Problem Tutorial: "Virus"

Consider we have x cells in the beginning, so the overall perimeter of the infected area is no more than $4x$. When the cell gets infected because of at least two infected neighbors, the total perimeter of the infected area doesn't increase, because at least two edges become internal, and at most two edges surrounding the new cell can lie on the border. In the end perimeter must be $2(n+m)$, so $4x \geq 2(n+m)$ and $x \geq \lceil \frac{n+m}{2} \rceil$.

We can reach this answer using the following construction: infect cells $(1, n-2i)$ for $i = 0, 1, \ldots, \lfloor \frac{n-1}{2} \rfloor$ and cells $(m-2i, 1)$ for $i = 0, 1, \ldots, \lfloor \frac{m-1}{2} \rfloor$. Then after the first milliseconds all cells in the first row and in the first column will be infected. After all cells in the second row and in the second column will be infected, and so on. We used $\lfloor \frac{m+1}{2} \rfloor + \lfloor \frac{n+1}{2} \rfloor$ cells, one can check that it is always equal to $\lceil \frac{n+m}{2} \rceil$.

Problem Tutorial: "Expected length of the minimum cycle"

Assume we have some permutation p. Consider the following algorithm: take the minimal element which is not chosen yet, write down the length of the cycle it lies in, and throw away all elements of the cycle. Consider we obtain a sequence of lengths of the cycles $a_1, a_2, \ldots, a_k$ after this algorithm. What is the probability of a random permutation p to generate such a sequence?

Let's count the number of such permutations at first. We can choose the first cycle of length a_1 containing 1 in $C_{n-1}^{a_1-1} \cdot (a_1-1)!$ ways (we choose the remaining elements of the cycle and then fix their order in the cycle). This is equal to $\frac{(n-1)!}{(n-a_1)!} = (n-1) \cdot (n-2) \cdot \ldots \cdot (n-a_1+1)$.

Now we have the remaining $(n-a_1)$ elements and need to build a cycle containing the minimal remaining element. By the same argument, number of ways to do it is $(n-a_1-1) \cdot (n-a_1-2) \cdot \ldots \cdot (n-a_1-a_2+1)$. Extending this argument further, we get that the number of such permutations is equal to

$$(n-1) \cdot \ldots \cdot (n-a_1+1) \cdot (n-a_1-1) \cdot \ldots \cdot (n-a_1-a_2+1) \cdot (n-a_1-a_2-1) \cdot \ldots \cdot (n-a_1-a_2-\ldots-a_k+1),$$

which is equal to

$$\frac{n!}{n \cdot (n-a_1) \cdot (n-a_1-a_2) \cdot \ldots \cdot (n-a_1-a_2-\ldots-a_{k-1})} = \frac{n!}{a_k \cdot (a_k+a_{k-1}) \cdot \ldots \cdot (a_1+a_2+\ldots+a_k)}.$$

So the probability is equal to

$$\frac{1}{a_k \cdot (a_k+a_{k-1}) \cdot \ldots \cdot (a_1+a_2+\ldots+a_k)}.$$

Let's now fix $1 \leq x \leq n$ so that all $a_i \geq x$ and calculate the sum of all such fractions — q_x. If we have these values, we can simply calculate the answer:

$$E(minimal\_cycle) = \sum_{x=1}^{n} x \cdot P(minimal\_cycle = x) =$$

$$= \sum_{x=1}^{n} P(minimal\_cycle \geq x) = \sum_{x=1}^{n} P(all\ cycles \geq x) = \sum_{x=1}^{n} q_x.$$

How to calculate q_x? Let's use dynamic programming $dp[m]$ — the sum of such fractions that $\sum_{i=1}^{k} a_i = m$ and $a_i \geq x$. Then the formula is:

$$dp[m] = \sum_{i=0}^{m-x} \frac{dp[i]}{m}.$$

So we can calculate the dp array and also the array of its prefix sums to obtain the answer for the next m in $O(1)$. For a fixed n we use $O(n)$ to calculate dynamic programming, so the overall complexity is $O(n^2)$. Note that we need to precalculate the inverse elements of numbers $1, 2, \ldots, n$ in the $\mathbb{Z}_p$ field so that we can divide by m in $O(1)$ in the formula of $dp[m]$ calculation.

