Xellos's blog

Codeforces Round #333 — editorial

By Xellos, history, 4 months ago,

Hints:

div2A: Try conversions between bases.

div2B: Solve a simpler version of the problem where $A_{i+1} \neq A_i$ for all i.

div1A: What are the shortest paths of the vehicles? what's the shorter of those paths?

div1B: Forget about the ceiling function. Draw points $(i, A[i])$ and lines between them — what's the Lipschitz constant geometrically?

div1C: Some dynamic programming. Definitely not for the exp. score of one person — look at fixed scores instead.

div1D: Compute $dif(v)$ in $O(N)$ (without hashing) and then solve the problem in $O(N^2)$. You need some smart merges.

div1E: Can you solve the problem without events of type 1 or 2? Also, how about solving it offline — as queries on subsets.

BORN TOO LATE TO EXPLORE THE EARTH
BORN TOO SOON TO EXPLORE THE GALAXY
BORN JUST IN TIME TO BROWSE
DANK MEMES

Div. 2 A: Two Bases

→ Pay attention

Before contest
Educational Codeforces Round 11
3 days

Like 28 people like this. Be the first of your friends.

→ Top rated

#	User	Rating
1	tourist	4126
2	TooDifficult	3553
3	Petr	3424
4	rng_58	3125
5	izrak	3095
6	jcvb	3058
7	Um_nik	3007
8	vepifanov	2999
9	Egor	2991
10	RomaWhite	2985

Countries | Cities | Organizations View all →

→ Top contributors

#	User	Contrib.
1	Errichto	168
2	Zlobober	164
3	Petr	163
4	Edvard	157
5	Xellos	155
6	amd	150
7	chrome	146
8	Swistakk	145
9	rng_58	142
10	Rubanenko	140

View all →

→ Find user

Handle:

Find

→ Recent actions

SkullSkin → High price for OpenCup's login

kAc → Mo's Algorithm

MehrdadAP → Got stuck in an interesting combinatorial problem

dtalamas24 → Which has been your

It's easy to compare two numbers if the same base belong to both. And our numbers can be converted to a common base — just use the formulas

$$X = \sum_{i=1}^{N} x_i B_x^{N-i}; \quad Y = \sum_{i=1}^{M} y_i B_y^{M-i}.$$

A straightforward implementation takes $O(N+M)$ time and memory. Watch out, you need 64-bit integers! And don't use `pow` — iterating $X \to XB_x + x_i$ is better.

Div. 2 B: Approximating a Constant Range

Let's process the numbers from left to right and recompute the longest range ending at the currently processed number.

One option would be remembering the last position of each integer using STL `map<>` / `set<>` data structures, looking at the first occurrences of A_i plus/minus 1 or 2 to the left of the current A_i and deciding on the almost constant range ending at A_i based on the second closest of those numbers.

However, there's a simpler and more efficient option — notice that if we look at non-zero differences in any almost constant range, then they must alternate: .., $+1$, -1, $+1$, -1, ... If there were two successive differences of $+1$-s or -1-s (possibly separated by some differences of 0), then we'd have numbers $a-1, a, a, ..., a, a+1$, so a range that contains them isn't almost constant.

Let's remember the latest non-zero difference (whether it was +1 or -1 and where it happened); it's easy to update this info when encountering a new non-zero difference.

When doing that update, we should also check whether the new non-zero difference is the same as the latest one (if $A_i - A_{i-1} = A_{j+1} - A_j$). If it is, then we know that any almost constant range that contains A_i can't contain A_j. Therefore, we can keep the current leftmost endpoint l of a constant range and update it to $j+1$ in any such situation; the length of the longest almost constant range ending at A_i will be $i-l+1$.

This only needs a constant number of operations per each A_i, so the time complexity is $O(N)$. Memory: $O(N)$, but it can be implemented in $O(1)$.

Bonus: the maximum difference permitted in an almost constant range is an arbitrary D.

favorite IOI?

ahmed_aly → Google Code Jam 2016 Warmups

Serh.kry → When java coders need multimap/multiset

NeverSayNever → Best Book for learning java

AmSen → Iran IOI team 2016

tweety → List of IOI 2016 Participants

maximaxi → Where can we find HackerEarth ratings?

xblwyc → Any data structure supporting log(n) update, delete and index query?

sahasumit288 → Floating To Integer problem:

King_Alexander → The Missing Differential

dragonslayerx → Invitation To Alkhwarizm 2016

Nickolas → April Fools Day Contest 2016

Inishan → An awesome list for competitive programming!

chrome → Topcoder Single Round Match 686

IlyaLos → Editorial for Codeforces Round #346 (Div. 2)

LongQuan → Compete Resources

mnbvmar → On "Mo's algorithm"

satyaki3794 → Getting TLE on "Shaass the Great", 178-DIV2-E

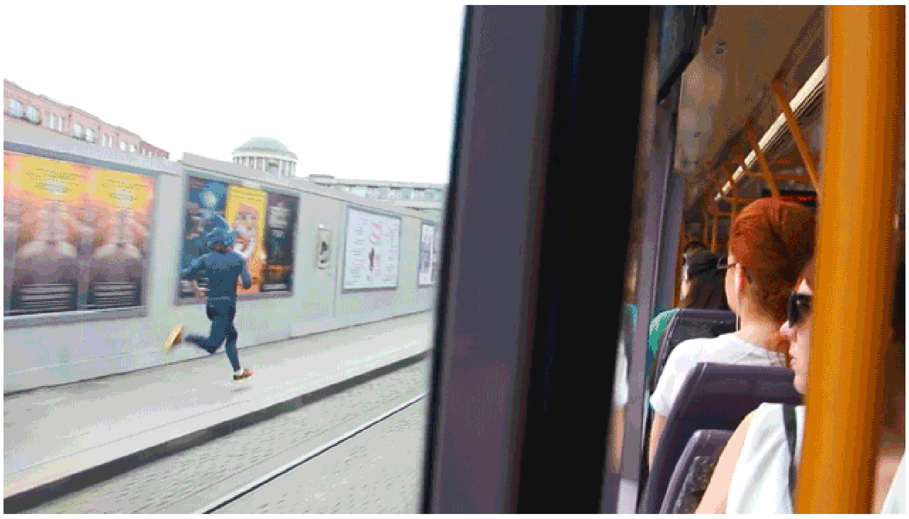
fmoraes → Announcing EclipseForces, an Eclipse plugin for Codeforces

kingofnumbers → [help] How many points under the line (queries problem)

INeverGetACs → Advice regarding Bad Habit of Seeing the solution too early

Edvard → Educational Codeforces Round 10

Detailed →



Div. 2 C / Div. 1 A: The Two Routes

The condition that the train and bus can't meet at one vertex except the final one is just trolling. If there's a railway $1 - N$, then the train can take it and wait in town N. If there's no such railway, then there's a road $1 - N$, the bus can take it and wait in N instead. There's nothing forbidding this :D.

