


DSAP期末報告-線段樹

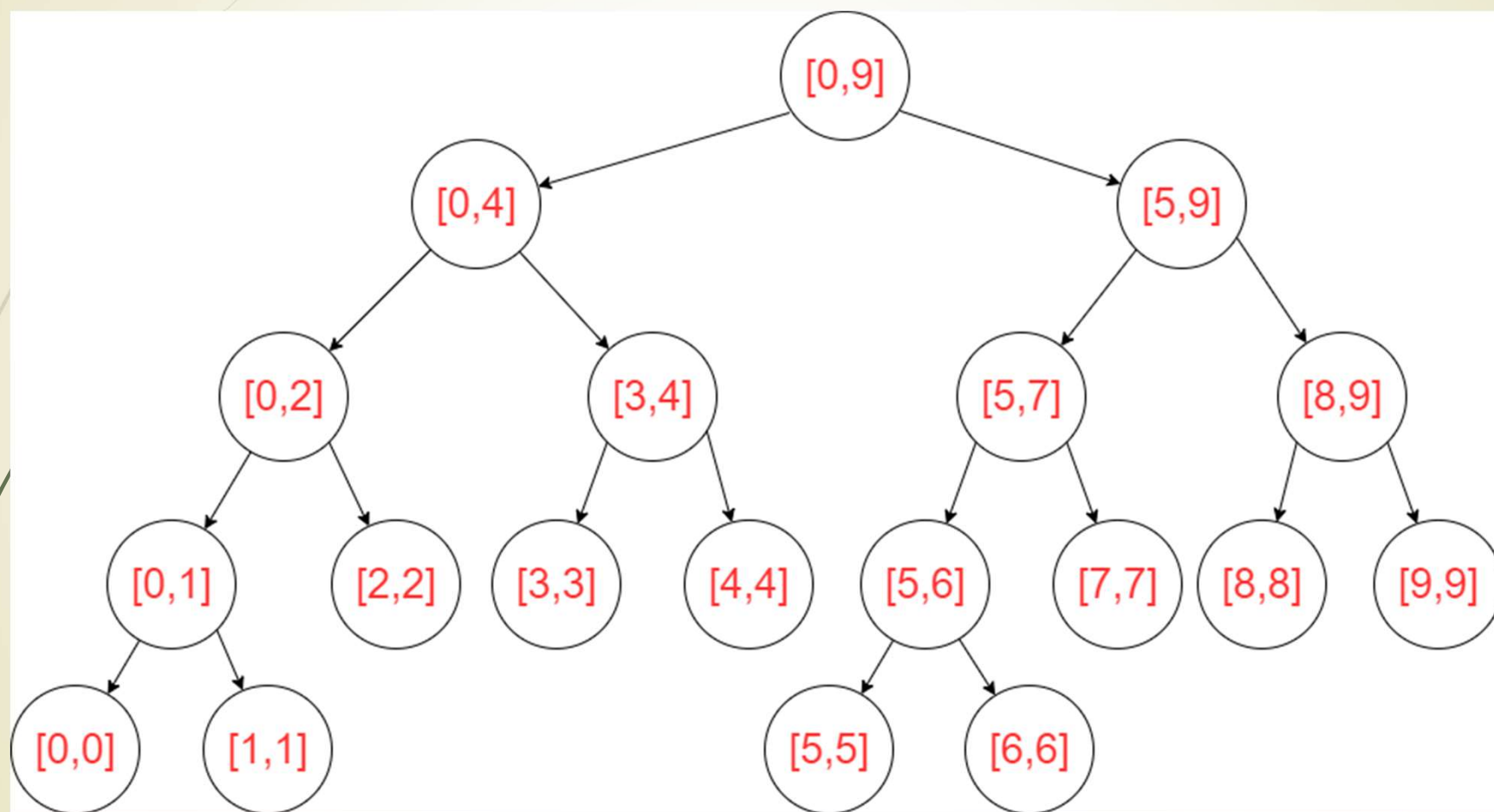
報告人:資管一楊中




主題簡介

- ➡ 甚麼是線段樹?
 - ➡ 線段樹能做甚麼?
 - ➡ 線段樹的優點?
- 

最一般的線段樹





動機


➡ 為何選擇線段樹來當作主題?



範例1. 求區間最大值

- 問題:給一個數列 $T_1, T_2, T_3, \dots, T_n$ ，求 T_a 到 T_b 之間(涵蓋 T_a 、 T_b)的最大值。
- EX:給定陣列 $[1, 2, 3, 4, 5]$ ，
求 $\max(2, 4)$? 求 $\max(1, 5)$? 求 $\max(2, 3)$?

