香港電腦奧林匹克競賽
Hong Kong Olympiad in Informatics

2026 Mini Competition 3 Editorial

2026-04-25

Statistics

Task	Attempts	Max	Mean	Std Dev	Subtasks
M2631 - Message Recovery	53	100	53.132	26.641	20: 47 8: 4	20: 28 16: 15 8: 6	60: 8 48: 1 36: 1 24: 12 12: 12			
M2632 - Sightseeing Attractions	42	100	49.023	37.463	14: 36	12: 30	22: 22	14: 16	23: 14	15: 11
M2633 - Marathon	32	76	20.437	16.649	11: 25	12: 14	14: 11	18: 2	21: 1	24: 0
M2634 - Paint and Guess	26	100	21.359	28.765	100: 1 83.999: 1 75.004: 1 60.028: 1 55.173: 1 31.139: 1 15: 10					
	55	**336.028**	**110.624**	**76.944**						

M2631
Message Recovery

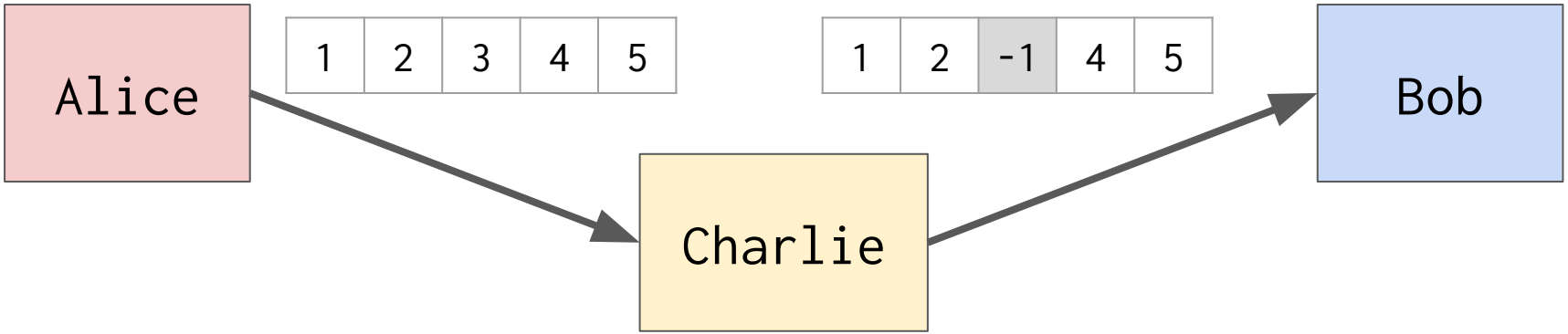
The Problem

Alice: wants to send a message A of length N to Bob, each between 0 and V - 1

Charlie: replace at most K **(= 1 or 2)** characters to -1

Minimise the length M of the actual message sent by Alice

While Bob can still recover the message.



Subtasks & Scoring

For all cases: N ≤ 100

Subtask	Score	K	V
1	20	1	2
2	20		101
3	60	2	101

prime

K = 1

M	%
N + 1	100
N + 2	80
2N	40

K = 2

M	%
N + 2	100
N + 3	80
N + 10	60
2N + 1	40
3N	20

Solution 1 (Repeat Each)

- **Solution 1** For each value, send K + 1 times. That way, all the same values cannot all be deleted. The actual length M is (K + 1) × N.

Expected Score: 8 + 8 + 12 = 28

- Clearly we cannot do anything better if we consider each position independently - we have to group all the values together in consideration. How to do that?

Solution 2.1 ~ 2.2 (Checksum)

- **Solution 2.1** (K = 1) We can append the **checksum** (A[0] + ... + A[N - 1]) % V to the array. Bob can recover the message with the check sum by simply subtract the check sum by all the existing values.

  Expected Score: 20 + 20 + 0 = 40

- **Solution 2.2** (K = 2) With checksum, we can send each value twice, followed by the checksum. If no values with both occurrence erased, then we are done; Otherwise, we can use the checksum to recover the erased character.

  Expected Score: 20 + 20 + 24 = 64

Solution 2.3 (Checksum++)

- (K = 2) We first append the checksum, followed by **checksum for each bit** (sum of all values with a specific bit as 1).

- **Observation**: Two distinct indices must have a bit that only one of them is included in the bitwise checksum(!)

