The J-th Number

原案：宮村
解答：宮村、森
スライド：宮村

問題概要

- N個の配列があって、最初は全部空。
- a[i]番目からb[i]番目までの配列にv[i]を追加する。（1≦i≦M）
- x[i]番目からy[i]番目までの配列に含まれる全ての数（重複あり）のうちj[i]番目に小さい数を出力する。（1≦i≦Q）

- 1 ≦ N ≦ 10^9
- 1 ≦ M, Q ≦ 10^5

解法1:愚直にやる(TLE)

- 問題文にかかれている操作を素直に実行する。
- 計算量O(Q*N*M)で絶望的。

解法2:入力をvでソートする(TLE)

- 入力の順番はどうでもよいので、vの値でソートしておく。
- v[i]の小さい順に|[a[i], b[i]]∩[x,y]|を足し上げていき、初めて和がj以上になるところを探す。
- 1クエリあたりO(M)なので全体ではO(Q*M)。
- 定数軽いとはいえ、流石にTLEしそう。
- 適当に高速化したら多分通ります。

着眼点

- 「j番目に小さい数」は二分探索で求められそう。
- 何が分かれば二分探索が成立するか？

 →ある数pに対して、x[i]番目からy[i]番目までの配列にp以下の値がいくつあるか知りたい。

着眼点

- 区間加算、区間和を効率的に処理できればうまくいきそう。セグメント木とか？
- 最初の値を追加していく部分で、v[i]の小さい順に追加が行われたものだと考えてよい。
- 区間に値を追加している途中の状態を取り出せればうまくいく。
- 途中の状態を取り出したいのだから、

永続データ構造を使えばよい！

解法3:永続セグメント木+二分探索

- 区間加算と区間和をサポートした永続セグメント木を用意する。
- v[i]の小さい順に[a[i], b[i]]の範囲に1を加算する。このとき途中の版を保持しておく。
- 各出力クエリについて、版番号について二分探索してj番目に小さい数を求める。

注意点

- Nが大きいので、座標圧縮するか、必要な領域を動的に確保するか、または同一視できるノードを使いまわす必要がある。
- メモリを大量に使うので、特に複数ケースの場合などはリソースの管理を適切に行うこと。

計算量

- 空間計算量は
  O(M * log N)とかO(M * log(M+Q))とか。
- 時間計算量は
  O(M * log N + Q * log N * log M)など。

- 一応通るが、メモリも時間も結構厳しい！
- 永続データ構造とかセグメント木は大抵定数重い

解法4:クエリをまとめて二分探索

- さっきの解法は、区間和を求めるのにセグメント木なんて高級なものを使っていたからlogがついて遅かった。
- 区間加算と区間和の計算は累積和を使い、より効率的に処理できそう。

解法4:クエリをまとめて二分探索

- 入力クエリをvでソートする。
- 入出力クエリを全部持っておく。
- 入力クエリを前半と後半に分ける。
- 座標圧縮する。
- 前半部分の区間加算を累積和で処理する。
- 各出力クエリに対応する区間和とjの値を比較して、適宜手を加えながら出力クエリを分割する。
- 入出力クエリを前半と後半に分割できたので再帰的に処理する。

解法4:クエリをまとめて二分探索

- これを素朴に実装すると座標圧縮する部分のO((M+Q)*log (M+Q))がボトルネックになる。
- とはいえ、座標圧縮にかかるlog は定数軽いので素朴に実装しても十分間に合う。
- 少し工夫すると、2回目以降の座標圧縮は線形時間で実装できる。

計算量

- 空間計算量はO(M+Q)。
- 時間計算量は、再帰の深さがO(log M)で各深さでO(M+Q)かかる。結局初回の座標圧縮がボトルネックで全体ではO((M+Q) * log (M+Q))。
- この解法だと余裕を持ってACできる。