進階資料結構 (Advanced Data Structure)

sam571128

目錄

- 離散化
- 離線
- 線段樹上二分搜
- 動態開點線段樹
- 持久化線段樹
- 更多線段樹題目

前言

- 這堂課前半的時間會先引導至兩種特殊的線段樹
- 剩下的時間會講各種例題
- 然後各位初選加油！
- 現在我這裡早上五點，~~然後我今天有兩個段考 + 一個 project，要死掉了~~
- 今天有資結 midterm，所以講資結來複習

離散化 (Coordinate Compression)

離散化 (Coordinate Compression)

- 一開始，我們首先要介紹的是名為「離散化」的技巧
- 這個技巧，相信大家在讀書會上學期的課程都已經學習過了
- 不過，我們要再來好好講講這個很重要的技巧是什麼

離散化 (Coordinate Compression)

- 首先，先來看一個很簡單的例子

離散化 (Coordinate Compression)

CSES - Increasing Subsequence

給你一個 n 項的陣列，請找到這個陣列中的 LIS 長度。

離散化 (Coordinate Compression)

CSES - Increasing Subsequence

給你一個 n 項的陣列，請找到這個陣列中的 LIS 長度。

- 可以很顯然的列出下面的 DP 式 (忘記或不會的要問喔

$$dp[i] = \max_{j<i,\ \ a[j]<a[i]} (dp[j] + 1)$$

離散化 (Coordinate Compression)

CSES - Increasing Subsequence

給你一個 n 項的陣列，請找到這個陣列中的 LIS 長度。

- 可以很顯然的列出下面的 DP 式（忘記或不會的要問喔

$$dp[i] = \max_{j<i,\ a[j]<a[i]} (dp[j]+1)$$

- 很顯然地可以在 $\mathcal{O}(n^2)$ 的時間做完

CSES - Increasing Subsequence

給你一個 n 項的陣列，請找到這個陣列中的 LIS 長度。

- 可以很顯然的列出下面的 DP 式（忘記或不會的要問喔

$$dp[i] = \max_{j<i,\ \ a[j]<a[i]} (dp[j] + 1)$$

- 很顯然地可以在 $\mathcal{O}(n^2)$ 的時間做完
- 太慢了！

離散化 (Coordinate Compression)

- 注意到，其實如果我們從 $i = 1$ 開始進行枚舉
- 那其實對於每一個數字，我們都只需要去找對於 $a[j] < a[i]$ 的最大 $dp[j]$

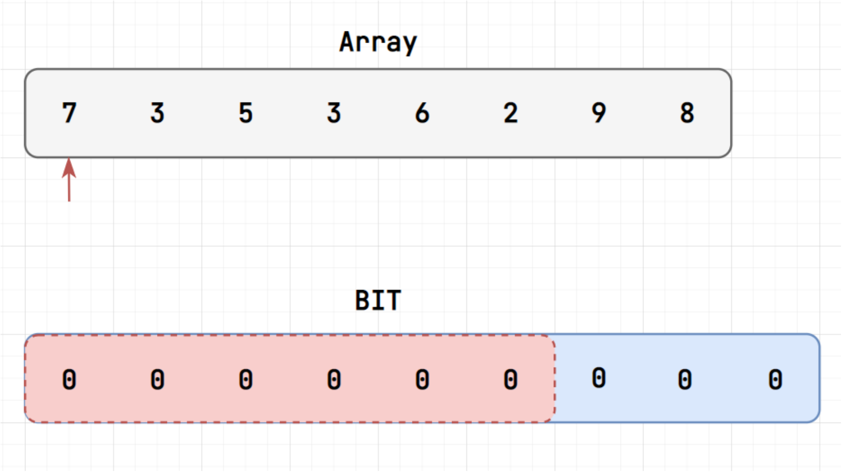
離散化 (Coordinate Compression)

- 注意到，其實如果我們從 $i = 1$ 開始進行枚舉
- 那其實對於每一個數字，我們都只需要去找對於 $a[j] < a[i]$ 的最大 $dp[j]$
- 要怎麼做呢？

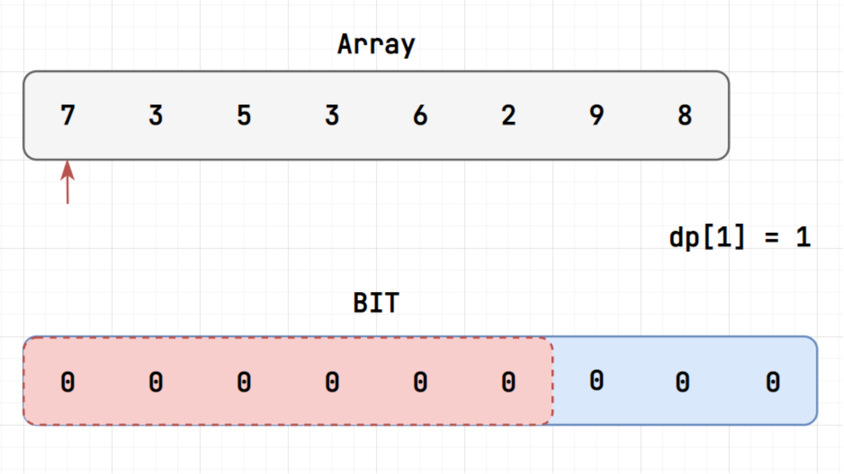
離散化 (Coordinate Compression)

- 注意到，其實如果我們從 $i = 1$ 開始進行枚舉
- 那其實對於每一個數字，我們都只需要去找對於 $a[j] < a[i]$ 的最大 $dp[j]$
- 要怎麼做呢？
- 用 BIT 存以 $a[j]$ 做結尾的最大答案！

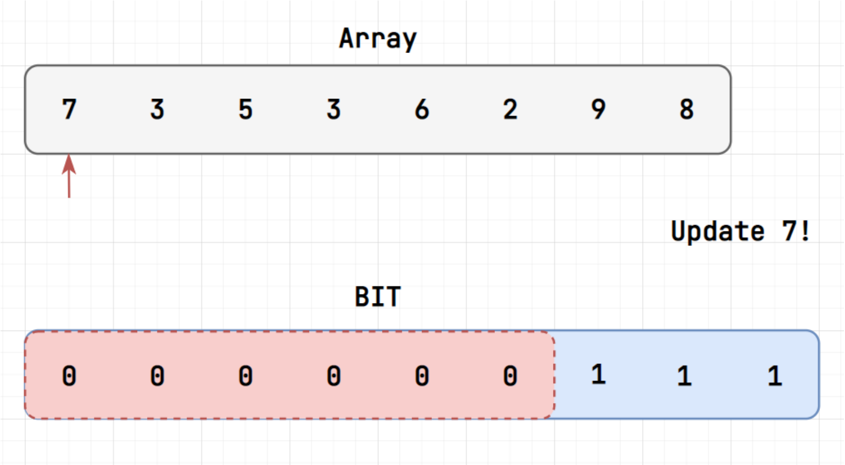
離散化 (Coordinate Compression)



