

非再帰セグ木サイコー！ 一番好きなセグ木です

えびちゃん

HCPC

November 21, 2019

今日の流れ

1. いつものコーナー
2. セグ木の導入
3. セグ木の実装
4. セグ木で解ける問題例
5. おわり

最近お勉強したもの (1/6)

えびちゃんが CHT の回以降に勉強したことたちです。興味があったら言ってね。

- simplex 法
 - 線形計画法を解くアルゴリズム
 - maximize $\mathbf{c}^\top \mathbf{x}$ subject to $\mathbf{Ax} \leq \mathbf{b}$ and $\mathbf{x} \geq 0$
 - 実用的には高速だが、最悪指数時間かかる
- wavelet matrix
 - 無限回言及したね
 - 整数の配列 $a[n]$ に関するいろんなクエリに答えられる
 - $i \in [s, t)$ のうちで $a_i \in [x, y)$ の個数は？
 - $i \in [s, t)$ のうちで x が k 番目に現れる場所は？
 - $i \in [s, t)$ のうちで k 番目に大きい値は？
 - $i \in [s, t)$ のうちで x 以上の要素の総和は？
 - $i \in [s, t)$ のうちの要素を頻出順に列挙して？

最近お勉強したもの (2/6)

- rolling hash の hack
 - 弱いものなら誕生日攻撃で終了
 - tree attack というのがあるらしい
- Mo's algorithm
 - Mo Tao (中国人)
 - 区間クエリ処理に関する枠組み
 - 区間の左端の位置でバケットに分ける
- 並列二分探索
 - クエリ処理に関する枠組み（に限らないかも？）

最近お勉強したもの (3/6)

- 永続配列
 - バージョン管理できる配列みたいなもの
- Karatsuba 法
 - 長さ n の畳み込みを $O(n^{\log_2 3}) \subset O(n^{1.59})$ 時間で行う
 - FFT と違って原始 n 乗根などがいらないので抽象化が楽そう?
- Fibonacci heap
 - amortized $O(1)$ で優先度を高くできる
 - Dijkstra 法や Prim 法で pop の回数を減らせて、 $O(|E| + |V| \log |V|)$ にできる
 - 改良版もいろいろある

最近お勉強したもの (4/6)

- disjoint sparse table
 - 静的なモノイドの配列で, $O(1)$ 時間で任意区間 fold できる
 - 空間は $n \cdot (\log_2 n + O(1))$ 要素ぶん
- sliding window aggregation
 - モノイドの両端キューで, ならし $O(1)$ 時間で fold できる
 - 空間は $2n$ 要素ぶん
- word-level parallel
 - word あたり w bits あるので w bits をまとめて演算できる
 - あるいは $w^{1/2}$ 個の $w^{1/2}$ bits の整数をまとめるとか
 - 最上位ビットを $O(1)$ で求められたりする

最近お勉強したもの (5/6)

- WQS binary search (trick from Aliens)
 - Qingshi Wang (中国人)
 - 関数がいい性質を満たすときのテク
 - 回数に関する制約をペナルティで置き換える
- removable CHT
 - $O((\log n)^3)$ でできるらしい
- x-fast trie
 - 整数 $[0, U)$ に関する集合演算を $O(\log \log U)$ 時間とかで
- 整数除算最適化
 - 除算は重くて加減乗算・ビット演算は軽い
 - $(n / 5) == ((n * 0xCCCCCCCCD_ju) >> 34)$
 - これは 32-bit 符号なし整数の例

最近お勉強したもの (6/6)

- monotone minima
 - monotone な $m \times n$ 行列について各行の最小値を $O(n \log m)$ 時間で求める
- SMAWK
 - すもうく～（線形時間）
 - totally monotone な $m \times n$ 行列について各行の最小値を $O(m + n)$ 時間で求める

非再帰セグ木の導入

ここから非再帰セグ木の話.

再帰セグ木を知っている人へ:

再帰セグ木を `stack` など無理やり書いたものではないです.

たぶんボトムアップセグ木とか呼ぶ方が適切?

セグ木の導入

配列 $a = (a_0, a_1, \dots, a_{n-1})$ が与えられる. 次のクエリを q 回処理してね. $1 \leq n \leq 10^5, 1 \leq q \leq 10^5$.