- Therefore, we can recover that value first, then use the original checksum to recover the second.

Expected Score: 20 + 20 + 36 = 76

Solution 3 (Weighted Checksum)

- (K = 2) In fact, we can append two checksums to the message:
  - (A[0] + A[1] + A[2] + ... + A[N - 1]) % V
  - (A[0] + A[1] × 2 + A[2] × 3 + ... + A[N - 1] × N) % V
- Then we may simply solve the linear equation! We only care about the case where two indices x and y are erased. We have the values for

$$P = (A[x] + A[y]) \% V \qquad Q = (A[x] * x + A[y] * y) \% V$$

- So $A[x] = (Q - P * x) * (y - x)^{-1} \bmod V$ and $A[y] = (P - A[x]) \bmod V$.
- You may refer to Math in OI (I) for the details in modular arithmetics.
- Note that the solution relies on the fact that V = 101 is prime!

Expected Score: 20 + 20 + 60 = 100

Remarks

- This is intended to be an easy communication task for your practice :)
- You can exhaust the value of A[x] and A[y] with some optimizations if you hate modular arithmetics

- Fact: there are a tons of “hard versions” on this task, but they are too hard (out-syl) to ever exist. For example:
  Subtask 4. $N \le 127$, $V = 128$, $K \le 2$
  Subtask 5. $N \le 50$, $V = 101$, $K \le 50$
  Subtask 6. $N \le 63$, $V = 128$, $K \le 63$

- Feel free to try them out if you think the easy version is too easy :)
- (Hint on the next page)

Remarks

- This is intended to be an easy communication task for your practice :)
- You can exhaust the value of A[x] and A[y] with some optimizations if you hate modular arithmetics

- Fact: there are a tons of “hard versions” on this task, but they are too hard (out-syl) to ever exist. For example:
  Subtask 4. $N \le 127$, $V = 128$, $K \le 2$
  Subtask 5. $N \le 50$, $V = 101$, $K \le 50$
  Subtask 6. $N \le 63$, $V = 128$, $K \le 63$

- Feel free to try them out if you think the easy version is too easy :)
- Hint: Finite Field and Cauchy Matrix (absolutely never will be tested!)

M2632
Sightseeing Attractions

The Problem

- Given K subsets of {1, 2, ..., N}.
- Find any permutation P[1], P[2], ..., P[N] of {1, 2, ..., N}, such that at every index i, at least one of the subsets have been “completed” when you add P[i] (or report that such a permutation does not exist).

- More formally, there exists some $1 \le j \le K$ such that:
  - the j-th subset is a subset of {P[1], P[2], ..., P[i]} and
  - P[i] belongs to the j-th subset

  for every index i.

Subtask 1 (14%): |S[j]| = j

- In this subtask, a permutation exists if and only if
  - S[j] is a subset of S[j + 1] for every $1 \le j < K$, and
  - N = K.
- The i-th element of the answering permutation is then the (unique) element that is newly added from the $(i - 1)^{th}$ set to the i^{th} set.

- You can check this naively (e.g. by using a frequency array).

- Time Complexity: O(sum of |S[j]|)
- Expected Score: 14

Subtask 2 (12%): N ≤ 8, K ≤ 100

- Exhaust over all N! permutations and check one-by-one whether it is valid or not.
- We can directly use check whether a fixed permutation is valid or not in O(NK) time.

- Time Complexity: O(N! * N * K)
- Expected Score: 12 (Cumulative: 26)

Subtask 3 (22%): N ≤ 3000, Σ |S[j]| ≤ 3000

It is intuitive to use a greedy algorithm (and prove that it works):

- At every step, pick any arbitrary element i which is
  - not yet in the answer permutation, and
  - there exists some subset such that i completes the subset (together with the elements already in the permutation).
- If at any time no such element i exists, then a valid permutation cannot exist too (unless we already have all N elements in the permutation, by then we can simply output it).
- Naively doing this gives an O(N * ∑ |S[j]|) solution.
- Expected Score: 34 (Cumulative: 48)

Subtask 4 (14%): $|S[j]| \leq 4$

- Essentially, each set S[j] is a “lock” that can be opened in a multiple ways.
- For example, if S[j] = {1, 2, 3}, then
  - If you have already visited 2 and 3, you are now allowed to visit 1.
  - Similarly, visited 1 and 3 ⇒ OK to visit 2; visited 1 and 2 ⇒ OK to visit 3.