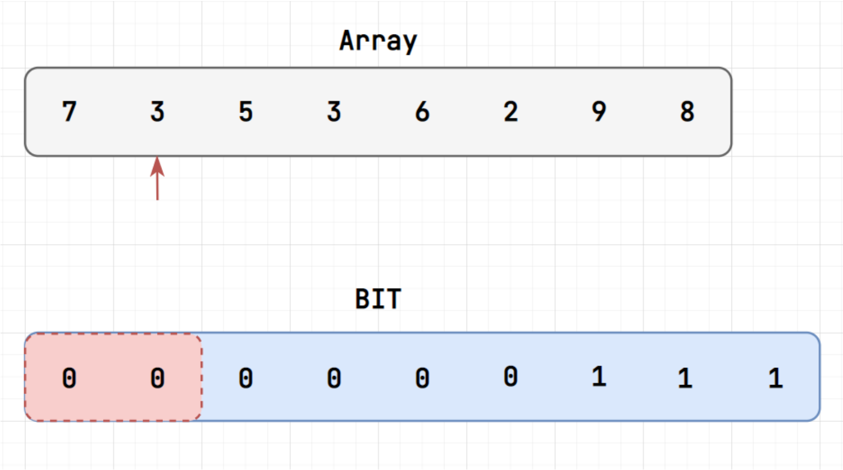
離散化 (Coordinate Compression)



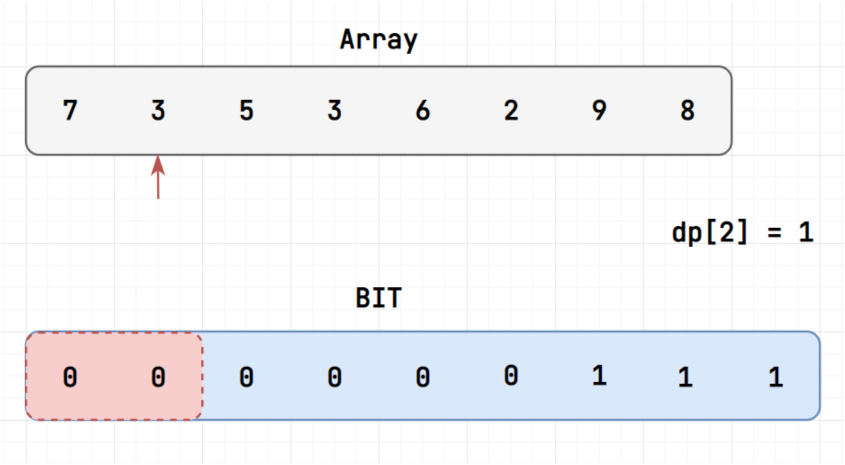
離散化 (Coordinate Compression)



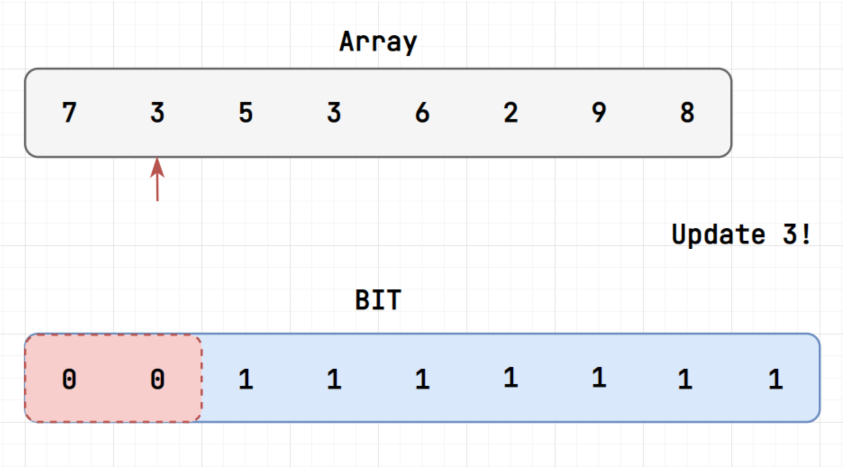
離散化 (Coordinate Compression)



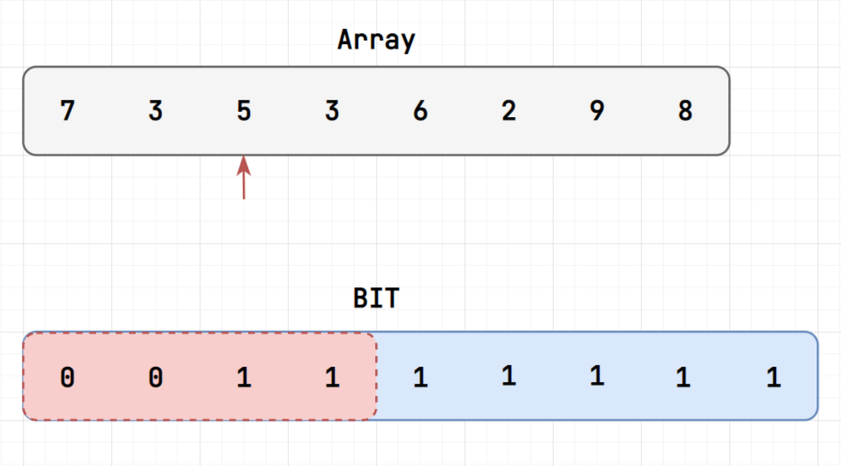
離散化 (Coordinate Compression)



