Segment/Interval Tree - 2

by 林品諺

強，還要更強

- 我們現在已經會的東西有：
  - 單點修改、區間詢問
  - 區間修改、單點詢問
- 單點修改、單點詢問？
  - 陣列O(1)解決
- 區間修改、區間詢問？
  - 怎麼好像有點困難QAQ

Sprout

範例題目

- 給你一個長度為N的序列以及Q個操作，操作分為兩種：
  - 1. 把一個區間的每個數字都加上x
  - 2. 詢問一個區間的最大值
- N, Q ≤ 100,000

題目說明

- 原始序列：

8	7	1	2	2	8

題目說明

- 詢問[3, 6]的最大值：8

8	7	1	2	2	8

題目說明

- 把[2, 5]加上7：

8	14	8	9	9	8

題目說明

- 詢問[3, 6]的最大值：9

8	14	8	9	9	8

just try it

- 先用我們已經會的線段樹知識嘗試一下
  - 每個節點記錄它所對應的區間的最大值
  - 區間詢問：一樣找到那logN個節點問一問答案，O(logN)
  - 區間修改：把它轉成N個單點修改，O(NlogN)
- 複雜度爆炸了！
- 把題目轉成區間修改跟N個單點詢問顯然也沒有解決問題
- 如果那N個東西都被加上了同個數字，我們真的有需要把它當成N個單點修改來操作嗎？

名詞解釋

- 「一個區間所對應的節點」
- 所有被該區間完全包含的節點們當中，深度最淺的那些節點
- 也就是區間詢問時會問到的那些節點

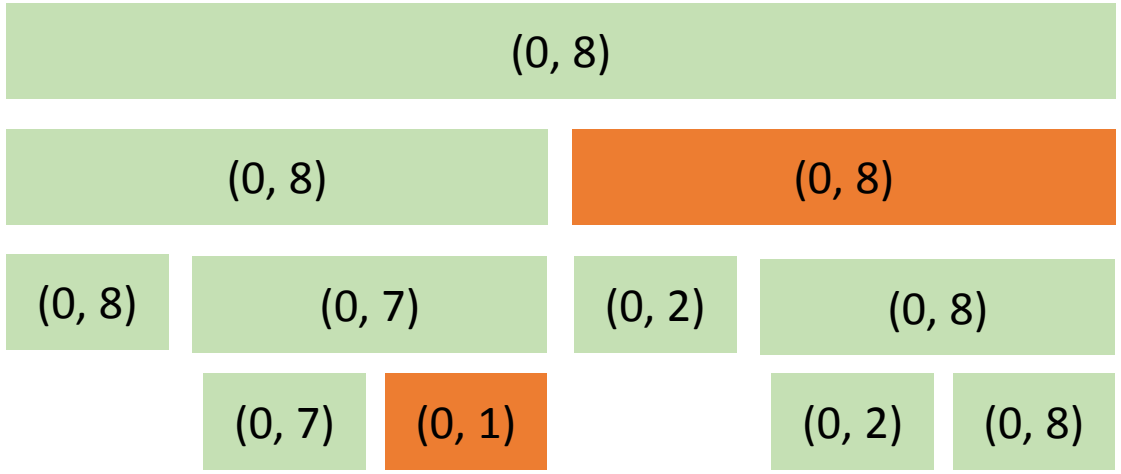
有的時候你不需要那麼勤勞

- 想想區間修改、單點詢問
  - 把修改放在該區間所對應的節點上
  - 詢問的時候會經過該點的祖先們，這時再去看剛剛的那些修改
- 區間詢問是不是也可以套用類似的想法？
- 對於每個修改，把它放在該區間所對應的節點上
- 區間修改：找到對應的節點們然後把修改貼上去，O(logN)
- 區間詢問：找到對應的節點們，把所記錄的data值加上修改的值後取max，O(logN)