- Every time you visit an element, you iterate over the sets that it belongs to, and naively check if you are allowed to visit any new elements.

- Time Complexity: $O(N * \sum |S[j]|^2)$

Subtasks 5 and 6 (23% + 15%): No additional constraints

- In fact, the greedy approach above is just a BFS.

- For each element, we first precompute which subsets it belongs to.
- For each subset, we keep track of the number of elements which has not been added towards the answer permutation yet.
  - Whenever this number of elements becomes 1, we push it into the BFS queue.

- This gives a $O(N + \sum |S[j]|)$ solution.

Subtasks 5 and 6 (23% + 15%): No additional constraints

In an ideal world,

- solutions with sqrt should get Subtask 4 only.
- solutions with a log factor (if you have the greedy idea, but are simulating the process using sets / priority queues instead of arrays) should get Subtasks 4 and 5, but not Subtask 6.
- Solutions with linear complexity should pass.

Remarks

- In general, *counting* the number of valid permutations (instead of determining whether one exists or not) is quite difficult.
- If the input subsets are arbitrary, then the counting version of the problem does not admit a polynomial time solution.

- However, if the input subsets have some specific structure, then there might be some hope of doing it relatively more efficiently.
  - In fact, (I hope) you have already worked on the case when the input subsets form a tree structure: this is M26CI.

M2633
Marathon

The Problem

- Given a tree of N vertices, each edge has two states: ‘on’ or ‘off’, initially all edges are ‘off’
- Q events occur
  - Type 1 event: toggle an edge
  - Type 2 event: given x, query max(dist(x, u)) for all vertices u that can be reached with edges that are ‘on’

Subtask 1 (11%): N, Q ≤ 5000

- Maintain toggles in O(1)
- For each type 2 event, run a BFS over ‘on’ edges from vertex x to find the maximum distance

Time complexity: O(NQ)

Expected Score: 11 (Cumulative: 11)

Subtask 2 (12%): Chain, each edge is toggled at most once

- As each edge is toggled at most once, no edge will go from ‘on’ to ‘off’
  - So edges are only ‘added’ and never ‘deleted’
- Maintain a DSU
  - Merge adjacent cities when an edge is added
  - Each set includes every city in some interval [l, r]
  - Answer for each query = max(x - l, r - x)

Time complexity: $O(N + Q\alpha(N))$

Expected Score: 12 (Cumulative: 23)

Subtask 3 (14%): Chain

Edges may be deleted

- Insteading of maintaining DSU, we maintain the intervals [l, r] directly with **`std::set`** or **`std::map`**

Time complexity: O(N + QlogN)

Expected Score: 26 (Cumulative: 37)

Subtask 4 (18%): $N \leq 10^5$, Type 1 events before Type 2 events

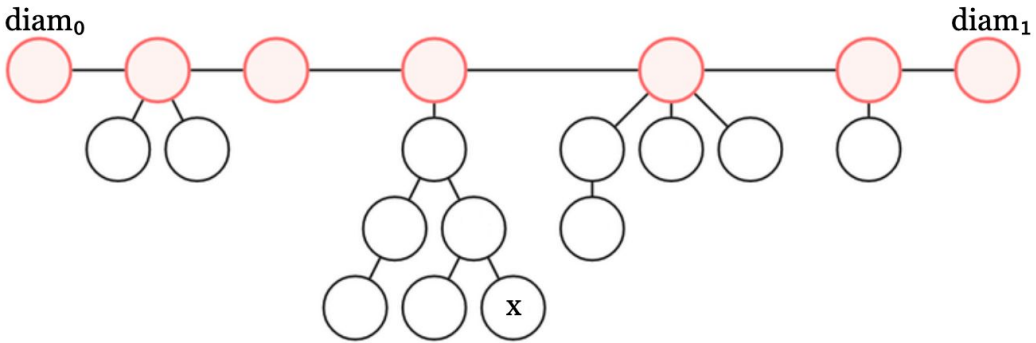
- As all toggles occur before any queries, the state of the edges in the tree are fixed, final structure will be some number of disconnected subtrees
  - Answer to each vertex x is fixed
- After doing type 1 toggles, precompute the answer for each vertex x with 2 DFS
  - Fix a root in each connected subtree, DFS from root to calculate maximum depth for each vertex u
  - Run another DFS from root, for each vertex u calculate the maximum distance from all vertices that are not in the subtree of u

Time complexity: O(N + Q)
Expected Score: 18 (Cumulative: 55)

Subtask 5 (21%): $N \le 10^5$, each edge is toggled at most once

Same as Subtask 2, but no longer a chain

Observation 1: For each query x, at least one of dist(x, $diam_i$) is the answer, where ($diam_0$, $diam_1$) are the endpoints of **any** diameter of the tree.