Problem Tutorial: "Antipalindrome"

Note, that if there's palindrome of length n, then there's palindrome of $n-2$ so it's enough to build a string that doesn't contains palindromes of length 2 and 3.

Cases with $k=1$ or $k=2$ may be solved separately.

Dynamic programming in $O(NK^3)$: State: $dp[i][last\_character][prev\_character] = minimal_cost$ is minimal cost to replace first i characters such that i-th one is replaced is $last\_character$, and $i-1$-th is replaced by $prev\_character$. To calculate bruteforce p - $i-2$-th character, that is not equal to $last\_character$ and $prev\_character$ and choose miminal $dp[prev\_character][p] + cost[s[i]][p]$

Now, dynamic programming in: $O(NK^2)$ it's not important what $prev\_character$ exactly, it's needed only to save two best options of $prev\_character$.

Now, note that each character may be replaced only to one of the 5 cheapest choices because in optimal answer there's only 4 restrictions (2 characters to the left and to the right), so we got DP in $O(N*5*5)$

Some teams also got accepted with $O(N*5*5*5)$ (only second optimization)

Problem Tutorial: "Long Nim"

It's known that second player wins if and only if $a_i \oplus ... \oplus a_n = 0$. We will prove that if $a_i^a..._n^a = 0$ than first player can force game to last $\sum_{i=0}^{n} a_i$ turns. Suppose that k is maximal such that all $2^k | a_i$ and choose such i that 2^{k+1} does not divide a_i and remove 1 stone. Now second player have to make such a turn so that a_j is xored by $2^{k+1}-1$ (k+1 ones in binary). If $2^{k+1}|a_j$ then $a_j \oplus 2^{k+1} > a_j$ and turn is impossible. If 2^{k+1} doesn't divide a_j then $a_j \oplus 2^{k+1} = a_j - 1$ because k is chosen to be maximal. So, second player have to remove only one stone to win. So, second player may force all turns to be ones.

Now, if $a_i \oplus ... \oplus a_n \neq 0$, then first player want to make maximal first turn such that he wins (so xor becomes 0) and it minimizes sum. We may choose such turn iterating over all heaps.

Problem Tutorial: "Guess Table"

At first lets find any row in a matrix. We will find a row in a two steps:

1. Lets start from empty string and add symbol '0' at the end and make a query. If the answer is 0 then lets change last symbol by '1' and try one more time. If the answer is 0 then the first step is finished. Otherwise we increased current string length by one.
2. Lets add symbol '0' at the start of the string. Similarly of case 1) if the answer to the query is 0 then lets change the first symbol by '1' and try again. If the answer again is 0 the second step is also finished. Otherwise we incresed current string length by ony.

Easy to see that this two steps will give us some string from guessed matrix with no more than $2 \cdot n$ queries.

Now lets find the first column. We will to that also in two steps:

1. Lets add a string with first symbol '0' and other symbols '?' to the bottom of the current table and make a query. If the answer is 0 lets change '0' by '1' and try again. If the answer is also 0 lets stop the first phase. Otherwise we found another symbol in the first column.
2. Lets add a string with first sumbol '0' and other symbol '?' to the top of the current table and make a query. If the answer is 0 again lets change '0' by '1' and try again. If the answer is 0 the second phase is finished. Otherwise we found another symbol in the first column.

Easy to see that after this two steps we will get a table with guessed some string, first column and question marks in other position. Also easy to see that we will use no more than $2 \cdot m$ queries for that.

At last lets find other symbol straightforwardly: we will try symbol '0' at each position and if the answer is 0 lets change it by '1'. Easy to see that we will get the answer in after another $(n-1)(m-1)$ queries.

So totally we will get the answer in $(n+1)(m+1)$ queries.

