Segment/Interval Tree

for Sprout 2014 by Chin Huang Lin

Sprout

本節概要

- 回顧分治法—分治為什麼有用？
- Segment/Interval Tree 概要與特性
- Segment/Interval Tree 定義與操作
- 複雜度分析

分治的力量

- 大事化小，小事化無
  - ex. 分地問題，複雜變簡單
  - ex. 排序問題，大量變小量
- 分工合作，權責分明
  - ex. 合併排序，兩邊已排更容易
  - ex. 快速取冪，切半不用重複算

分工合作，權責分明

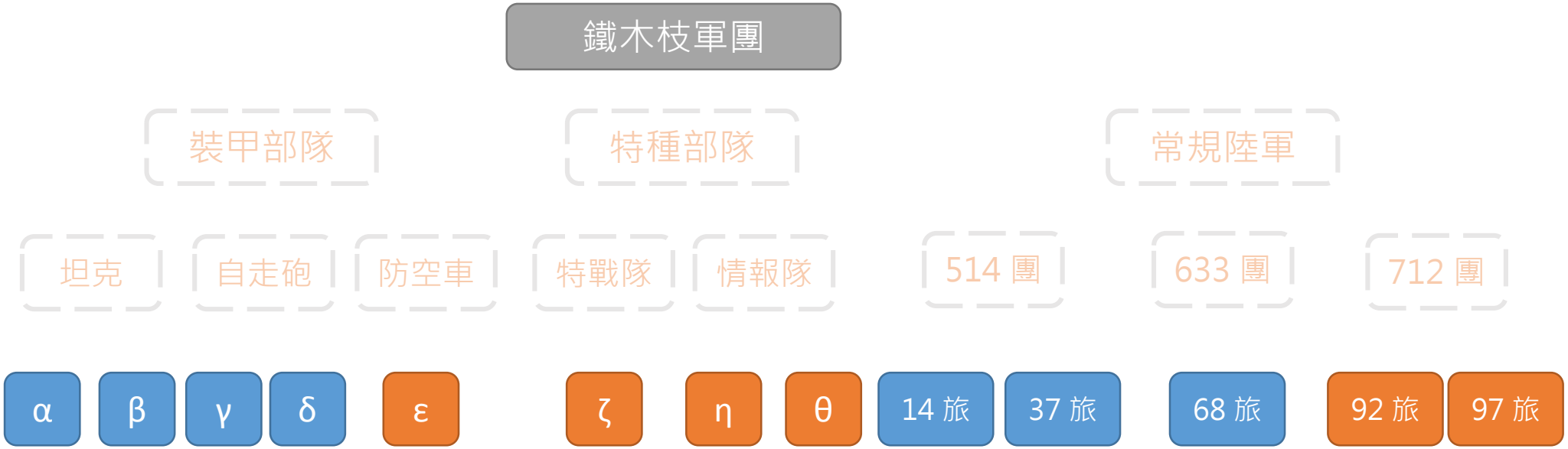
- 一個有效率的組織應該長這樣

分工合作，權責分明

- 下令調度防空車、特種部隊和 712 團

越俎代庖，大權在握

- 下令調度防空車、特種部隊和 712 團



分工合作 vs 越俎代庖

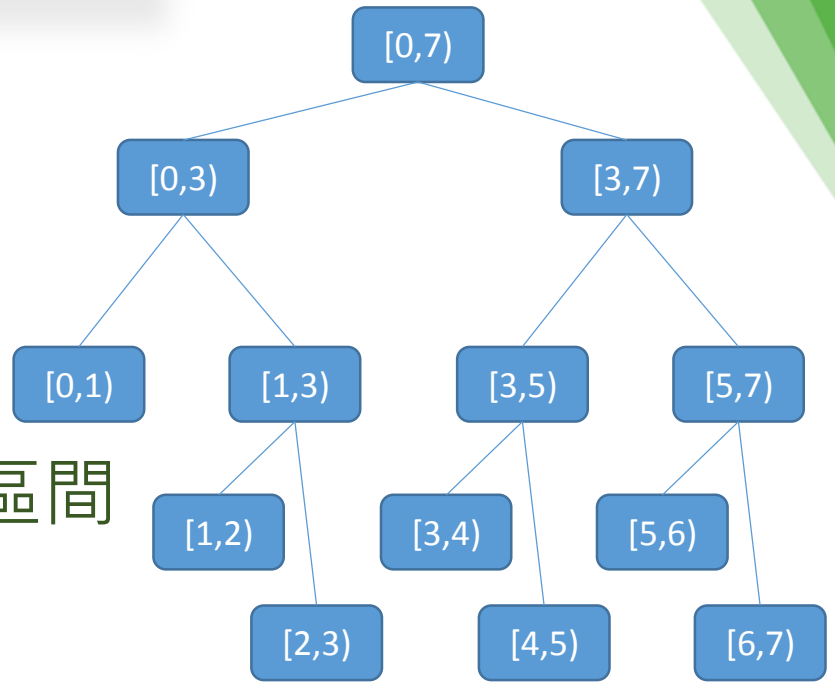
- 組織複雜 vs 組織簡單
- 依法行政 vs 隨心所欲
- 資料量大 vs 資料量小
- 管理方便 vs 管理困難