クエリセット (かんたん)

- i が与えられて, a_i の値を答える
- (i, x) が与えられて, a_i を x に書き換える

セグ木の導入

配列 $a = (a_0, a_1, \dots, a_{n-1})$ が与えられる。次のクエリを q 回処理してね。 $1 \leq n \leq 10^5, 1 \leq q \leq 10^5$ 。

クエリセット（かんたん）

- i が与えられて、 a_i の値を答える
- (i, x) が与えられて、 a_i を x に書き換える

配列を知っていますか？

セグ木の導入

配列 $a = (a_0, a_1, \dots, a_{n-1})$ が与えられる。次のクエリを q 回処理してね。 $1 \leq n \leq 10^5, 1 \leq q \leq 10^5$ 。

クエリセット（むずかしい）

- (l, r) が与えられて、 $\sum_{i=l}^{r-1} a_i$ の値を答える
- (i, x) が与えられて、 a_i を x に書き換える

セグ木の導入

配列 $a = (a_0, a_1, \dots, a_{n-1})$ が与えられる。次のクエリを q 回処理してね。 $1 \leq n \leq 10^5, 1 \leq q \leq 10^5$ 。

クエリセット (むずかしい)

- (l, r) が与えられて, $\sum_{i=l}^{r-1} a_i$ の値を答える
- (i, x) が与えられて, a_i を x に書き換える

for 文を知っていますか？

セグ木の導入

配列 $a = (a_0, a_1, \dots, a_{n-1})$ が与えられる. 次のクエリを q 回処理してね. $1 \leq n \leq 10^5, 1 \leq q \leq 10^5$.

クエリセット (むずかしい)

- (l, r) が与えられて, $\sum_{i=l}^{r-1} a_i$ の値を答える
- (i, x) が与えられて, a_i を x に書き換える

for 文を知っていますか?

↑計算量を知っていますか?

セグ木の導入

配列 $a = (a_0, a_1, \dots, a_{n-1})$ が与えられる. 次のクエリを q 回処理してね. $1 \leq n \leq 10^5, 1 \leq q \leq 10^5$.

クエリセット (むずかしい)

- (l, r) が与えられて, $\sum_{i=l}^{r-1} a_i$ の値を答える
- (i, x) が与えられて, a_i を x に書き換える

for 文を知っていますか?

↑計算量を知っていますか?

→セグ木を使うと $O(n + q \log n)$ で解ける.

基本アイディア

隣同士の和を求める。そのペアごとに、和を求める。というのを繰り返す。

$a_0 + a_1 + a_2 + a_3 + a_4 + a_5 + a_6 + a_7$							
$a_0 + a_1 + a_2 + a_3$				$a_4 + a_5 + a_6 + a_7$			
$a_0 + a_1$		$a_2 + a_3$		$a_4 + a_5$		$a_6 + a_7$	
a_0	a_1	a_2	a_3	a_4	a_5	a_6	a_7

Figure: セグ木の概念図

基本アイディア

隣同士の和を求める。そのペアごとに、和を求める。というのを繰り返す。

$a_0 + a_1 + a_2 + a_3 + a_4 + a_5 + a_6 + a_7$							
$a_0 + a_1 + a_2 + a_3$				$a_4 + a_5 + a_6 + a_7$			
$a_0 + a_1$		$a_2 + a_3$		$a_4 + a_5$		$a_6 + a_7$	
a_0	a_1	a_2	a_3	a_4	a_5	a_6	a_7

Figure: $a_0 + a_1 + \cdots + a_6$ に対応する区間

基本アイディア

隣同士の和を求める。そのペアごとに、和を求める。というのを繰り返す。

$a_0 + a_1 + a_2 + a_3 + a_4 + a_5 + a_6 + a_7$							
$a_0 + a_1 + a_2 + a_3$				$a_4 + a_5 + a_6 + a_7$			
$a_0 + a_1$		$a_2 + a_3$		$a_4 + a_5$		$a_6 + a_7$	
a_0	a_1	a_2	a_3	a_4	a_5	a_6	a_7

Figure: a_2 が関与する区間

木の表現

配列を用いる．根の添字を 1 として，幅優先順に番号をつける．

[1]			
[2]		[3]	
[4]	[5]	[6]	[7]