Problem Tutorial: "Restrooms"

Suppose we've already placed first i restrooms, did't fail any request yet and the restroom at position i is for men. Then we have only unsatisfied requests for men's restrooms with left border $\geq i+1$ and unsatisfied requests for women's restrooms with right border $\geq i+1$. All requests for women's restrooms with left border $\leq i$ will be satisfied iff request with the smallest right border will be satisfied. Therefore among all placements of the first i restrooms with i-th men's restroom we are interested only in the placement which maximizes the smallest right border of unsatisfied requests for women's restrooms. We can now write dynamic programming solution with state (prefix length, last restroom gender) containing largest position when we must place restroom of the opposite gender. Using this states of dynamic programming it's easy to restore the answer.

Problem Tutorial: "Flying Doors"

Fix some h and v and the order of doors. When Kostya flies through coordinate i, the position of the i-th door is $a_i + \frac{b_i \cdot i}{v}$. So, the condition of Kostya's win is

$$h \leq a_i + \frac{b_i \cdot i}{v}$$

for all $i = 1, 2, \ldots, n$.

Consider these doors as lines in coordinates $(\frac{1}{v}, h)$. Then all possible h are those that there exists some point $(\frac{1}{v}, h)$ such that all lines are upper than this point. In particular, this implies that the set of winning heights for Kostya is always some segment $[0; h_0]$.

How to find this h_0? Let's do a binary search on it, assume our guess is g. If there exists some i that $b_i = 0, a_i < g$, this g is more than h_0. Consider (a, b) such that $b > 0$. If it has number i in the final order, than is gives the condition on the velocity which Kostya chooses for the height g:

$$g \leq a + \frac{b \cdot i}{v} \Rightarrow \frac{1}{v} \geq \frac{g-a}{b \cdot i}.$$

Similarly, if $b < 0$, we have the condition:

$$g \leq a + \frac{b \cdot i}{v} \Rightarrow \frac{1}{v} \leq \frac{g-a}{b \cdot i}.$$

Note that if $a < g$ for some $b < 0$, then g is more than h_0 too.

So Artem's aim is to make

$$\max_{b<0} \frac{g-a}{b \cdot i} > \min_{b>0} \frac{g-a}{b \cdot i},$$

or, the same

$$\max_{b<0} \frac{a-g}{-b \cdot i} > \min_{b>0} \frac{g-a}{b \cdot i}.$$

To maximize the left part, we should take the maximum $\frac{a-g}{-b}$ with $b < 0$ and take $i = 1$ for it. To minimize the right part, we should take the minimum $\frac{g-a}{b}$ with $b > 0$ and take $i = n$ for it. In this case the other doors are not important, because only these two doors can give Artem the desired condition.

So the solution is as follows: make the binary search, find these two doors which should be arranged first and last, if the desired condition is satisfied, then Kostya will not win for this g. The complexity is $O(n \cdot \log C)$.

Problem Tutorial: "MIPT Campus"

Consider one student going from dormitory at point a to study building at point b. If $a > b$, we can swap a and b, the answer to the problem will not change. If there exists a crosswalk inside segment $[a; b]$, then this student needs to go $b - a$ meters, otherwise consider the crosswalk at point x closest to this segment (it can be either to the left of the segment or to the right of the segment). Then student has to go $(b-a)+2 \cdot dist(x, [a; b])$ meters. This x can be found using the binary search. So we can find the initial sum of distances in $O(n \log m)$.

For students who doesn't have a crosswalk inside their segments we can decrease the distance they have to go. If we build a crosswalk closer to the segment than this closest x, we have a profit of $(dist(x, [a; b]) - dist(y, [a; b]))$. Let $d = dist(x, [a; b])$. Then building a crosswalk at point y inside $[a-d; a)$ gives profit $2(y-a+d)$, inside $[a; b)$ gives profit $2d$, inside $[b; b+d)$ gives profit $2(b+d-y)$. We can rewrite this events in form: on segment $[u; v)$ add a linear function $p \cdot y + q$, and after that find y with maximal value, in which we will build a new crosswalk. Note that since all functions are linear, the maximum will be reached in the end of some event segment. This problem is solved using sweep line: for each such segment $[u; v)$ make two events: add from u value $p \cdot y + q$ and add from v value $-p \cdot y - q$. Now go from left to right accumulating this events in the form of linear function and check its values in each coordinate which has an event starting in it. The overall complexity of the solution is $O(n \log (nm))$.