圖解

- 詢問[3, 6]的最大值：8

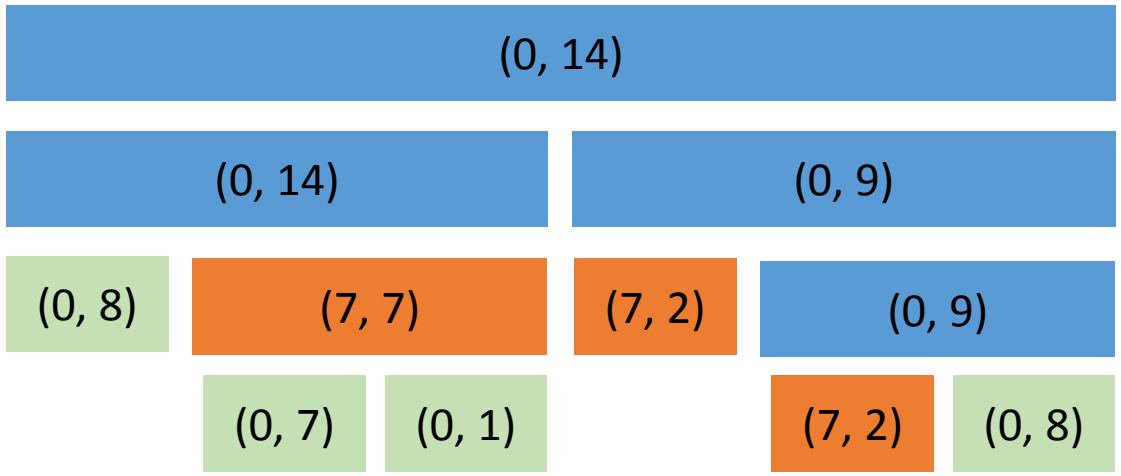
8	7	1	2	2	8



圖解

- 把[2, 5]加上7

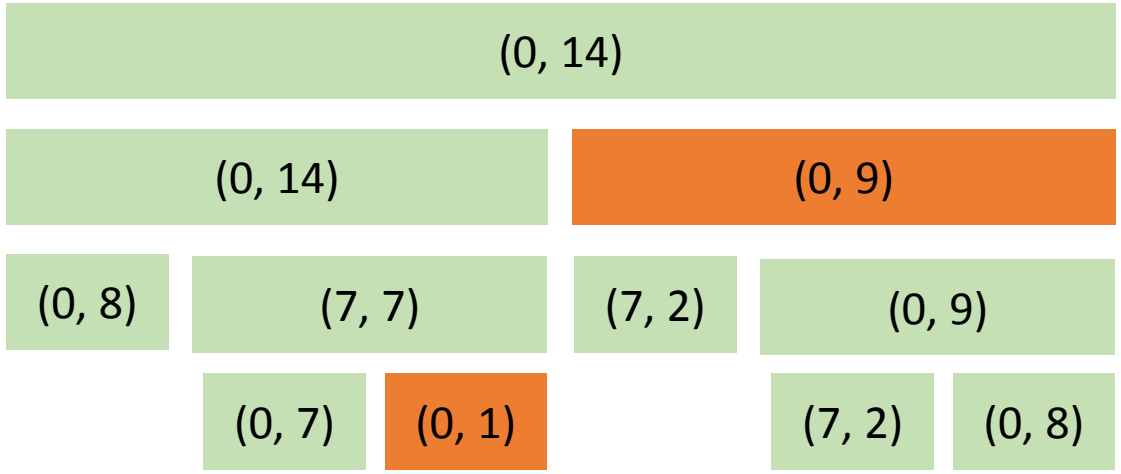
8	14	8	9	9	8



圖解

- 詢問[3, 6]的最大值：9

8	14	8	9	9	8



實作

- chg: 修改的值, data: 我們要維護的東西(最大值)

```
20 struct Node {
21     int chg, data;
22 } ST[MAXN*4];
```

區間修改

[a, b]:詢問區間,i:節點編號, [l, r]:該節點對應的區間, x:要加上去的數字

```
24 void add(int a, int b, int i, int l, int r, int x) {
25      if(a<=l && r<=b)    ST[i].chg += x;
26      else if(r<a || b<l) return ;
27      else {
28          int m = (l+r) / 2, lson = i*2+1, rson=i*2+2;
29          add(a, b, lson, l, m, x);
30          add(a, b, rson, m+1, r, x);
31          ST[i].data = max(ST[lson].data+ST[lson].chg,
32                           ST[rson].data+ST[rson].chg);
33      }
34 }
```

區間詢問

```
36 int query(int a, int b, int i, int l, int r) {
37     if(a<=l && r<=b)    return ST[i].data+ST[i].chg;
38     else if(r<a || b<l) return -1;
39     else {
40         int m = (l+r) / 2, lson = i*2+1, rson=i*2+2;
41         int x1 = query(a, b, lson, l, m);
42         int x2 = query(a, b, rson, m+1, r);
43         return max(x1, x2)+ST[i].chg;
44     }
45 }
```

故事總是有個轉折

- 有了剛剛的技巧，看起來什麼區間加值、區間改值什麼的我們都會做了
- 那如果兩個一起來呢？

例題二

- 給你一個長度為N的序列以及Q個操作，操作分為兩種：
  - 1. 把一個區間的每個數字都加上x
  - 2. 詢問一個區間的最大值
  - 3. 把一個區間的每個數字都變成x
- N, Q ≤ 100,000

思考一陣

- 這不是一樣ㄇ……等等，真的一樣嗎？
- 如果開兩個變數分別存兩種修改，那我怎麼知道是先改再加、還是先加再改？
- 到頭來好像又只能一個一個改了。
- 難道我們就要在這裡停下腳步了嗎？