Segment/Interval/Range Tree?

- 概念型的資料結構！
  - 沒有特定的形式與功能
  - 支援連續區間型的可分治操作
  - 可隨需求有多種變形
- 完全類似的核心概念
  - 底下以 segment tree 作為統一代稱
- 最早出現於計算幾何
  - 給你很多平面上的線段，每次動態詢問某條垂直線與多少條線段相交
  - 平面上有許多矩形，問所有矩形的聯集面積
  - 平面上有許多矩形，問某條垂直線上有多少個矩形
  - ……

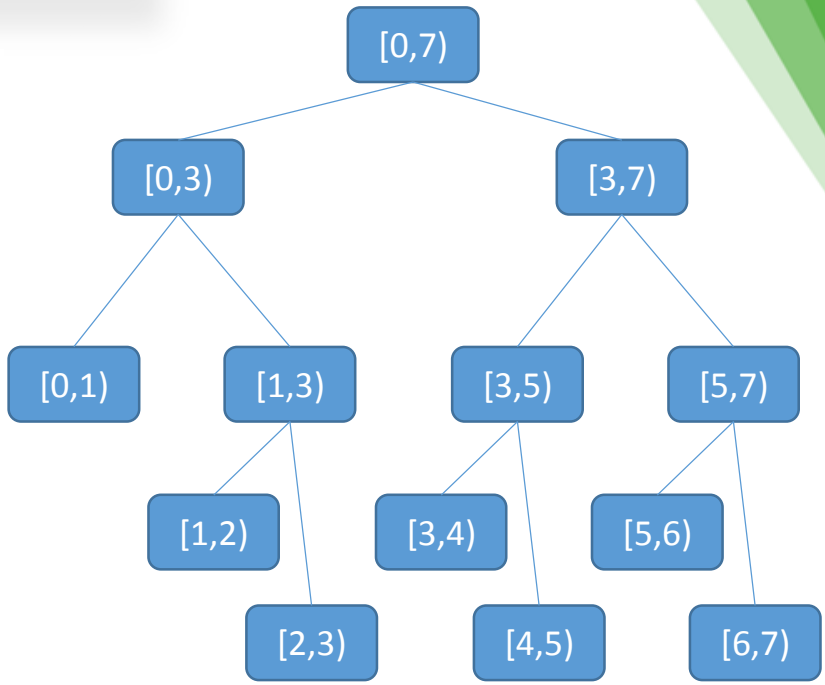
基本框架

- 一棵「平衡」的二元樹
  - 不一定是滿二元樹！
- 每個節點都對應到一個連續區間
- 根節點為總區間
- 左兒子為左半區間，右兒子為右半區間
- 葉節點區間長度為 1



