ENS Lyon. Day 2. Basic group: Problem Analysis

October 27, 2015

Problem A. Sum

Statement

- Given array of n elements;
- Queries: find sum in the interval, change value of an element.

Solution

- Segment tree

Problem B. Range Variation Query

Statement

- Given array of n elements;
- Queries: find difference between the maximal and minimal values in the range, change value of an element.

Solution

- Keep two segment trees: one for minimum and one for maximum;
- Update elements in both trees.

Problem C. Sum 2

Statement

- Given an array of n elements;
- Queries: find the sum over all elements in the range, change value of all elements in the range.

Solution

- Segment tree with range update;
- Lazy propagation.

Problem D. RMQ

Statement

- Given array of n elements;
- Queries: find the maximum in the range, add value to all elements in the range.

Solution

- Segment tree with range update;
- Lazy propagation;.

Problem E. Signchange

Statement

- Given array of n elements;
- Queries: find the alternating sum of elements in the range, change value of an element;

Solution

- Keep two segment tree: one for the elements with odd indices and one for the elements with even indices;
- Update element in the corresponding tree.

Problem F. K-inversions

Statement

- k-inversion is a sequence of numbers $i_1, \ldots, i_k$ such that $1 \leq i_1 < \cdots < i_k \leq n$ and $a_{i_1} > \cdots > a_{i_n}$;
- Find the number of k-inversions in the given permutation;
- It is the same to find the number of decreasing subsequences of length k.

Problem F. K-inversions

Solution

- DP: $dp_{j,i}$ is the number of decreasing subsequences of length j that ends in i;
- $dp_{j,i} = \sum_{k<i, a_k>a_i} dp_{k,j-1}$;
- Do not keep all matrices, only two last layers;
- Sum could be calculated efficiently using segment tree.

Problem G. RMQ Inverse Problem

Statement

- $Q(i, j)$ — minimum element in the range from i to j;
- Restore array, given the sequence of queries and responces;

Solution

- Initialy array is filled with infinity;
- For each query $Q(i, j) = x$ set value x for all elements in range from i to j;
- Array does not exist if you try to set value in the vertex where it was set before;

Problem H. Bus

Statement

- Passengers enters and exists;
- Minimize the number of times, when one passenger passes another

Problem H. Bus

Solution (idea)

- Three cases:
  - $[a_i, b_i] \subset [a_j, b_j]$, seat of i should be closer to entry;
  - $[a_i, b_i] \cap [a_j, b_j] = \emptyset$, they will not pass each other;
  - Otherwise, they will pass each other exactly once anyway;
- Place passengers in order of increasing of theirs a_i;
- To calculate a number of passes algorithm use segment tree.

Problem H. Bus

Solution

- Scanline with two kinds of events: enters and exits;
- When i-th passenger enters, assign 1 to his seat;
- When i-th passenger exits, assign 0 to his seat;
- For both kinds of events we use set and sum in segment tree.

Problem I. Rectangles

Statement

- Given n rectangles, any two of them do not have common points;
- B is `farther` than A if B's top left corner lies strictly below and strictly right than bottom right corner of A;
- `Chain` is sequence $R_1, \ldots, R_k$, s.t. for all i R_i is father than R_{i-1};
- `Weight of chain` is sum of the numbers inside rectangles;
- Find the chain with maximal weight.

Problem I. Rectangles

Solution

- Sort rectangles by their x value;
- Compress y coordinates, scanline, two kinds of events:
  - ST keeps length of maximal weight of chain;
  - Rectangle i starts; add a_i to range $[y_m in, y_i]$;
  - Find the maximum in the segment tree;
  - To restore the answer.

Problem J. Windows

Statement

- Given n overlapping rectangular windows;
- Find point covered by the maximal number of windows.

Problem J. Windows

Solution

- Scanline, sort rectangles by x coordinates, compress y coordinates;
- ST stores how many rectangles cover y-range;
- Events of two kinds:
  - Rectangular starts; add $+1$ in the range $[y_1, y_2]$;
  - Rectangular ends; add -1 in the range $[y_1, y_2]$;
- Find maximum in the array after each event of first type.

Problem K. Windows 2

Statement

- Given n overlapping rectangular windows
- Find the area covered by this windows

Problem K. Windows 2

Solution

- Scanline, sort rectangles by x coordinates, compress y coordinates;
- Calculate uncovered area, so the result is equal to the difference between all area and uncovered area;
- Events of two types:
  - Rectangular starts; add $+1$ in the range $[y_1, y_2]$;
  - Rectangular ends; add -1 in the range $[y_1, y_2]$;
- The minimum (0) corresponds to uncovered y-s;

Problem L. Two captains

Statement

- Two captains, three kinds of orders:
- `send` l r — send sailors toward the cannons with numbers from l to r;
  `back` l r — recall all his sailors from the cannons with numbers from l to r
  `rum` — bring bottle of rum
- If sailors obeing different captains stay on the same cannon, they kill each other
- Insert minimal number of `rum` orders to the plans to save sailors

Problem L. Two captains

Solution

- DP: $canFirst[i][j]$ — can the 1-st captain give i-th orders, if the 2-nd captain gave j orders; $canFirst[i][j]$ is *true* in the following cases:
  - i-th order is `back` or `rum`
  - order is `send` l r and there is no sailors of the second captain (after j orders) staying on cannons $[l, r]$
- The same dynamic for the 2-nd captain: $canSecond[i][j]$.

Problem L. Two captains

Solution

- The numbers of sailors of 2-nd captain on the cannons stores in ST;
  - `send` l r; add $+1$ from l to r;
  - `back` l r; add -1 from l to r;
  - `rum` — nothing happens;
- For each j-th order of the 2-nd captain do the following:
  - update ST;
  - Calculate $canFirst[i][j]$ for all values of i.

Problem L. Two captains

Solution

- DP: $rum[i][j]$ — minimal number of `rum` orders to save sailors after i orders of the 1-st captain and j orders of the 2-nd captain.

Problem L. Two captains

Solution

- How to calculate?
  - If $canFirst[i][j]$ and $canSecond[i][j]$ and they don't send their sailors to the same cannon, then $rum[i][j] = rum[i-1][j-1]$;
  - If $canFirst[i][j]$, then $rum[i][j] = rum[i-1][j] + 1$; 2-nd gave `rum` order;
  - If $canSecond[i][j]$, then $rum[i][j] = rum[i][j-1] + 1$; 1-nd gave `rum` order.

Questions

Questions?