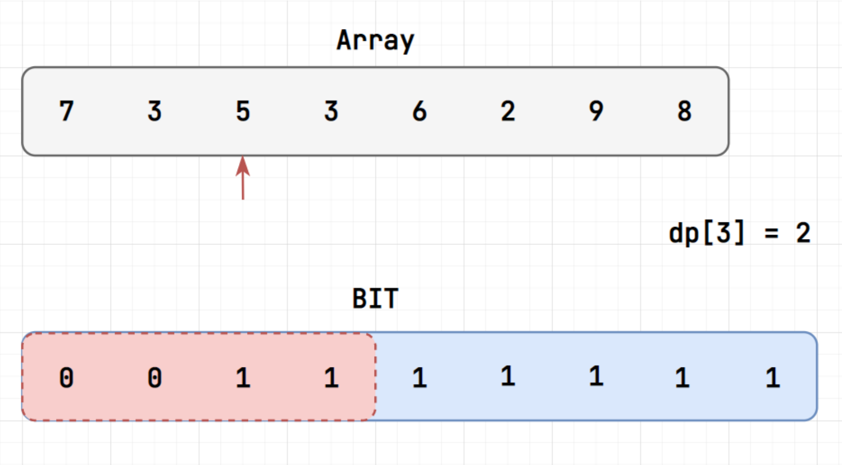
離散化 (Coordinate Compression)



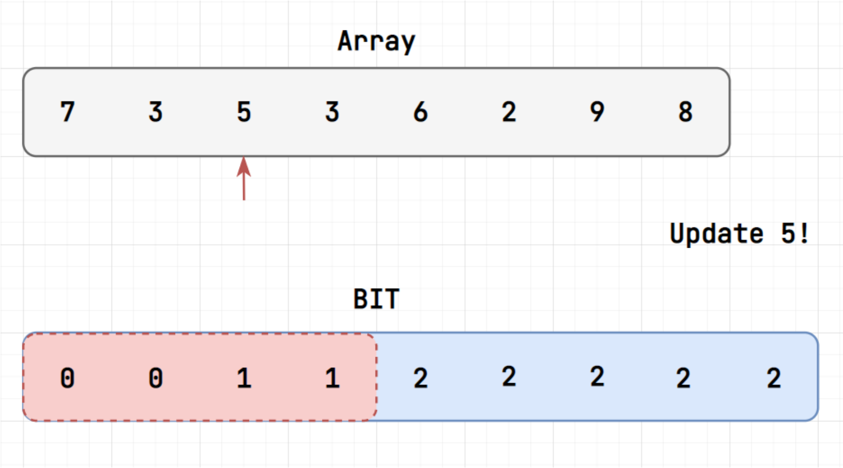
離散化 (Coordinate Compression)



離散化 (Coordinate Compression)



離散化 (Coordinate Compression)



離散化 (Coordinate Compression)

- 雖然剛剛那個樣子很合理，但是有沒有注意到什麼事情？

離散化 (Coordinate Compression)

- 雖然剛剛那個樣子很合理，但是有沒有注意到什麼事情？
- 我們對於每一個數字，都會在 BIT 上開一個位置給它

離散化 (Coordinate Compression)

- 雖然剛剛那個樣子很合理，但是有沒有注意到什麼事情？
- 我們對於每一個數字，都會在 BIT 上開一個位置給它
- 因此，當 $\max(a_i)$ 很大（$10^8, 10^9$）的時候，空間會存不下！

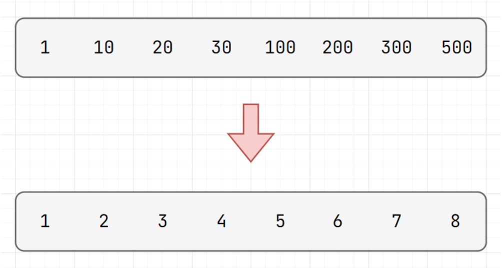
離散化 (Coordinate Compression)

- 注意到其實在做這件事情時，重要的只有大小關係！(找比 $a[i]$ 小的 dp 答案)
- 而我們實際上在 BIT 上修改和詢問的位置，其實最多只有 n 個

離散化 (Coordinate Compression)

- 注意到其實在做這件事情時，重要的只有大小關係！(找比 $a[i]$ 小的 dp 答案)
- 而我們實際上在 BIT 上修改和詢問的位置，其實最多只有 n 個
- 因此，將這些數字進行轉換，讓他們對應到 $1 \sim n$ 之間的數字，保留大小關係

假設我們將出現的數字排序好，則他們會如下圖一一對應



離散化 (Coordinate Compression)

- 那我們要怎麼做到這一點，有兩種常見的方法
  1. 二分搜 (unique + lower_bound)
  2. map + set
- 第二種方法的常數會比第一種大很多，因此，建議使用第一種

離散化 (Coordinate Compression)

- 那我們要怎麼做到這一點，有兩種常見的方法
  1. 二分搜 (unique + lower_bound)
  2. map + set
- 第二種方法的常數會比第一種大很多，因此，建議使用第一種

- 範例 code ←

離線 (Offline Method)

離線 (Offline Method)

- 接著，我們要來介紹的技巧為「離線」
- 首先，我們先來講講，「離線」是什麼，「在線」又是什麼？

離線 (Offline Method)

- 接著，我們要來介紹的技巧為「離線」
- 首先，我們先來講講，「離線」是什麼，「在線」又是什麼？
- 兩者的差異：
  - 在線：必須照著詢問與操作的順序下去執行
  - 離線：可以以不同的順序去進行操作

離線 (Offline Method)

- 接著，我們要來介紹的技巧為「離線」
- 首先，我們先來講講，「離線」是什麼，「在線」又是什麼？
- 兩者的差異：
  - 在線：必須照著詢問與操作的順序下去執行
  - 離線：可以以不同的順序去進行操作
- 第一次聽到可能會不理解兩者差在哪，因此我們實際來看看例題吧

CSES - Distinct Value Queries

有一個 n 項的陣列以及 q 筆詢問，每筆詢問請找到區間 $[l, r]$ 內有幾個不同的數字。

- $1 \leq n, q \leq 2 \cdot 10^5$
- $1 \leq a_i \leq 10^9$

CSES - Distinct Value Queries

有一個 n 項的陣列以及 q 筆詢問，每筆詢問請找到區間 $[l, r]$ 內有幾個不同的數字。

- $1 \leq n, q \leq 2 \cdot 10^5$
- $1 \leq a_i \leq 10^9$

- 乍看之下，應該會發現這題沒有那麼好做 (?
- 想法可能會是對線段樹的每個節點開 `set` 等等的方式去維護

離線 (Offline Method)

