Editorial - Monuments

Problem summary

There are N points on the number line, where the i-th point is located at coordinate $X[i]$. Of these points, the $P[0], P[1], \ldots, P[M-1]$-th points are **red**, and the remaining $N-M$ points are **blue**. The red points cannot be moved, but the blue points can be moved.

We want to make the points symmetric with respect to $x = 0$, that is, for all $x > 0$, the number of points at coordinate x and coordinate $-x$ are the same. Determine whether it is possible, and if so, find the minimum total distance to move points.

Solutions

Subtask 1

Constraint: $M = N$, i.e. all points are red (3 points).

Since we cannot move any points, we need to check if the initial points are symmetric. We can do so by checking $X[i] = -X[N-i-1]$ for each i, since the array X is sorted. The answer is 0 if all points are symmetric, and -1 otherwise.

Subtask 2

Constraint: $M = 0$, i.e. all points are blue (4 points).

When all the points are blue, it is not optimal for a point to "overtake" another point. Hence, we need to pair the i-th point with the $(N-i-1)$-th point for a cost of $|X[i] + X[N-i-1]|$.

Note that if N is odd, we need to move the middle point to coordinate 0, for a cost of $|X[\frac{N-1}{2}]|$.

Subtask 4

Constraint: $N \leq 10$ (6 points).

Let's assume N is even. Since the constraints are small, we can brute force all the possible ways to create $\frac{N}{2}$ pairs of symmetric points in $O(N!)$ time. To make two points at coordinates x and y symmetric, we incur the following cost:

- $|x + y|$ if at least one of the points is blue;
- 0 if both points are red and $x + y = 0$;
- ∞ otherwise, i.e., this is an invalid pair.

If N is odd, we can just add a "dummy" red point at coordinate 0 and solve the problem for the even case.

Subtask 5

Constraint: $N \leq 19$ (5 points).

We can improve the $O(N!)$ solution by using bitmask DP. Let $dp[S]$ be the minimum cost to make all points in set S symmetric. For each set S, we can try adding two symmetric points or one point at coordinate 0, and minimize the cost. The number of transitions for each set will be $O(N^2)$, giving us a complexity of $O(2^N \cdot N^2)$.

Subtask 6

Constraint: $N \leq 32$ (5 points).

We can assume that there are no red points at symmetric coordinates. Since they are immovable and already in symmetric positions, simply removing them does not change the answer. With similar reasoning, we can assume there are no red points at coordinate 0.

Now we can assume that each red point will be paired with a blue point. If $M < N - M$, i.e., there are more blue points than red points, then the rest of the blue points will be paired with each other, with some (possibly 0) being moved to coordinate 0. If $M > N - M$, however, we can immediately return -1, since there will be at least one red point at a non-zero coordinate without a symmetric pair.

Let's brute force all the $\binom{N-M}{M}$ possibilities to choose the set of blue points paired with red points. To optimally assign the red-blue pairs, we can sort all red points and all blue points, and pair the i-th red point with the $(M-i-1)$-th blue point among the chosen M. To optimally assign the blue-blue pairs, we can use the solution for subtask 2.

The total complexity becomes $O\left(N \cdot \binom{N-M}{M}\right)$. For $N = 32$, the maximum value of $\binom{N-M}{M}$ is achieved when $M = 9$ with $\binom{23}{9} = 817,190$, so this solution passes the subtask.

Dividing into two halves

From subtask 2, we know that in an optimal solution no blue point "overtakes" another blue point. Also, if we assume there are no red points in symmetric coordinates, then the number of red points in negative (and positive) coordinates is fixed. Finally, in the final configuration of points, there will be exactly $\lfloor \frac{N}{2} \rfloor$ points on each half (if there are no 0s), so we can easily determine which blue points will end up in negative and positive coordinates.

Formally, let M_L and M_R be the number of red points in negative and positive coordinates, respectively. Also, let N_L and N_R be the number of blue points in the *final* configuration, in negative and positive coordinates, respectively. Then we have $M_L + N_L = M_R + N_R$. We can set both values equal to $\lfloor \frac{N}{2} \rfloor$ and get $N_L = \lfloor \frac{N}{2} \rfloor - M_L$ and $N_R = \lfloor \frac{N}{2} \rfloor - M_R$.

