Long Contest Editorial

AIM Fund

November 18, 2015

A. Substrings and Subsequences

Observation:

Strings that have same substrings and subsequences have the form "aa... ab... b" or "aaaa... aaa"

Proof:

Suppose there are u characters "a" in string, consider a subsequence "a... a" (u times). It's also a substring, so all "a"'s are placed continuously in s.

Suppose there are at least 3 different letters used. So the string looks like "a... ab... bc... c...". Subsequence "ac" is not a substring.

Now fix a pair of different letters, a and b, then probability of all strings of the form "a... ab... b" is $\sum_{i=0}^{n} p(a)^i p(b)^{n-i}$. It's a geometric progression, that can be calculated in $O(\log n)$.

Probability of the string "a... a" is $p(a)^n$

Overall complexity is $O(m^2 \log n)$, where $m = 26$ is the number of different letters.

B. Bus stop

Probability that minimum is less than x equals $F(x) = 1 - \prod_{i=1}^{n}(1 - x/t_i)$, so we need to calculate integral $\int xF'(x)dx$. $F(x)$ is a product of polynomials, so we could calculate it by divide-and-conquer and FFT in $O(n \log^2 n)$, but coefficients will be too big and it may not work with appropriate precision. Let $t_{min} = min(t_i)$. Calculate probability p_k of exactly k of random variables less than t_{min}. These probabilities equal coefficients of polynomials $\prod_{i=1}^{n}((t_i - t_{min})/t_i + t_{min}/t_i \cdot x)$. Then the answer equals $\sum_{i=0}^{n} p_i \cdot \frac{t_{min}}{i+1}$, because expected value of minimum of k independent uniformly distributed on $(0; t)$ variables equals $\frac{t}{k+1}$. Complexity $O(n \log^2 n)$

C. Collection of sets

Let's find a solution for $k = 7$ at first. Divide 14 elements into groups of two elements, and compose a collection of 2^7 subsets as follows: for $i = 0, 1, \ldots, 2^7 - 1$, build the i-th subset: if the j-bit of i is 0, take the first element from the j-th group, else take the second element from

the j-th group. If the weights in some group are equal, then the minimum is not unique. So, the probability of the good assignment is 2^{-7} (in each group the weights are different).

To obtain a solution for $k \geqslant 7$, we can take a solution for $14 - k$ and take the complements of the subsets in this collection. Now we assume that $k < 7$.

Consider the system of m one-element subsets. The probability of the good assignment for it is obviously $m \cdot 2^{-m}$ — one element's weight equal to 1 and others equal to 2. Let's merge the answers for $k = 7$ and for $k = 1$ then. Divide the set into t pairs and the remaining $14 - 2t$ elements. Compose the collection of the subsets in the following way: take 2^t subsets from the t pairs using the previous construction, and from each such subset compose a subset for the collection adding one element from the last $14 - 2t$ elements — overall, there are $2^t \cdot (14 - 2t)$ such subsets. The probability of the good assignment is the product of those probabilities for the two collections, so it is equal to $2^{-t} \cdot (14 - 2t) \cdot 2^{2t-14} = 2^{t-14} \cdot (14 - 2t)$. It is the solution for $k = t + 1$, so the probability of the good assignment for that collection is equal to $2^{k-15} \cdot (16 - 2k)$. It is easy to check that for $k \leqslant 6$ that probability is less than $\frac{1}{100}$.

D. Decomposition

First of all, how to determine if $k > 0$? That is, how to determine if the given graph can be represented as a product of another graph and a single edge?

The product of a graph G and an edge is the graph H that consists of two copies of G with corresponding vertices of copies connected by an edge. We have to find a suitable partition of vertices of H into two parts so that the edges between the parts form a perfect matching, and two induced subgraphs on the parts are equal with accordance to the matching.

Let the graph H be connected. Suppose we know a pair of corresponding vertices from different parts. Then, we can unambigiously restore the whole partition. Let's start the BFS-like process: each vertex has one of three states, 0 ("unvisited"), 1 ("part is determined, but corresponding vertex in another part is not found"), 2 ("corresponding vertex is found"). Initially, two vertices of the pair have state 2. The BFS will traverse the halves of the graph in parallel while matching corresponding vertices.

Consider a vertex v with state 2. All its yet unvisited neighbours must have the same part as v, so their states become 1 and they're added to the queue.

Consider a vertex v with state 1. If the original pair was chosen correctly, exactly one of unvisited neighbours of v must lie in the different part than v, and it must be its corresponding vertex. Mark them with 2 and add them to the queue once more.

This process will build a partition into two halves in $O(m)$ time, given a pair of corresponding vertices. Since we do not know a pair from the start, choose a vertex v and try to build a partition with all pairs $(v, \text{neighbour of } v)$. This works in $O(m \deg(v))$ time. If we choose v to be the vertex with minimal degree, the complexity becomes $O(m^2/n)$, since the minimal degree is $O(m/n)$. In a connected graph $n = \Omega(\sqrt{m})$, thus the complexity is $O(m^{3/2})$.

