $1 + 2 + \ldots + n$.

It turns out that this condition is also a sufficient condition (so-called separation oracle for polytope).

Perform a procedure similar to Caratheodory theorem proof: pick arbitrary permutation p as a starting point and find out point y where ray px intersects the boundary of permutahedron (it may be done using the binary search). On that boundary at least one of the inequalities that are checked in the oracle turn into equation. It means that y is divided into two independent parts y_1 and y_2 of smaller size which may be processed recursively. Express y_1 as the convex combination of at most k permutations of length k and and y_2 as the convex combination of at most $n - k$ permutations of length $n - k$. It is possible to combine them into $(n - k) + k - 1 = n - 1$ permutations of length n. Together with the initial permutation, we get the desired convex combination of length n.

In order to deal with the precision issues, after making a binary search it may be useful to find exact coordinates of y by solving the linear equation arising from the most tight inequality.

Problem Tutorial: “Planar Max Cut”

Problem author and developer: Maxim Akhmedov inspired by MIT Combinatorial Optimization course

Consider the actual answer: $V = A \sqcup B$. Consider edges that go from A to B, denote them as $\delta(A, B)$. Note that $\delta(A, B)$ is a bipartite graph, i.e. our problem is reformulated as follows: remove edges of minimum total cost from the graph so that the remaining graph is bipartite.

Observation: planar graph is bipartite iff all of its faces are even cycles. Indeed, they have to be even, and any other cycle decomposes into several faces.

Removal of one edge joins two faces together. Consider all odd faces of the original graph (there is an even number of them, btw). The optimum way to remove edges in order to get the desired result is to distribute odd faces into pairs and connect each pair with a shortest path along the dual graph (recall that the edges of the planar graph correspond to the edges in the dual graph).

Let us sum up: we build a dual graph, calculate shortest distances between the odd faces, and now we need to find out the perfect matching of minimum cost. This problem may be solved in polynomial time, it is a good chance for you to learn this algorithm and put it into your team notebook :)

References:

- `https://theory.stanford.edu/~jvondrak/CS369P/lec6.pdf` — theoretical explanation, requires knowledge of LP theory;
- `https://github.com/zlobober/algorithms/tree/master/min-weighted-matching/` — non-obfuscated detailed implementation in $O(n^4)$ by Zlobober, somehow efficient;

Problem Tutorial: “Battle Royale”

Problem author and developer: Alexey Dergunov

Let $L = 0$ wlog. We describe the time the player with parameters (x_i, a_i) stays alive as a function of M:

$$t_i(M) = \begin{cases} \frac{R-x_i}{R-M} \cdot R + a_i, & M \le x_i \\ \frac{x_i}{M} \cdot R + a_i, & M > x_i \end{cases}$$

We need to find the function $t(M) = \max_i t_i(M)$. Consider the left and the right parts of the functions separately. If we multiply the left part of t_i by M then we get a linear function: $\hat{t}_i(M) = x_i \cdot R + a_i \cdot M$. It is a linear function on M that is present for $x_i \le M \le R$. We proceed these functions in order of increasing x_i and maintain the dynamic convex hull of slopes to find the maximums. This way we find a set of segments (of linear size) with some of $\hat{t}_i$ being the maximum on all the segment.

Now we need to intersect these sets of segments for left and right parts of t_i functions. There are two possible maximums for each subsegment; we may determine which of them is by solving a quadratic equation.

Problem Tutorial: “Jeopardy”

Problem author: Maxim Akhmedov and Ivan Smirnov, **problem developer**: Ivan Smirnov

Suppose that the answer is at most x, replace larger numbers with ones and the remaining with zeroes.

Suppose that there is a row with all ones. In this case first player may leave this row and obtain answer that is larger than x.

Suppose that in any row there is a zero. In this case after each turn of first player second is able to pick a column that does not contain a zero, i.e. the game ends up with a zero cell that corresponds to a number at most x.

So, we obtain a solution with binary search, but the binary search indeed is not needed: the answer is maximum of row minimums.

Problem Tutorial: "Slippers"

Problem author: Maxim Akhmedov, **problem developer**: Ivan Smirnov

First of all, disregard slipper directions. Consider the standard grid graph, it is a bipartite graph. Leave only edges corresponding to the pairs of left and right slippers, find maximum matching in this graph. Suppose its size is k.

If this matching does not cover all cells, the answer is exactly k: it is easy to see that using one uncovered cell as temporary cell, we may arbitrarily rotate any cell.

If this matching covers all cells, the answer is either $k - 1$ (similarly to previous argument) or k. The key idea here is to track the total orientation of all slippers: assign each slipper with the value of 0, 1, 2 or 3 according to its direction, for example, in counter-clockwise order starting from R. It is easy to see that the total orientation modulo 4 is invariant. On the other hand, it may be proven that the total orientation of any perfect matching also does not depend on matching. So, it is enough to check the maximum matching if it has the correct orientation modulo 4, if yes, the answer is k, otherwise it is $k - 1$.

Problem Tutorial: "The Good, the Bad and the Ugly"

Problem author and developer: Ivan Smirnov inspired by `http://puzzling.stackexchange.com`

If we arrive at zero at least once, it is easy to determine of the player in $O(1)$ moves, so the problem is in fact to reach zero in less than $30m$ moves. As we get no response during the intermediate moves, we need to find some deterministic strategy that will eventually move the player of each kind to zero.

Here is an example of the strategy. Make one `+` request, c `-` requests, c^2 `+` requests, so on. One can verify that if c is sufficiently large constant then this strategy will move each kind of player to zero in $O(m)$ steps. Optimal value of c turns out to be around 3.5, is solves all possible testcases in $25m$ steps.