CSES - Distinct Value Queries

有一個 n 項的陣列以及 q 筆詢問，每筆詢問請找到區間 $[l, r]$ 內有幾個不同的數字。

- $1 \leq n, q \leq 2 \cdot 10^5$
- $1 \leq a_i \leq 10^9$

- 乍看之下，應該會發現這題沒有那麼好做 (?
- 想法可能會是對線段樹的每個節點開 set 等等的方式去維護
- 不論是時間複雜度 $\mathcal{O}(q \log^2 n)$ 還是空間複雜度都有點太高！

離線 (Offline Method)

- 有沒有什麼辦法，可以讓我們更簡單的處理這個問題？
- 假如，一開始你就已經知道有哪些詢問的區間呢？

- 考慮將所有的詢問，按照詢問的左界，由大到小排序

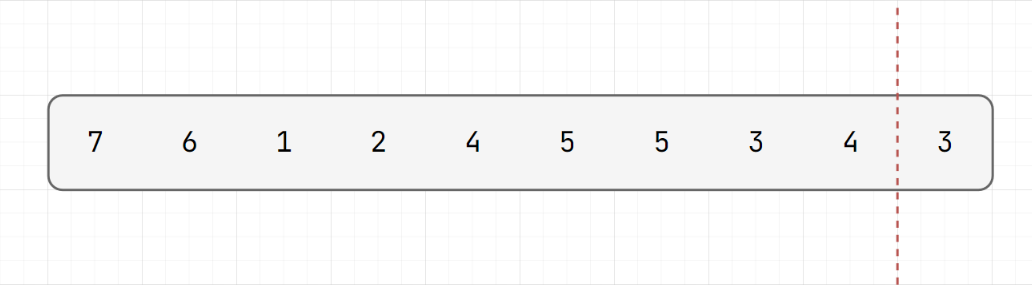
離線 (Offline Method)

- 思考當我們將左界向左移動時，以 r 當右界的答案會發生什麼事？

7	6	1	2	4	5	5	3	4	3

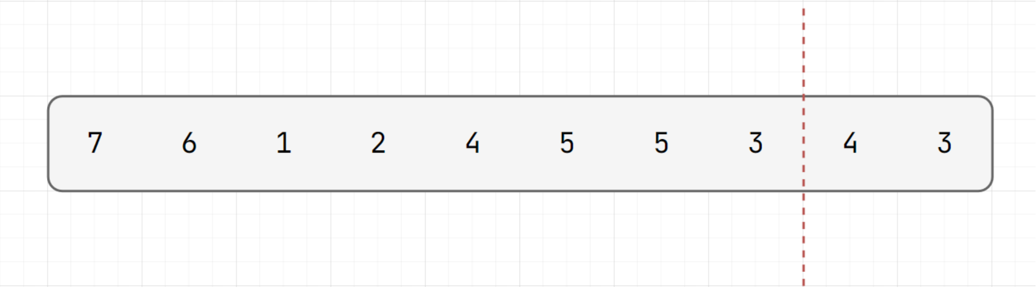
離線 (Offline Method)

- 思考當我們將左界向左移動時，以 r 當右界的答案會發生什麼事？



- $r \in [10, 10]$ 的答案會 $+1$！

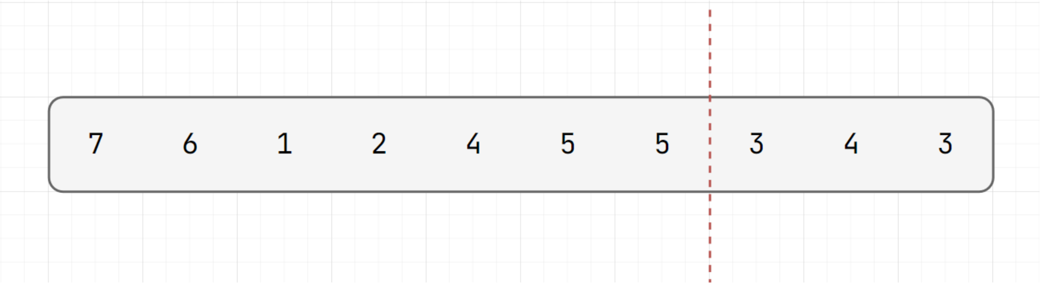
- 思考當我們將左界向左移動時，以 r 當右界的答案會發生什麼事？



- $r \in [9, 10]$ 的答案會 $+1$！

離線 (Offline Method)

- 思考當我們將左界向左移動時，以 r 當右界的答案會發生什麼事？



- $r \in [8, 9]$ 的答案會 $+1$！(注意到為什麼是到 9 了嗎？)

離線 (Offline Method)

- 會發現，其實我們真正需要的，只有一個能夠區間加值，單點求和的資料結構

離線 (Offline Method)

- 會發現，其實我們真正需要的，只有一個能夠區間加值，單點求和的資料結構
- 這些都是 BIT 或線段樹可以做到的！

離線 (Offline Method)

- 會發現，其實我們真正需要的，只有一個能夠區間加值，單點求和的資料結構
- 這些都是 BIT 或線段樹可以做到的！
- 參考程式碼：https://cses.fi/paste/a38a13351d94e2bd316f86/

線段樹上二分搜 (Binary Search on Segment Tree)

線段樹上二分搜 (Binary Search on Segment Tree)

- 接著，讓我們同樣來複習另外一個技巧「線段樹上二分搜」(還有 BIT)
- 一樣先來看看一個例子

線段樹上二分搜 (Binary Search on Segment Tree)

PBDS (?

現在，請你維護一個 `multiset`，可以做到以下四種操作

1. 插入 x
2. 刪除 x
3. 詢問第 k 大的數字是多少（或 -1）
4. 詢問 x 比集合中多少數字還要大（第 k 大）

線段樹上二分搜 (Binary Search on Segment Tree)

- 對 STL 熟悉的人，應該可以很直接地想到，這不就是 PBDS 的 Tree 嗎？

線段樹上二分搜 (Binary Search on Segment Tree)

- 對 STL 熟悉的人，應該可以很直接地想到，這不就是 PBDS 的 Tree 嗎？
- 沒有錯！不過，假設你忘了，而且也不會 Treap 等等的平衡二元樹

線段樹上二分搜 (Binary Search on Segment Tree)

- 對 STL 熟悉的人，應該可以很直接地想到，這不就是 PBDS 的 Tree 嗎？
- 沒有錯！不過，假設你忘了，而且也不會 Treap 等等的平衡二元樹
- 那我們可以使用 BIT 或線段樹來做到這件事情！

線段樹上二分搜 (Binary Search on Segment Tree)

