Editorial: Teams

No preprocessing, O((n + m) log n) per query

The following greedy assignment works: take the smallest team size k and repeat k times: assign the child with a_i <= k and smallest b_i >= k. It can be implemented in time O((n + m) log n) by processing the teams and children in increasing order and maintaining a priority queue of children available for assignment.

Constructive approach

Now we enter the world of geometry. If we map the children to points (a_i, b_i), forming a set P, a team of size k can be mapped to a rectangle R(k) = [0, k] x [k, +oo]. We can reformulate our query as follows: - is it possible to simultaneously assign k_i points to a rectangle R(k_i)? (no point can be assigned to 2 or more rectangles)

From now on we assume that we preprocess the set P, so that we can query the number of points in some rectangle [x1,x2] x [y1,y2] in time O(log n). This can be done for example using a persistent segment tree or some other method.

We again try to form teams in the order of increasing ks. The idea is to maintain the set of "forbidden" points, that is, the region of plane containing the points which are either assigned to some previously processed rectangle, or not available because y_i < k.

We need to assign k lowest points (smallest y) from outside the forbidden region and update the forbidden region. If we represent the region by corners c_1, c_2, ..., c_z, the update will remove some (maybe 0) of consecutive corners starting before c_2 and insert a single new corner.

Therefore, we only need to find the right place for the new corner.

O(n log n) preprocessing, O(m^2 log n + m log^2 n) query

We can find the appropriate corner c_l such that there are at least k allowed points in a rectangle [0, k] x [0, c_l] in time O(l log n) (we test sequentially c_1, c_2, ..., each time asking about points in some rectangle and summing them up).

Once we found c_l, we know that the y coordinate of our new corner will be somewhere between y(c_{l-1}) and y(c_l). We can binary search for the exact y in time O(log^2 n).

O(n log n) preprocessing, O(m sqrt(n) log n) amortized query

We can combine the two solutions above to obtain a better bound: - we run the first solution if m > sqrt(n), - we run the second solution if m <= sqrt(n).

O(n log n) preprocessing, O(m log^2 n) query

If we store the corners in some kind of binary search tree and decompose the forbidden area into rectangles (picture) so that each corner knows how (is responsible for) many points are there in its rectangle, then we can binary search for the y of new corner directly: each guess will involve a range sum in the corners structure and a single query to the points structure.

Thus, the running time is O(m log^2 n).

Non-constructive approach

The above solutions, although implicitly, construct the assignments. However, our question is binary and thus another approach is possible.

The Hall theorem says the following:

It's not possible to assign children to teams if there exists such subset A of team sizes k_i, that the set of points that can be assigned to any team from A (let's call those points neighbors) is smaller than the sum of numbers in A.

Therefore, we want to construct such set A, that the number c(A) = |neighbors of A| - (sum of numbers in A) is smallest possible. If the smallest c(A) turns out to be negative, then the answer is NO, otherwise it's YES.

Assume that k_i's are distinct and sorted (just for clarity).

We can propose a simple dynamic programming solution.

Let D_i be the minimal c(A) such that k_i is the greatest element of A. Then: D_i = min { D_j + Z(j, i) - k_i : j < i}, where Z(j, i) = |children [a,b] s.t. a \in [k_j + 1, k_i], b >= k_i|

Optimal c(A) is thus equal to min { D_i }.

Another O(m^2 log n) solution

As computing Z(j, i) is asking about number of points in some rectangle, we can implement this DP in O(m^2 log n).

Another O(m sqrt(n) log n) amortized solution

This can be again combined with the brute force solution to speed it up.

O(m log n) solution

Let us assume, that we have three indices i < j < k, such that it's more beneficial to take index i than j, while computing the minimum in the formula for D_k. Then, for any l > k, it's also more beneficial to take index i instead of j. Why? We have from out assumption:

D_i + Z(i, k) <= D_j + Z(j, k), which is equivalent to:

D_j - D_i >= | children [a,b] s.t. a \in [k_i + 1, k_j], b >= k_k |.

If we replace k_k with k_l >= k_k, the right side won't increase, so indeed D_i + Z(i, l) <= D_j + Z(j, l).

Therefore, for any indices i < j we can compute the exact moment W(i, j) of DP computation, when the index i will be better than j.

It can be done in time O(log n).

The improved DP goes like this: when computing D_k, we maintain a set of those indices that might be useful according to our current knowledge. It also means that if indices i and j, i < j are in the set right now, then j is more beneficial. Hence, the last index from the set constitutes an optimal transition for D_k.

Maintaining the set of indices involves a queue of events. For each two indices i < j that happen to be neighbors in the set at some point of time, we push the event "remove j from the set once you reach computation of D_l", for some l.

There are O(m) set updates/accesses and each time we use O(log n) time to compute a moment when j > i is useless.

It remains to show, how to preprocess the input points to be able to find W(i, j) in time O(log n). Let B = D_j - D_i.

We need to find smallest y, such that there are at least B input points in [k_i + 1, k_j] x [y, +oo]. This can be solved with a variant of a segment tree. However, in a node [A, B] of the segment tree we store all the points (children) with y in [A, B]. The points inside a single node are sorted by increasing x and each point stores the pointers to:

- the first point with x' >= x and last with x' <= x in [A, mid]

- the first point with x' >= x and last with x' <= x in [mid + 1, B].

This allows us to binary search for k_i and k_j only in the root of the segment tree, and then just follow the pointers on a path to the leaf.