Proof by contradiction:

- Assume there exists vertex u where dist(x, u) > max(dist(x, $diam_0$), dist(x, $diam_1$))
- Then dist(x, u) + dist(x, $diam_1$) > dist(x, $diam_0$) + dist(x, $diam_1$)
- So (u, $diam_1$) is longer than ($diam_0$, $diam_1$) and hence ($diam_0$, $diam_1$) is not a diameter -> **contradiction!**

Subtask 5 (21%): $N \le 10^5$, each edge is toggled at most once

How to compute diameter after an edge is 'added'?

Observation 2: Let T and T' be two disjoint trees, and denote some diameter of each tree as $(diam_0, diam_1)$ and $(diam'_0, diam'_1)$ respectively. Let $S = \{diam_0, diam_1, diam'_0, diam'_1\}$.
After an edge e connects T and T', at least one of $S \times S$ is a diameter of the new tree combining T and T'.

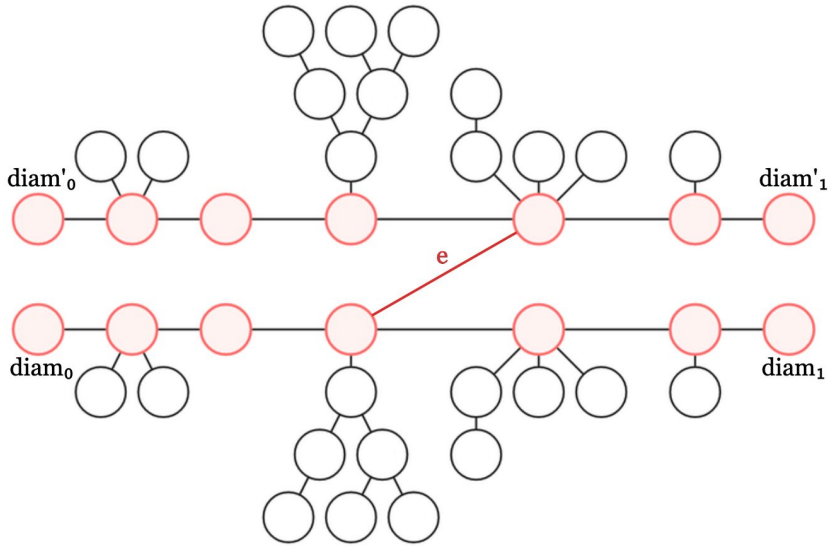
- In other words, at least two vertices in S form the endpoints of a new diameter

1. *Case 1*: There exists a new diameter that does **not** include e

- If e is not included in some diameter, then some diameter only includes nodes in either one of T or T'
- At least one of $\{(diam_0, diam_1), (diam'_0, diam'_1)\}$ is a new diameter

Subtask 5 (21%): $N \le 10^5$, each edge is toggled at most once

How to compute diameter after an edge is 'added'?



1. *Case 1*: There exists a new diameter that does **not** include *e*
   - If *e* is not included in some diameter, then some diameter only includes nodes in either one of T or T'
   - At least one of {($diam_0$, $diam_1$), ($diam'_0$, $diam'_1$)} is a new diameter

Subtask 5 (21%): $N \leq 10^5$, each edge is toggled at most once

How to compute diameter after an edge is 'added'?

Observation 2: Let T and T' be two disjoint trees, and denote some diameter of each tree as ($diam_0$, $diam_1$) and ($diam'_0$, $diam'_1$) respectively. Let S = {$diam_0$, $diam_1$, $diam'_0$, $diam'_1$}.
After an edge e connects T and T', at least one of S × S is a diameter of the new tree combining T and T'.