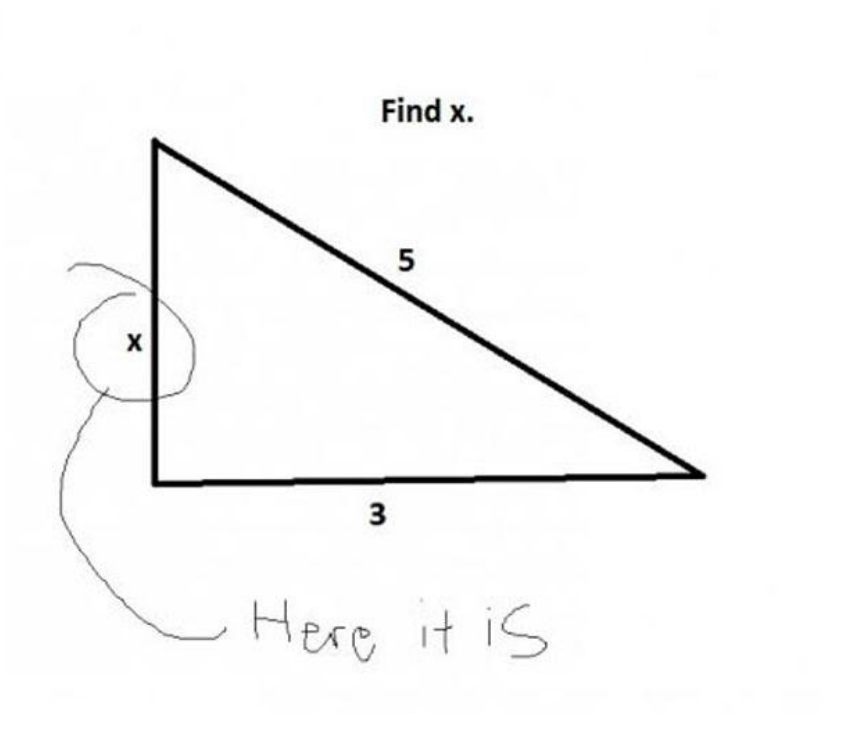
The route of one vehicle is clear. How about the other one? Well, it can move as it wants, so the answer is the length of its shortest path from 1 to N... or -1 if no such path exists. It can be found by BFS in time $O(N+M)=O(N^2)$.

In order to avoid casework, we can just compute the answer as the maximum of the train's and the bus's shortest distance from 1 to N. That way, we compute $\max(1, \text{answer})$; since the answer is ≥ 1, it works well.

In summary, time and memory complexity: $O(N^2)$.

Bonus: Assume that there are M_1 roads and M_2 railways given on the input, all of them pairwise distinct.

Bonus 2: Additionally, assume that the edges are weighted. The speed of both vehicles is still the same — traversing an edge of length l takes l hours.



Div. 2 D / Div. 1 B: Lipshitz Sequence

Let $L_1(i,j) = \frac{|A_i - A_j|}{|i-j|}$ for $i \neq j$.

Key observation: it's sufficient to consider $j = i + 1$ when calculating the Lipschitz constant. It can be seen if you draw points (i, A_i) and lines between them on paper — the steepest lines must be between adjacent pairs of points.

In order to prove it properly, we'll consider three numbers A_i, A_j, A_k ($i < j < k$) and show that one of the numbers $L_1(i,j)$, $L_1(j,k)$ is $\geq L_1(i,k)$. W.l.o.g., we may assume $A_i \leq A_k$. There are 3 cases depending on the position of A_j relative to A_i, A_k:

- $A_j > A_i, A_k$ — we can see that $L_1(i,j) > L_1(i,k)$, since $|A_j - A_i| = A_j - A_i > A_k - A_i = |A_k - A_i|$ and $j - i < k - i$; we just need to divide those inequalities
- $A_j < A_i, A_k$ — this is similar to the previous case, we can prove that $L_1(j,k) > L_1(i,k)$ in the same way
- $A_i \leq A_j \leq A_k$ — this case requires more work:
  - we'll denote $d_{1y} = A_j - A_i$, $d_{2y} = A_k - A_j$, $d_{1x} = j - i$, $d_{2x} = k - j$
  - then, $L_1(i,j) = d_{1y} / d_{1x}$, $L_1(j,k) = d_{2y} / d_{2x}$, $L_1(i,k) = (d_{1y} + d_{2y}) / (d_{1x} + d_{2x})$
  - let's prove it by contradiction: assume that $L_1(i,j), L_1(j,k) < L_1(i,k)$
  - $d_{1y} + d_{2y} = L_1(i,j)d_{1x} + L_1(j,k)d_{2x} < L_1(i,k)d_{1x} + L_1(i,k)d_{2x} = L_1(i,k)(d_{1x} + d_{2x}) = d_{1y} + d_{2y}$, which is a contradiction

We've just proved that to any L_1 computed for two elements $A[i], A[k]$ with $k > i + 1$, we can replace one of i, j by a point j between them without decreasing L_1; a sufficient amount of such operations will give us $k = i + 1$. Therefore, the max. L_1 can be found by only considering differences between adjacent points.

This is actually a huge simplification — the Lipschitz constant of an array is the maximum abs. difference of adjacent elements! If we replace the array $A[1..n]$ by an array $D[1..n - 1]$ of differences, $D[i] = A[i + 1] - A[i]$, then the Lipschitz constant of a subarray $A[l, r]$ is the max. element in the subarray $D[l..r - 1]$. Finding subarray maxima actually sounds quite standard, doesn't it?

No segment trees, of course — there are still too many subarrays to consider.

So, what do we do next? There are queries to answer, but not too many of them, so we can process each of them in $O(N)$ time. One approach that works is assigning a max. difference $D[i]$ to each subarray — since there can be multiple max. $D[i]$, let's take the leftmost one.

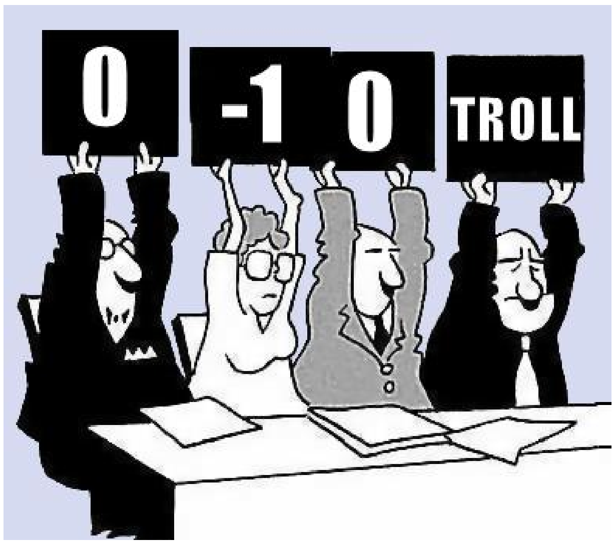
We can invert it to determine the subarrays for which a given $D[i]$ is maximum: if $D[a_i]$ is the closest difference to the left of $D[i]$ that's $\geq D[i]$ or $a_i = 0$ if there's none, and $D[b_i]$ is the closest difference to the right that's $> D[i]$ or $b_i = n - 1$ if there's none (note the strict/non-strict inequality signs — we don't care about differences equal to $D[i]$ to its right, but there can't be any to its left, or it wouldn't be the leftmost max.), then those are all subarrays $D[j..k]$ such that $a_i < j \leq i \leq k < b_i$.

If we don't have the whole array $D[1..n - 1]$, but only some subarray $D[l..r]$, then we can simply replace a_i by $\max(a_i, l-1)$ and b_i by $\min(b_i, r+1)$. The number of those subarrays is $P_i = (i - a_i)(b_i - i)$, since we can choose j and k independently.

All we have to do to answer a query is check all differences, take a_i, b_i (as the max/min with some precomputed values) and compute P_i; the answer to the query is $\sum_{i=l}^{r} D[i] \cdot P_i$. We only need to precompute all a_i, b_i for the whole array $D[1..n - 1]$ now; that's a standard problem, solvable using stacks in $O(N)$ time or using maps + Fenwick trees in $O(N \log N)$ time.

The total time complexity is $O(NQ)$, memory $O(N)$.

