培正喇沙 編程挑戰賽
LA SALLE – PUI CHING PROGRAMMING CHALLENGE

第九屆 · 2025

2025 年 8 月 13 日（星期三）

香港培正中學

題解

名稱			作者	編製	難易度
A		Advanced Preparations	袁樂勤	何翊蓁	★☆☆☆☆
B		Broken Joy-Con	龍瑞希	龍瑞希	★★☆☆☆
C		Copper Mining	黃敏恆	黃卓翹	★★★☆☆
D		Diving Groupmates	鄭希哲	黃梓謙	★★★★☆
E		Express Trains	黃進	黃進	★★★★☆
F		Faster Route	袁樂勤	招朗軒	★★★★★★
G		Greedy Merchants	鄭羽辛	鄭羽辛	★★★★★
H		Hurricane Signal	黃敏恆	黃浩恩	★★★☆☆
I		Interesting PuiLaTiu	鄭羽辛	黃浩恩	★☆☆☆☆
J		Jumping Game	盧沂鋒	黃進	★★★★★
K		Keep Sorting	楊汶璁	楊承羲	★★★★☆
L		Linesweeper	楊汶璁	郭正謙	★★☆☆☆

難易度是指作者團隊認為能在比賽中解決該題的隊伍比例

由 1 星至 6 星為: >75%, 50-75%, 25-50%, 5-25%, 1-5%, <1%

A - Advanced Preparation

Problem Summary

Classes are suspended based on a rule that if rainfall on day i is >=200 mm, classes on day i+1 are suspended. If classes are suspended on day i+1 and its rainfall is also >=200 mm, then classes on i+2 are also suspended, and so on. If classes are suspended on a day with 0 mm rainfall, it counts as a Failed Prediction.

Your Goal is to construct a rainfall schedule for N days with total rainfall exactly S mm, with each day's rainfall is a non-negative integer <= 350 mm, that maximizes the number of Failed Prediction.

Solution

To maximize the number of Failed Prediction, we can strategically place a day with >=200 m and the next day followed by 0 mm. Structured the schedule in repeating blocks of [200 mm, 0mm], until the N-th day, or no more rainfalls. If there are rainfalls remaining, distribute it across the days with 200 mm without exceeding 350 mm per day. If there is still rainfall leftover, distribute it across the days with 0 mm without exceeding 350 mm. This guarantees the maximum number of Failed Prediction.

Comments

The number of teams solved was quite aligned to my expectations. I predicted around 46 teams out of 62 teams (~75%), and the actual result is 50 teams (~80%). However, I thought the solving time would be faster.

B - Broken Joy-con

The grid dimensions can be up to 10^9, which makes it impossible to simulate the movement cell by cell or to store the grid in memory.

The key insight is that the player only changes its state (location or direction) at a few specific Points of Interest (POIs):

1. The starting cell, (1, 1)
2. The 2K portal cells
3. The finishing cell, (R, C)

The vast empty space between these points is irrelevant. The journey is just a sequence of straight-line movements between these POIs.

To create a sorted list of POIs, we can insert the starting cell to the head of the portal list and the finishing cell to the tail. The player then starts at the first position in this list.

Here lists the conditions of failing (cannot reach the finishing cell):

