Segment Tree inclass

Lecture by 8e7

Credit to double、minson123、林品諺

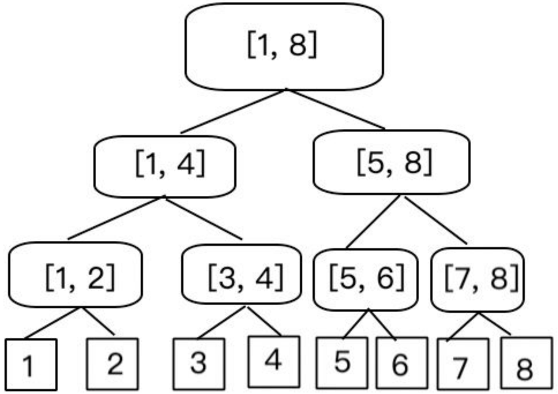
2024/05/16

Q & A

- 影片看了嗎？
- 有任何問題嗎？

Quick Review

- 線段樹可以做什麼
  - 操作可以分割
  - 答案可以合併
- 可做區間查詢、區間修改等
- 空間複雜度:O(N)
- 時間複雜度:O(log N)



線段樹與紀錄分治

線段樹與紀錄分治

- 線段樹的本質到底是什麼？
- 課程影片說是分治
  - 具體來說是把分治的過程存下來，面對多筆詢問的時候就可以直接回答，節省複雜度
- 知道這件事情有幫助嗎？
- 線段樹可以解決什麼樣的問題？
  - 可以將問題區間分成很多段，然後合併得到答案的問題。
- 線段樹可以紀錄什麼樣的資訊？

區間最大差

- 給定一個長度為 N 的序列跟 Q 筆操作:
  - 單點修改
  - 求區間 [l, r] 中的 max(a[j] - a[i])
- N, Q <= 10^5, 序列的元素 <= 10^9

區間最大差

- 拆解問題:
  - max(a[j] - a[i]) = [l, r]內的最大值 - [l, r]內的最小值
  - 只要在線段樹上同時紀錄最大和最小值即可
- 如何合併兩個區間？
  - 最大值變成兩邊最大, 最小值變成兩邊最小

區間最大順向差

- 給定一個長度為 N 的序列跟 Q 筆操作:
  - 單點修改
  - 求區間 [l, r] 中的 max(a[j] - a[i]), l <= i < j <= r
- N, Q <= 10^5, 序列的元素 <= 10^9

區間最大順向差

- 考慮兩個 index 在一段區間內的情況
- 分成 3 種 case:
  - i 跟 j 都在左邊
  - i 跟 j 都在右邊
  - i 在左邊, j 在右邊
- 根據分治的想法:
  - 前兩種可以遞迴下去解決
  - 第三種很明顯是找 i = 左邊最小值, j = 右邊最大值

區間最大順向差

- 套用到線段樹
  - 維護區間答案、區間最大值跟區間最小值
  - query 的時候要回傳 3 個值
  - 利用分治去「合併」兩個區間

```
max(max_l, max_r), min(min_l, min_r),
max(ans_l, ans_r, max_r - min_l)
```

```
max_l, min_l, ans_l
```

```
max_r, min_r, ans_r
```

區間最大連續和

- 給定一個長度為 N 的序列跟 Q 筆操作:
  - 單點修改
  - 求區間 [l, r] 中的 max(a[i] + a[i+1] + … + a[j]), l <= i <= j <= r
- N, Q <= 10^5, 序列的元素 <= 10^9

區間最大連續和

- 其實跟上一題非常像
- 先做一次前綴和，把 a[1], a[2], …, a[n] 變成 p[1], p[2], …, p[n]，那麼問題就會變成：

  找到 max(p[j] - p[i]), l-1 <= i < j <= r

區間最大連續和

- 另外一種作法(其實跟上一頁的方法相同):
  - 對於每一段區間紀錄答案 ans, 最大前綴和 pref, 最大後綴和 suf, 總和 sum
  - 自己想想看合併方式!

ans=???, pref=???, suf=???, sum=???

