

Team Parade — Editorial

Intended Solution

For every team c , store the sorted list of positions where contestants of team c appear. These lists are naturally sorted if the input is read from left to right.

We want to minimize the maximum allowed distance D between two consecutive chosen contestants.

The key observation is monotonicity: if the parade can be formed for some D , then it can also be formed for every larger D .

Therefore, we can binary search the answer D .

It remains to describe how to check, for a fixed value of D , whether a valid parade exists.

Feasibility Check for Fixed D

For team 1, every occurrence of team 1 is a possible ending position of a partial parade.

Suppose we have already processed teams $1, 2, \dots, c-1$. Let `previous` be the sorted list of all positions where a valid partial parade ending with team $c-1$ may end. Now we want to compute the list `current` for team c .

A position q of team c is valid if there exists some position p in `previous` such that

$$p < q$$

and

$$q - p \leq D.$$

Equivalently,

$$q - D \leq p < q.$$

Thus, for every position q of team c , we need to know whether `previous` contains at least one element in the interval

$$[q - D, q - 1].$$

Both lists are sorted, so this can be done with two pointers.

While the current pointer in `previous` points to a value smaller than $q - D$, we move it to the right.

After that, the pointer points to the first previous position which is not too far to the left. If that position exists and is smaller than q , then q is a valid ending position for the current team.

If after processing some team c the list of valid ending positions is empty, then no valid parade exists for this value of D . If after processing all K teams at least one valid ending position remains, then the parade can be formed for this value of D .

Complexity Analysis

For a fixed D , each position is considered at most once while processing its team. The two-pointer scan also moves only forward, so the total complexity of one feasibility check is

$$O(N).$$

The binary search performs $O(\log N)$ checks.

Therefore, the total time complexity is

$$O(N \log N).$$

The memory complexity is

$$O(N + K),$$

because we store the positions of all teams.

Solutions of the Subtasks

Subtask 1: $N = K$

Since every team from 1 to K appears at least once and there are exactly K contestants, every team appears exactly once.

For every team c , let p_c be the unique position of its contestant.

A valid parade exists if and only if

$$p_1 < p_2 < \dots < p_K.$$

If this condition does not hold, the answer is -1 .

Otherwise, there is only one possible parade, and the answer is

$$\max_{1 \leq c < K} (p_{c+1} - p_c).$$

The complexity is

$$O(N).$$

Subtask 2: $K = 2$

We only need to choose one contestant from team 1 and one contestant from team 2 to the right of them.

The goal is to minimize their distance.

Let P_1 be the sorted list of positions of team 1, and let P_2 be the sorted list of positions of team 2.

For every position q of team 2, we need the largest position p of team 1 such that

$$p < q.$$

Then

$$q - p$$

is the best distance for this fixed q .

This can be done with two pointers in $O(N)$ time. If no such pair exists, the answer is -1 .

Subtask 3: $K = 3$

We need to choose positions

$$p < q < r$$

such that

$$a_p = 1, \quad a_q = 2, \quad a_r = 3.$$

The value of such a choice is

$$\max(q - p, r - q).$$

We can try every possible middle position q where $a_q = 2$.

For this fixed q , the best choice for p is the largest position of team 1 which is smaller than q . Choosing any smaller position can only increase the distance $q - p$.

Similarly, the best choice for r is the smallest position of team 3 which is greater than q . Choosing any larger position can only increase the distance $r - q$.

Therefore, for each position q of team 2, we find

$$p = \max\{x < q \mid a_x = 1\}$$

and

$$r = \min\{x > q \mid a_x = 3\}.$$

If both positions exist, then this middle position gives the candidate answer

$$\max(q - p, r - q).$$

The answer is the minimum of these values over all positions q such that $a_q = 2$. If no such position gives a valid triple, the answer is -1 .

This can be implemented in linear time. For example, we may scan the array from left to right and store for every position the closest previous occurrence of team 1. Then we scan from right to left and store for every position the closest next occurrence of team 3. Finally, we check all positions of team 2.

The complexity is

$$O(N).$$

Subtask 4: The array a is sorted in nondecreasing order

Since the array is sorted and every team from 1 to K appears at least once, all contestants from the same team form one contiguous block, and these blocks appear in the order

$$1, 2, \dots, K.$$

Let the block of team c be

$$[L_c, R_c].$$

The first team should clearly be represented by the last contestant from its block, that is by position R_1 . Similarly, the last team should be represented by the first contestant from its block, that is by position L_K .

It remains to choose one position from every middle block.

We binary search the answer D . For a fixed value of D , we check whether such a choice is possible.

Let `best` be the rightmost position where a valid partial parade can currently end. Initially,

$$\text{best} = R_1.$$

Now process teams

$$2, 3, \dots, K - 1.$$

Suppose we are processing team c . We need to choose some position from the interval

$$[L_c, R_c]$$

which is at distance at most D from best .

Such a position exists if and only if

$$L_c \leq \text{best} + D.$$

If this condition is false, then this value of D is impossible.

Otherwise, among all reachable positions in the block of team c , it is always best to keep the rightmost one, because this can only help with the following teams. Therefore, we update

$$\text{best} = \min(R_c, \text{best} + D).$$

After all middle teams have been processed, we only need to check whether the first position of the last block is reachable:

$$L_K \leq \text{best} + D.$$

If this holds, then D is feasible; otherwise, it is not.

The feasibility check is linear in the number of teams. Therefore, together with binary search, this subtask can be solved in

$$O(N + K \log N)$$

time.

Subtask 5: $N \leq 2000$

A simple dynamic programming solution is sufficient.

For every position i , define dp_i as the smallest possible value of the maximum distance in a valid partial parade ending at position i .

If

$$a_i = 1,$$

then

$$dp_i = 0.$$

If

$$a_i = c > 1,$$

then we may come to position i from any earlier position $j < i$ such that

$$a_j = c - 1.$$

In that case, the new maximum distance is

$$\max(dp_j, i - j).$$

Therefore,

$$dp_i = \min_{j < i, a_j = a_i - 1} \max(dp_j, i - j).$$

The answer is the minimum value of dp_i over all positions i such that

$$a_i = K.$$

If no such valid value exists, the answer is -1 .

The complexity of this solution is

$$O(N^2),$$

which is sufficient for $N \leq 2000$.

Subtask 6: Each team has at most 50 contestants

Let P_c be the list of positions of team c .

Since each team appears at most 50 times, we can use dynamic programming over these lists.

For every position

$$q \in P_c,$$

we compute the best possible answer for a partial parade ending at q .

For every pair of consecutive teams $c - 1$ and c , we try all transitions

$$p \in P_{c-1}, \quad q \in P_c,$$

where

$$p < q.$$

The transition value is

$$\max(dp(p), q - p).$$

Since every list has size at most 50, the total number of transitions is at most

$$50^2 \cdot K.$$

This is easily fast enough.

Subtask 7: Full Constraints

For the full constraints, we use the intended solution.

We binary search the answer D .

For a fixed D , we run the two-pointer feasibility check described above.

The check keeps all possible ending positions after each team. When moving from team $c - 1$ to team c , a position q of team c is kept if the previous list contains some position p satisfying

$$q - D \leq p < q.$$

The lists are sorted, so this can be checked in linear time using two pointers.

The final complexity is

$$O(N \log N).$$