Bonus: $Q \leq 10^5$.



Div. 1 C: Kleofáš and the n-thlon

As it usually happens with computing expected values, the solution is dynamic programming. There are 2 things we could try to compute: probabilities of individual overall ranks of Kleofáš or just some expected values. In this case, the latter option works.

- "one bit is 8 bytes?"
- "no, the other way around"
- "so 8 bytes is 1 bit?"

After some attempts, one finds out that there's no reasonable way to make a DP for an expected rank or score of one person (or multiple people). What does work, and will be the basis of our solution, is the exact opposite: we can compute the expected number of people with a given score. The most obvious DP for it would compute $E(i, s)$ — the exp. number of people other than Kleofáš with score s after the first i competitions.

Initially, $E(0, 0) = m - 1$ and $E(0, s > 0) = 0$. How can we get someone with score s in competition i? That person can have any score k from 1 to m except x_i (since Kleofáš has that one) with the same probability $\frac{1}{m-1}$. The expected values are sums with probabilities $P(i, s, j)$ that there are j people with score s:

$$E(i, s) = \sum_j j \cdot P(i, s, j) .$$

Considering that the probability that one of them will get score k is $\frac{j}{m-1}$, we know that with probability $P(i, s, j)\frac{j}{m-1}$, we had j people with score s before the competition and one of them had score $s + k$ after that competition — adding 1 to $E(i + 1, s + k)$. By summation over j, we'll find the exp. number of people who had overall score s and scored k more:

$$\sum_j P(i, s, j)\frac{j}{m-1} = \frac{E(i,s)}{m-1} .$$

Lol, it turns out to be so simple.

We can find the probability $E(i + 1, t)$ afterwards: since getting overall score t after $i + 1$ competitions means getting score k in the currently processed competition and overall score $s = t - k$ before, and both distinct k and expectations for people with distinct s are totally independent of each other, then we just need to sum up the exp. numbers of people with those scores (which we just computed) over the allowed k:

$$E(i+1, t) = \sum_k \frac{E(i,t-k)}{m-1} .$$

The formulas for our DP are now complete and we can use them to compute $E(n, s)$ for all $1 \le s \le mn$. Since $E(n, s)$ people with s greater than the overall score s_k of Kleofáš add $E(n, s)$ to the overall rank of Kleofáš and people with $s \le s_k$ add nothing, we can find the answer as

$$1 + \sum_{s > s_k} E(n, s) .$$

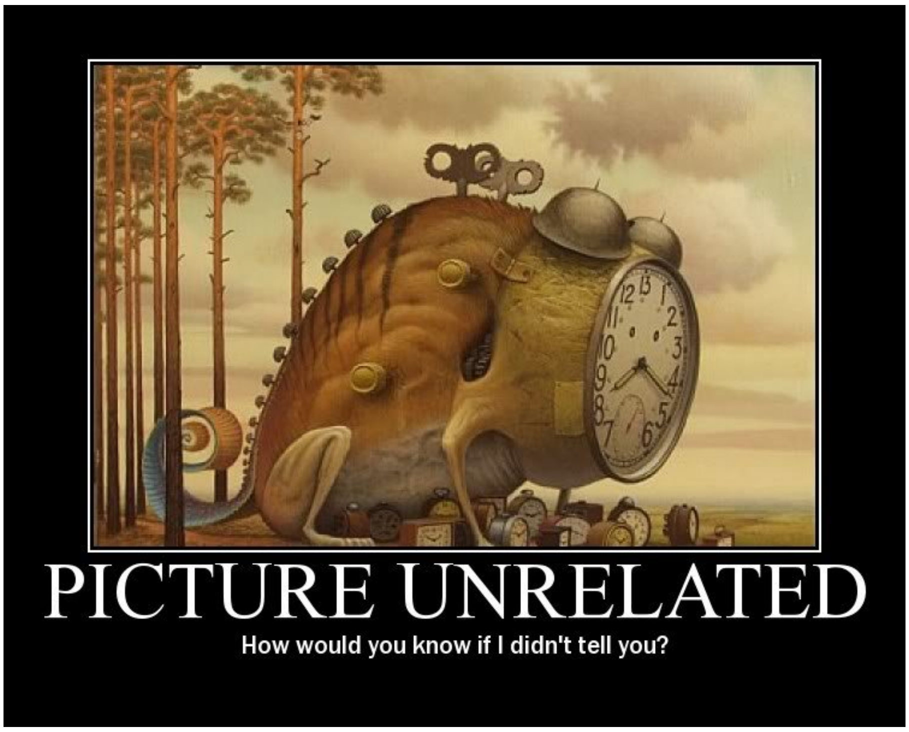
This takes $O(m^2n^2)$ time, since there are $O(mn)$ scores, $O(mn^2)$ states of the DP and directly computing each of them takes $O(m)$ time. Too slow.

We can do better, of course. Let's forget about dividing by $m - 1$ for a while; then, $E(i + 1, t)$

is a sum of $E(i, s)$ for one or two ranges of scores — or for one range minus one value. If you can solve div1C, then you should immediately know what to do: compute prefix sums of $E(i, s)$ over s and find $E(i+1, t)$ for each t using them.

And now, computing one state takes $O(1)$ time and the problem is solved in $O(mn^2)$ time (and memory).

Bonus: Really, how fast can you solve this problem?



Div. 1 D: Acyclic Organic Compounds

The name is really almost unrelated — it's just what a tree with arbitrary letters typically is in chemistry.

If you solved problem TREEPATH from the recent Codechef November Challenge, this problem should be easier for you — it uses the same technique, after all.

Let's figure out how to compute $\mathrm{dif}(v)$ for just one fixed v. One more or less obvious way is computing hashes of our strings in a DFS and then counting the number of distinct hashes (which is why there are anti-hash tests :D). However, there's another, deterministic and faster way.

Compressing the subtree T_v into a trie.

Recall that a trie is a rooted tree with a letter in each vertex (or possibly nothing in the root), where each vertex encodes a unique string read along the path from the root to it; it has at most σ sons, where $\sigma = 26$ is the size of the alphabet, and each son contains a different letter. Adding a son is done trivially in $O(\sigma)$ (each vertex contains an array of 26 links to — possibly non-existent — sons) and moving down to a son with the character c is then possible in $O(1)$.

Compressing a subtree can be done in a DFS. Let's build a trie H_v (because T_v is already used), initially consisting only of one vertex — the root containing the letter s_v. In the DFS, we'll remember the current vertex R of the tree T and the current vertex cur of the trie. We'll start the DFS at v with cur being the root of H_v; all we need to do is look at each son S of R in DFS, create the son cur_s of cur corresponding to the character s_S (if it didn't exist yet) and run $DFS(S, cur_s)$. This DFS does nothing but construct H_v that encodes all strings read down from v in T_v. And since each vertex of H_v encodes a distinct string, $\mathrm{dif}(v)$ is the number of vertices of H_v.

This runs in $O(|T_v|\sigma)$ time, since it can create a trie with $|T_v|$ vertices in the worst case.

Overall, it'd be $O(N^2\sigma)$ if T looks sufficiently like a path.

The HLD trick

Well, what can we do to improve it? This trick is really the same — **find the son w of v that has the maximum $|T_w|$, add s_v to H_w and make it H_v; then, DFS through the rest of T_v and complete the trie H_v as in the slow solution.** The trick resembles HLD a lot, since we're basically remembering tries on HLD-paths.

If v is a leaf, of course, we can just create H_v that consists of one vertex.