Figure: セグ木の添字

$[i > 1]$	
$[i]$	$[i-1]$
$[i < 1 0]$	$[i < 1 1]$

Figure: 親子の添字（綺麗！）

木の表現（悪い例）

配列を用いる．根の添字を 0 として，幅優先順に番号をつける．

[0]			
[1]		[2]	
[3]	[4]	[5]	[6]

Figure: セグ木の添字

親子の添字を求めるのが面倒なことがわかる（はず）．
なんでも 0-indexed にしたがる人は考え直した方がよい．

和を求めるクエリ

- 少ない区間で所望の区間をカバーしたい.
- 上の要素ほど大きな区間をカバーする.

→ できるだけ上の要素で足したい.

ある要素を足さなきゃいけない条件は？

→ その親の要素では所望の区間をはみ出してしまうとき.

和を求めるクエリ

所望の区間を半开区間で表す.

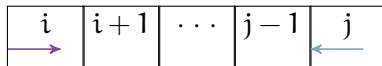


Figure: 半开区間 $[i, j)$ を意味する矢印

区間の左端・右端と, 左の子・右の子の4パターンを考える.

和を求めるクエリ

まず左端について考える.

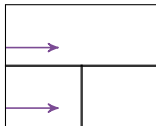


Figure: 左端・左の子

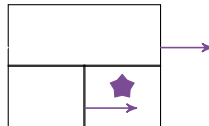


Figure: 左端・右の子

★ は, その要素を足すことを意味する. 足したら区間をずらす.

和を求めるクエリ

次に右端について考える.



Figure: 右端・左の子

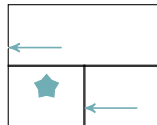


Figure: 右端・右の子

★ は, その要素を足すことを意味する. 足したら区間をずらす.

和を求めるクエリ

要素数 n の配列に対応するセグ木で, $[l, r)$ の和を求める.

```
x = 0;
l += n, r += n;
while (l < r) {
    if (l & 1) x += c[l++];
    if (r & 1) x += c[--r];
    l >>= 1, r >>= 1;
}
return x;
```

交換法則を仮定したくないときも, 少しいの手直しで対処可能 (読者への課題).

動作例

$a_0 + a_1 + a_2 + a_3 + a_4 + a_5 + a_6 + a_7$							
$a_0 + a_1 + a_2 + a_3$				$a_4 + a_5 + a_6 + a_7$			
$a_0 + a_1$		$a_2 + a_3$		$a_4 + a_5$		$a_6 + a_7$	
a_0	a_1	a_2	a_3	a_4	a_5	a_6	a_7

Figure: $[1, 7)$ の和を求めてみる

動作例

$a_0 + a_1 + a_2 + a_3 + a_4 + a_5 + a_6 + a_7$							
$a_0 + a_1 + a_2 + a_3$				$a_4 + a_5 + a_6 + a_7$			
$a_0 + a_1$		$a_2 + a_3$		$a_4 + a_5$		$a_6 + a_7$	
a_0	a_1	a_2	a_3	a_4	a_5	a_6	a_7

Figure: $[1, 7)$ の和を求めてみる

動作例

$a_0 + a_1 + a_2 + a_3 + a_4 + a_5 + a_6 + a_7$							
$a_0 + a_1 + a_2 + a_3$				$a_4 + a_5 + a_6 + a_7$			
$a_0 + a_1$		$a_2 + a_3$		$a_4 + a_5$		$a_6 + a_7$	
a_0	a_1	a_2	a_3	a_4	a_5	a_6	a_7

Figure: $[1, 7]$ の和を求めてみる

変更するクエリ

自分の項が関係するのは祖先ノードのみ.
→親を辿っていけばよい.

```
i += n;  
c[i] += x;  
while (i > 1) {  
    i >>= 1;  
    c[i] = c[i<<1|0] + c[i<<1|1];  
}
```

動作例

$a_0 + a_1 + a_2 + a_3 + a_4 + a_5 + a_6 + a_7$							
$a_0 + a_1 + a_2 + a_3$				$a_4 + a_5 + a_6 + a_7$			
$a_0 + a_1$		$a_2 + a_3$		$a_4 + a_5$		$a_6 + a_7$	
a_0	a_1	a_2	a_3	a_4	a_5	a_6	a_7