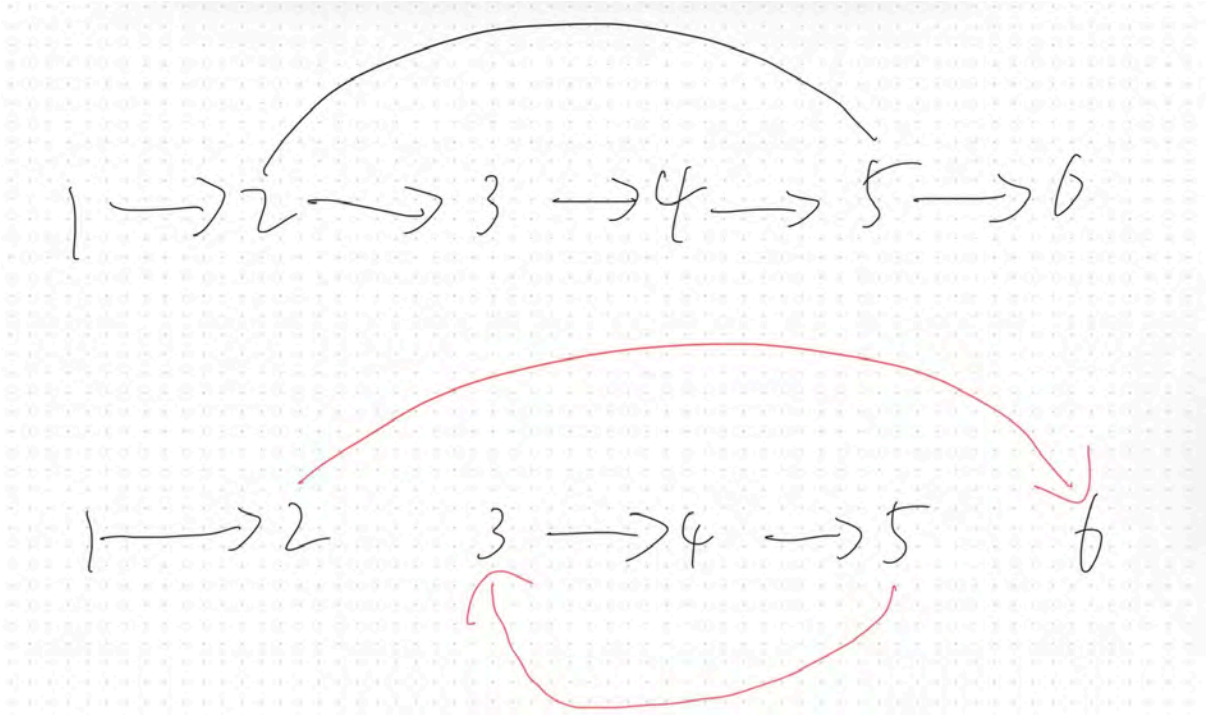
1. Hits a wall
2. Enters a portal that was visited before (visited as an entrance)

To check whether the Ben can reach the finishing cell, follow this steps:

1. Check to see if Ben can reach the next portal. Assuming Ben is at the i-th cell in the list, he can reach the next portal if and only if the (i+1)-th cell shares the same row index with the i-th cell. Jump to (3) if he can, jump to (2) if he cannot.
2. There are two cases, in either case, the answer can be determined directly.
    a. Ben hits the wall (fail condition 1): The finishing cell doesn't share the same row index with the i-th cell, or the finishing cell shares the same row index with the i-th cell but has a smaller column index.
    b. Ben reaches the finishing cell: The finishing cell shares the same row index with the i-th cell and has a larger column index.
3. Ben has no choice but to move into the next portal and teleport to the corresponding position. Go back to (1) after updating the current position.

Note that it is impossible for Ben to enter a loop. So the solution can always be found within 2K+1 steps.

Proof of no loop
For a portal pair ((x1, y1), (x2, y2)).
It is actually building an edge in between (x1, y1) and (x2, y2 + 1) and an edge in between (x1, y1 + 1) and (x2, y2).



Then notice that there is at most 1 in-degree and 1 out-degree for each cell, so it is impossible to reach an entrance visited before.
(Figure cr: firewater)

C - Copper Mining

In this task, you are given a N * N grid with exactly 3 copper cells (‘***X***’) and the remaining are stone cells (‘**.**’). The goal is to mine the minimum number of stone cells to connect all 3 copper cells with copper cells and mined cells (‘*****’).

Solution

Define the median cell as the cell at the X and Y coordinate median of copper cells. E.g. if copper cells are at (3, 2), (2, 4), and (4, 5), the median cell is at (3, 4). Observe that the optimal solution is to connect all 3 copper cells to the median cell separately in the respective shortest paths. One of the easiest ways to connect 2 cells in the shortest path would be mining out a ‘L’ shape.

Comments

It is expected to have about 25 teams to be able to solve this task during contest time.

The solution requires some simple observation.

D - Diving groupmates

First, let's consider a simplified version of the problem, where each groupmate must be the PIC of exactly one task. We can see that for every task i, it is handled if and only if there exists a task j such that the reminder for task i is sent earlier than task j and the PIC of task i is a peanuter of task j.

We can build a directed graph of M nodes, where each node represents a task. For every pair of nodes, we build an edge from node i to node j if the PIC of task j is a peanuter of task i, which means that sending the reminder for task i would cause task j to be handled if the reminder for task j has already been sent. We can see that for each connected component in the graph, the last reminded task can never be handled.

For each of the connected components, we can divide the graph into multiple SCCs forming a tree, using an algorithm like Tarjan or Kosaraju. We can then start a BFS (or DFS) from any node in the root SCC, which every non-root node must be visited after visiting a node that has an edge pointing to it. We can then send reminders for the tasks in the reverse order that we visited the nodes, so that every task except the last one will be handled, which must be an optimal order. We can get the answer to the whole problem by combining the answers of each connected component.