- 考慮用一棵 BIT 或線段樹去維護每個數字出現的頻率（可能要離散化）
- 對於前兩種操作，其實就只是 add(x,1) 和 add(x,-1) 這樣的操作而已
- 那麼後兩種操作呢？

線段樹上二分搜 (Binary Search on Segment Tree)

- 最直覺的想法：直接另外寫一個二分搜檢查！
- 由於第三四種操作概念差不多，我們這裡只示範第三種

```
int get(int k){
    int l = 0, r = MAXA;
    while(l < r){
        int mid = l+r>>1;
        if(sum(1,mid) >= k) r = mid;
        else l = mid+1;
    }
}
```

線段樹上二分搜 (Binary Search on Segment Tree)

- 因此，可以在 $O(\log^2 n)$ 做完！

線段樹上二分搜 (Binary Search on Segment Tree)

- 因此，可以在 $O(\log^2 n)$ 做完！
- NO!!!! 太慢了

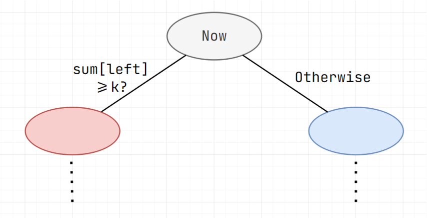
線段樹上二分搜 (Binary Search on Segment Tree)

- 對於 BIT，有一種方法叫做倍增法，可以在 $O(\log n)$ 的時間完成二分搜
- 至於正確性，可以想想看 BIT 以 lowbit(i) 作為儲存區間大小來去思考
- 範例如下：

```
int get(int k){
    int sum = 0, idx = 0;
    for(int i = LOGN; i >= 0; i--){
        if(idx + (1<<i) < MAXN && sum + bit[idx + (1<<i)] < k){
            idx += (1<<i);
            sum += bit[idx];
        }
    }
    return idx+1;
}
```

線段樹上二分搜 (Binary Search on Segment Tree)

- 至於線段樹呢，由於是一棵二元樹
- 我們可以根據目前節點的總和，決定要往左還右邊進行 DFS
- 直到達到我們要找的數字後終止



線段樹上二分搜 (Binary Search on Segment Tree)

- 這裡不再另外放教學，請回去看資料結構課的簡報或者 山姆的競程維基 學習

動態開點線段樹 (Dynamic Segment Tree)

動態開點線段樹 (Dynamic Segment Tree)

- 終於來到今天的重點之一了，也就是所謂的「動態開點線段樹」(和 BIT)

- 在這堂課的最一開始，我們就講到了所謂「離散化」的技巧
- 不過，它卻有一個缺點：這個技巧只能用在 **已知所有詢問** 時

動態開點線段樹 (Dynamic Segment Tree)

- 在這堂課的最一開始，我們就講到了所謂「離散化」的技巧
- 不過，它卻有一個缺點：這個技巧只能用在 **已知所有詢問** 時
- 因為離散化通常會需要是離線的，只要遇到沒有辦法的題目就會卡住！

- 題目通常會用各式各樣的方法來防範你使用「離線的技巧」

You are asked to perform some queries on the graph. Let $last$ be the answer to the latest query of the second type, it is set to 0 before the first such query. Then the queries are the following:

- $1\ x\ y\ (1 \le x, y \le n,\ x \ne y)$ — add an undirected edge between the vertices $(x + last - 1)\ mod\ n + 1$ and $(y + last - 1)\ mod\ n + 1$ if it doesn't exist yet, otherwise remove it;
- $2\ x\ y\ (1 \le x, y \le n,\ x \ne y)$ — check if there exists a path between the vertices $(x + last - 1)\ mod\ n + 1$ and $(y + last - 1)\ mod\ n + 1$, which goes only through currently existing edges, and set $last$ to 1 if so and 0 otherwise.

- 最常見的即為這種，「下一筆詢問與上一筆詢問的答案有關」的題目

動態開點線段樹 (Dynamic Segment Tree)

- 第二種可能會需要使用到這種線段樹的可能性則為「需要開好多棵」的情況
- 經典例題：TIOJ 1169 - 氣球博覽會
- 假設你今天要開 10^5 棵值域為 10^5 的線段樹 $\Rightarrow$ Explosion!

動態開點線段樹 (Dynamic Segment Tree)

- 看完上述的兩種例子以後，應該能理解離散化所沒辦法做到的原因了吧！
- 那麼，我們要怎麼樣讓我們的線段樹有辦法解決這樣的缺點呢？

動態開點線段樹 (Dynamic Segment Tree)

- 看完上述的兩種例子以後，應該能理解離散化所沒辦法做到的原因了吧！
- 那麼，我們要怎麼樣讓我們的線段樹有辦法解決這樣的缺點呢？
- 很簡單！用不到的空間，那我們就不要開就好了啊！

動態開點線段樹 (Dynamic Segment Tree)

- 看完上述的兩種例子以後，應該能理解離散化所沒辦法做到的原因了吧！
- 那麼，我們要怎麼樣讓我們的線段樹有辦法解決這樣的缺點呢？
- 很簡單！用不到的空間，那我們就不要開就好了啊！
- 這裡用一個簡單的例子來看看動態開點的線段樹

單點加值區間取 min

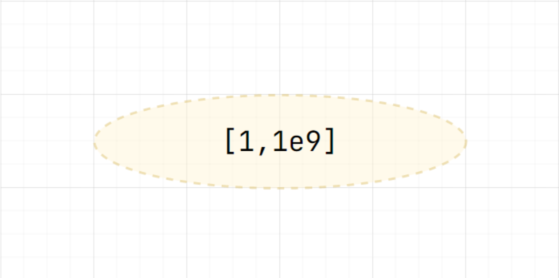
現在有一個 10^9 項的陣列，每個位置都是 ∞，有 q 種操作或詢問，每次有兩種可能

1. 對 $arr[x]$ 設成 v
2. 詢問區間 $[l, r]$ 的最小值

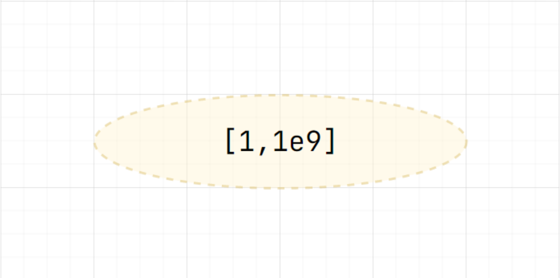
- $1 \leq x \leq 10^9$
- $1 \leq l \leq r \leq 10^9$