基本框架

```
struct Node{
    int l,r;
    int lson, rson;
    int data;
};
```

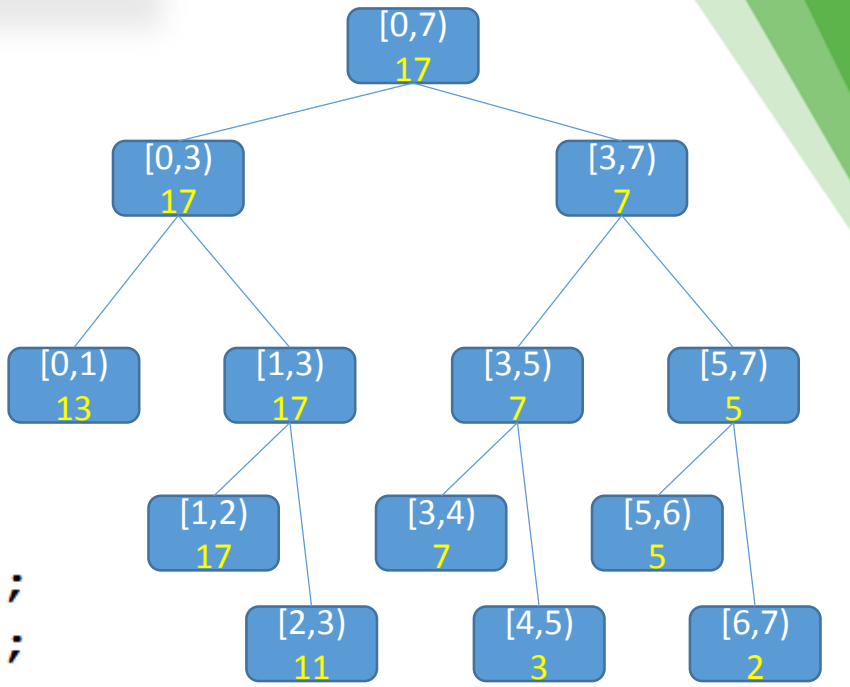


- l, r 記錄節點對應到的區間
- lson, rson 為左兒子與右兒子的 index
- data 為該節點對應到的區間的資料
  - 底下先以 RMQ 問題為例：求區間最大值

初始建構

```
int array[MAXN];
Node ST[MAXN*2];
int stptr = 0;
void build(int l,int r,int index){
    ST[index].l=l, ST[index].r=r;
    if( l == r-1 )
        ST[index].data=array[l];
    else {
        int lson=ST[index].lson=stptr++;
        int rson=ST[index].rson=stptr++;
        build(l,(l+r)/2,lson);
        build((l+r)/2,r,rson);
        ST[index].data=max(ST[lson].data,ST[rson].data);
    }
}
```

build(0, 7, 0)

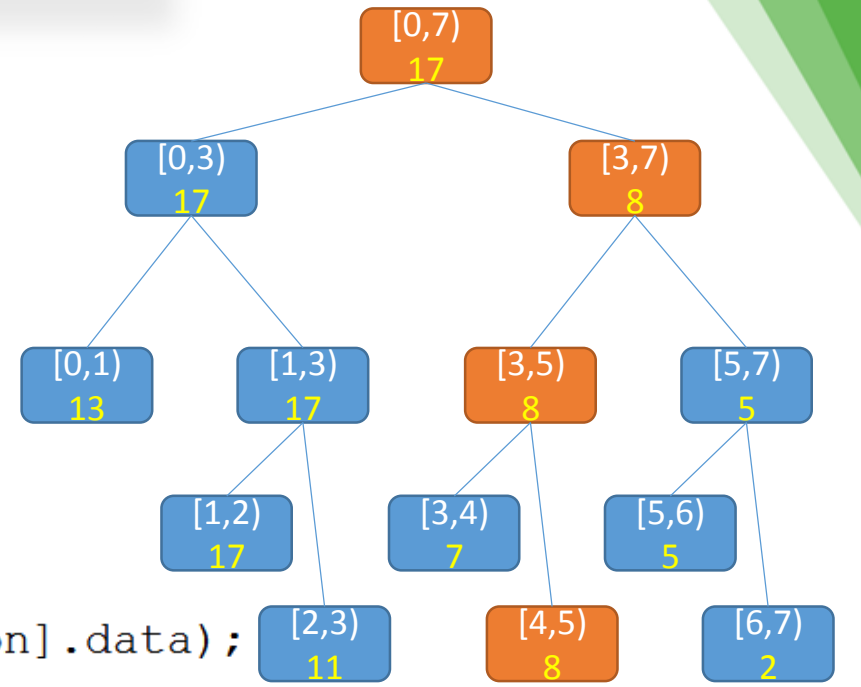


13	17	11	7	3	5	2

單點修改

```
void modify(int x,int v,int index){
    if( ST[index].l == ST[index].r-1 )
        ST[index].data=v;
    else {
        int mid = (ST[index].l+ST[index].r)/2;
        int lson=ST[index].lson;
        int rson=ST[index].rson;
        if( x < mid ) modify(x,v,lson);
        else modify(x,v,rson);
        ST[index].data=max(ST[lson].data,ST[rson].data);
    }
}
```