Now, let's go back to the original problem, where each groupmate can be the PIC of multiple tasks. We can build a directed graph of N + M nodes, where each of the M nodes represents a task and each of the N nodes represents a groupmate. For every pair of groupmate node i and task node j, we build an edge from node i to node j if the groupmate i is the PIC of task j, which means that if groupmate i upwaters, they will handle task j if the reminder for task j has already been sent. We also build an edge from node j to node i if the groupmate i is a peanuter of task j, which means that sending the reminder of task j would cause groupmate i to upwater. We can then solve the problem in a similar way that we used to solve the simplified version, start a BFS from the root SCC, visit all nodes in the same SCC, then visit the next SCC that has an edge pointing to it. Please note that we should skip the groupmate nodes when outputting the answer.

Comments

The author expected this task to be solved by 3-5 teams, but only 1 team had solved it during contest time. It may require some time to think about how to build the graph, but the solution is quite straightforward after the graph is built.

E - Express Trains

Problem Summary

Given a linear train system with N stations and M type of trains, where trains of type j stop at stations i where station level $A_i \leq j$ in order. All trains take 1 minute to travel between consecutive stops.
For Q pairs of travel plans (U_i, V_i), find the minimum travel time from U_i to V_i.

Solution

First, let x be any station such that $U_i \leq x \leq V_i$ and $A_x = \min(A_{U_i}, A_{U_i+1}, \ldots, A_{V_i})$. Observe that we must stop at the station x.

Then, we can travel using a two-step greedy method.

- From station U_i to station x, we will greedily change to a train of the smallest possible type in every station we stop at. Notice that we will not skip station x.
- From station x to station V_i, we can consider the problem in reverse, traveling from station V_i to station x, also changing to a train of the smallest possible type in every station we stop at.

Directly implementing this greedy method leads to a $O(NQ)$ solution.

To optimize it, for every station i we can precompute the first station on both sides of it j such that $A_j \leq A_i$. If they exist, name them l_i and r_i, respectively. We can show that we can go from station i to station l_i or station r_i in 1 minute.
To implement this, we can use a monotone stack or any range query data structure and binary search.

For the first step, when we are at station i, we always "jump" to station r_i in 1 minute as our next station, and station r_i always exists. Therefore, the time taken in the first step is the number of "jumps" needed from station U_i to station x using r_i.
Similarly, the time taken in the second step is the number of "jumps" needed from station V_i to station x using l_i.

However, the solution still has a time complexity of $O(NQ)$. To further speed up the solution, we can use binary lifting to find the number of "jumps" needed in $O(\log N)$.
Alternatively, we can consider a graph of N nodes with (i, r_i) as edges. Note that it is a rooted forest where all parent nodes are greater than their child nodes. Node x will be an ancestor of node U_i and the number of "jumps" needed is the difference in depth of two nodes. The time complexity of each query is $O(1)$ using this method.

The time complexity of the full solution is $O(N \log N + Q \log N)$ or $O(N \log N + Q)$ for the alternative solution.

Comments

The author expected this task to be solved by around 10 teams. However, the overall performance is slightly worse than our expectations. Contestants should familiarize themselves with more "standard" tricks, such as greedy approach, reversing the problem and binary lifting.

F - Faster Route

The setting of this task is a $1 \times C$ rectangle: starting from lower-left corner,

- Alice would pass the pedestrian crossing 1 taking 1 unit time, and then walk right taking C unit time;
- Bob would walk right taking C unit time, and then pass the pedestrian crossing 2 taking 1 unit time.
- Each pedestrian crossing has a traffic light, with periods of A and B unit times respectively, such that the lights turn red during odd periods and turn green during even periods. Both lights just turned from green to red at time 0.
- A person can immediately cross the road if he/she arrives at the crossing with the light being green, otherwise he/she must wait till the light turns green.

Your task is to determine the number of pairs (u, v), where $0 \le u \le U$ and $0 \le v \le V$, such that if Alice starts the journey at time u and Bob at time v, the person starting strictly later will arrive strictly sooner.