繼續思考

- 問題發生在，兩種修改會互相影響。
- 假設所有加值操作都在左半邊，所有改值操作都在右半邊，那麼就直接做就好了，對我們而言這世界就依舊美好。
- 兩種操作不影響的時候，就沒有不需要把修改壓到底了。
- 每一次修改只會動到O(logN)個節點。
- 如果我們在修改的時候，才順便把遇到的修改壓下去…？
- ……好像發現了什麼？

不小心得到了一個做法

- 修改的部分跟剛剛一樣，先放在該區間對應的節點上。
- 如果遞迴的路上發現這個節點有其他的修改資訊，就把那些資訊往下推一層。
- 因為每次修改還是只會走過O(logN)個節點，所以複雜度不變。
- 詢問就照樣回答。

順序問題

- 剛剛的做法還有一個細節要注意，那就是兩種修改值的順序問題。
- 一個常見的做法是定義一個順序，比如說如果兩個修改值皆非0(或任何表示他沒被修改的數字)時，假設是先改再加。
- 加值的時候就直接加上去。
- 改值的時候，因為改值這件事本身就會優先於加值，所以做改值操作的時候要把加值的修改值改成0。
- 就這題而言，定義順序是先加再改不太能做。

實作

- chg1: 加值的修改, chg2: 改值的修改(以chg1=0, ch2=0表示沒有修改)

```
20 struct Node {
21     int chg1, chg2, data;
22 } ST[MAXN*4];
23
```

把修改往下壓

```
24 void push(int i, int lson, int rson) {
25     if(ST[i].chg2 > 0) {
26         ST[i].data = ST[i].chg2;
27         if(lson >= 0)   ST[lson].chg2 = ST[i].chg2, ST[lson].chg1 = 0;
28         if(rson >= 0)   ST[rson].chg2 = ST[i].chg2, ST[rson].chg1 = 0;
29         ST[i].chg2 = 0;
30     }
31     if(ST[i].chg1 > 0) {
32         ST[i].data += ST[i].chg1;
33         if(lson >= 0)   ST[lson].chg1 += ST[i].chg1;
34         if(rson >= 0)   ST[rson].chg1 += ST[i].chg1;
35         ST[i].chg1 = 0;
36     }
37 }
```

區間加值

```
39 void add(int a, int b, int i, int l, int r, int x) {
40      int lson = i*2+1, rson = i*2+2;
41      if(l == r)  lson = rson = -1;
42      push(i, lson, rson);
43      if(a<=l && r<=b)     ST[i].chg1 += x;
44      else if(r<a || b<l) return ;
45      else {
46          int m = (l+r) / 2;
47          add(a, b, lson, l, m, x);
48          add(a, b, rson, m+1, r, x);
49          int x1 = (ST[lson].chg2>0 ? ST[lson].chg2 : ST[lson].data)+ST[lson].chg1;
50          int x2 = (ST[rson].chg2>0 ? ST[rson].chg2 : ST[rson].data)+ST[rson].chg1;
51          ST[i].data = max(x1, x2);
52      }
53 }
```

區間改值

```
55 void chg(int a, int b, int i, int l, int r, int x) {
56      int lson = i*2+1, rson = i*2+2;
57      if(l == r)  lson = rson = -1;
58      push(i, lson, rson);
59      if(a<=l && r<=b)     ST[i].chg2 = x;
60      else if(r<a || b<l) return ;
61      else {
62           int m = (l+r) / 2;
63           chg(a, b, lson, l, m, x);
64           chg(a, b, rson, m+1, r, x);
65           int x1 = (ST[lson].chg2>0 ? ST[lson].chg2 : ST[lson].data)+ST[lson].chg1;
66           int x2 = (ST[rson].chg2>0 ? ST[rson].chg2 : ST[rson].data)+ST[rson].chg1;
67           ST[i].data = max(x1, x2);
68      }
69 }
```

區間詢問

```
71 int query(int a, int b, int i, int l, int r) {
72     int lson = i*2+1, rson = i*2+2;
73     if(l == r)  lson = rson = -1;
74     push(i, lson, rson);
75     if(a<=l && r<=b)    return ST[i].data;
76     else if(r<a || b<l) return -1;
77     else {
78         int m = (l+r) / 2;
79         int x1 = query(a, b, lson, l, m);
80         int x2 = query(a, b, rson, m+1, r);
81         return max(x1, x2) + ST[i].chg1;
82     }
83 }
```

懶人標記

- 剛剛的技巧被稱為「懶人標記」(lazy tag)。
- 修改就直接先放在那，要用到的時候就路過順便壓一下。
- 可分治性
- 要能夠把修改先放著，就必須能夠在修改沒壓下去的時候就回答詢問。
- 懶真是人類進步的原動力
- 只在你需要做事的時候才做事，有時候你其實不需要那麼勤勞。