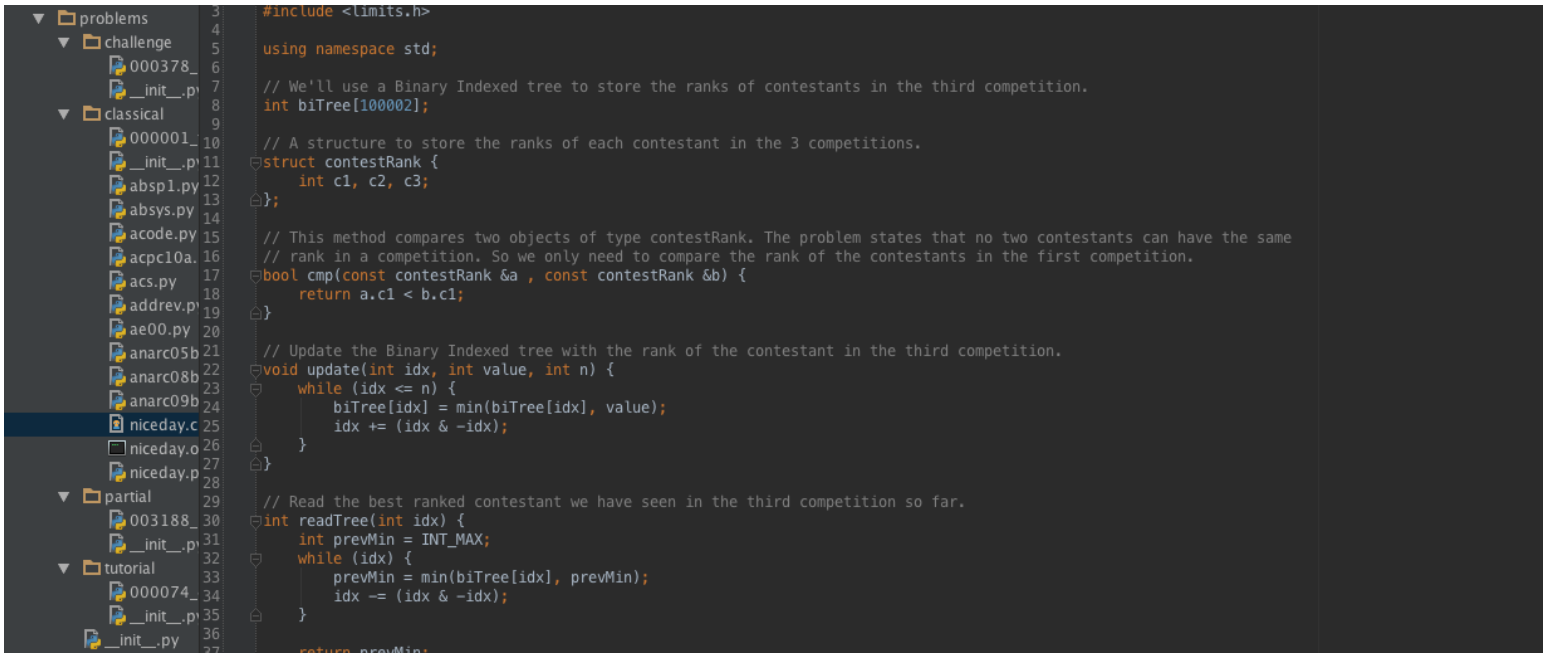
Anuvrat Singh Follow
Apr 27 · 6 min read

SPOJ: NICEDAY — The Day of the Competitors



Problem

Contestants are evaluated in 3 competitions. We say that:

- A contestant A is better than B if A is ranked above B in all of the three competitions they were evaluated in.
- A is an excellent contestant if no other contestant is better than A.

Given the ranks of all the contestants that participated in the 3 competitions we are required to find out how many of them are excellent. We also know that no two contestants can have the same rank in a competition.

Brute Force Solution:

The brute force way would be to compare each contestant with another one and eliminate those that are worse than a contestant in each of the three competitions. Whoever we are left with must be excellent. This solution is $O(n^2)$ in complexity and will not scale with increasing number of contestants. We obviously need to do better.

A Hint:

Indexed Trees help in this case.

Binary Indexed Trees:

I did not know about the Binary Indexed Trees, so I started reading up. I found good reading materials on BIT at:

- Geeks for Geeks
- TopCoder
- Theory of Programming
- Stack Exchange

This is an amazing data structure to perform operations over multiple elements of an array in O(log n) when they would have taken O(n) otherwise. They should be used if you have to perform operations like:

- Increment the value of the elements in an array for indices from i to j.
- Get the sum of consecutive elements of an array.

Implementing a BIT is very easy. We can use an array to hold the tree. From a child node you can go to the parent by a simple bit operation—replace the least significant 1 in the binary value of the child index with 0. Getting from parent to child involves performing the same bit operation in reverse.

Observations:

The following two observations make the problem a piece of cake:

Ob1: Sort The Contestants

Sorting the contestants by their rank in one of the competition simplifies the problem significantly. By definition a contestant is excellent if no other contestant is better than him. Also remember that a contestant is better than another only if he is ranked above the other in each of the three competitions. So if we sort everyone by their ranks in competition C1 then we can simply forget about that competition.