ans_l, pref_l,
suf_l, sum_l

ans_r, pref_r,
suf_r,sum_r

sprout

難題:Optimal Milking

- 給定一個長度為 N 的序列跟 Q 筆操作:
    - 單點修改
    - 詢問整個序列中，選取任意多個互不相鄰元素的最大和
- N, Q <= 10^5，序列的元素 <= 10^9

線段樹與二分搜

區間最早高分值

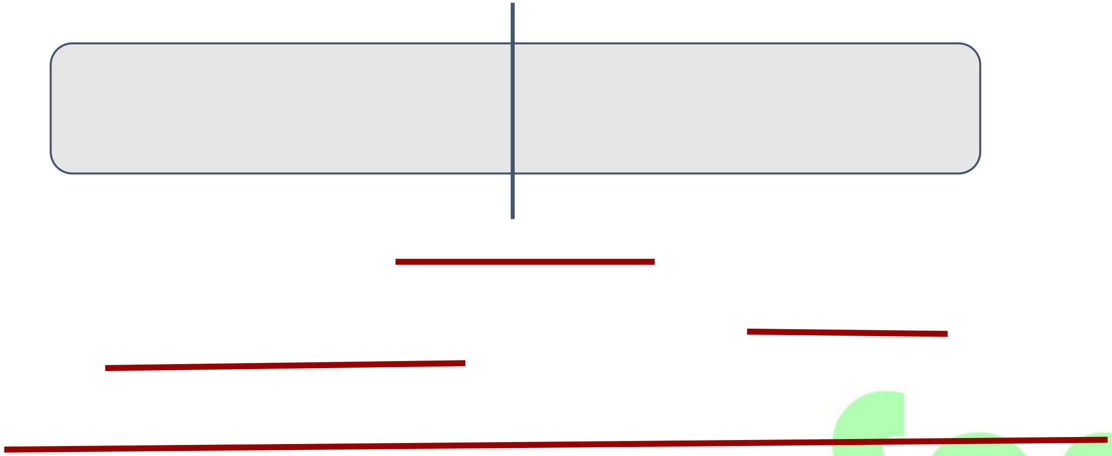
- 給定一個長度為 N 的序列跟 Q 筆操作:
  - 單點修改
  - 找到區間 [l, r] 裡最小的 i 使得 a[i] >= x。
  - 不存在的話輸出 -1
- N, Q <= 10^5, 序列的元素 <= 10^9

區間最早高分值

- 使用二分搜！
- 假設目前答案的可能區間在 [l, r] 這個節點。
  - 先看 [l, mid) 的最大值有沒有 >= x
  - 如果有，答案就在 [l, mid)，遞迴求解。
  - 如果沒有，答案就在 [mid, r)。
  - 當我們到只剩一個元素的節點，就能直接判斷出答案。
- 正確實作的話，一次二分搜的複雜度是 O(logn)

區間最早高分值

- 詢問的區間與當前節點的區間有一些不同的關係：



再談懶惰標記

Overview

- 懶標的意義
  - push 與 pull
- 可以使用 Lazy Tag 的條件
  - 一次操作產生的懶標可以分割
  - 兩次操作產生的懶標可以合併
  - 懶標的順序

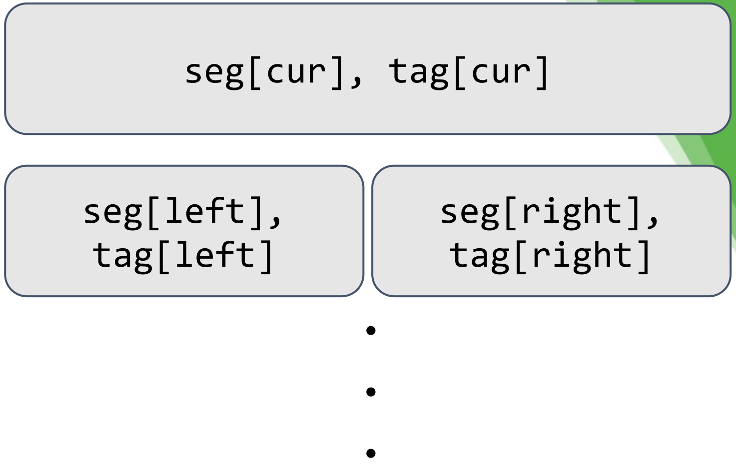
懶標的意義

- 對於一個節點 cur，他儲存的值是 seg[cur]，他的懶標是 tag[cur]
- tag[cur] 的意義是：針對 cur 底下所有元素進行的修改量
- seg[cur] 的意義是：cur 的區間內，不考慮 tag[cur] 以及 cur 上層節點的修改時，該區間紀錄的答案。