If we have succeeded to represent the graph as a product with an edge, we can further repeat the process on one of the resulting parts, and so on. Thus we will obtain the representation of the graph as a product of another graph and a Boolean cube of maximal order, since the Boolean cube is a Cartesian power of a single edge. If on some step there is more than one possible way to partite the graph into two halves, each of them will produce isomorphic parts, thus it doesn't matter which one we'll choose. The total complexity of the repeated process is still $O(m^{3/2})$.

It suffices to run the process for all connected components of the graph and choose minimal k over all components. The actual representation should be then carefully restored.

E. Easy Everest

If two climbers are initially on heights g_i and h_j and should be on the same height in the end, their total spent energy is $|g_i - h_j|$.

Sort both arrays g and h. In the optimal answer, for all i climbers on heights g_i and h_i should be grouped together on the same height. Otherwise, there are pairs (g_a, h_b) and (g_c, h_d) such that $g_a < g_c$ and $h_b > h_d$, but they can be swapped to make the answer better. Thus, the answer is $\sum_{i=1}^{n} |g_i - h_i|$.

F. Sum of divisors

We need to express n as a sum $n/a_1 + n/a_2 + \ldots + n/a_k$, where all a_i divide n.

Obviously, minimum of $a_1, a_2, \ldots, a_k$ does not exceed k (otherwise the sum is less than n). Consider some set $a_1, a_2, \ldots, a_k$ such that $1/a_1+1/a_2+\ldots+1/a_k = 1$. It gives us the representation for all n divisible by $lcm(a_1, a_2, \ldots, a_k)$. So we need to generate all lcm-s of such subsets.

Do bruteforce using the mentioned observation and halt it if the current denominator is already divisible by some previously considered $lcm(a_1, a_2, \ldots, a_k)$. Having generated them, you can see that for $k \leqslant 7$ all lcm-s are less than 200, and for each k there are no more than 15 of them. Now our problem is to count the number of integers not exceeding n such that they are divisible by at least one of these lcm-s. These can be done using preprocessing with inclusion-exclusion formula. For each possible lcm $prod$ of these numbers, store the coefficient the number $\lfloor \frac{n}{prod} \rfloor$ has when we use inclusion-exclusion formula for n. There are no more than 800 different $prod$'s for $k \leqslant 7$. After that, we can solve one test using approximately 800 operations.

G. Guess sinus

Send ? 10^{-9}, the sign of answer determines $sign0$ — the sign of a. Then run binary search on absolute value of answer: $l = 0$, $r = 10^9$. Consider one step of binary search: $mid = (l+r)/2+eps$. Send ? π/mid and if $sign0 \cdot sign > 0$ then $r = mid$, else $l = mid$. When $r - l$ become less than 10^{-6} the answer $(l + r)/2$ will be acceptable.

H. Matrices dot product

$O(n^2k)$ dynamics. Let's calculate number $dp[k][i][j]$ = dot product of matrix $dp[i...i+2^k)[j...j+2^k)$ with T_k. $dp[0][i][j] = t[i][j]$

$dp[k+1][i][j] = dp[k][i][j]a_{00} + dp[k][i][j+2^k]a_{01} + dp[k][i+2^k][j]a_{10} + dp[k][i+2^k][j+2^k]a_{11}$

Answer for k-th query is maximum and minimum of $dp[k][i][j]$ for all i, j.

I. I18n

In this problem you need to process all words consequently and store for each $i18n(x)$ value all words with same $i18n$. You can store this info in map or dictionary for example. Then for every new word calculate $i18n$ and check that there is only one word with same value and this word equals the current word.

J. Segment Sort

Notice that costs satisfy the triangle inequality $w_{i+j} \leqslant w_i + w_j$. Split array into atoms — the minimal segments $[l; r]$ such that sets of elements at $[1; l-1]$, $[l; r]$ and $[r+1; n]$ in the given and in the sorted arrays coincide. The minimality here means that there is no atom $[l'; r']$, which is a proper subsegment of $[l; r]$. It is easy to prove that we need to cover all atoms of length more than one by nonintersected segments with minimal total cost, and sort these segments. We can find the atoms in linear time: iterate from 1 to n over initial array and sorted one and mark a point to split the segments if the counts of every number have been processed in both arrays are equal. Then calculate the following array: for every position i where right end of its atom r_i is situated. Then calculate dynamics with n states. i-th state is a minimal cost of sorting prefix of length i. Each state could be recalculated in $O(\sqrt{n})$ time: for every possible cost of last sorted segment $c \leqslant \sqrt{n}$:

```
for (int t = 1; t * t <= n; ++t) {
    int j = r[max(0, i - t*t)];
    d[i] = min(d[i], t + d[j]);
}
```

Another approach is to calculate dynamics $dp[c]$ — the maximal prefix which can be sorted using c operations. The are $O(\sqrt{n})$ states, the transition takes $O(atoms)$ operations. The total complexity of both solutions is $O(n\sqrt{n})$.

K. Tree Generation

In this problem you need to write a test generator at first and then calculate different statistics for different values of *type*. For example, you can calculate the number of leaves in the generated trees for all values of *type* and observe that their average values form a decreasing sequence. Than, given a tree, calculate the number of its leaves and output the *type* for which the expected number of leaves is the closest to that number. This gives an average result of 96-97%, with the minimum result 92% on tests.