Figure: [2] の値を x に変更してみる

動作例

$a_0 + a_1 + a_2 + a_3 + a_4 + a_5 + a_6 + a_7$							
$a_0 + a_1 + a_2 + a_3$				$a_4 + a_5 + a_6 + a_7$			
$a_0 + a_1$		$a_2 + a_3$		$a_4 + a_5$		$a_6 + a_7$	
a_0	a_1	x	a_3	a_4	a_5	a_6	a_7

Figure: [2] の値を x に変更してみる

動作例

$a_0 + a_1 + a_2 + a_3 + a_4 + a_5 + a_6 + a_7$							
$a_0 + a_1 + a_2 + a_3$				$a_4 + a_5 + a_6 + a_7$			
$a_0 + a_1$		$x + a_3$		$a_4 + a_5$		$a_6 + a_7$	
a_0	a_1	x	a_3	a_4	a_5	a_6	a_7

Figure: [2] の値を x に変更してみる

動作例

$a_0 + a_1 + a_2 + a_3 + a_4 + a_5 + a_6 + a_7$							
$a_0 + a_1 + x + a_3$				$a_4 + a_5 + a_6 + a_7$			
$a_0 + a_1$		$x + a_3$		$a_4 + a_5$		$a_6 + a_7$	
a_0	a_1	x	a_3	a_4	a_5	a_6	a_7

Figure: [2] の値を x に変更してみる

動作例

$a_0 + a_1 + x + a_3 + a_4 + a_5 + a_6 + a_7$							
$a_0 + a_1 + x + a_3$				$a_4 + a_5 + a_6 + a_7$			
$a_0 + a_1$		$x + a_3$		$a_4 + a_5$		$a_6 + a_7$	
a_0	a_1	x	a_3	a_4	a_5	a_6	a_7

Figure: [2] の値を x に変更してみる

驚愕の事実？

上のコードは要素数 n を二ベキにしなくても動く．

$O(\log n)$ 個の完全二分木のセグ木が存在していると思わせる．

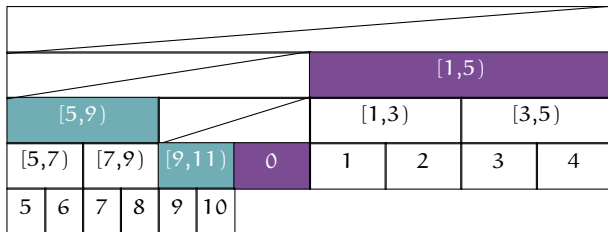


Figure: 11 要素の配列に対応するセグ木

驚愕の事実？

					[1,5)				
[5,9)					[1,3)		[3,5)		
[5,7)	[7,9)	[9,11)	0		1	2	3	4	
5	6	7	8	9	10				

Figure: 配列に詰め込んだ形

	[1,5)				[5,9)					
	[1,3)		[3,5)		[5,7)		[7,9)		[9,11)	
0	1	2	3	4	5	6	7	8	9	10

Figure: 実質的に扱う概念

抽象化

和に限ったデータ構造ではなく，モノイドならなんでもよい．

- $(a \circ b) \circ c = a \circ (b \circ c)$
- $\exists e \text{ s.t. } a \circ e = e \circ a = a$

$(\{i, i+1, \dots, i+n\}, \min)$ はモノイドを成す．単位元は $i+n$ ．

中級者向け：ロリハは接続についてモノイドを成す．

基数 b ，法 p とする．

長さ n_1, n_2 の文字列のハッシュ値 h_1, h_2 について，

$$(n_1, h_1) \circ (n_2, h_2) = (n_1 + n_2, h_1 \cdot b^{n_2} + h_2).$$

抽象化

モノイドになるようにうまく言い換えられるとうれしい.

たとえば, 正しい括弧列を考えてみる.

開き括弧を $+1$, 閉じ括弧を -1 として, 正しい括弧列である条件をうまく表せないか考えてみよう.

セグ木上のにぶたん (セグメント木上の二分探索)

非負整数の配列 $a = (a_0, a_1, \dots, a_{n-1})$ について, 先頭からの累積和が x を超えない限界はどこ? 形式的には, 以下のものが知りたくなってみる.

$$\max. s \text{ s.t. } \sum_{i=0}^{s-1} a_i \leq x.$$

$s \in [0, n+1)$ を決め打ちして, $[0, s)$ での和を求めてにぶたん?
→これを見ると $O((\log n)^2)$.