懶標的意義

seg[cur] 紀錄的答案會包含
seg[left], tag[left],
seg[right], tag[left]

所以 seg[cur] 看不到
tag[cur] 或是更上層的 tag!



push and pull

push(cur) 會將 tag[cur] 對答案的修改作用在 seg[cur] 上，並且把這個修改紀錄在 tag[left] 和 tag[right]，最後清空 tag[cur] 的修改。

pull(cur) 是計算 seg[cur] 的函式，他需要考慮 seg[left], tag[left], seg[right], tag[right]。

- 有一種方法是在 pull 裡面對左右節點 push，只要看 seg 就好

線段樹修改的一般方法

```
modify(cur, ql, qr, x): //對 [ql, qr) 改 x
    if [ql, qr) 不在 cur 的範圍:
        return
    push(cur)
    if [ql, qr) 包含 cur 的範圍:
        在 tag 上修改然後 return
    modify(left, ql, qr, x); modify(right, ql, qr, x);
    pull(cur)
```

線段樹詢問的一般方法

```
query(cur, ql, qr):
    if [ql, qr) 不在 cur 的範圍:
        return
    push(cur)
    if [ql, qr) 包含 cur 的範圍:
        return seg[cur]
    lans = query(left, ql, qr);
    rans = query(right, ql, qr);
    return lans, rans 合併後的答案 (類似 pull)
```

可以使用懶標的條件

- 一次操作產生的懶標可以分割：被「推下去」的時候可以拆散程左右節點的懶標
  - e.g. 區間加等差數列 (O)
  - e.g. 區間離散化（按照大小順序變成 1, 2, …, siz）(X)

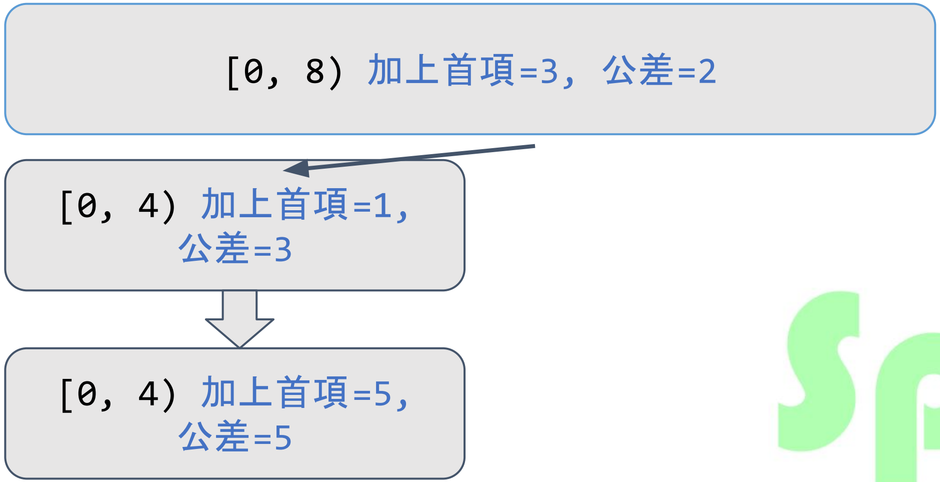
[0, 8) 加上首項=1, 公差=3

[0, 4) 加上首項=1, 公差=3

[4, 8) 加上首項=1+4*3, 公差=3

可以使用懶標的條件

- 兩次操作產生的懶標可以合併：當某個已經有 tag 的點，上面的點又呼叫 push 時，必須要合併兩個修改
  - e.g. 區間加等差數列



懶標的順序

- 回到這題：給定一個長度為 N 的序列跟 Q 筆操作：
  - 區間加值
  - 區間改值
  - 區間和
- N, Q <= 10^5，序列的元素 <= 10^12