動態開點線段樹 (Dynamic Segment Tree)

- 一開始的線段樹，會是一個不存在的節點

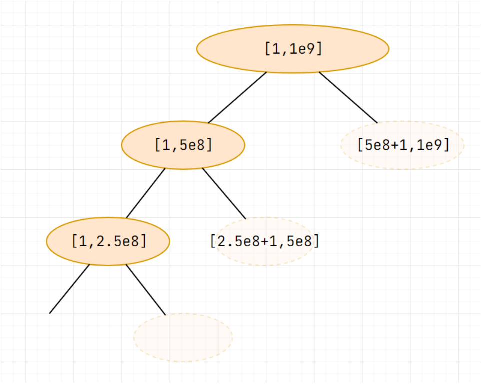


- 假設在這個時候，我們詢問 $[1,2]$ 的最小值，由於節點不存在，直接回傳是 ∞



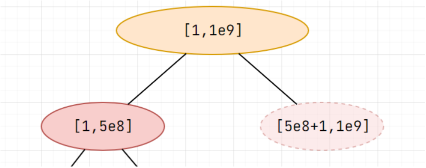
動態開點線段樹 (Dynamic Segment Tree)

- 假設現在，我們對 $arr[1]$ 加 1，則我們要新增從根到葉子路徑上的所有點



動態開點線段樹 (Dynamic Segment Tree)

- 那當我們這時候要再詢問 $[1, 5e8+2]$ 的時候，實際覆蓋到的節點會是這兩個節點



- 由於右邊的節點預設是 ∞ 了，我們不必向下走也能知道那裡的答案會是 ∞
- 答案取 $[1, 5e8]$ 的 1 和 ∞ 取 min

動態開點線段樹 (Dynamic Segment Tree)

- 因此，動態開點線段樹的概念就是
- 當我們修改時，走到空的點，就新增點
- 詢問時，遇到空的點，就直接回傳「初始值」，否則就正常回傳

動態開點線段樹 (Dynamic Segment Tree)

- 而會發現，在每一次修改時，我們最多新增 $\log n$ 個節點
- 每次詢問時，與線段樹的詢問依舊相同

動態開點線段樹 (Dynamic Segment Tree)

- 而會發現，在每一次修改時，我們最多新增 $\log n$ 個節點
- 每次詢問時，與線段樹的詢問依舊相同
- 因此，空間複雜度為 $O(q_1 \log n)$（q_1 是修改次數）
- 至於時間複雜度，皆與一般的線段樹相同（不過 n 通常比一般情況大）

動態開點線段樹 (Dynamic Segment Tree)

- 而會發現，在每一次修改時，我們最多新增 $\log n$ 個節點
- 每次詢問時，與線段樹的詢問依舊相同
- 因此，空間複雜度為 $O(q_1 \log n)$（q_1 是修改次數）
- 至於時間複雜度，皆與一般的線段樹相同（不過 n 通常比一般情況大）
- 實作的話建議自己用這概念實作看看，我會在最後附上 code

動態開點 BIT！(Dynamic BIT)

- 你可能會想，那 BIT 可不可以動態開點

動態開點 BIT！(Dynamic BIT)

- 你可能會想，那 BIT 可不可以動態開點
- 答案是可以！

動態開點 BIT！(Dynamic BIT)

- 思考看看 BIT 的實作方式

```
int bit[MAXN];
void update(int pos, int val){
    while(pos < MAXN){
        bit[pos] += pos;
        pos += pos&-pos;
    }
}
```

- 有沒有什麼東西可以像陣列一樣使用，但又可以動態開新的空間？

- 實際上，將 bit 的宣告，從陣列替換成 map, unordered_map，即可有動態開點的 BIT！

```
map<int,int> bit;
void update(int pos, int val){
    while(pos < MAXN){
        bit[pos] += pos;
        pos += pos&-pos;
    }
}
```

動態開點 BIT！(Dynamic BIT)

- 然而，使用 map, unordered_map 的 BIT 在實際上並不常用
- 實際寫下去之後，你也會發現常常會得到一個 TLE
- 因為 map 有額外的 log，unordered_map 亂戳可能也會戳到重複的

動態開點 BIT！(Dynamic BIT)

- 然而，使用 map, unordered_map 的 BIT 在實際上並不常用
- 實際寫下去之後，你也會發現常常會得到一個 TLE
- 因為 map 有額外的 log，unordered_map 亂戳可能也會戳到重複的
- 有沒有更好的 STL 可以幫我們達到這點呢？

動態開點 BIT！(Dynamic BIT)

- 實際上，在 policy based data structure 裡面
- 還真的有一個能夠在 $O(1)$ 時間完成詢問的 hash table！
- 也就是很有名的 gp_hash_table（黑魔法）
- 這樣就可以無痛的使用動態開點 BIT 了（不過空間可能還是會逼你離散化）
- 詳細可以參考這篇 cf: https://codeforces.com/blog/entry/60737

持久化線段樹 (Persistent Segment Tree)

什麼是持久化資料結構（Persistent Data Structure）？

- 來看看一個例子

什麼是持久化資料結構（Persistent Data Structure）？

持久化 stack

現在你有一個 stack，你希望能夠維護以下幾種操作或詢問

1. 將一個數字 push 進去
2. 將一個數字 pop 掉
3. 詢問在第 k 次操作後，stack 的最上方是誰

什麼是持久化資料結構（Persistent Data Structure）？

假設操作依序為

1. push 1
2. push 3
3. push 4
4. pop
5. push 5
6. pop
7. ask 2
8. ask 3
9. ask 1
10. ask 4
11. ask 5

當前的指令	stack 內容物	版本編號
push 1	1	**1**
push 3	1 3	**2**
push 4	1 3 4	**3**
pop	1 3	**4**
push 5	1 3 5	**5**

什麼是持久化資料結構（Persistent Data Structure）？

- 因此，我們只要能夠儲存每一個版本即可
- 不過，你會發現，如果要把每一個版本的存起來
- 我們的空間又會 Explode!