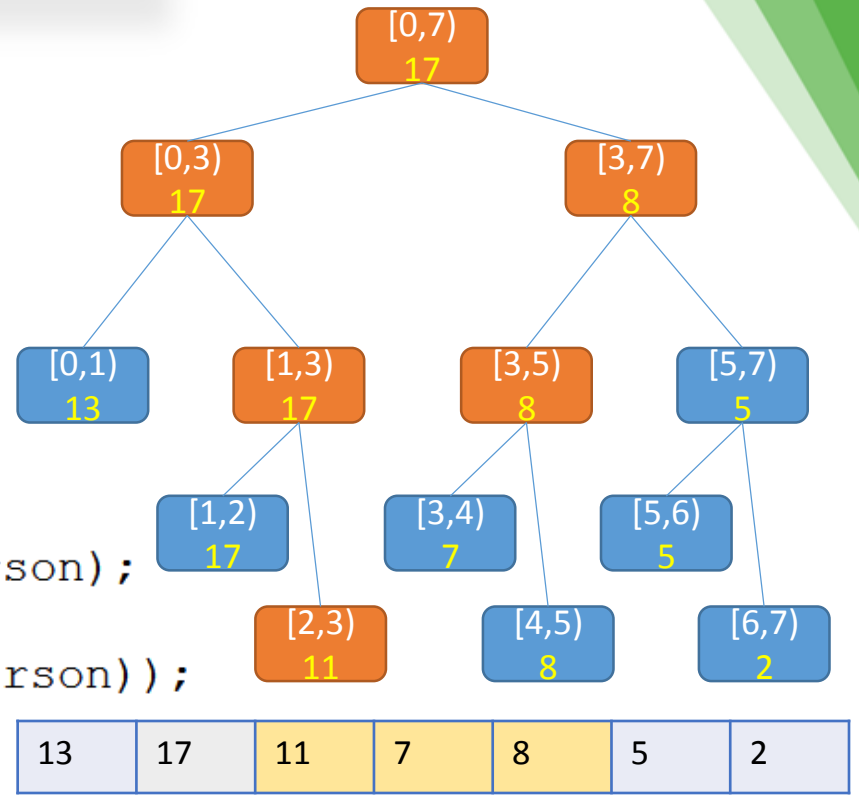
modify(4, 8, 0)



區間查詢

```
int query(int l,int r,int index){
    if( ST[index].l == l && ST[index].r == r )
        return ST[index].data;
    else {
        int mid = (ST[index].l+ST[index].r)/2;
        int lson=ST[index].lson;
        int rson=ST[index].rson;
        if( r <= mid ) return query(l,r,lson);
        else if( l >= mid ) return query(l,r,rson);
        else return
            max(query(l,mid,lson),query(mid,r,rson));
    }
}
```

query(2, 5, 0)



簡單嗎？

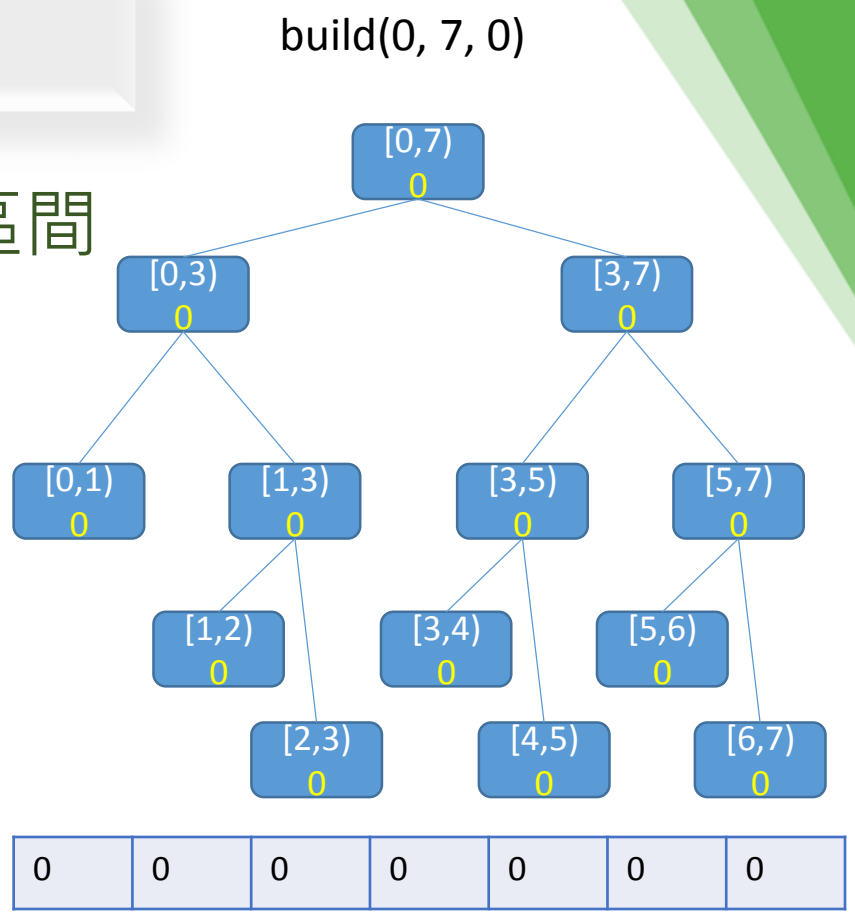
- 好簡單！只要把握每個節點 `data` 的定義，就能輕鬆掌握
- 複雜度分析：
  - 空間複雜度：可以證明節點數為 $2n$，$O(n)$
  - 時間複雜度：
    - 初始建構：所有節點恰會建構一次，每個節點 $O(1)$，配合節點樹可得為 $O(n)$
    - 單點修改：該點的所有祖先節點都會被修改到，其他都不會被修改到，$O(\log n)$
    - 區間查詢：每筆詢問最多詢問到深度為 $O(\log n)$ 的節點；在一次詢問中，每一層不會有超過 2 個節點被詢問 (想一想，為什麼？)，總複雜度為 $O(\log n)$
- 有沒有辦法區間修改、單點查詢之類的呢……？