為什麼紀錄的懶標要「先改再加」？

懶標的順序

- 為什麼紀錄的懶標要「先改再加」？
- 想想看在 push 的時候，兩組 tag 合併時發生的事情
  - 上面的修改會覆蓋掉下面的加值
- 如果是「先加再改」呢？
  - 加值會依據下面的點有沒有修改而做不同的事情
  - 事實上也可以做，只是比較麻煩。

例題:區間複製

- 給定兩個長度為 N 的序列 A 跟 B，還有 Q 筆操作:
  - 給定 x, y, l，把 B[y + i] 改成 A[x + i]，0 <= i < l
  - 給定 i，求 B[i]
- N, Q <= 10^5，序列的元素 <= 10^12

例題：區間複製

- Lazy Tag
- 在 Tag 裡面紀錄節點的左界對應到的 a[x]
- 在查詢的時候直接經由 Tag 算出應該 mapping 到陣列 A 哪個位置

[0, 8) 改成 a[3, 11)

[0, 4) 改成 a[3, 7)

[4, 8) 改成 a[7, 11)

例題：區間 XOR

- 給定一個長度為 N 的序列跟 Q 筆操作：
  - 給定區間跟 x，把裡面每個元素都 XOR x
  - 求區間和
- N, Q <= 10^5，序列的元素 <= 10^6

例題：區間 XOR

- 好像沒辦法把 XOR 的標記直接改變總和？
- 觀察到 XOR 其實是把某些 bit 反轉
  - 對於每個 bit，只需要知道有幾個 0 跟 1 即可
- 每個 bit 分開蓋線段樹！
- 問題變成：
  - 給定一個長度為 N 的 01 序列跟 Q 筆操作：
    - 給定區間，把裡面每個元素都 01 反轉
    - 求區間和
- 複雜度是 O((N + Q) log N log C)

例題：區間 XOR

- 這 20 棵線段樹其實可以塞成 1 棵
  - 每個節點存一個長度是 20 的陣列 A，A[i] 代表這個區間第 i 個 bit 有幾個 1
  - 合併就 O(20) 合併
  - Lazy Tag 比起存 20 個 bool，直接塞成一個 int 可以節省空間
- 基本上要做的操作數量跟 20 棵是一模一樣的
  - 可以節省遞迴的常數(stack 的 push 和 pop 常數蠻大的)
  - 可以節省中途的運算量(l, r, mid 等等)

離線與排序

離線與排序是最強大的武器！

許多線段樹問題會需要對資料或是詢問進行排序。

為了方便維護，將詢問重新排序的作法稱為**離線。**接下來，我們會用一些例題來展示這樣的方法搭配線段樹的強大。

例題：CSES Distinct Values

- 給一個長度為 n 的序列，回答 q 筆詢問
  - 給定 l, r, 回答 a[l], a[l+1], …, a[r] 有幾個相異的數字
- n, q <= 2 * 10^5, a_i <= 10^9

例題:[CSES Distinct Values]()

- 線段樹的每個區間要紀錄什麼答案？
  - 感覺沒有辦法很快速的合併多個區間...
- 重點:沒有修改 -> 詢問沒有時間順序 -> 可以離線！
- 考慮把詢問按照右界排序
  - 用「掃描線」的方式，由左到右「加入」一個元素
  - 當掃描線到達該詢問的右界時就回答這個詢問

例題:[CSES Distinct Values]

- 考慮把詢問按照右界排序
  - 用「掃描線」的方式，由左到右「加入」一個元素
  - 當掃描線到達該詢問的右界時就回答這個詢問
- 此時，我想要知道所有左界可能對應的答案
  - 當左界往左，區間裡的數字只會變多
  - 我們只在乎每一種數字在右界以左第一次出現的時間

例題:CSES Distinct Values

- 舉例:

2 1 4 1 3 2 1

例題：[矩形覆蓋面積計算]