Problem Tutorial: "Tickets"

Let's construct a graph, such that he distance from each vertex to the capital would be exactly the needed distance. After we constructed such graph, we can use Dijkstra algorithm to calculate all the distances and process all queries afterwards.

The most obvious solution would be the following: for each ticket (v_i, k_i, w_i) construct the edges too all vertices u such that $dist(u, v_i) \le k_i$ with cost w_i. However, the number of edges would be up to $O(V^2)$ and this solution would be too slow in such case.

Let's construct the graph more carefully. We need to add the edges from vertex v_i to all the vertices u no further than k_i. Let's perform Divide-And-Conquer on tree approach.

We iterate though all the layers in Divide-And-Conquer, let's fix one of them with vertices V and root vertex $root$. In this layer, let's add all the edges from $v \in V$ to all the vertices u such that $root \in path(v, u)$.

We can sort all the vertices v in increasing order by $dist(root, v)$. It's easy to see that for each ticket (v_i, k_i, w_i) we need to add edges from vertex v_i to all vertices u such that $dist(root, u) \le k_i - dist(root, v_i)$. That means, we need to add edges from v_i to some prefix in the sorted list.

To do that, we add $|V|$ vertices, distance of each of them represent minimum on prefix in sorted list. That means, let the sorted vertices list be $sorted_i$ and additional vertices list be $additional_i$. Then, we add edge from $additional_i$ to $additional_{i-1}$ with price 0 and from $additional_i$ to $sorted_i$ with price 0.

To add the edges with cost w_i to some prefix $sorted_1, sorted_2, \ldots, sorted_k$ we can just add a single edge to $additional_k$. The total count of additional vertices in $O(N \log N)$ and total number of edges in also $O(N \log N)$.

Total complexity is $O(N \log^2 N)$, solution uses $O(N \log N)$ memory.

Problem Tutorial: "Total control"

Let S_old be the square of the initial polygon and S_ext be the square of the extended polygon. If $S_old \ge S$ then answer is 0.

Consider one edge of length l and polygon extension along this edge. The extended area is a rectangle $l \times d$. So the extended area along all edges is a sum of areas of rectangles: $perimeter \times d$. Then consider extension around vertices. It is easy to see that these parts are sectors of a circle, moreover the sum of all these sectors is a full circle with area $\pi \times d^2$.

So $S_ext = S_old + \pi \times d^2 + perimeter \times d$. We can solve quadratic equation or binary search to find d.

Problem Tutorial: "Möbius"

Firstly calculate using Eratosthenes-like routine mobius functions for all numbers $1, \ldots, 10^6$. Then remove from arrays the numbers with zero mobius function, they make contribution only in k_0 and this contribution could be easily calculated. Further we consider arrays a and b free of values with zero mobius function.

Assume k_0 is number of pairs (a_i, b_j) having common prime number. In other case $\mu(a_i \cdot b_j) = \mu(a_i) \cdot \mu(b_j)$. Consider all 4 cases of pairs of values $\mu(a_i), \mu(b_j)$, for each case one could calculate number of zeros $\mu(a_i \cdot b_j)$ and subtract it from $k_{\mu(a_i) \cdot \mu(b_j)}$. So we reduced problem to calculation of number of pairs (a_i^x, b_j^y) having common prime divisors (array a^x is part of array a having $\mu(a_i) = x$). It could be done using inclusion-exclusion formula and Eratosthenes-like routine:

$$\sum_{d=1}^{10^6} \mu(d) \cdot cnt_{a^x}(d) \cdot cnt_{b^y}(d),$$

where $cnt_{a^x}(d)$ – number of elements of a^x divisible by d, $cnt_{b^y}(d)$ – number of elements of b^y divisible by d.