セグ木上のにぶたん

セグ木をよく見ると，すでに半々に分けられた構造をしていることがわかる．→左右どちらに辿るかを判定すればよい．

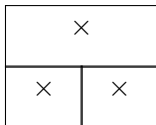


Figure: 左の子を辿る

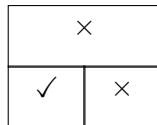


Figure: 右の子を辿る

左の子を足しても条件を満たすなら，足して右の子を辿る．そうでなければ，そのまま左の子を辿る．

✓はそこまで足しても条件を満たすことを，×はそこまで足すと条件を満たさないことを意味する．

セグ木上のにぶたん

完全二分木でないときは次のような感じになっている。

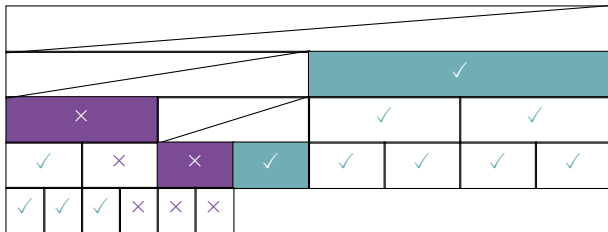


Figure: 11 要素の配列に対応するセグ木

木は $O(\log n)$ 個しかないので、どの根から始めるかを線形探索できる。任意位置を始点とすることもできる（考えてみよう）。

ヒストグラム: 区間 $[1, t)$ でいぶんすると t に見るべき根は, $[1, t)$ の和を求める t までに足す要素と同じ.

動的配列の模倣

以下のクエリを処理してね.

クエリセット (むずかしそう)

- 集合に要素 x を追加する
- 集合から要素 x を削除する
- 小さい方から k 番目の要素を答える

ただし, $0 \leq x < 10^5$. x を先読み可能なら $|x| \leq 10^9$ でも.

動的配列の模倣

以下のクエリを処理してね.

クエリセット (むずかしそう)

- 集合に要素 x を追加する
- 集合から要素 x を削除する
- 小さい方から k 番目の要素を答える

ただし, $0 \leq x < 10^5$. x を先読み可能なら $|x| \leq 10^9$ でも.

`std::set` では k 番目へのアクセスはできない.

動的配列の模倣

以下のクエリを処理してね.

クエリセット (むずかしそう)

- 集合に要素 x を追加する
- 集合から要素 x を削除する
- 小さい方から k 番目の要素を答える

ただし, $0 \leq x < 10^5$. x を先読み可能なら $|x| \leq 10^9$ でも.

`std::set` では k 番目へのアクセスはできない.

AVL 木や赤黒木を貼るしかない...? → セグ木でできます.

~~つよい bit set で 64 倍高速化してもいいです.~~

動的配列の模倣

$(\mathbb{N}, +)$ を管理するセグ木を使う.

要素 i が集合に入っているとき $a_i = 1$, otherwise $a_i = 0$ とする.

追加・削除は簡単に行える (0 または 1 を代入すればよいので).

k 番目の要素は, このセグ木で和が k になる添字に対応するので, セグ木上でのにぶたん.

図があった方がわかりやすそう. 次ページ.

動的配列の模倣

集合 $\{1, 4, 6, 9\}$ に対応する配列（実際にはセグ木で管理する）.
ここから 0-indexed で 2 番目の要素を探してみる.

0	1	0	0	1	0	1	0	0	1
[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]

$$\longrightarrow \sum_{i=0}^{6-1} a_i \leq 2$$

$$\longrightarrow \sum_{i=0}^{7-1} a_i > 2$$

（セグ木上のにぶたんができて）6 が答えだとわかる.

動的配列の模倣（抽象化）

以下の条件を満たして，入りうる全要素を並べられればよい．

- 同時に配列に入る要素間では，その順序が保たれている
- そうでない要素については不問

あとは，前ページ同様に $(\mathbb{N}, +)$ で添字を管理すればよい．

(T, \circ) のセグ木を用意して， x が入っているなら x ，そうでないなら単位元 e を入れておくと，添字 k までの fold など可能．

転倒数

転倒数というのを求めてみよう.