- 給你 n 個平面上的矩形，請求出它們覆蓋的總表面積。
  - n <= 10^5, 0 <= 座標範圍 <= 10^6

例題：矩形覆蓋面積計算

- 看起來不像線段樹問題？
  - 一維的陣列呢...
- 試試看把他變成很多個一維陣列：對於每一個 y (從 0 到 10^6)，數有多少個 x 被至少一個矩形覆蓋到。
  - 當 y 增加時，可能會「進入」一個矩形的下界，或是「離開」那個矩形的上界，這兩種事件會改變這個切面的長相。
  - 掃描線！

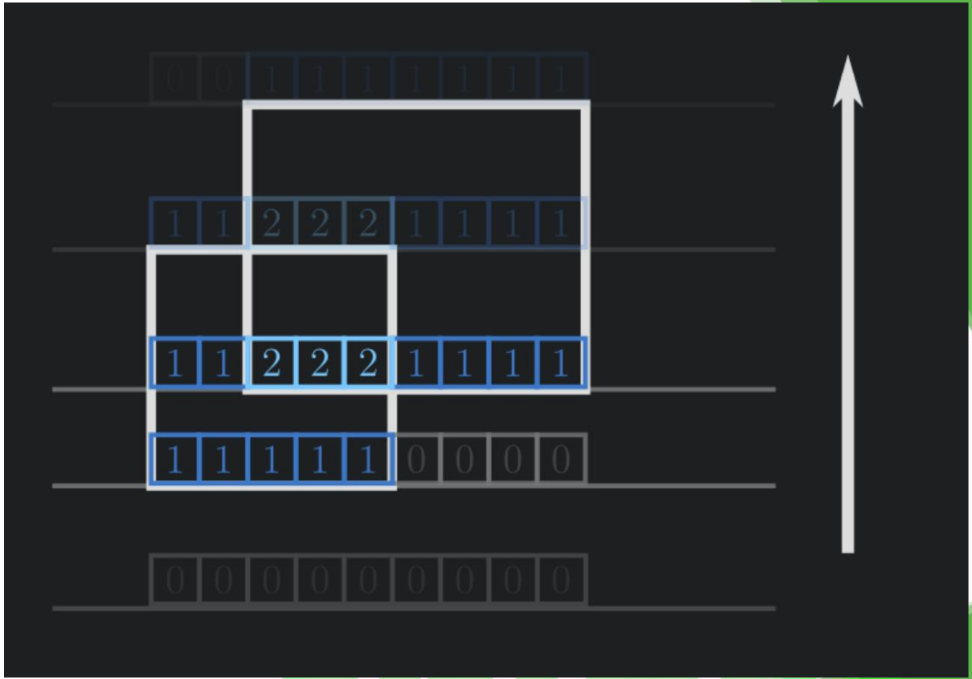
例題：矩形覆蓋面積計算

把每一個矩形當成兩個區間，並且將區間照 y 值排序。

加入矩形時，要將對應的區間 +1

移除矩形時，要將對應的區間 -1

對於每一個 x，計算有多少個數字大於 0



Credit: LittleCube

例題: 矩形覆蓋面積計算

轉換成這個問題:

給定一開始都是 0 的陣列, 請處理 Q 個操作:

- 區間 +1
- 區間 -1
- 詢問全部有幾個數字大於 0

保證數字在任何時間都>= 0 (Why?)

例題：矩形覆蓋面積計算

使用懶惰標記！

- tag[cur] 紀錄當前區間的加值
- seg[cur] 紀錄區間內有幾個非零的數字

例題:矩形覆蓋面積計算

Live Coding 時間！

補充:這題有兩種作法

- Push 懶標(需要紀錄最小值與最小值個數)
- 不推懶標(比較好寫但是不好推廣)

結語

線段樹還有非常非常多可以分享的東西...

在許多 OI 競賽中，困難的題目往往都用得到線段樹，不過它只會是解題步驟中的一小部份。

因此打好實作的基礎，並且學會靈活運用才是精通線段樹的關鍵！

補充資源

- 基礎資結 & 線段樹簡介
- 稍微難一點的線段樹
- 很難很難的線段樹