Then, N_L leftmost blue points will end up in negative coordinates, N_R rightmost blue points will end up in positive coordinates. If N is odd, then there will be one more point in-between, which we can move to coordinate 0.

New problem

We can rewrite the problem as follows. There are two parallel lines A and B, each with $K = \lfloor \frac{N}{2} \rfloor$ colored points: $A[0] \leq A[1] \leq \cdots \leq A[K-1]$ and $B[0] \leq B[1] \leq \cdots \leq B[K-1]$. Here, line A corresponds to the points in positive coordinates, whereas line B corresponds to the points in negative coordinates where the coordinates are *negated*, for simplicity.

Create a perfect matching between line A and line B, where the cost of matching the i-th point in line A and the j-th point in line B is $|A[i] - B[j]|$. However, these points cannot be matched if they are both red. Calculate the minimum total cost of matching all points.

Also, for simplicity, we divide each matching $(A[i], B[j])$ into four types:

1. Blue-blue matching – if both points $A[i]$ and $B[j]$ are blue.
2. Blue-red matching – if point $A[i]$ is blue and point $B[j]$ is red.
3. Red-blue matching – if point $A[i]$ is red and point $B[j]$ is blue.
4. Red-red matching – if both points $A[i]$ and $B[j]$ are red, which is not allowed.

Subtask 3

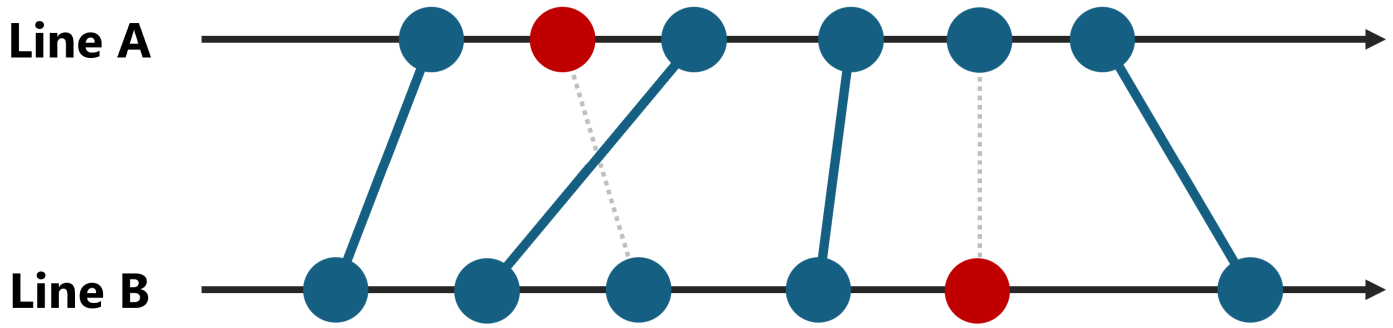
Constraint: $X[P[j]] < 0$ for each j such that $0 \leq j < M$, i.e. all red points are located in negative coordinates (5 points).

Since all red points are located in negative coordinates, there is no risk of red-red matching. So we can simply match the i-th point in A with the i-th point in B. Complexity: $O(N)$.

Subtask 7

Constraint: $N \leq 200$ (13 points).

Observe that, as shown in the figure below, the blue-blue matches never cross with each other. This is because, if there are two crossing matches, the solution will not be worse if we "uncross" these matches. So we can consider the DP from left to right.



We note that, for similar reasons, no blue-red matches will cross with each other, and no red-blue matches will cross with each other.

Let us move on to the DP solution. Let $dp[i][j][k]$ be the minimum cost for the state when we already considered the i leftmost blue points in line A, j leftmost blue points in line B, and created k blue-blue matches. In fact, this DP also takes blue-red and red-blue matches into account, because the blue points which are "skipped" (for the blue-blue matches) will be matched to red points.