區間覆蓋數問題

- 你有一個一維的數線，座標是 $[0,2n)$ 內的整數
- 每次操作可能會：
  - 新增一條線段，覆蓋 $[l,r)$ 區間
  - 詢問某個單位區間 $[l,l+1)$ 上覆蓋的線段數
  - 移除某條先前覆蓋的區間
- 操作次數 $\leqq 10^5$，$n \leqq 5 * 10^5$
- 想一想：如果座標不是 $[0,2n)$，而可能是 $[0,2147483647)$ 怎麼辦呢？

初始建構

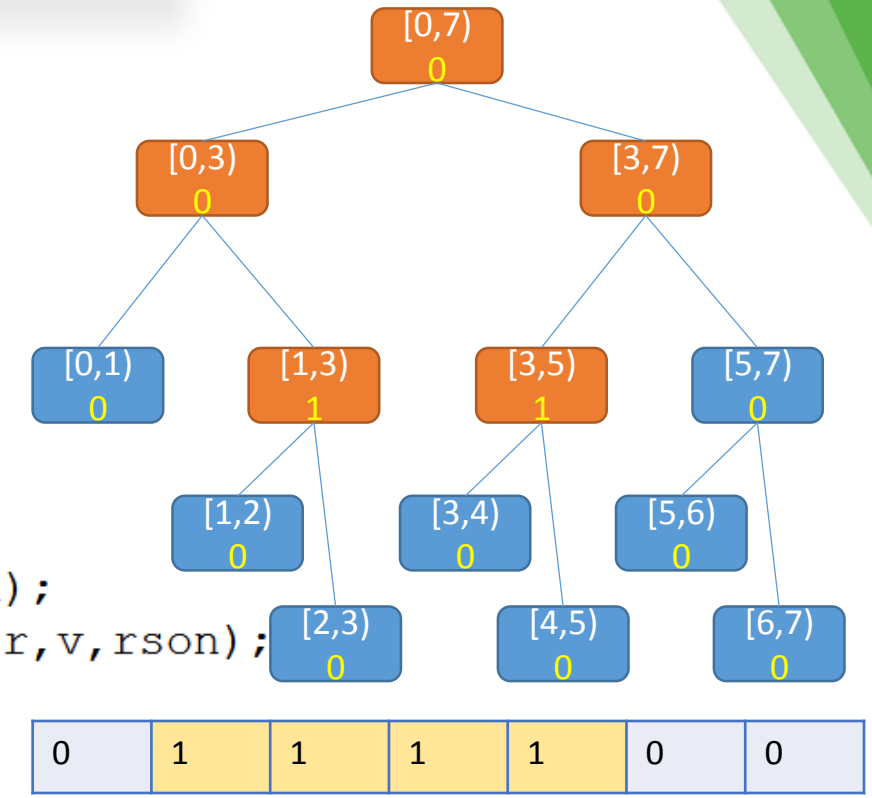
- data 的定義改成「完整覆蓋當前區間的線段數量」
- 和 RMQ 問題的建構沒本質的不同，只是 data 全部初始化為 0