什麼是持久化資料結構（Persistent Data Structure）？

- 然而，持久化資料結構其實就是處理這樣情況的一種方式！

持久化線段樹 (Persistent Segment Tree)

- 讓我們回到線段樹的主題，來看看一個最基本的例子

CSES - Range Queries and Copies

你現在一個 `list`，一開始上面只有一個 n 項的陣列，接下來你會進行 q 次三種操作

1. 將 `list` 上第 k 個陣列設成 x
2. 詢問 `list` 上第 k 個陣列 $[l, r]$ 區間的和
3. 將目前的陣列加進 `list` 的最後面

持久化線段樹 (Persistent Segment Tree)

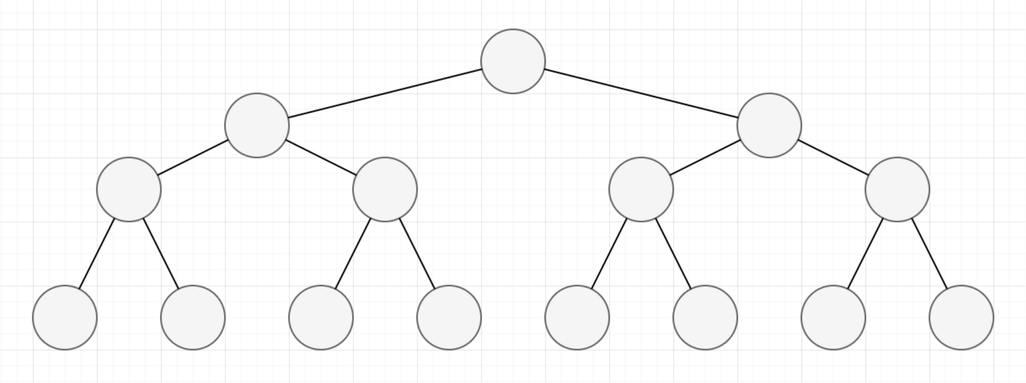
- 應該可以發現這其實跟我們剛剛講的 stack 很像
- 不過這前兩個操作其實是線段樹可以做到的操作吧！

持久化線段樹 (Persistent Segment Tree)

- 應該可以發現這其實跟我們剛剛講的 stack 很像
- 不過這前兩個操作其實是線段樹可以做到的操作吧！
- 但是，就跟剛剛講的一樣，你最多有可能存 q 個長度為 n 的陣列
- 當 n 很大時，根本就連空間都開不下阿！

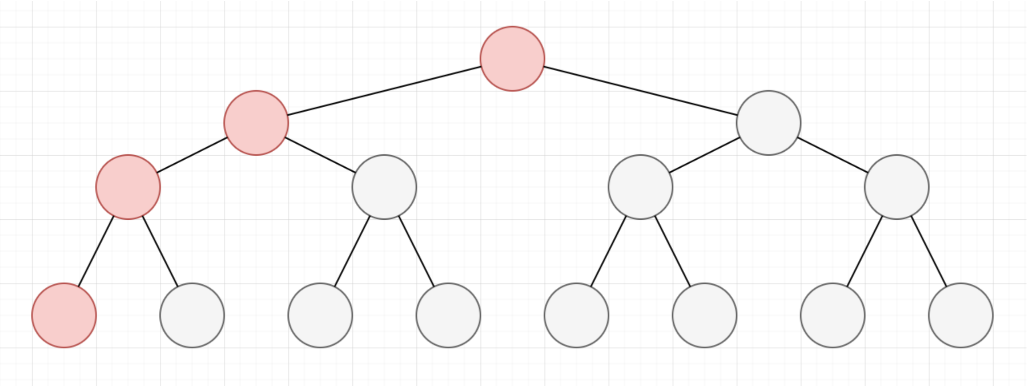
持久化線段樹 (Persistent Segment Tree)

- 仔細思考看看線段樹修改的過程



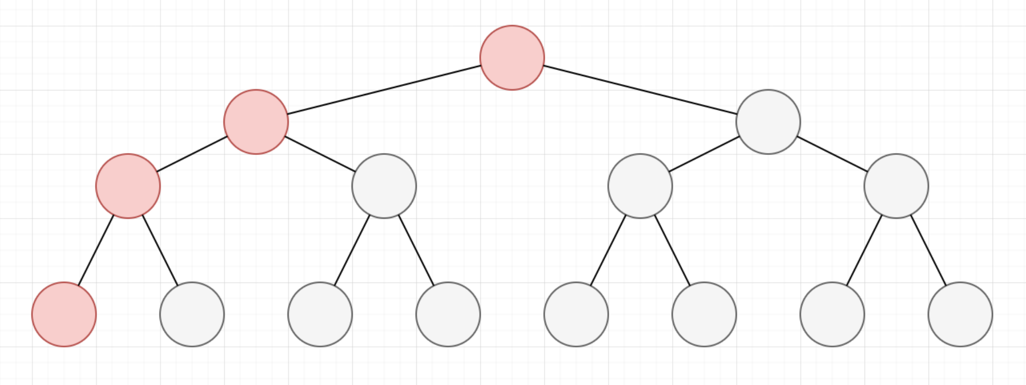
持久化線段樹 (Persistent Segment Tree)

- 仔細思考看看線段樹修改的過程



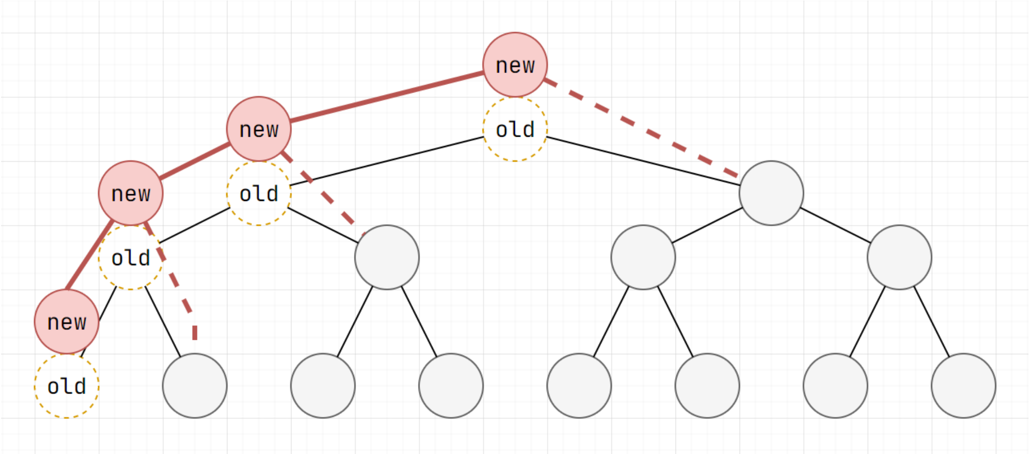
持久化線段樹 (Persistent Segment Tree)