転倒数

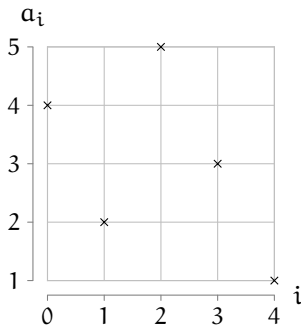
配列 $a = (a_0, a_1, \dots, a_{n-1})$ について, 次の条件を満たす (i, j) のペアの個数を転倒数と呼ぶ.

- $i < j$
- $a_i > a_j$

ウェーブレット行列はこのクエリに直接答えられるが...
→セグ木でできます.

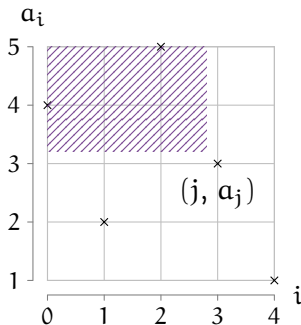
平面走査

例として, 配列 $a = (4, 2, 5, 3, 1)$ は次のように見なせる.



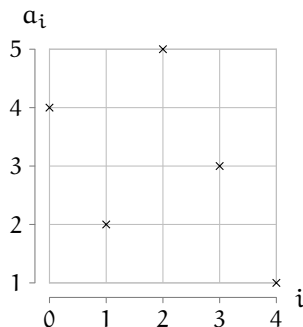
平面走査

点 (j, a_j) に関して, $i < j$ かつ $a_i > a_j$ は次の領域に相当する.



平面走査

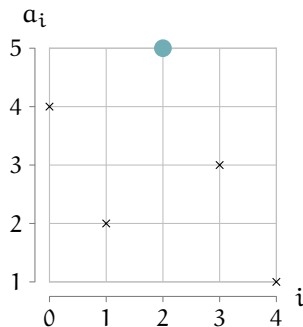
a_j が大きい方から順に見て, $b_j \leftarrow 1$ で更新する.



$$b = (0, 0, 0, 0, 0), \quad x = 0$$

平面走査

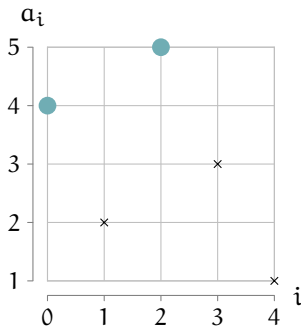
a_j が大きい方から順に見て, $b_j \leftarrow 1$ で更新する.



$$b = (0, 0, 1, 0, 0), \quad x = 0 + 0$$

平面走査

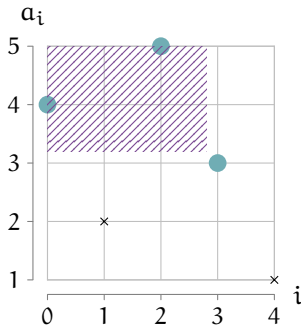
a_j が大きい方から順に見て, $b_j \leftarrow 1$ で更新する.



$$b = (1, 0, 1, 0, 0), \quad x = 0 + 0 + 0$$

平面走査

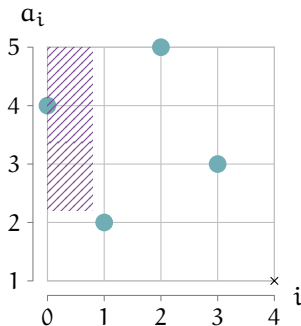
a_j が大きい方から順に見て, $b_j \leftarrow 1$ で更新する.



$$b = (1, 0, 1, 1, 0), \quad x = 0 + 0 + 0 + 2$$

平面走査

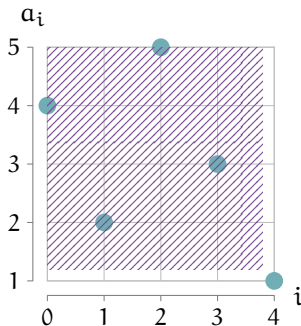
a_j が大きい方から順に見て, $b_j \leftarrow 1$ で更新する.