How do we "add" v to a trie H_w of its son w? Well, v should be the root of the trie afterwards and the original H_w's root should become its son, so we're rerooting H_w. We'll just create a new vertex in H_w with s_v in it, make it the root of H_w and make the previous root of H_w its son. And if we number the tries somehow, then we can just set the number of H_v to be the number of H_w.

It remains true that $dif(v)$ is $|H_v|$ — the number of vertices in the trie H_v, which allows us to compute those values directly. After computing $dif(v)$ for each v, we can just compute both statistics directly in $O(N)$.

Since each vertex of T corresponds to vertices in at most $O(\log N)$ tries (for each heavy edge that's on the path from it to the root), we aren't creating tries with a total of $O(N^2)$ vertices, but $O(N\log N)$. The time complexity is therefore $O(N\log N\sigma)$. However, the same is true for the memory, so you can't waste it too much!

Bonus: you have an additional tiebreaker condition for vertices with identical $c_r + \mathrm{dif}(r)$. Count the number of distinct strings which occurred exactly k times for each k in an array $P_r[]$; take the vertex/vertices with lexicograhically maximum $P_r[]$ (as many strings as possible which occur only once, etc).

Bonus 2: Can you get rid of the logarithm in the time complexity?

SUPER EFFECTIVE
57
OUT WITH IT! Where'd you hide the fossils?!
It.. IT BELONGS IN A MUSEUM!
THWACK!
~~RED~~ CLEFAIRY WANTS TO FIGHT!
R
VGCATS.COM/SUPER

Comic strip name: Indy. Go read the whole thing, it's not very long, but pretty good.

Div. 1 E: A Museum Robbery

In this problem, we are supposed to solve the 0-1 knapsack problem for a set of items which changes over time. We'll solve it offline — each query (event of type 3) is asked about a subset of all N exhibits appearing on the input.

Introduction

If we just had 1 query and nothing else, it's just standard knapsack DP. We'll add the exhibits one by one and update $s(m)$ (initially, $s(m) = 0$ for all m). When processing an exhibit with (v, w), in order to get loot with mass m, we can either take that exhibit and get value at least $s(m - w) + v$, or not take it and get $s(m)$; therefore, we need to replace $s(m)$ by $\max(s(m), s(m-w)+v)$; the right way to do it is in decreasing order of m.

In fact, it's possible to merge 2 knapsacks with any number of items in $O(k^2)$, but that's not what we want here.

Note that we can add exhibits this way. Thus, if there were no queries of type 2, we would be able to solve whole problem in $O(Nk)$ time by just remembering the current $s(m)$ and updating it when adding an exhibit. Even if all queries were of type 2 (with larger n), we'd be able to solve it in $O(nk)$ time in a similar way after sorting the exhibits in the order of their removal and processing queries/removals in reverse chronological order.

The key

Let's have q queries numbered 1 through Q in the order in which they're asked; query q is asked on some subset S_q of exhibits.

MAGIC TRICK: Compute the values $s(m)$ only for subsets $S_{2q} \cap S_{2q+1}$ — the intersections of pairs of queries $2q, 2q + 1$ (intersection of the first and the second query, of the third and fourth etc.), **recursively**. Then, recompute $s(m)$ for all individual queries in $O((N + Q)k)$ time by adding elements which are missing in the intersection, using the standard knapsack method.

What?! How does this work?! Shouldn't it be more like $O(N^2)$ time? Well, no — just look at one exhibit and the queries where it appears. It'll be a contiguous range of them — since it's displayed until it's removed (or the events end). This element will only be missing in the intersection, but present in one query (so it'll be one of the elements added using knapsack DP), if query $2q + 1$ is the one where it appears first or query $2q$ the one where it appears last. That makes at most two addittions of each element and $O(N)$ over all of them; adding each of them takes $O(k)$ time, which gives $O(Nk)$.

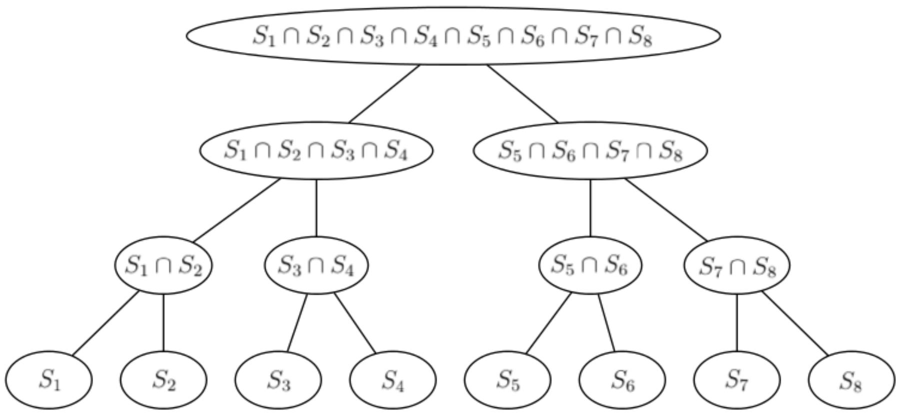
The second part of the complexity, $O(Qk)$ time, is spent by copying the values of $s(m)$ first from the intersection of queries $2q$ and $2q + 1$ to those individual queries.

If we're left with just one query, we can solve it in $O(Nk)$ as the usual 0-1 knapsack.

Since we're halving the number of queries when recursing deeper, we can only recurse to depth $O(\log Q)$ and the time complexity is $O((N + Q)k \log Q)$.

A different point of view (Baklazan's)

We can also look at this as building a perfect binary tree with sets $S_1, \ldots, S_Q$ in leaves and the intersection of sets of children in every other vertex.


$S_1 \cap S_2 \cap S_3 \cap S_4 \cap S_5 \cap S_6 \cap S_7 \cap S_8$
$S_1 \cap S_2 \cap S_3 \cap S_4$
$S_5 \cap S_6 \cap S_7 \cap S_8$
$S_1 \cap S_2$
$S_3 \cap S_4$
$S_5 \cap S_6$
$S_7 \cap S_8$
S_1
S_2
S_3
S_4
S_5
S_6
S_7
S_8

For each vertex v of this tree, we're solving the knapsack — computing $s(m)$ — for the set D_v of displayed exhibits in it. We will solve the knapsack for the root directly and then proceed to the leaves. In each vertex v, we will take $s(m)$, the set D_p of its parent p and find $s(m)$ for v by adding exhibits which are in D_v, but not in D_p.

We know that the set D_p is of the form $\bigcap_{i=a}^{b} S_i$ for some a, b and D_v is either of the form $\bigcap_{i=a}^{m} S_i$ or $\bigcap_{i=m+1}^{b} S_i$ for $m = \frac{b+a-1}{2}$ (depending on whether it's the left or the right son). In the first case, only elements removed between the m-th and b-th query have to be added and in the second case, it's only elements added between the a-th and $m+1$-th query. Since each element will only be added/removed once and the ranges of queries on the same level of the tree are disjoint, we will do $O((N+Q)k)$ work on each level and the overall time complexity is $O((N+Q)k\log Q)$.

Finding the intersections and exhibits not in the intersections

Of course, bruteforcing them in $O(NQ)$ isn't such a bad idea, but it'd probably TLE — and we can do better. We've just described how we can pick those exhibits based on the queries between which they were added/removed. Therefore, we can find — for each exhibit — the interval of queries for which it was displayed and remember for each two consecutive queries the elements added and removed between them; finding the exhibits added/removed in some range is then just a matter of iterating over them. Since we're actually adding all of them, this won't worsen the time complexity.