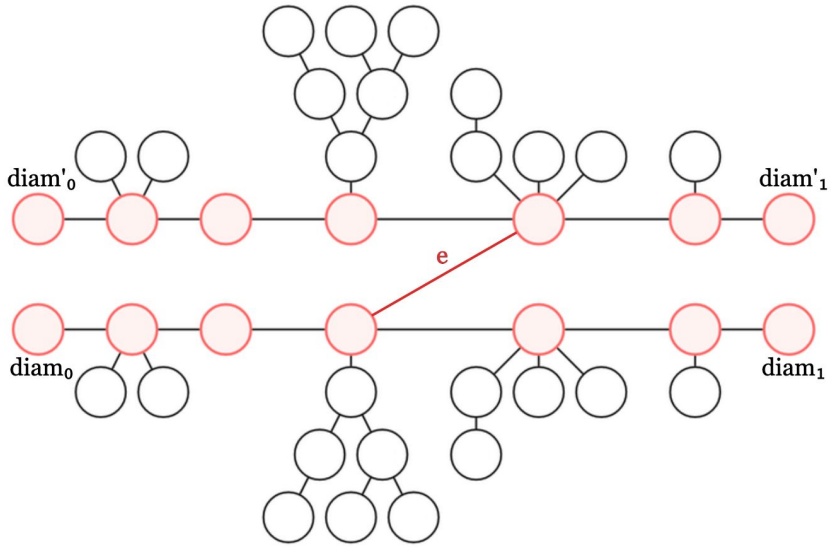
- In other words, at least two vertices in S form the endpoints of a new diameter

2 *Case 2*: All new diameters include e

- If e connects u from T and u' from T', any new diameter has length = $\max_{v \in T}(dist(u, v)) + \max_{v' \in T'}(dist(u', v'))$ for all vertices v in T and v' in T'.
- From **Observation 1**, we know that $\max_{v \in T}(dist(u, v)) = \max(dist(u, diam_0), dist(u, diam_1))$, and a similar argument can be made with T'
- So some pair in S forms a new diameter

Subtask 5 (21%): $N \leq 10^5$, each edge is toggled at most once

How to compute diameter after an edge is 'added'?



2 *Case 2*: All new diameters include *e*

- If *e* connects u from T and u' from T', any new diameter has length = $\max_{v \in T}(\text{dist}(u, v)) + \max_{v' \in T'}(\text{dist}(u', v'))$ for all vertices v in T and v' in T'.
- From **Observation 1**, we know that $\max_{v \in T}(\text{dist}(u, v)) = \max(\text{dist}(u, \text{diam}_0), \text{dist}(u, \text{diam}_1))$, and a similar argument can be made with T'
- So some pair in S forms a new diameter

Subtask 5 (21%): $N \le 10^5$, each edge is toggled at most once

- Root the tree and precompute binary lifting
- Maintain DSU
  - Maintain some diameter ($diam_0$, $diam_1$) in each set
  - For union, iterate through all possible pairs in S and compare dist(S_i, S_j)
    - Compute dist(S_i, S_j) from LCA by binary lifting

Time complexity: $O((N + Q)\log N)$

Expected Score: 33 (Cumulative: 76)

Subtask 6 (24%): No additional constraints

Solution 1: Segment Tree on Time with DSU Rollback

Segment Tree on Time is a classic way to solve DSU with edge deletions

- Maintain a segment tree over the Q events
  - For each edge, it is 'on' in some disjoint intervals of the Q events $[l_0, r_0]$, $[l_1, r_1]$, ...
  - Range update for each $[l_i, r_i]$: store these edge insertions in the nodes of the segment tree that represent $[l_i, r_i]$
- Do a preorder traversal on the segment tree so the Q events are traversed in order
  - If edge insertions are stored in current node, union in the DSU
  - When exiting the current node of the DSU, rollback to delete the inserted edges
- When the i-th leaf node is reached, the DSU reflects the tree after doing all toggles from events [1, i]
- Combine with Subtask 5's solution to maintain diameters