$$b = (1, 1, 1, 1, 0), \quad x = 0 + 0 + 0 + 2 + 1$$

平面走査

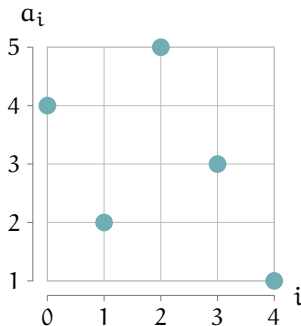
a_j が大きい方から順に見て, $b_j \leftarrow 1$ で更新する.



$$b = (1, 1, 1, 1, 1), \quad x = 0 + 0 + 0 + 2 + 1 + 4$$

平面走査

a_j が大きい方から順に見て, $b_j \leftarrow 1$ で更新する.

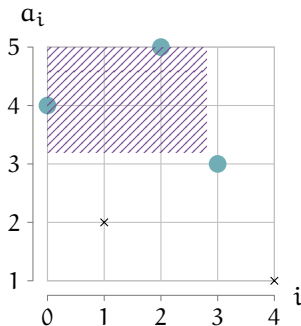


$$b = (1, 1, 1, 1, 1), \quad \kappa = 7$$

平面走査

以上により, $(4, 2, 5, 3, 1)$ の転倒数は7だとわかった.

a_j の順で行うことで, a_j の大小関係と処理順が対応づけられるのがうれしい.



平面走査

実装の際は, $i > j$ かつ $a_i < a_j$ とみて, a_j の小さい方から走査して, 右下の点の個数を数えてもよい.

同じ値の組が入っているときに, それらを余分に数えないように注意しよう. たとえば, (x, x) の転倒数が正しく 0 となるか確認しよう.

少し考えると, 同じような方法で最大増加部分列 (LIS) も平面走査で解けることがわかる. これは, 蟻本 pp. 63–65 に載っている方法とは異なる. ∞ に相当する要素を用意しなくていいのが利点としてありそう. 計算量は $\Theta(n \log n)$ のまま.

平面走査（応用例）

たとえば，ある条件を満たす区間 $[l, r)$ の個数を求めたいとする．ある f が存在して，その条件を満たすことと $f(l) \leq f(r)$ が同値であるとする．

→これは平面走査で解ける形をしているので，平面走査で解ける．

問題例

条件 P と配列 $a = (a_0, a_1, \dots, a_{n-1})$ が与えられる．次の条件を満たす区間 $[l, r)$ の個数を数えよ．ただし $1 \leq n \leq 10^5$ ．

- P を満たす要素の方が満たさない要素より真に多い

$P(a_i)$ の計算は高速にできると仮定してよい．

まとめ

セグ木では，モノイドの配列への更新と fold を高速に行える．
モノイドを利用できる形に言い換えられるとうれしい．
更新順序をうまく変えて平面走査するテクも有効．

非再帰セグ木の実装：

- 内部実装で 0-indexed にこだわるべきでない
- 二ベキに丸める必要もない

ところで再帰セグ木の紹介をしていませんね．

問題たち

- https://onlinejudge.u-aizu.ac.jp/courses/library/3/DSL/all/DSL_2_A
- <https://yukicoder.me/problems/no/875>
- https://atcoder.jp/contests/abc127/tasks/abc127_f
- https://atcoder.jp/contests/bitflyer2018-final/tasks/bitflyer2018_final_c
- https://atcoder.jp/contests/arc033/tasks/arc033_3
- <https://yukicoder.me/problems/no/877>
- https://atcoder.jp/contests/chokudai_S001/tasks/chokudai_S001_j
- https://atcoder.jp/contests/arc075/tasks/arc075_c
- https://atcoder.jp/contests/arc101/tasks/arc101_b
- https://atcoder.jp/contests/dp/tasks/dp_q

発展的な話

静的なら disjoint sparse table で $O(1)$ 回の演算で fold 可能.

よい性質を満たすとき, fold に加えて区間の更新も $O(\log n)$ 回の演算で可能. 遅延伝搬セグメント木 (遅延セグ木).

- 区間 min ・ 区間加算は可能
- 区間和 ・ 区間加算は可能
- 区間和 ・ 区間 min 更新は不可能 → segment tree beats!

beats は謎. なんかいろいろできるらしい.

終

制作・著作