In order to efficiently determine the exhibits in some set $\bigcap_{i=a}^{b} S_i$, we can remember for each exhibit the interval of time when it was displayed. The exhibit is in the set $\bigcap_{i=a}^{b} S_i$ if and only if it was displayed before the a-th query and remained displayed at least until the b-th query.

To conclude, the time complexity is $O((N+Q)k\log Q) = O(qk\log q)$ and since we don't need to remember all levels of the perfect binary tree, but just the one we're computing and the one above it, the memory complexity is $O(qk)$.

Tutorial of Codeforces Round #333 (Div. 1)
Tutorial of Codeforces Round #333 (Div. 2)
codeforces, round, 333, editorial, pictures!

+490 Xellos 4 months ago 119

Comments (119)

Write comment?

404Error
4 months ago, # | +8
You trolled many users. Many would be like what the heck is that?
→ Reply

4 months ago, # ^ | ← Rev. 3 0
the best trick i have ever seen. (and also the funniest tags.)
→ Reply

smahdavi4

→ Reply

Xellos

4 months ago, # | 0

Auto comment: topic has been updated by ***Xellos*** *(previous revision, new revision, compare).*

→ Reply

ruhan.habib39

4 months ago, # | +2

If I could've upvoted n-times!(n > 1) :)

→ Reply

misterfourtytwo

4 months ago, # ^ | +11

At least u tried

→ Reply

abgnwl

4 months ago, # | 0

good trick :D and can dynamic programming solve the bonus?

→ Reply

Xellos

4 months ago, # ^ | 0

Yes.

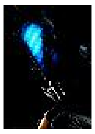
→ Reply

Abdelrahman_

4 months ago, # ^ | 0

can you provide the recurrence please ?

→ Reply

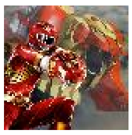


m.Elbarbary

4 months ago, # | +7

Great Contest , I appreciate your effort! I just wanna comment that Div.2/C is easier than Div.2/B for me.

→ Reply



GaoRed

4 months ago, # | +6

Still I wonder if Div2.C statement did not include the condition each of different pairs is always a rail/road, could we solve it? I was managing to do C with BFS sequentially anyway.

→ Reply

Xellos

4 months ago, # | ← Rev. 2 +6

Auto comment: topic has been updated by ***Xellos*** *(previous revision, new revision, compare).*

Now with ∞ percent more images!

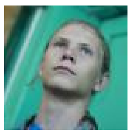
→ Reply

I_love_Hoang_Yen

4 months ago, # | +18

You are so generous :) letting O(N^3) pass Div1A and O(N*log^2) pass div1D :)

→ Reply



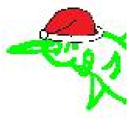
ifsmirnov

4 months ago, # | +15

Another simple solution for div1D (with hashing, although one can get rid of it with a "suffix array on a tree" method).

For each vertex, let h(v) be the polynomial hash of a string from root to v. It can be calculated using one dfs. Easy to notice that number of different

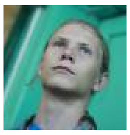
can be calculated using one dfs. Easy to notice that number of different strings reachable from some vertex v is exactly the number of different hash values in its subtree, and this is a standard task.
→ Reply



gendelpiekel

4 months ago, # ^ | 0

Actually **Xellos** mentioned that before the suffix explanation. "One more or less obvious way is computing hashes of our strings in a DFS and then counting the number of distinct hashes (which is why there are anti-hash tests :D)." Although evidently the "anti-hash tests" weren't so effective.
→ Reply



ifsmirnov

4 months ago, # ^ | 0

Oops, skipped it while reading :(Thank you.
→ Reply

Xellos

4 months ago, # ^ | 0

> *Although evidently the "anti-hash tests" weren't so effective.*

How would you imagine making anti-hash tests for any other large modulo than 2^{64}? One modulo close to 10^9 fails comfortably, but two are practically impossiblt to beat. But it's $O(N \log^2 N)$, so it's possible for it to TLE.
→ Reply

gendelpiekel

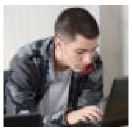
4 months ago, # ^ | 0

Oh right. I thought you meant you had general breaking cases for hashing solutions since you seemed so happy when you mentioned them (":D"), which seemed pretty unbelievable to me.
→ Reply

Xellos

4 months ago, # ^ | +8

Yes, that would be pretty unbelievable :D
→ Reply



dj3500

4 months ago, # ^ | 0

Hashing solutions are provably correct w.h.p. if you pick the prime randomly from a large enough (polynomial-sized) interval, so one can only hope to spoil the fun for some concrete numbers like 1000000007 ;)
→ Reply

rng_58

4 months ago, # ^ | +10

Do you know a simple solution for the "standard task" in O(N log N)? My solution was O(N log N) but it was relatively complicated.
→ Reply

Kostroma

4 months ago, # ^ | 0

You can compress the hash values into the integers from 1 to n and then solve the task "compute the number of different values on $[L_i, R_i]$", where $[L_i, R_i]$ is a segment of the Euler tour on a tree.
→ Reply

→ Reply

4 months ago, # ^ | ← Rev. 2 +2

ffao

I don't know what you call "relatively complicated", as your code seems to be really short.

Anyway, an alternative approach is to use the merging technique in which you keep track of all of the hashes present in a subtree, then calculate the hashes present in the parent subtree by merging the results for the children's subtrees. If you always merge by placing elements from the smallest set in the largest set, you can show that the number of jumps each element makes is at most log n, as in 14499298.

→ Reply

4 months ago, # | +15

gendelpiekel

A simpler deterministic solution for Div 1 D is to simply merge children with the same letter, so that every node has at most one child of each letter. We can do this recursively and since each node can get merged at most once, the complexity amortizes out.

Code

→ Reply

4 months ago, # | +8

AngryBacon

Div1E's idea is almost the reduction from #319E, did you notice that?

→ Reply

4 months ago, # ^ | +9

Laakeri

It's also really similar to Good Bye 2014 F

→ Reply

4 months ago, # ^ | +17

Baklazan

In fact, this problem was really inspired by Good Bye 2014 F. When solving Good Bye 2014, I came up with a solution which was different from the author's one and it also had a different time complexity. So I decided to make a problem, where my approach will still work, but author's solution of Good Bye 2014 F will not.

→ Reply

4 months ago, # | +5

amd

I think the trick you called "HLD trick" is called smaller-to-greater trick since we're actually using dsu basics.

→ Reply

4 months ago, # ^ | 0

Xellos

I've practically never heard of it called "smaller to greater" in any other context than actual DSU (where the notion of HLD-paths is unclear, in addittion).

→ Reply

4 months ago, # | +21

Al.Cash

I don't understand why going so complicated about Div1D. Just build a trie for each node in a dfs, and recursively merge tries for children with equal letters before exiting the node. Total number of nodes in all tries is exactly N, and every merge written in the most straight-forward way takes $O(\sigma)$, so the total complexity is $O(N\sigma)$. Code for reference:

http://codeforces.com/contest/601/submission/14454218

→ Reply

→ Reply

Xellos

4 months ago, # ^ | 0

No, merging two tries in the *most straightforward* way takes O(the sum of their sizes) time. That's why people are complaining about too short editorials on CF :D

→ Reply

Al.Cash

4 months ago, # ^ | ← Rev. 3 +8

No! Every node is merged to some other node not more than once, so the total number of merges is less than N. Of course single recursive call to merge can take linear time, but the total amortized time is still $O(N\sigma)$. I wanted to say that merging one node is $O(\sigma)$ not including recursive calls to merge children.