Time complexity: $O((N + Q)\log^2 N)$

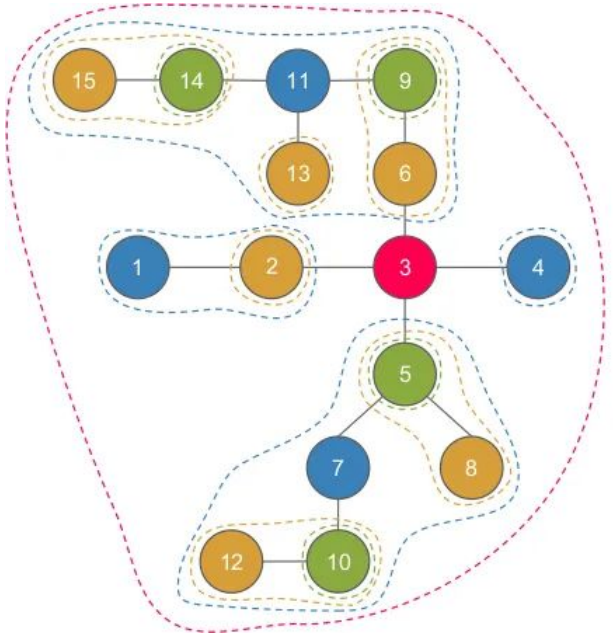
Expected Score: 100 (Cumulative: 100)

Subtask 6 (24%): No additional constraints

Solution 2: Centroid Decomposition

Fact: For each query x, there exists centroid ***C*** on the path x->u where dist(x, u) is maximum and ***C***'s component includes **both** x and u

- Since there are at most logN components that include x (height of centroid decomposition tree ≤ logN), we try find max(dist(x, u)) for each component



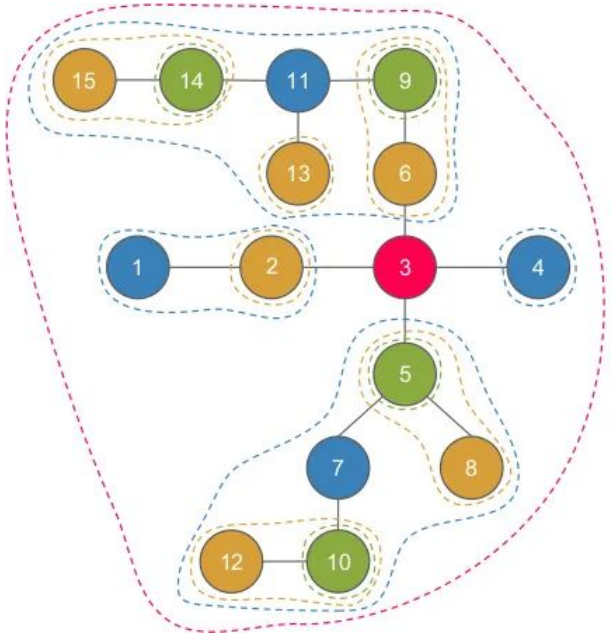
Subtask 6 (24%): No additional constraints

Solution 2: Centroid Decomposition

How to maintain max(dist(x, u)) for a component?

*Assume that **C** lies on the path x->u, so root at **C***

- Flatten the component's tree with preorder traversal order
- Use segment tree(+, max) to maintain f(***C***, u) for each vertex u
  - ***C*** can reach u: f(***C***, u) = dist(***C***, u)
  - otherwise: f(***C***, u) = dist(***C***, u) - INF
  - Edge toggle: range add INF or -INF
  - Maintain range max

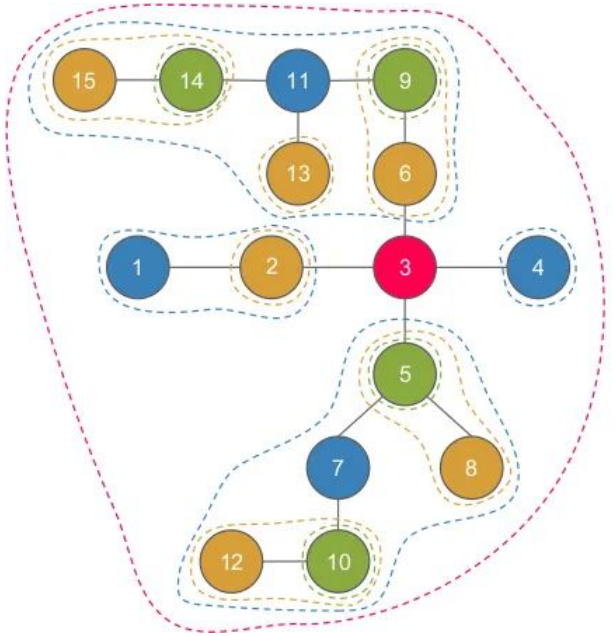


Subtask 6 (24%): No additional constraints

Solution 2: Centroid Decomposition
How to maintain max(dist(x, u)) for a component?
*Assume that **C** lies on the path x->u, so root at **C***