Let $arriveA(u)$ and $arriveB(v)$ be their arrival times respectively, and let S be the set of pairs that satisfy the task requirement. Further define $S = S_A \cup S_B$, where S_A refers to the set of pairs which Alice departs first and S_B is which Bob departs first. Then

$$(u, v) \in S_A \quad \Leftrightarrow \quad u < v < arriveB(v) < arriveA(u)$$
$$(u, v) \in S_B \quad \Leftrightarrow \quad v < u < arriveA(u) < arriveB(v)$$

First some simple observations:

- The overall scenario repeats itself after $2 \times \mathrm{LCM}(A, B)$ unit time. For ease of discussion we shall compute the results for which $0 \le \min(u, v) < 2AB$.
- The waiting time for Alice at the crossing is at most A, purely determined by $\mathrm{mod}(u, 2A)$. Similarly, the waiting time for Bob at the crossing is at most B, purely determined by $\mathrm{mod}(v + C, 2B)$.
- $arriveA$ and $arriveB$ are non-decreasing functions.

In the following discussion we shall consider the case that Alice starts earlier and arrives later, i.e. compute the values for S_A. The computation for S_B, in which Bob starts earlier and arrives later, is analogous with a shift of C in starting time. Based on the above observations we can draft an $\mathrm{O}(AB \times \log \max(A, B))$ solution as follows:

- For each $0 \le u < 2AB$ for Alice, $arriveA(u)$ is fixed.
- Since $arriveB$ is non-decreasing, there is a maximum v^* such that $arriveB(v^*) \le arriveA(u) - 1$. It is possible $v^* \le u$, or $\boldsymbol{v^* \ge 2AB}$, or even $v^* < 0$.

- For all $u < y \le v^*$, we have $arriveB(y) \le arriveB(v^*) < arriveA(u)$, so $(u, y) \in S_A$.
- The number of desired pairs is therefore $\max(v^* - u, 0)$ for each u.
- We can use **binary search** to find v^* for each u, hence $O(AB \times \log \max(A, B))$.

Observation 4: If both Alice and Bob get to their respective crossings at a green light, their waiting times are both 0, and so both their travel times are exactly $C + 1$, the one starting earlier must arrive sooner, thus not meeting our requirements. In other words, the only time difference between the routes is waiting time at traffic lights, and the one starting earlier arriving later **must have waited at his/her crossing** (for at least 2 unit time).

But that would only eliminate half of the u's during computation. Alice still needs to wait when $u = 2kA + r$, with $0 \le k < B$ and $0 \le r < A$ (k here actually indicates which red light (0-based) of Crossing 1 that Alice is waiting at), and finding v^* for each u is still required. Luckily,

Observation 5: For the above $u = 2kA + r$, we have $arriveA(u) = (2k + 1)A + 1 + C$, which is independent of r. In plain words, whenever Alice needs to wait for the k-th red light at Crossing 1, she will cross the road **at the moment the light turns green**, resulting in the same arrival time regardless of her starting time.

Thus there is only B distinct values of $arriveA(u)$, and so B distinct values of v^* amongst the u's we are interested in. We can therefore handle the computation by each red light:

- For each $0 \le k < B$ for Alice, let $u^* = 2kA$, corresponding to the beginning of the k-th red light. We can use **binary search** to find v^* for each u^*.
- If $v^* \le u^*$ we move on to the next red light.
- Otherwise $(u^*, v^*) \in S_A$ and $(u^*, v^* + 1) \notin S_A$. It can be easily shown that $v^* < (u^* + A - 2)$ since A is Alice's maximum waiting time. It is possible $\boldsymbol{v^* \ge 2AB}$.
- Then for all $u^* \le x < y \le v^*$, we have

$$arriveB(y) \le arriveB(v^*) < arriveA(u^*) = arriveA(x)$$

  so $(x, y) \in S_A$, and boundary $(x, v^*) \in S_A$, $\boldsymbol{(x, v^* + 1) \notin S_A}$.
- The number of desired pairs is therefore $(v^* - u^*) + \ldots + 1 = \frac{(v^* - u^*)(v^* - u^* + 1)}{2}$.
- Time complexity = $O(A \log B + B \log A)$, including S_A and S_B altogether.

It is important to be aware that v^* can be $\ge 2AB$ when u^* is very close to $2AB$. Therefore, to better handle the boundary parts of U and V, to avoid affecting our "complete blocks", it would be implementation-wise easier to separate out the last $4AB$ instead. Also because of U and V, some of the triangular formulae for our last $4AB$ may have to be "clipped" into a "trapezoid", **especially for S_B with a shift of C**.

Comments:

This problem is rated the most difficult task in this year's problemset; at the end no team managed to solve it during contest time, and only 1 team made a submission attempt. Even for the unofficial team, they solved 11 out of 12 tasks, and the only task they missed is this Problem F.

The background setting of red-light-green-light should be easy to understand, as a common yet crucial part in our everyday lives. The difficulty of this problem, being an ad-hoc math-ish task with no advanced algorithms required, comes not only from the complexity of incorporating boundary case handling into our formulas (the large values of C, U, V can be intimidating), but also that some of the observations may appear too trivial to be taken into serious consideration to make further progress.