→ Reply

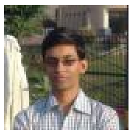
Xellos

4 months ago, # ^ | +3

You said the most straightforward way. That's DFSing fully through both trees.

I know what you mean by your solution, but everything seems much simpler in retrospect. That's why this is not div1B.

→ Reply



triveni

4 months ago, # ^ | +35

I too feel that the way **Al.Cash** is merging the tries is **the most-straightforward way**. Doing DFS in each child (except the heavy one) is not more straightforward to think than thinking about simply merge two tries. However, I do agree that realizing that his method will do at-most N merges overall, may not be that straightforward to many.

BTW, my solution is different than both of yours. My idea goes as follows —

- Do dfs and compute hash values for each node for string represented from root (node 0) to that particular node.
- Do pre-order traversal of the tree so that any subtree will appear continuously in the pre-order array.
- Then the problem boils down to find number of distinct numbers in a subarray for a given array (pre-order array), for N such queries.
- This reduced problem can be solved using fenwick tree (see DQUERY) in (N + Q) log N.

My solution using this Idea — code.

→ Reply

ffao

4 months ago, # ^ | +13

I know he's an incurable troll, but I have to side with **Xellos** in this one. Merging two tries by making a full DFS on the second tree and adding nodes one by one in the first tree is a lot more

straightforward than the trick **Al.Cash** used, by DFSing through both tries simultaneously.
→ Reply

MPeti

4 months ago, # | ← Rev. 3 +14

For Div1B, a different solution — O(Q * N log N), just passed for me (873 ms):

Build an array of differences as in the editorial solution, then use a segment tree to find maxima in it. But don't calculate it for each subarray — instead, first just find the index of the greatest element in the entire queried subarray. Calculate the amount this number contributes to the sum with some basic combinatorics, then call a similar query to the two subarrays on the two sides (if they exist) and sum the results. We'll have to find the maximum N times in each query, so it's O(N log N) per query.
→ Reply

sebinechita

4 months ago, # ^ | +5

This is the solution I used too! Except that instead of Segment Tree I used an RMQ so the complexity becomes O(N log N + Q * N)
→ Reply

waterfalls

4 months ago, # | +8

I wanted to say I really enjoyed the problemset this time around! D is actually pretty cool — somehow I've never seen merging tries before. After seeing that solution, I mostly wish there was no solution with hashing at all :P

I guess I just am a bit unclear on the purpose of pretests and partial feedback — most major contests don't have them. Is there a blog post anywhere that talks about this?
→ Reply

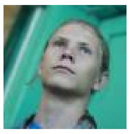
misof

4 months ago, # ^ | ← Rev. 3 +20

Pretests are used for two reasons:

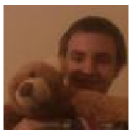
1. Codeforces is nice to you and tells you your solution is dumb so that you can go back and fix it :)
2. Only solutions that pass pretests can then be hacked, so there are no extra cheap points for just noticing and hacking a dumb submission that doesn't even attempt to solve the correct problem.

→ Reply


ifsmirnov

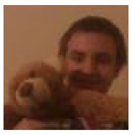
4 months ago, # ^ | +20

And point 3: you can hack others only if your own solution passes pretests. So in order to hack you must write something reasonably clever.
→ Reply


Swistakk

4 months ago, # ^ | +9

Is it an advantage?
→ Reply



4 months ago, # | +25

I like problem D, however I am confused because of that statement: "If you solved problem TREEPATH from the recent Codechef November Challenge, this problem should be easier for you — it uses the same technique, after all." I haven't solved that Codechef problem, however

setting two similar problems in two different contests at merely the same

Swistakk

time and openly admitting to it doesn't seem to be a very good idea (inb4, my judgment is based only on that statement, I haven't read Codechef problem, maybe they are not *that* similar).
→ Reply

4 months ago, # ^ | +28

No, they aren't that similar. The Codechef problem is about counting ways to split a tree with a number in each vertex into paths with non-negative sums. It uses a lot more ideas, but the trie compression that's here is not one of them. But the not DFSing through the largest son's vertex is present in both (of course, this can be solved without that with smarter merges as mentioned by **Al.Cash** above).

Xellos

I agree that it isn't very good, but neither is just casually making another div1D 3 weeks before the contest just because I'm editorialing something that uses the same technique. And where's the line, then? Should div1E not be used because its solution was made for 500F - New Year Shopping? Hey, there's probably been a problem that uses probability theory stuff, so div1C isn't suitable either! Nah, that's just not worth it — stuff happens to be similar sometimes, there's just no helping it. It does have a good side, too — if there's an easier and a harder problem using the same trick, then you can practice it more gradually.

As to openly admitting to it, well it is the truth. Why should I act like a politician?
→ Reply

4 months ago, # | ← Rev. 2 +89

misof

I hate sleep so instead having some I tested this round yesterday. Links to my codes are below, if somebody is interested in them.

- Div2A: http://ideone.com/U8NJL9
- Div2B: http://ideone.com/BpjNkY
- Div1A: http://ideone.com/DVBc2v
- Div1B: http://ideone.com/Lukj24
- Div1C: http://ideone.com/RAsfz6
- Div1D: http://ideone.com/VTJ7dH
- Div1E: http://ideone.com/h6Bb0c

→ Reply

4 months ago, # | 0

bvd

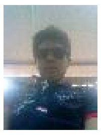
I used Dijsktra 4 times in Div2C (Div1A) http://codeforces.com/contest/602/submission/14451579 and my friends said my solution was not only long but also wrong. Was I just lucky this time?
→ Reply

4 months ago, # ^ | +5

Ramtin

of course instead of writing dijkstra 4 times u can write it once as a function once and call it four times :|
→ Reply

4 months ago, # ^ | 0


amrondonp

Look how easy it can be done. You don't even need dijkstra, just BFS http://codeforces.com/contest/601/submission/14486558
→ Reply

4 months ago, # | ← Rev. 2 0

r4huln

Why do I get a runtime error all the time for 2A when my code works fine on my system? Here is my code http://ideone.com/Y5g0T8
→ Reply

Dushyant

4 months ago, # ^ | 0

n1 and n2 are given garbage value by default. You have to first input n1 then define array of size n1. Same applies for n2.
→ Reply

r4huln

4 months ago, # ^ | ← Rev. 2 0

tt
→ Reply

Dushyant

4 months ago, # ^ | 0

Welcome. :) I see you don't write return 0 in the end this can also sometimes give Runtime Error.
→ Reply

misof

4 months ago, # ^ | +5

In plain C it can but in C++ it cannot. In C++, if you don't return anything from main, the standard requires the program to terminate with exit code 0.
→ Reply

Dushyant

4 months ago, # ^ | 0

Didn't know that. Thanks for correction.
→ Reply

determinism

4 months ago, # | ← Rev. 3 +21

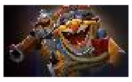
It's one of the most detailed editorials I have seen. Awesome!

