Kick Start 2021 - Round D

Analysis: Final Exam

Test set 1

For consistency, we will use $maxDifficulty$ to represent the maximum difficulty of a problem which is equal to 1000 for this test set. As the difficulty of any problem can be at most 1000, we can keep track of all the problems that can be used to test the students. This can be done by maintaining a boolean array $problemAvailable$ of size $maxDifficulty$ where $problemAvailable[i]$ is true if a problem is still available for difficulty i. We can iterate over the given $\mathbf{N}$ sets of problems and for each set i, mark that a problem exists for each difficulty x such that $\mathbf{A_i} \leq x \leq \mathbf{B_i}$. As no two sets contain the problems with same difficulty level, we will iterate over a particular difficulty level at most once. Hence, we can fill the boolean array $problemAvailable$ in $O(maxDifficulty)$ time.

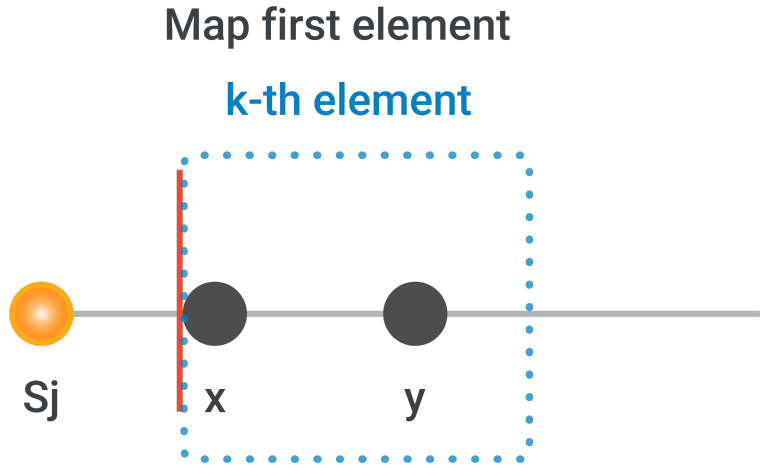
For each query j, we can iterate through all remaining problems and find out which problem has the difficulty d such that $abs(d - \mathbf{S_j})$ is minimum. This can be done in $O(maxDifficulty)$ time. After finding such problem, we can remove this problem from the remaining problems by marking $problemAvailable[d]$ as false. This can be done in constant time. Hence, we can answer each query in $O(maxDifficulty)$ time. The overall complexity of the solution is $O(\mathbf{M} \times maxDifficulty)$.

Test set 2

We cannot list problems with all possible difficulties as maximum difficulty can go upto 10^{18} in this test set. As the problem sets contain problems with disjoint difficulties, we can store the problem sets as a range. This can be done by using a map where key denotes the starting value of the range of difficulty and value denotes the ending range of difficulty. To store the ranges from the initial problem sets, we would need insertion operation in map. Hence, we can store the ranges in $O(\mathbf{N} \times \log \mathbf{N})$ time complexity initially.

For a query j, we need to lookup into the map to find the problem with difficulty d such that $abs(d - \mathbf{S_j})$ is minimized. We can perform an upper bound operation on map. In C++, one can use std::map::upper_bound method. upper_bound(z) returns first element greater than z. Suppose upper_bound($\mathbf{S_j}$) returned k-th element of the map. There are several cases that could occur:

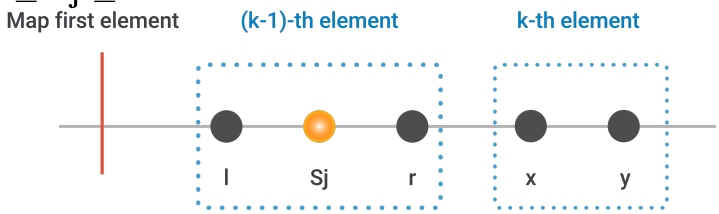
- $k = 1$.



This means that there is no range in map which could contain $\mathbf{S_j}$ as the first range in map

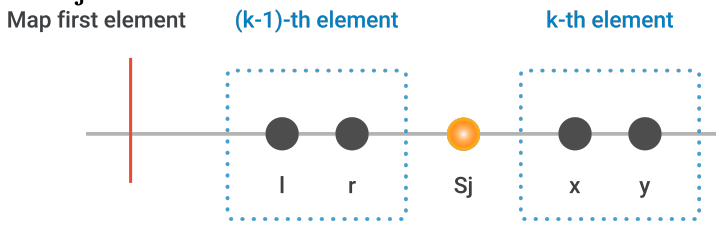
contains all values greater than $\mathbf{S_j}$. Let the range corresponding to first entry by (x, y). The problem with difficulty x will be the one with closest value to $\mathbf{S_j}$. Thus, we can choose problem with difficulty x for this query. We can now update this range to $(x+1, y)$ as problem with difficulty x is not available anymore. If $x+1 > y$, we can remove this range.

- Let the range corresponding to $(k-1)$-th element in map be (l, r). There are two cases here:
  - $l \leq \mathbf{S_j} \leq r$.



This means that the problem with difficulty $\mathbf{S_j}$ exists. The answer to this query would be the problem with difficulty $\mathbf{S_j}$. As this problem is not available for the remaining queries, we can remove the range (l, r) and insert 2 separate ranges $(l, \mathbf{S_j}-1)$ and $(\mathbf{S_j}+1, r)$. If any of the ranges is invalid, we can ignore that range.

  - $r < \mathbf{S_j}$.



This means that there is no problem with difficulty $\mathbf{S_j}$. If $\mathbf{S_j} - r \leq x - \mathbf{S_j}$, then we can select the problem with difficulty r as it is closest to $\mathbf{S_j}$. We can update the range (l, r) to $(l, r-1)$ if it is a valid range. Otherwise, we can select the problem with difficulty x. We can update the range (x, y) to $(x+1, y)$ if it is a valid range.

Each query can lead to addition of at most 1 new entry into the map. Hence, the size of the map would be $O(\mathbf{N}+\mathbf{M})$. We perform $O(1)$ operations each of which takes $O(\log(\mathbf{N}+\mathbf{M}))$ time to compute answer for a single query. So, we can answer $\mathbf{M}$ queries in $O(\mathbf{M} \times \log(\mathbf{N}+\mathbf{M}))$ time. Hence, the overall complexity of the solution is $O(\mathbf{N} \times \log \mathbf{N} + \mathbf{M} \times \log(\mathbf{N}+\mathbf{M}))$.