區間修改

```
void modify(int l,int r,int v,int index){
    if( ST[index].l == l && ST[index].r == r )
        ST[index].data += v;
    else {
        int mid = (ST[index].l+ST[index].r)/2;
        int lson=ST[index].lson;
        int rson=ST[index].rson;
        if( r <= mid ) modify(l,r,v,lson);
        else if(  l >= mid ) modify(l,r,v,rson);
        else modify(l,mid,v,lson), modify(mid,r,v,rson);
    }
}
```

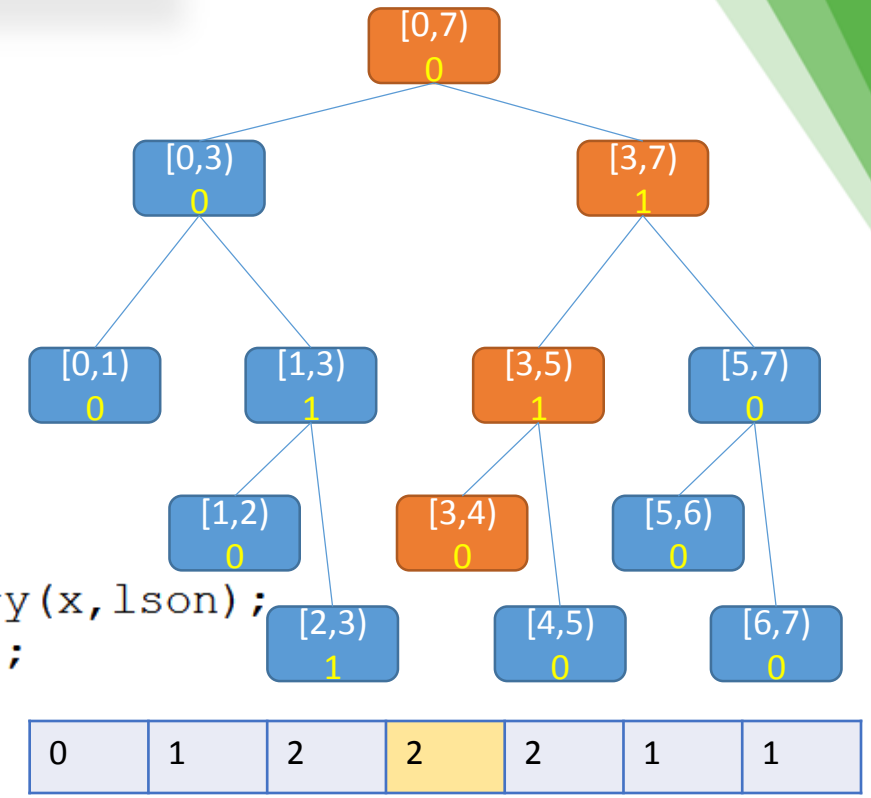
modify(1, 5, 1, 0)



單點查詢

```
int query(int x,int index){
    if( ST[index].l == ST[index].r-1 )
        return ST[index].data;
    else {
        int mid = (ST[index].l+ST[index].r)/2;
        int lson=ST[index].lson;
        int rson=ST[index].rson;
        if( x < mid ) return ST[index].data+query(x,lson);
        else return ST[index].data+query(x,rson);
    }
}
```

query(3, 0)



簡單嗎？

- 還是很簡單！乍看之下只是把 `modify` 和 `query` 交換而已
- 不過，有些點的 `data` 似乎不完全滿足定義？
  - 假設某個節點對應到 $[l,r]$，且有 k 條線段覆蓋它，那麼它的兒子們一定也被這 k 條線段覆蓋
  - 在上面的作法中，兒子的 `data` 可能不是「新鮮的」—它會在查詢的時候才順便看看哪些被長輩們「暗蓋」起來了！
  - 想一想：這意味著線段樹在什麼情況下可能會有潛在的問題呢……？
- 複雜度分析：
  - 空間複雜度：毫無改變
  - 時間複雜度：
    - 初始建構：毫無改變
    - 區間修改：跟區間查詢一樣，$O(\log n)$
    - 單點查詢：跟單點修改一樣，$O(\log n)$

線段樹怎麼可能這麼給力？

- 關鍵前提：面對的問題須具有可分治性
  - 區間的最大值
  - 區間的總和
  - 區間的標準差
  - 區間同加某個值（單點查詢）
  - 區間開根號（單點查詢）
  - 區間同加某個值（區間查詢）
  - 區間開根號（區間查詢）
  - 區間的逆序數對數
  - 區間的最大公因數
  - 區間的最小公倍數
  - ……