Let's say the sorted contestants and their ranks in the three competitions are:

- P1: 1, 2, 3
- P2: 2, 3, 4
- P3: 3, 4, 1
- P4: 4, 1, 2

Note: It's easy to see that all except P2 are excellent contestants.

we only need to worry about the ranks of the contestants in the remaining two competitions.

Ob2: Better Than the Best Seen So Far

Let's take the contestant P3. We know two contestants are better than P3 in the first competition. P3 is excellent if those above him are worse than him in at least one competition. Here's my claim:

> *If P3 is better in third competition than the best rank we have seen so far among all candidates having rank less than P3s rank in second competition then P3 is an excellent candidate.*

To show that the reason above claim holds true let's consider the counter case. Let's assume that P3s rank in the third competition is not better than the best rank we have seen in that competition so far among all candidates having rank less than P3 in the second competition. This implies that there is a candidate who got better rank than P3 in each of the three competitions and so that candidate is a better candidate than P3, which means that P3 is not an excellent candidate.

You must have noticed that we don't really care about the candidates who are ranked below P3 in the first competition as they can never be better than P3.

How does BIT help?

BIT works wonders here. Really!! In our BIT the index is rank in second competition. The value of each index is the best rank in the third competition that we know of. And given the way read() operation is implemented in BIT we can get the best rank in the third competition for all contestants ranked at most r in the second competition by looking up the value of index r in the BIT. The read complexity is just O(log n).

Implementation:

We initialize a BIT with positive infinity (or any number greater than the number of contestants we have). We sort the contestants by their rank in first competition. While we iterate over the sorted array and are looking at the ranks for player P:

- Get the best rank we have seen in third competition for all players having rank less than the rank of P in the second competition from BIT. This takes O(log n) time.
- If the rank of P in third competition is less than the best rank we got above increment the count of excellent candidates by 1.
- Update the BIT by calling update() method for index = rank of P in second competition and value = rank of P in the third competition. The update() method updates the best rank in the BIT in O(log n) complexity.

And we are done! We can now implement a O(n log n) solution in 15 minutes.

Oh, by the way, my Python code timed out. I spent 3 hours trying to port the same code in C++ (it was the first C++ code I have written in years now)! I have provided both the codes below. Pardon my C++ code style as I am too used to coding in Java and Python now.

```
#include <stdio.h>
#include <algorithm>
#include <limits.h>

using namespace std;

// We'll use a Binary Indexed tree to store the ranks of
contestants in the third competition.
int biTree[100002];

// A structure to store the ranks of each contestant in the
3 competitions.
struct contestRank {
    int c1, c2, c3;
};

// This method compares two objects of type contestRank.
The problem states that no two contestants can have the
same
// rank in a competition. So we only need to compare the
rank of the contestants in the first competition.
bool cmp(const contestRank &a , const contestRank &b) {
    return a.c1 < b.c1;
}

// Update the Binary Indexed tree with the rank of the
contestant in the third competition.
void update(int idx, int value, int n) {
    while (idx <= n) {
        biTree[idx] = min(biTree[idx], value);
        idx += (idx & -idx);
    }
}

// Read the best ranked contestant we have seen in the
third competition so far.
int readTree(int idx) {
    int prevMin = INT_MAX;
    while (idx) {
        prevMin = min(biTree[idx], prevMin);
        idx -= (idx & -idx);
    }

    return prevMin;
}

int main() {
    int t;
    scanf("%d", &t);

    // Iterate over all the test cases.
    while (t--) {
        int n;
        scanf("%d", &n);

        // Read all the contestant ranks as input.
        contestRank ranks[n];
        for (int idx = 0; idx < n; idx++) {
            scanf("%d%d%d", &ranks[idx].c1, &ranks[idx].c2,
&ranks[idx].c3);
        }

        // We sort all the contestants by their ranks in
the first competition.
        sort(ranks, ranks + n, cmp);

        // Prepare the tree by setting each elements value
```

```
        fill(bitree, bitree + n + 2, INT_MAX);

        // For each contestant figure out if it is an
excellent contestant or not.
        int excellent = 0;
        for (int idx = 0; idx < n; idx++) {
            int curr = readTree(ranks[idx].c2);

            if (curr > ranks[idx].c3) {
                excellent++;
            }

            update(ranks[idx].c2, ranks[idx].c3, n);
        }

        printf("%d\n", excellent);
    }

    return 0;
}
```

And now the same stuff in sweet Python:

```
from sys import stdin


def read(pos):
    prev_min = max_value
    while pos > 0:
        prev_min = min(bi_tree[pos], prev_min)
        pos -= (pos & (-1 * pos))
    return prev_min


def update(pos, value):
    while pos <= competitors:
        bi_tree[pos] = min(bi_tree[pos], value)
        pos += (pos & (-1 * pos))

if __name__ == '__main__':
    for _ in xrange(int(stdin.readline())):
        competitors = int(stdin.readline())

        ranks = []
        for _ in xrange(competitors):
            ranks.append(map(int,
stdin.readline().split()))

        ranks = sorted(ranks, key=lambda r: r[0])

        excellent = 0
        max_value = competitors + 1
        bi_tree = [max_value] * (competitors + 1)
        for idx in xrange(competitors):
            curr = read(ranks[idx][1])
            if curr > ranks[idx][2]:
                excellent += 1
            update(ranks[idx][1], ranks[idx][2])

        print excellent
```