來源



- ➡ 暴力解:每問一次就回頭遍歷陣列

- ➡ 時間複雜度: $O(N)$

- ➡ 如何優化?



EX:給定陣列[3,2,4,5,6,8,1,2,9,7]

```
for(int l = 0; l < testcase; l++){  
    int left, right = 0;  
    cin >> left >> right;  
    int max = arr[left-1];  
    for(int j = left; j <= right-1; j++){  
        if(max < arr[j])  
            max = arr[j];  
    }  
    cout << max << '\n';  
}
```



線段樹解

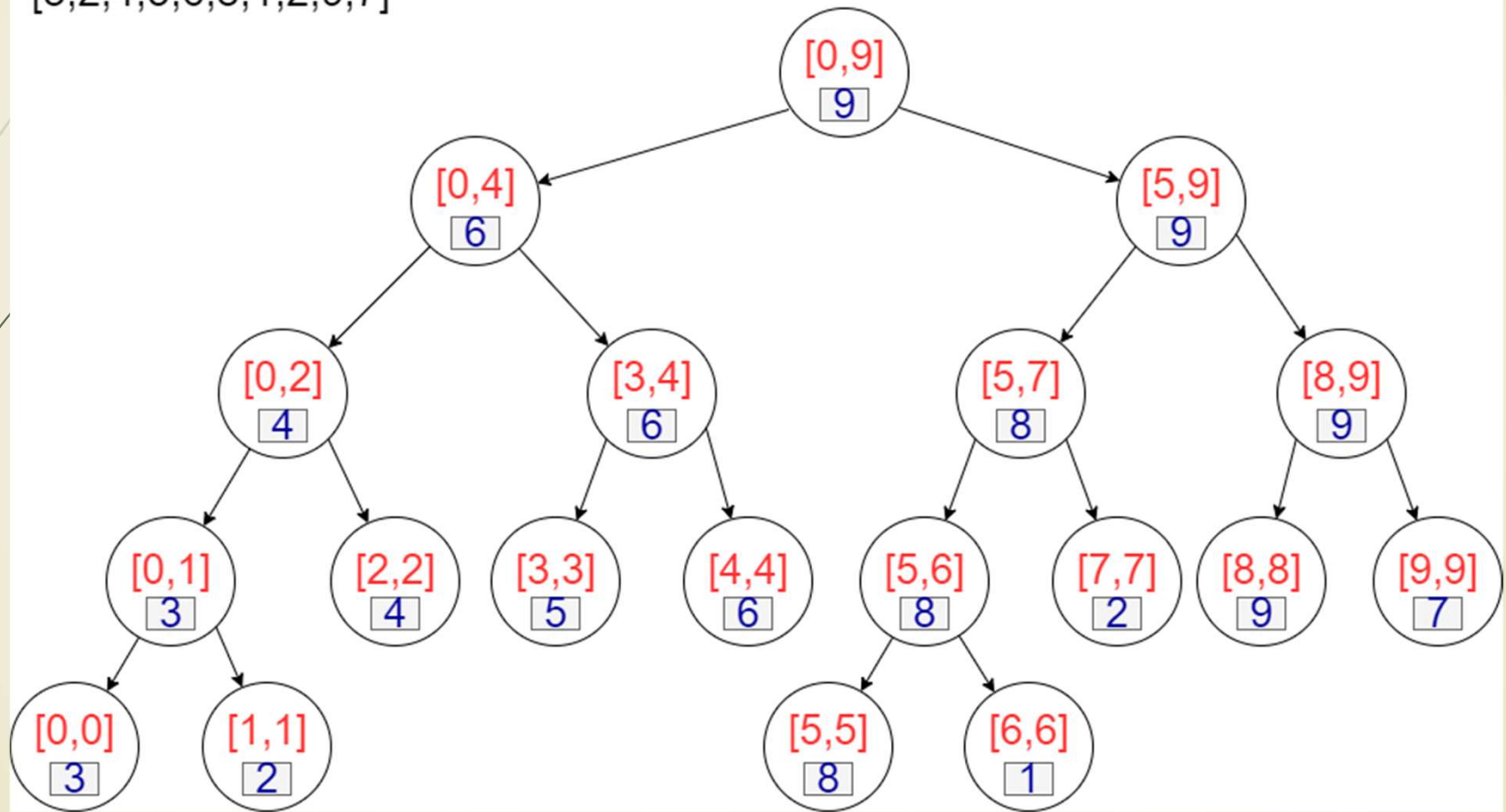
➡ 最差情況: $O(\log N)$

➡ 如何實現?

➡ 代碼

EX: 給定陣列[3,2,4,5,6,8,1,2,9,7]

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





實作線段樹類別

➡ 應該具備甚麼功能?