Let [l(a), r(a)] be the corresponding preorder traversal interval of vertex a, and let v be the node on the path x->u where par(v) = ***C***

- u cannot be in the subtree of v or it breaks the assumption
- Therefore, max(dist(x, u)) =
  query(l(x), r(x)) + max(query(0, l(v)-1),
  query(r(v)+1, size(component)-1)



Subtask 6 (24%): No additional constraints

Solution 2: Centroid Decomposition

- Take max over all log N components

Time complexity: $O((N + Q)\log^2 N)$

Expected Score: 76(if big constant) / 100 (Cumulative: 100)

Subtask 6 (24%): No additional constraints

Solution 3: Heavy Light Decomposition

Left as practice!

Time complexity: $O((N + Q)\log^2 N)$
Expected Score: 76(if big constant) / 100 (Cumulative: 100)

Remarks

From an exhausted tester:

- I used 2 hours to code the HLD solution, and got wrong answer and used AI to debug
- I used 1 hours to code the centroid decomposition solution, got MLE, optimize for 30 minutes and still get MLE and used AI to optimize further
- Choose your own journey :)

- Codeforces blog on diameters: https://codeforces.com/blog/entry/101271

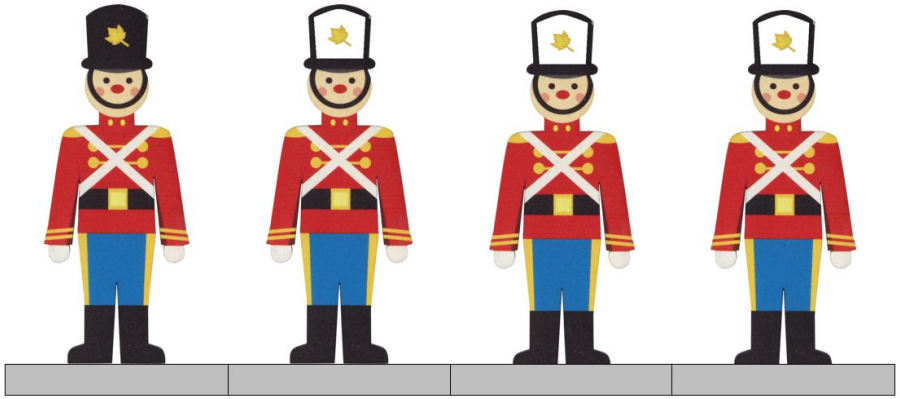
M2634
Paint and Guess

The Problem

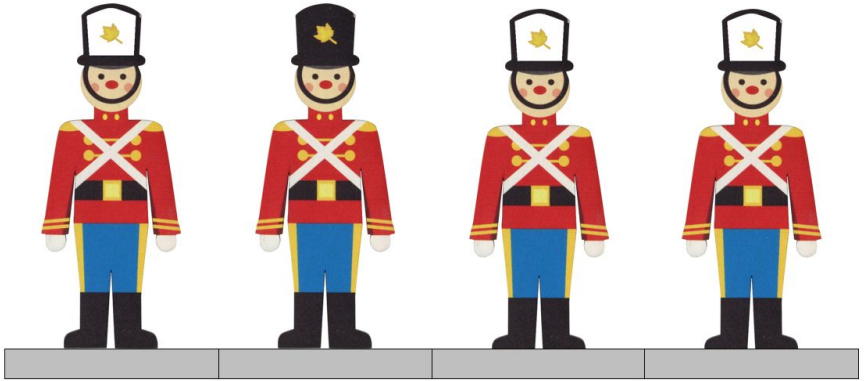
- There's N soldiers with unknown position P[i] (distinct, from 0 to N - 1).
- Queries: pass a length N 01 array C, paint the soldiers according to C, return the number of consecutive segments of same color.
- Find array P with the least number of queries.

Solution 1.1 (Find one by one)

- Paint every single soldier black to find the soldier at position 0.