As a final note, the constraints of this task is set such that the above solution can comfortably pass, yet in fact $v*$ for each $u*$ can be computed in **constant time**, yielding $O(A + B)$ full solution; the detailed formula is left as an exercise to the readers. Hint: Prove that Bob's waiting time correspond to $v*$ **must be 0**.

G – Greedy Merchants

Define $f(a, b)$ as the minimum number of trades required for merchant a to obtain item b. Specially, $f(a, b) = -1$ if it is impossible for merchant a to obtain item b.

Since the problem can be modeled as a shortest path problem on an undirected graph, we have $f(a, b) = f(b, a)$.

Hence, we can ensure that $V_u \leq V_v$ by swapping u and v if it does not satisfy the inequality.

Let x be the item that merchant u currently owns. Let r be any item with the maximum value that can be traded with item x.

If at any point where $x \neq v$, and r does not exist or $V_r < V_x$, then the answer is impossible, since we can no longer find a way to trade for items with value greater than or equal to V_x. Otherwise, there are two cases to be handled.

Case 1: $V_r < V_v$.

Observation: In this case, it is always optimal to trade item x for item r.

Proof: Assume there is a sequence of trades $x \to y_1 \to y_2 \to \cdots \to y_m \to z$, where $V_z > V_r$ ... (1) and $V_{y_k} \leq V_r$ ... (2) for all $1 \leq k \leq m$. We have $V_{y_m} \geq C_z$ ... (3). Note that $V_r \geq V_x$ ... (4) and $V_x \geq C_r$ ... (5). From (2) and (3), we have $V_r \geq C_z$. From (1), (4) and (5), we have $V_z \geq C_r$. Hence, the sequence of trades can always be optimized to $x \to r \to z$.

Hence, we can greedily trade item x for item r as long as $V_r < V_v$. The number of trades required for this part can be quickly calculated by pre-computing r for every x and do binary lifting for each query in $O(\log N)$.

Case 2: $V_r \geq V_v$.

In this case, it may not be optimal to trade item x for item r. However, the following observation enables us to reduce the problem to simple case handling.

Observation: If $f(x, v) \neq -1$, then $f(x, v) \leq 3$.

Proof: If $f(x, v) \neq -1$, then there exists an item w where $V_w \geq C_v$ ... (1) and $V_v \geq C_w$ ... (2). Note that $V_r \geq C_x$ ... (3) and $V_x \geq C_r$ ... (4). From (2) and since $V_r \geq V_v$, we have $V_r \geq C_w$ ... (5). We will then handle the following two cases separately.

- $V_w \geq V_r$

  If $f(x, v) = 1$, it is trivial to prove that this can only be achieved through $x \to v$.

  If $f(x, v) = 2$, then it is always possible to trade from item x to item r, and from item r to item v. We assume there is a valid sequence of trades $x \to p \to v$, where $V_p \leq V_r$ ... (6). We have $V_p \geq C_v$ ... (7). From (6) and (7), we know that $V_r \geq C_v$. From (4) and since $V_x \leq V_v$, we have $V_v \geq C_r$. Hence, it is possible to trade from r to v. Thus, $x \to r \to v$ is always achievable given that $f(x, v) = 2$.

  Otherwise, $f(x, v)$ must be equal to 3. From (4) and since $V_r \geq V_x$, we have $V_r \geq C_r$, and hence $V_w \geq C_r$ ... (8). By (5) and (8), we know that item r can be traded for item w. Hence, it is always possible to trade from item x to item v within 3 trades ($x \to r \to w \to v$).

- $V_w < V_r$

  If $f(x, v) = 1$, it is trivial to prove that this can only be achieved through $x \to v$.

  Otherwise, $f(x, v)$ must be equal to 2. From (1), we have $V_r \geq C_v$ ... (9). From (4) and since $V_v \geq V_x$, we have $V_v \geq C_r$ ... (10). By (9) and (10), we know that we can trade item r for item v. Hence, it is always possible to trade from item x to item v within 2 trades ($x \to r \to v$).

So, $f(x, v) \leq 3$.

With this observation, it is enough by checking if it is possible for the merchant to trade from item x to item v through the following three ways:

1. $x \to v$ in one trade.
2. $x \to r \to v$ in two trades.
3. $x \to r \to w \to v$ in three trades.

For every item x, its respective r and w can be found by storing the item with the maximum value while running a sweep line algorithm. Combine the above solutions for the two cases to obtain the full solution.