- 有沒有發現其實我們只會改到 $\log n$ 個節點，而其他根本不會被改變！



持久化線段樹 (Persistent Segment Tree)

- 與動態開點的概念相同，當我們修改一個節點時，我們就開一個新的點
- 而新的點如果右節點或左節點沒被修改，就讓它連接到舊的版本（共用）



- 這個概念可能對與指標不熟的人來說有點難理解
- 不過可以想成，左右孩子原本是連接到 $\mathtt{idx} * 2$ 和 $\mathtt{idx} * 2 + 1$ 的
- 但現在，連接的可以是不同編號的，甚至是和其他線段樹共用這些節點

持久化線段樹 (Persistent Segment Tree)

- 我們實際來看 code 解釋一次吧
- 範例 code $\Leftarrow$

持久化線段樹 (Persistent Segment Tree)

- 有了持久化線段樹後的我們能夠做什麼呢？
- 事實上，我們現在可以解決原本要離線才能解決的問題了！

CSES - Distinct Value Queries (強制在線板)

有一個 n 項的陣列以及 q 筆詢問，每筆詢問請找到區間 $[l, r]$ 內有幾個不同的數字。

- $1 \leq n, q \leq 2 \cdot 10^5$
- $1 \leq a_i \leq 10^9$

CSES - Distinct Value Queries (強制在線板)

有一個 n 項的陣列以及 q 筆詢問，每筆詢問請找到區間 $[l,r]$ 內有幾個不同的數字。

- $1 \le n, q \le 2 \cdot 10^5$
- $1 \le a_i \le 10^9$

- 應該會發現，其實我們可以模仿我們在離線時的做法

持久化線段樹 (Persistent Segment Tree)

CSES - Distinct Value Queries （強制在線板）

有一個 n 項的陣列以及 q 筆詢問，每筆詢問請找到區間 $[l, r]$ 內有幾個不同的數字。

- $1 \leq n, q \leq 2 \cdot 10^5$
- $1 \leq a_i \leq 10^9$

- 應該會發現，其實我們可以模仿我們在離線時的做法
- 我們可以在一開始，就將所有左界的線段樹版本給建出來（與離線相同）

CSES - Distinct Value Queries （強制在線板）

有一個 n 項的陣列以及 q 筆詢問，每筆詢問請找到區間 $[l, r]$ 內有幾個不同的數字。

- $1 \leq n, q \leq 2 \cdot 10^5$
- $1 \leq a_i \leq 10^9$

- 應該會發現，其實我們可以模仿我們在離線時的做法
- 我們可以在一開始，就將所有左界的線段樹版本給建出來（與離線相同）
- 接著，每次我們就可以直接去取用左界為 l 的版本，詢問 r 的值了！

區間第 k 大

給你一個 n 項的陣列，接下來有 q 筆詢問，每次請找到區間 $[l, r]$ 中第 k 大的數字

- $1 \leq n, q \leq 2 \cdot 10^5$
- $1 \leq a_i \leq 10^9$

持久化線段樹 (Persistent Segment Tree)

- 考慮一個簡單一點的問題，假設你今天想要找陣列 $[1, r]$ 第 k 大的數字，你會怎麼做？

持久化線段樹 (Persistent Segment Tree)

- 考慮一個簡單一點的問題，假設你今天想要找陣列 $[1, r]$ 第 k 大的數字，你會怎麼做？
- 前面才剛講過的「線段樹上二分搜」！

更多例題

?

- 在這之後的題目，我整理了各種類型的題目給大家做練習
- 我會口頭上與大家講這些題目的做法

欸？ 這看起來是不是就要持久化阿

AtCoder Beginner Contest 273 E - Notebook

你現在手上有一個 stack 和一本筆記本，接下來有四種操作

1. 將 x 推進 stack
2. 將 stack 最上面的數字 pop 掉
3. 將現在的 stack 紀錄在筆記本的第 y 頁
4. 把手上的 stack 替換成筆記本第 z 頁的 stack

蝦不是建一棵線段樹就做完了嗎

AtCoder Beginner Contest 273 E - Notebook

你現在手上有一個 stack 和一本筆記本，接下來有四種操作

1. 將 x 推進 stack
2. 將 stack 最上面的數字 pop 掉
3. 將現在的 stack 紀錄在筆記本的第 y 頁
4. 把手上的 stack 替換成筆記本第 z 頁的 stack

蝦不是建一棵線段樹就做完了嗎

TIOJ 1872 - 最小公倍數

給你一個陣列，每次詢問請輸出區間 $[l, r]$ 的最小公倍數 $\mod 10^9 + 7$ 後的結果

- $1 \leq n, q \leq 10^6$
- $1 \leq c_i \leq 10^6$

剛剛那題的強制在線？

Codeforces 1422F - Boring Queries

給你一個陣列，每次詢問請輸出區間 $[l, r]$ 的最小公倍數 $\mod 10^9 + 7$ 後的結果（強制在線）

- $1 \leq n, q \leq 10^6$
- $1 \leq c_i \leq 10^6$

蛤這題要怎麼做 R

CSES - Movie Festival Queries

有 n 部電影，每部電影的播映時間為 $[a_i, b_i]$. 有 q 筆詢問，每次輸出從時間 l_i 到時間 r_i 一共可以完整看完幾部電影

- $1 \leq n, q \leq 2 \cdot 10^5$
- $1 \leq a < b \leq 10^6$

超級超級經典題！！！

IOI 2014 - Wall

你有 n 面牆，接下來你要做 q 個操作，一共有兩種操作

1. 將區間 $[l, r]$ 高度不到 x 的牆高度提升至 x
2. 將區間 $[l, r]$ 高度超過 x 的牆高度降低至 x

請在最後輸出所有牆的高度

欸？ 困難版的 Wall？

111 學年薇閣高中校內賽 - pG

某天，山姆決定要來整理電腦中的檔案。在某個資料夾中，一共有 n 個檔案排成一列，每個檔案的名稱有 a_i 個字。
接下來，山姆會對這些檔案進行一些操作。

1. 將區間 $[l, r]$ 的檔案依照名稱長度**由小到大排序**
2. 將區間 $[l, r]$ 的檔案依照名稱長度**由大到小排序**
3. 將區間 $[l, r]$ 檔案名稱長度不到 x 的，增加到 x
4. 將區間 $[l, r]$ 檔案名稱長度超過 x 的，減少到 x

在整理過後，他希望能夠知道有哪些位置的檔案名稱長度會是 k，請你幫他解決這個問題。