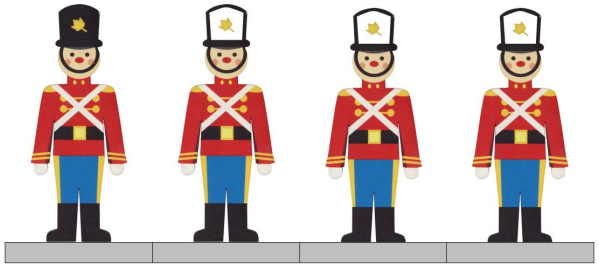
Count =2



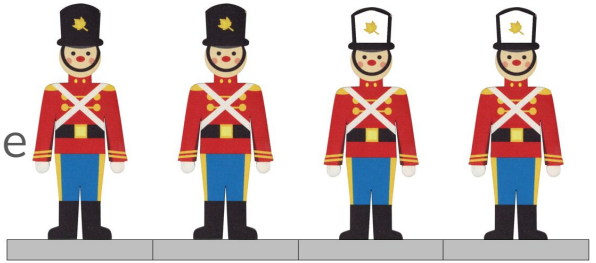
Count = 3

Solution 1.1 (Find one by one)

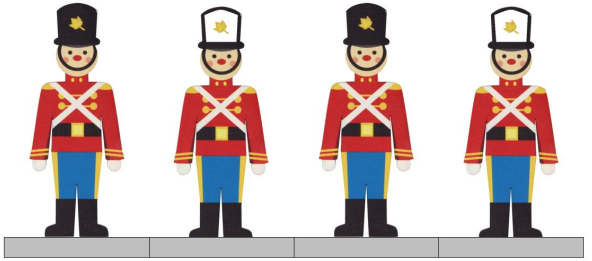
- Paint every single soldier black to find the soldier at position 0.
- Paint another soldiers until we find the soldier at position 1.
- If the soldier is beside soldier at position 0, the count remain unchanged.



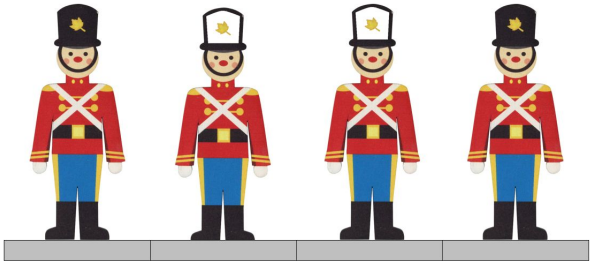
Count unchange



Count += 2



Count += 1



Solution 1.1

- Paint every single soldier black to find the soldier at position 0.
- Paint another soldiers until we find the soldier at position 1.
- If the soldier is beside soldier at position 0, the count remain unchanged.
- From left to right, find every single soldier.
- Number of queries = $N \times (N+1)/2$

Solution 1.2

- Paint every single soldier black to find the soldiers at position 0 and N-1.
- Paint another soldiers until we find the soldier at position 1 or N-2.
- Paint one more time to determine whether the soldier is at position 1 or N-2.
- Search from both sides, check the position once find a expected soldier.
- Random shuffle the index.
- Number of queries is ≈ N×(N+1)/6 + N

Solution 2

- Paint every single soldier black to find the soldier at position 0.
- Paint half of the remaining soldiers black.
- Paint the half of soldiers and soldiers at position 0 black.
- Check the difference to see whether one of the soldiers in this half is next to soldier at position 0.
- Binary search to find the soldiers one by one.
- Number of queries ≈ 2N log N

Solution 2.5

- Every soldiers are at most next to two soldiers.
- Maintain three sets of soldiers, soldiers at each set is not next to each others.
- For each soldier, use at most 2 queries to put it into a set. (Amortized N queries)
- For each soldier, use binary search to find the soldiers next to it in other set.
- Number of queries ≈ 8N/3 + N log N

Solution 3.1

- Preserve the adjacency relationships for the subset of soldiers from **index** 0 to i.
- For i+1, if it's next to any soldier in the set, the count will remain unchanged or decrease after adding it.
- Binary search to find the soldier(s) next to it.
- Number of queries ≈ N log N

Solution 3.2

- Optimizations on top of it:
- Only binary search on soldiers that has less than two neighbours found
- Don’t need to search the position of the last one.
- Number of queries ≤ 1415

Remarks

- Try to find the answer in different ways.
- Remove the useless things which increase the number of queries.
- Try to add random things to your solution, few lines but sometimes increase the score.

https://codeforces.com/blog/entry/88134?#comment-765758