Comments

This problem requires a deep understanding of advanced topics, such as sweep line algorithm and binary lifting, along with several critical observations that may be challenging for contestants to observe. Hence, it is expected that only a small number of teams (approximately 1-3) is able to solve this problem during the contest.

Eventually, only the unofficial team managed to solve the problem, with a different approach than the intended solution. Their method took advantage of the fact that for all items i, if there exists an item j such that $V_i \leq V_j$ and $C_i \geq C_j$, then it is always suboptimal to use item i as an intermediate item instead of item j. By eliminating all such items i, one can obtain a sorted list of items where every item k in the list satisfy $V_{k-1} \leq V_k$ and $C_{k-1} \leq C_k$. Then for every item x, the possible intermediate items can be represented by a range in this sorted list. If any item in the range can swap with item v, we know that we can trade from x to v within 2 steps. Otherwise, it is always optimal to trade x with the item with the maximum value in the range. This can be done by binary lifting too.

There are many other alternative solutions. Hence, contestants are encouraged to explore different approaches when tackling a problem during the contest, as some may lead to unexpected breakthroughs and ultimately to the full solution.

H - Hurricane Signal

Problem Summary

You are given two points H, C on a Cartesian coordinate plane. Let A be the angle formed by the two tangents of a circle with center H and radius 1 that pass through C.
You are also given two parameters D, U which defines another angle B. The two lines that form B are $D - U$ degrees and $D + U$ degrees relative to the positive x-axis, counterclockwise.
Your task is to find the percentage of B that intersects with A.

Solution

To simplify the problem, we first translate the graph so that C is the origin.
The two tangents are $\alpha_1 = \tan^{-1}(\frac{H_y}{H_x}) - \sin^{-1}(\frac{1}{\sqrt{{H_y}^2+{H_x}^2}})$ and $\alpha_2 = \tan^{-1}(\frac{H_x}{H_y}) + \sin^{-1}(\frac{1}{\sqrt{{H_x}^2+{H_y}^2}})$ degrees relative to the positive x-axis counterclockwise, respectively.

We then rotate the graph such that the predicted trajectory is along the negative x-axis,
i.e. rotating the graph by $180 - D$ degrees counterclockwise.
Now $\alpha_1' = \tan^{-1}(\frac{H_y}{H_x}) - \sin^{-1}(\frac{1}{\sqrt{{H_y}^2+{H_x}^2}}) + 180 - D$ and $\alpha_2' = \tan^{-1}(\frac{H_x}{H_y}) + \sin^{-1}(\frac{1}{\sqrt{{H_x}^2+{H_y}^2}}) + 180 - D$
Since the overlapping region must be between 90° and 270°, we can take modulo with respect to 360 for all relevant values.
If $\alpha_1' \geq 180 + U$ or $\alpha_2' \leq 180 - U$, the answer is 0; otherwise, the answer is $\frac{\min(180+U, \alpha_2') - \max(180-U, \alpha_1')}{2U}$.

Alternative Solution

Again, we first translate the graph so that C is the origin.
Since the maximum allowable error is 10^{-6}, it is not necessary to use an exact algorithm to compute the required probability. In fact, this task can be solved by scanning through the range $D - U$ to $D + U$ in very small steps to estimate the probability.

We have to be able to tell whether the cyclone, if it travels along a particular angle θ, would enter the circle of 1-unit radius. We know that the length of the tangent is $l = \sqrt{{H_x}^2 + {H_y}^2 - 1}$. Therefore we can simply check whether this point $(p_x, p_y) = (l\cos\theta, l\sin\theta)$ is inside the circle or not. i.e. $\sqrt{(H_x - p_x)^2 + (H_y - p_y)^2} \leq 1$.

Then, we exhaust $\theta = D + \frac{2i+1}{10^6}U$ where $i = -5 \times 10^5, \ldots, 5 \times 10^5 - 1$ (1 million values). For each θ that the cyclone would enter the circle, we add 10^{-6} to the answer. The maximum error is 10^{-6}.

Comments

The author expected 15 to 30 teams can solve this task during contest time. However, the onsite performance is slightly worse, with only 5 teams able to solve it within the contest.

I - Interesting PuiLaTiu

Problem Summary

Given two integers $2021 \leq L \leq R \leq 10^9$, find the number of integers $L \leq k \leq R$ such that k and $k - 2016$ are both perfect squares.

Solution

For any such k, denote $k = a^2$ and $k - 2016 = b^2$ for some integers $a > b \geq 0$.