For example, when we transition from the state of $dp[i][j][k]$ to $dp[i+1][j][k]$, it means that the i-th blue point in line A will be matched to $(i-k)$-th red point in line B.

Therefore, the optimal cost can be calculated in $O(N^3)$ time.

Subtask 8

Constraint: $N \leq 4000$ (17 points).

Another observation is that no blue-blue match crosses a blue-red match, since "uncrossing" these matches will not make the solution worse. Therefore, we can consider the improved DP that decides the blue-blue and blue-red matches from left to right.

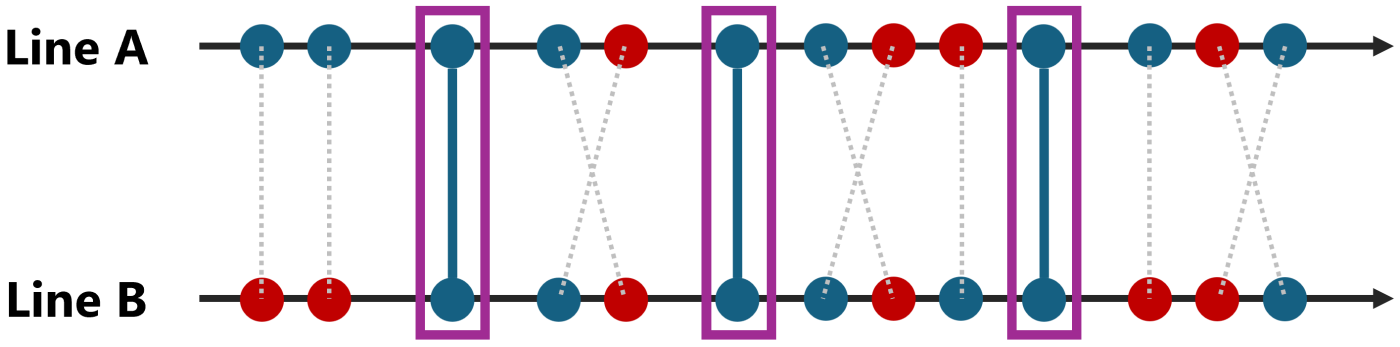
Let $dp[i][j]$ be the minimum cost for the state when we already considered the i leftmost blue points in line A and j leftmost (blue or red) points in line B. It means that we already created $j - i$ red-blue matchings, and increasing j by one means we match this point to the $(j - i)$-th leftmost red point in line A.

Therefore, the optimal cost can be calculated in $O(N^2)$ time.

More observations

We have shown that no blue-blue match crosses a blue-red match. Symmetrically, no blue-blue match crosses a red-blue match. Therefore, the crossing happens only between blue-red match and red-blue match (in this case, "uncrossing" these matches will generate an invalid "red-red match").

It means that any blue-blue match does not cross with any other matches. Therefore, the structure of the optimal solution is as shown in the figure below. Between two blue-blue matches, there are only blue-red or red-blue matches.



With this structure, we obtain a very useful property: all blue-blue matches connect the points with the same indices (i.e., the i-th point in line A and the i-th point in line B).

Therefore, we can consider the following DP. Let $dp[i]$ be the optimal cost when the i leftmost points in line A and the i leftmost points in line B are matched.

There are two types of transitions:

- $dp[i] \to dp[i + 1]$: make a blue-blue match between the i-th point in line A and the i-th point in line B. We can use this transition only if $A[i]$ and $B[i]$ are both blue.
- $dp[i] \to dp[j]$: make a chunk of blue-red and red-blue matches between the $i, \ldots, (j - 1)$-th points in lines A and B. We can use this transition only if the number of red points in line A equals the number of blue points in line B, and vice-versa. We can calculate the cost as follows:
  - Each red-blue matching is formed by the k-th smallest red point in line $A[i \ldots j - 1]$ and the k-th smallest blue point in line $B[i, \ldots, j - 1]$;

- Each blue-red matching is formed by the k-th smallest blue point in line $A[i \dots j-1]$ and the k-th smallest red point in line $B[i, \dots, j-1]$.