Problemset was also really good. Congratulations!
→ Reply

sankar95

4 months ago, # | 0

Xellos Could you provide your solution for the Div2 B problem ?
→ Reply


LashaBukhnikashvili

4 months ago, # | 0

why the codes with algorithm O(n*m*log(n)) in problem D(div 1 B) passed system test? I'm wondering how that codes working in 0.6second :\
→ Reply

Xellos

4 months ago, # ^ | +5

My code works in 0.12 seconds, so it's really possible. CF is fast.
→ Reply

xmrisha

4 months ago, # | +1

Div2 B can also be solved using DP:

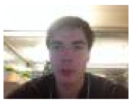
```
dp[i][0] : The length of the longest constant range ending
at 'i', where all elements in the range are equal or 1 less
than than arr[i].

dp[i][1] : The length of the longest constant range ending
at 'i', where all elements in the range are equal or 1
greater than than arr[i].<br/>
```

My submission: 14451702

My submission: [illegible]
→ Reply

kewl
4 months ago, # | 0
Xellos, You are too good !
→ Reply


Pixar
4 months ago, # | -7
Readable solution for problem `div2A` :)
→ Reply

m.ajaj94
4 months ago, # | 0
in div2 C i don't quite understand how not meeting at a node at the same time is meaningless and i noticed people computing only the shortest path for one of the vehicles i used bfs 4 times where i calculate shortest distance for train then car then calculate for car then train and printing minimum
→ Reply

Xellos
4 months ago, # ^ | ← Rev. 2 0
Ehm, either the car or the train can go to town n directly. That way, they won't meet anywhere at the same time.
→ Reply

m.ajaj94
4 months ago, # ^ | 0
how i was unable to see that is beyond me thank you
→ Reply

gaziz
4 months ago, # ^ | ← Rev. 5 0
Can you please explain, what logic for this graph would be? Please see screenshot. And what happens in town 4, when bus and train meet there simultaneously? I cannot understand, why they don't meet in 4, if they started from 1 simultaneously. graph screenshot
→ Reply

Xellos
4 months ago, # ^ | 0
Your picture is missing a LOT of roads.
→ Reply

gaziz
4 months ago, # ^ | 0
Wow! I think I got it now. Thanks, Xellos! That makes it max(1, someBFSResult), right?
→ Reply

Xellos
4 months ago, # ^ | 0
Yes.
→ Reply

suggi_d_luffy
4 months ago, # | 0
This code is not working Solution for Div2 B.

Please suggest the flaw.
→ Reply

4 months ago, # | +29
"In fact, it's possible to merge 2 knapsacks with any number of items in

"In fact, it's possible to merge 2 knapsacks with any number of items in O(k), but that's not what we want here."

How?
→ Reply

Errichto

4 months ago, # ^ | +43

typo, $O(k^2)$:D
→ Reply

Xellos

4 months ago, # | 0

whats wrong with the library pow function for Div. 2A?
→ Reply

DarkDextrous

4 months ago, # ^ | +11

Codeforces + C++ Pow = Error

The pow function works with doubles, and it may generate precision errors in the process.

For Integer powers it's better to use your own power functions!
→ Reply

sam721

4 months ago, # ^ | 0

you can use powl instead of that. my code: 14470229
→ Reply

Arpa

4 months ago, # ^ | 0

Using powl will reduce the risk, however it's still a floating point number, so I think it's better to use your own power function for integers
→ Reply

sam721

4 months ago, # ^ | 0

No! powl uses long double and it is good enough for long long.
→ Reply

Arpa

4 months ago, # ^ | 0

With long double there won't be overflow if results fit in long long, but there is still chance of getting precision errors.

I prefer to keep on the safe side xP
→ Reply

sam721

4 months ago, # | ← Rev. 2 0

i solved div 1 B with set and i get tle,however my complexity is O(n log n * q).
→ Reply

reality

4 months ago, # | 0

Problem Div2-D/Div1-B can be solved using Range Maximum Queries without segment trees.

Given that the elements of the array D will not change, a Sparse Table can

Given that the elements of the array D will not change, a Sparse Table can be used. This way we can pre-compute the RMQ Sparse Table with complexity O(N*log(N)) and answer RMQ queries in O(1).

Nice problem! ^_^
→ Reply

sam721

4 months ago, # | ← Rev. 2 0

So,could you give me some hints on div1.B bonus? Thx.
→ Reply

bshu

4 months ago, # | +2

you are creative; thanks for hints!
→ Reply

Arpa

4 months ago, # | -10

What is the meaning of anti-hash test when anyone can use triple hash! (601D - Acyclic Organic Compounds)
→ Reply

Kuzey

4 months ago, # ^ | +3

The purpose of an anti-hash test *isn't* to break all solutions that use hashing. (In fact, when hashing is used properly, such a solution can even be extremely hard to hack!)

The anti-hash test breaks *incorrect implementations of hashing* that use hashing modulo 2^32 / 2^64.
→ Reply

misof

4 months ago, # ^ | +9

Then why sometimes they create an Anti-Hash test for hashes that use 31(as their base) and $10^9 + 7$ (as their Modulo)?
→ Reply

.-.---.

4 months ago, # ^ | 0

Because they can.
→ Reply

muratakburak

4 months ago, # ^ | ← Rev. 2 0

I think you are wrong.The purpose of any kind of input is to break all the breakable codes.If you can break a code uses *1e9+7* as mod, then why not to hack it ?
→ Reply

muratakburak

4 months ago, # | 0

Could anyone tell me more about problem E div2? I can not understand what tutorial said. Thanks in advance.
→ Reply

santa_calus

4 months ago, # ^ | ← Rev. 6 +8

Kleofáš's expected rank = 1 + E, where E is the expected number of people who have a smaller score.

By the linearity of expectation:

E = (the number of other people) * P, where P is the probability

that Zachariáš (one specific other person) has a smaller score

misof

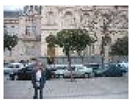
Let P(s) be the probability that Zachariáš has the score exactly s. P can be computed as the sum of P(s) over all s smaller than Kleofáš's score.

The values P(s) can be computed using dynamic programming.

More precisely, we can compute all values P(s,i): the probability that Zachariáš has score exactly s after i competitions.

This DP can then be computed in a clever way to make the program an order of magnitude faster.
→ Reply

4 months ago, # ^ | 0


sam721

I don't understand the linearity part :(

If P is the probability that some person will get a smaller score than Kleofáš's, how the expected value comes to be (number of other people)*P ?

I'm not very good at probability theory :'(.

Thanks!
→ Reply

4 months ago, # ^ | +11

misof

Linearity of expectation simply means that if you do two experiments X and Y that each return a random value (e.g., flip a coin and roll a die), the expected value of X+Y is the expected value of X + the expected value of Y. (The nontrivial part is that this is true even if X and Y are not independent.)

This is actually pretty easy to prove. In this problem, we can do it like this.

We will count the same thing twice, using two different methods.

Method 1: Let's take an empty piggy bank. We will now run 1,000,000 n-athlons. After each n-athlon, each person who was better than Kleofáš will put a coin into the piggy bank.

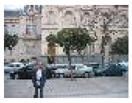
At the end, what is the number of coins in the piggy bank? On average, this should be 1,000,000 * the expected (i.e., average) number of people who are better than Kleofáš.

Method 2: Let's focus on one competitor, Zachariáš. Suppose that he has a better score than Kleofáš with probability p.

What is the number of coins Zachariáš puts into the piggy bank? On average, this should clearly be 1,000,000 * p.

Zachariáš is one of N-1 identical people. Each of these N-1 people will, on average, put 1,000,000 * p coins into the piggy bank.

Both methods are counting the results of the same process. Therefore, it must be the case that the expected number of people who are better than Kleofáš must be (N-1) * p.
→ Reply


sam721

4 months ago, # ^ | 0

I see! Thank you very much! :D

By the way, the fact that the expected value is (n-1)*p could mean that we are dealing with a binomial distribution? Or am I overthinking the problem? :P

→ Reply

saurv4u

4 months ago, # ^ | ← Rev. 3 0

I found the answer as the sum over probability of each rank * rank. Probability of given rank(k + 1) = ${}^{m-1}C_k P^k(1-P)^{m-k-1}$ which sums over to $(P+(1-P))^{m-1}$ that is 1. The idea behind was that I will choose k which have score lower than s_k and rest with score above s_k. Something like Bernoulli distribution.