We have $a^2 - b^2 = 2016$
and factorizing it gives $(a + b)(a - b) = 2016$

As a, b are both integers, we can factorize 2016 to find the possible values of $a + b$ and $a - b$. This can be done by (almost) all factorization methods.

We then solve for a, b for each pair of factors, and check if $a > b > 0$ are integers.
To answer the question, simply count the number of a such that $L \leq a^2 \leq R$.

Alternative Solution

Alternatively, observe the following:

- $a + b = 2016$ and $a - b = 1$ has $a = 1008.5$ and $b = 1007.5$,
  which should be rejected as a, b are not integers
- $a + b = 1008$ and $a - b = 2$ has $a = 505$ and $b = 503$
- ...

Hence, the maximum integral value of a is when $a = 505$, and the maximum value of k is $505^2 = 255025$. Iterate from L to $\min(255025, R)$ and count the number of k that satisfy the constraints.

Comments

The author expected almost all teams can solve this task during contest time.
The onsite performance was as expected, with 55 teams solving the task within the contest.
Contestants should be aware of the time complexity of their solutions, as the naive implementation will net a Time Limit Exceeded (TLE) verdict due to the large constraints on L and R

J - Jumping Game

Problem summary

There is a starting platform and n jumping platforms on a line. The width of the i-th platform is A_i. The distance between the i-th platform and the $(i-1)$-th platform is D_i.

Starting from the starting platform, you will jump n times. In the i-th jump, you can jump over $0 \sim K_i$ blocks to the i-th platform. When you jump to the middle of a platform, you earn a point.

You should find the highest point you can earn when you finish the n jumps, or report that it's impossible to finish the n jumps.

Solution

Let f_u be the highest point when jumping to block u without adding the point in this jump, v be a block on the last platform, $p_v = [\text{v is the middle of the platform}]$.

Jumping from v to u, the point you can earn is $f_v + p_v$, so f_u is the maximum value of $f_v + p_v$ among all the valid block v, which means $f_u = \max\limits_{v+k_i \geq u} f_v + p_v$.

Observation 1

For two blocks on a platform u' and $u(u' < u)$, we have $f_{u'} \geq f_u$.

Proof

$$f_u = \max_{v+k_i \geq u} f_v + p_v$$
$$f_{u'} = \max_{v+k_i \geq u'} f_v + p_v$$

Since $u' < u$, we have $v + k_i \geq u > u'$. So $\{v | v + k_i \geq u\}$ is a subset of $\{v | v + k_i \geq u'\}$ and $f_u \leq f_{u'}$.

Since f_v is non-increasing, we can maintain the range for each value of f_v on each platform(number of ranges on a platform $\leq$ maximum value of $f_u + 1 \leq n + 1$).

Let $[L_x, R_x]$ be the range such that $f_v = x (L_x \leq v \leq R_x)$.

- According to the above transition formula, for $u \in [L_x + k_i, r_x + k_i]$, $f_u = \max\limits_{v \geq u - k_i} f_v + p_v = x$, so the jump will only shift all the range by k_i units without changing their size. You can just maintain the offset and cut the ranges lying outside the platform.
- For the middle of platform m, you can use binary search to find the range $[L_x, R_x]$ such that $L_x \leq m \leq R_x$. Since you will earn a point on block m, $f_u (L_x + k_i \leq u \leq m + k_i)$ will increase by one. Hence, x-th range $[L_x, R_x]$ will become $[m + k_i + 1, R_x + k_i]$, $(x+1)$-th range $[L_{x+1}, R_{x+1}]$ will become $[L_{x+1} + k_i, m + k_i]$.

So in each jump, the transition can be done in $O(\log n)$.

Eventually, the value of the last range will be the answer.

Time complexity $O(n \log n)$

Comments

The author expected this task to be solved by at least 5 teams. However, the overall performance is slightly worse than our expectations. Contestants are encouraged to apply different DP optimization techniques, such as processing entire layers in 2D DP in this task, which can significantly reduce time complexity.

K – Keep Sorting

Problem Summary

Given a permutation of $1, 2, \ldots, 3N$, where you can sort a contiguous subarray of length $\leq 2N$ in one operation. Find the shortest sequence of operations to sort the entire permutation.