Since we can calculate the cost of second transition in $O(j-i)$ time, the total complexity becomes $O(N^3)$, which solves subtask 7.

Subtask 8. Alternative solution

We can use prefix sums to speed up the previous solution. Let $cnt_{red}[i]$ be the number of red points among $A[0, \dots, i-1]$ and $B[0, \dots, i-1]$. Similarly, let $cnt_{blue}[i]$ be the number red points among $A[0, \dots, i-1]$ and $B[0, \dots, i-1]$. Then, let $S[i] = cnt_{red}[i] - cnt_{blue}[i]$.

It's not hard to notice that the second transition $dp[i] \to dp[j]$ is allowed if and only if $S[i] = S[j]$. However, we can notice that if there is a k ($i < k < j$) such that $S[i] = S[k] = S[j]$, then we can split the chunk of $i, \dots, j-1$ into two chunks $i, \dots, k-1$ and $k, \dots, j-1$.

Thus, we can replace the $dp[i] \to dp[j]$ transition with $dp[i] \to dp[k] \to dp[j]$ transitions for the same cost. If for each i, we choose the smallest j ($i < j$) such that $S[i] = S[j]$, then the number of transitions would decrease to $O(N)$, giving us a complexity of $O(N^2)$.

Subtask 9

Constraint: $M \leq 4000$ (13 points).

We can observe that the solution in the previous section runs in $O(N + M^2)$ time, because there are at most $2(M+1)$ transitions of the second type, and for each $dp[i] \to dp[j]$ transition, we must have $j - i \leq M$.

Full solution

From the alternative solution of subtask 8, we can predetermine all the possible $O(N)$ transitions without their costs. We can calculate the total cost of all red-blue and blue-red matchings in the $i, \dots, j-1$ range more efficiently.

More formally, to calculate the total cost of red-blue matchings, we can solve the following subproblem.

- Given two arrays $r[0] \leq r[1] \leq \ldots \leq r[v-1]$ (red points from line A) and $b[0] \leq b[1] \leq \ldots \leq b[w-1]$ (blue points from line B). You need to answer q offline queries of the form (x, y, k), where each answer is $\Big|r[x] - b[y]\Big| + \Big|r[x+1] - b[y+1]\Big| + \ldots + \Big|r[x+k-1] - b[y+k-1]\Big|$.

We consider the contribution of each element $r[i]$ to the final answer. For $\Big|r[i] - b[j]\Big|$, the contribution of $r[i]$ is $(+r[i])$ when $r[i] > b[j]$, and is $(-r[i])$ if $r[i] \leq b[j]$. Since both arrays are sorted, the contribution of $r[i]$ is $(+r[i])$ if and only if $j - i$ is less than the some threshold, which is determined for each i individually. A symmetric statement holds for $b[j]$.

Therefore, we can process the queries in the increasing order of $y - x$. We can use a Fenwick tree to maintain the contribution of each element in the array, so the answer can be calculated by a simple range-sum query. When $y - x$ exceeds a threshold, we can update the contribution of the corresponding element.

All operations cost logarithmic time per query, solving the subproblem in $O((v+w+q)\log(v+w))$ time.

Total cost for all blue-red matchings can be calculated symmetrically.

Therefore, by using this solution to calculate the costs of transitions in DP, we can solve the original problem in $O(N \log N)$ time.

Further remarks

It turns out this problem can be solved in $O(N)$ time. Instead of deciding, chunk by chunk, which point of A is matched to which point of B, we scan all $2K$ points of both lines in one sorted order and only decide, for each point, whether it opens a match (contributing $-a$) or closes one (contributing $+a$).

Let $dp[i][c]$ be the minimum cost after processing the first i points of the merged sequence, provided that:

- all open points lie on one line, namely:
  - line A, if there are more points from line A than from line B;
  - line B, otherwise.
- all open points have color c, where $c \in \{\mathrm{red}, \mathrm{blue}\}$.

By using prefix sums and handling the case-work in transitions, one can calculate the DP in $O(N)$, solving the entire problem in linear time.