→ Reply

ds_95

4 months ago, # | 0

Hi Xellos ! I think in second bullet point in **Div. 2 D / Div. 1 B: Lipshitz Sequence** instead of "L1(i,j)>L1(j,k)" it should be "L1(j, k) > L1(i, k)"

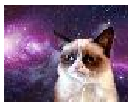
→ Reply

Xellos

4 months ago, # ^ | 0

You're right, thanks.

→ Reply


maverick_10

4 months ago, # | 0

O(n^2) logic passed in div2 B

include<bits/stdc++.h>

using namespace std; int main() { int n; cin>>n; int a[n]; for(int i=0;i<n;i++) cin>>a[i];

```
int maxlen=1;
int ans=0;
for(int i=0;i<n;i++)
{
    int maxi=INT_MIN;
    int mini=INT_MAX;
    int count=0;
    for(int j=i;j<n;j++)
    {
        maxi=max(maxi,a[j]);
        mini=min(mini,a[j]);
        if(maxi-mini>1)
         break;
        count++;
    }
    ans=max(count,ans);
    if(n-i<=ans)
    break;
}
cout<<ans <<endl;

}
```

→ Reply

4 months ago, # ^ | +3

There's a shitload of optimisations, you know. Especially the line `if(n-i<=ans) break;` , which cuts down the time for large answers significantly — to $O((N - ans) \cdot ans)$.

Xellos

You forgot to mention those 3 TLE before it and that it takes 1900 ms, though. If the TL wasn't raised to 2 seconds before the round, it'd fail for sure. If someone actually made those optimisations during the round and got lucky enough not to get hacked, it's pretty much deserved.

→ Reply

aranjuda 4 months ago, # ^ | 0

Can you explain how to solve the bonus problems?

→ Reply

zodiacleo 4 months ago, # | 0

Hi, Can somebody help me with Div1B/Div2D. I am clearly missing out something in this —

Xellos: "One approach that works is assigning a max. difference D[i] to each subarray — since there can be multiple max. D[i], let's take the leftmost one." zodiacLeo: Why the leftmost one ? Even after multiple reads (of the editorial i couldn't get a convincing answer)

Xellos: "We can invert it to determine the subarrays for which a given D[i] is maximum: if D[ai] is the closest difference to the left of D[i] that's ≥D[i] or ai =0 if there's none, and D[bi] is the closest difference to the right that's >D[i] or bi=n-1 if there's none (note the strict/non-strict inequality signs — we don't care about differences equal to D[i] to its right, but there can't be any to its left, or it wouldn't be the leftmost max.), then those are all subarrays D[j..k] such that ai<j≤i≤k<bi." zodiacLeo: Lost the explanation completely! :|

In my defense, I went through code of **misof** and **mmaxio** (both implementation, which I found to be along the lines of the editorial) but still I couldn't get it.

→ Reply

Xellos 4 months ago, # ^ | ← Rev. 2 0

Why not the leftmost one? It doesn't matter which, as long as it's a unique and useful choice.

Do you understand that we need to describe/count the subarrays for which a given difference is maximum? (Because we use it afterwards.)

Try drawing pictures.

→ Reply

zodiacleo 4 months ago, # ^ | +5

I did the drawing part again after what you said, but I am still finding myself in the middle of desert :|. **sigh**

I guess I'll have to keep probing on such question (involving subarrays etc) which might eventually put my lack of familiarity with subarrays (counting it, describing them, etc) at bay. I'll revisit this problem sometime again in future (hope that's not too far away).

Just for reference - Drawing part

I used the first sample example to understand-

10 4

1 5 2 9 1 3 4 2 1 7

I haven't uploaded the image where-in I actually run the code (manual tracing/debugging) because it appears nothing more than gibberish.

If you happen to think of a different way to explain this then please do post it over here. (If that's not much to ask already)
→ Reply

Xellos
4 months ago, # ^ | 0

That's more of a picture to the original problem, not to the reduced one that you asked about (summing up subarray maxima).

The subproblem we're solving there is "for how many subarrays will $D[i]$ be the leftmost maximum?".
→ Reply

zodiacleo
4 months ago, # ^ | 0

Epiphany!!! It all makes sense now. Why the subarrays and the maximum in each of those subarrays. :)

Wohu! Thanks **Xellos**
→ Reply

xennygrimmato
4 months ago, # | 0

> *"The condition that the train and bus can't meet at one vertex except the final one is just trolling."*

Can some one please help me understand this:

Why don't we have to handle the condition that the bus and train may reach at an intermediate node ($\neq 1$ & $\neq N$) at the same time? Won't there be any possible case such that the bus/train may have to take a longer path, because the shortest path was intersecting with a railway/road ?
→ Reply

Xellos
4 months ago, # ^ | +3

What is the shortest path of one of them?
→ Reply

xennygrimmato
4 months ago, # ^ | 0

The path obtained by performing BFS from 1 to N, correct?
→ Reply

Xellos
4 months ago, # ^ | 0

No. Not in a general shortest path problem. In this problem.
→ Reply

4 months ago, # ^ | 0

Ok, so as per this:

> *"You may assume that there is at most one railway connecting any two towns"*

If this statement means that there is at xennygrimmato

most one path connecting any 2 towns by railway, then the shortest path of one of them is equal to 1, if such a railway exists.
→ Reply

4 months ago, # ^ | 0

Xellos

I have no idea what you're trying to say.

The shortest path of one of the vehicles is the edge from 1 to n. It will always exist, either as a railway or as a road.
→ Reply

4 months ago, # | 0

Lord.of.AMC

I thought about problem (div. 2) D without looking at this editorial first, but the only part I couldn't figure out was the "standard" problem described at the end: the one about precomputing all a_i, b_i. Do you mind explaining to me how the approach using a stack works?

For example, take this array of consecutive differences: [14, 12, 9, 7, 11, 8]. How would you use a stack to figure out that the latest element before 11 that is $\geq$ 11 is 12? I went through people's code, but still couldn't figure it out. Thanks for your help!
→ Reply

4 months ago, # ^ | +6

Xellos

https://duckduckgo.com/?q=next+greater+element
→ Reply

4 months ago, # ^ | 0

Lord.of.AMC

thanks :D
→ Reply

4 months ago, # | 0

chachachowdhury

div2B what am i doing wrong ? my submission :14555108
→ Reply

4 months ago, # | 0

electro

Damn. Got trolled in div2C/div1A. :P
→ Reply

3 months ago, # | 0

nikitosoleil

I have just solved Div2 B in time O(n) and memory O(1). Also, my solution can be updated to solve this problem with different D (according to bonus task here). I am dividing each element by 2 and using, in one case, floor(), in another — ceil(), next I am counting simillar elements — this is very easy to implemet. My (awful, I know) code: http://codeforces.com/contest/602/submission/15226134 To solve it with different D we need to divide by D+1, I guess.
→ Reply

2 months ago, # | 0

Xellos: In div2 B, why is checking only for the last position of A[i] + 2 and A[i] -2 enough? What if A[i] + 3 occurs after the last position where A[i] + 2

occurred? Wouldn't the solution be screwed then?

TomBombadil

→ Reply

2 months ago, # ^ | +1

No, because $|a_{i+1} - a_i| \leq 1$. That condition isn't there just for fun.

→ Reply

Xellos

5 weeks ago, # | +3

I think Lipshitz Sequence was good problems not only in terms of graph property but also in terms of stack operation.

→ Reply

petil777