Solution

Observe that there is no reason to sort a subarray shorter than $2N$ elements – it's always not worse to sort more elements in a single operation (so the resulting array is more "sorted"). Having this in mind, it would be reasonable to guess that the answer is bounded by a small constant. We can try enumerating the possibilities one by one:

- Case #1: Answer is 0. Obviously, this could only happen when the permutation is already sorted.
- Case #2: Answer is 1. With one operation, we can only relocate a contiguous subarray of $2N$ elements and the other locations remain unchanged. This inspires us to look out for the first and last misplaced element, and sort the subarray between them. If they are less than $2N$ elements apart, then the whole array can be sorted after the operation. Otherwise, either one of them will still be mispositioned after any one operation.
- Case #3: Answer is 2. From the observation in Case #2, we already know that the array has an unsorted segment of length $> 2N$. Since we only have 2 operations, the 2 operations must involve the leftmost and rightmost misplaced element, respectively. Let's denote them as l and r. Since it's always no worse to sort a subarray of $2N$ elements, the two operations would involve the subarrays $[l, l + 2N)$ and $(r - 2N, r]$ respectively. It suffices to test whether these two operations would work. Don't forget to try applying them in the other order as well!
- Case #4: Answer is 3. In fact, it is always possible to sort *any* permutation using 3 operations. The 3 operations are: $[1, 2N], [N + 1, 3N], [1, 2N]$, effectively "dividing" the array into three parts, $[1, N], [N + 1, 2N], [2N + 1, 3N]$, and sorting two parts at a time.
  - Consider the elements $1, 2, \ldots, N$. If the element is originally within the subarray $[1, 2N]$, then it will be moved to $[1, N]$ after the first operation and stay there forever. Otherwise, if it is originally within the subarray $[2N + 1, 3N]$, then it will be moved to $[N + 1, 2N]$ after the second operation and moved to $[1, N]$ after the third operation. Therefore, it ends up at the correct part.
  - Consider the elements $2N + 1, 2N + 2, \ldots, 3N$. If the element is originally within the subarray $[N + 1, 3N]$, then it will stay in $[N + 1, 3N]$ after the first operation and moved to $[2N + 1, 3N]$ after the second operation. Otherwise, if it is originally within the subarray $[1, N]$, then it will be moved to $[N + 1, 2N]$ after the first operation and moved to $[2N + 1, 3N]$ after the second operation. The third operation does not affect the element and hence, it ends up at the correct part.
  - We can deduce from here that $N + 1, \ldots, 2N$ must be placed in the correct part as well.

  Due to the second and third operation, the order of the elements within each part should also be sorted. Therefore, these 3 operations always sort the array correctly.

  Side Note: This is actually the core idea of a sorting algorithm called Stooge Sort, which divides the array into three thirds and sorts 2/3 of the array recursively. Unfortunately, it is ridiculously slow, with a $O(n^{2.7095\cdots})$ time complexity. One can view Stooge sort as bubble sort but with a batch size of $N/3$. Interestingly, the optimal batch size is simply 1 – any larger batch size would make it slower than bubble sort.

Comments

The author expected around 10 teams to solve this problem, and the performance is slightly better than expected. As "machine time" is a valuable resource in ICPC contests, teams are encouraged to try making more observations before coding – some observations would help reducing the coding time significantly despite not being essential!

L – Linesweeper

Problem Summary

You are given a minesweeper game in progress in a single row with N cells. Find the number of unopened cells that are guaranteed not to contain mines.

Solution

There are only 3 types of available hints, `0`, `1` and `2`. We can try to deduce as much as possible using the hints we have.

The hints `0` and `2` are straightforward – if we see a `0`, we can mark both neighbouring cells as safe. If we see a `2`, we can mark both neighbouring cells as containing mines.

Next, we can work on the hints with `1`. Here are things that we can possibly deduce:

- If the cell only has one neighbour (i.e. the cell is on the left or right border), then the only neighbour must contain a mine.
- If the cell has two neighbours,
  - If one of the neighbours is known to be safe, the other neighbour must contain a mine.
  - If one of the neighbours is known to contain a mine, the other neighbour must be safe.

However, the deduction might come in an arbitrary order. Consider the following two examples:

- `1?1?1?1?1?1?`
- `?1?1?1?1?1?1`

The first input should be deduced from left to right, as `1` is at the left border. However, the second input should be deduced from right to left, as `1` is at the right border. It is also possible that deduction has to be done with a mix of both orders, e.g. for the input `1?1?????1?1`. To solve this issue, we should try doing this deduction in *both* directions (from left to right, and from right to left).

One can show that there are no more deductions possible other than what we have already done, i.e. the remaining cells can either contain or not contain a mine.

Comments

The author expected this task to be solved by at least half of the teams. The overall performance is close to our expectations. However, many teams made numerous penalty attempts in this task. Teams are reminded to consider corner cases more carefully, such as handling the 1-st and N-th cells, and to deduce the answer from both left to right and right to left.