➡ (1) 單點查詢、區間查詢

➡ (2) 單點加值、區間加值

➡ (3) 單點賦值、區間賦值

Q : 線段樹內儲存的資料是甚麼?


A : 最大最小值、區間和等等



具體要有的函式(非一定，因人而異)

(1)建構子：以遞迴方式建樹

```
void SegmentTree::push_up(int vertex)
{
    //合併子節點的結果
    maxVal[vertex] = max(maxVal[vertex * 2 + 1], maxVal[vertex * 2 + 2]);
    minVal[vertex] = min(minVal[vertex * 2 + 1], minVal[vertex * 2 + 2]);
    sumVal[vertex] = (sumVal[vertex * 2 + 1] + sumVal[vertex * 2 + 2]);
}
```



```
void SegmentTree::buildTree(vector<int>& v, int vertex, int l, int r)
{
    if(l == r){
        maxVal[vertex] = v[r];
        minVal[vertex] = v[r];
        sumVal[vertex] = v[r];
        return;
    }
    int middle = (l + r) / 2;
    buildTree(v, vertex * 2 + 1, l, middle);
    buildTree(v, vertex * 2 + 2, middle + 1, r);
    //將子節點的結果合併後回傳給父節點
    push_up(vertex);
}


SegmentTree(vector<int>& v, int l, int r)
{
    buildTree(v, 0, l, r);
};
```

(2) 區間、單點查詢: 以遞迴方式查找

```
int SegmentTree::maxi(int vertex, int ql, int qr, int l, int r)
{
    // if(l != r)//當前區間為一個點，不可再往下找最大值(因為該點為leaf)
    //     push_up(vertex);

    if(ql == l && qr == r){//當前節點範圍等同欲查詢區間，直接回傳結果
        return maxVal[vertex];
    }
    push_down(vertex, l, r);//將lazy tag往下傳遞

    int middle = (l + r) / 2;
    if(qr <= middle){
        return maxi(vertex * 2 + 1, ql, qr, l, middle);/*如果欲查詢的區間範圍完全在前半段*/
    }else if(ql > middle){
        return maxi(vertex * 2 + 2, ql, qr, middle + 1, r);/*如果欲查詢的區間範圍完全在後半段*/
    }else{
        /*如果欲查詢的範圍橫跨左右半段*/
        return std::max(maxi(vertex * 2 + 1, ql, middle, l, middle), maxi(vertex * 2 + 2, middle + 1, qr, middle + 1, r));
    }
}
```



(3) 區間、單點加值: 遞迴、懶人標記

➡ Q : 為甚麼要用懶人標記?

A : 若要逐點更新，複雜度達 $O(N\log N)$
而用懶人標記，複雜度 $O(\log N)$

➡ Q : 甚麼是懶人標記?

```
void SegmentTree::push_down(int vertex, int l, int r)
```

```
{
```

```
    //先加後改
```

```
    //將lazy tag傳遞給子節點
```

```
    if(lazyAdd[vertex] != 0){
```

```
        lazyAdd[vertex * 2 + 1] += lazyAdd[vertex];
```

```
        lazyAdd[vertex * 2 + 2] += lazyAdd[vertex];
```

```
        maxVal[vertex * 2 + 1] += lazyAdd[vertex];
```

```
        maxVal[vertex * 2 + 2] += lazyAdd[vertex];
```

```
        minVal[vertex * 2 + 1] += lazyAdd[vertex];
```

```
        minVal[vertex * 2 + 2] += lazyAdd[vertex];
```

```
        int middle = (r + 1) / 2;
```

```
        sumVal[vertex * 2 + 1] += lazyAdd[vertex] * (middle - l + 1);
```

```
        sumVal[vertex * 2 + 2] += lazyAdd[vertex] * (r - middle);
```

```
        lazyAdd[vertex] = 0;
```

```
    }
```

```
    if(lazyChg[vertex] == 0) return;
```

```
    else{
```

```
        lazyChg[vertex * 2 + 1] = lazyChg[vertex];
```

```
        lazyChg[vertex * 2 + 2] = lazyChg[vertex];
```

```
        maxVal[vertex * 2 + 1] = lazyChg[vertex];
```

```
        maxVal[vertex * 2 + 2] = lazyChg[vertex];
```

```
        minVal[vertex * 2 + 1] = lazyChg[vertex];
```

```
        minVal[vertex * 2 + 2] = lazyChg[vertex];
```

```
        int middle = (r + 1) / 2;
```

```
        sumVal[vertex * 2 + 1] = lazyChg[vertex] * (middle - l + 1);
```

```
        sumVal[vertex * 2 + 2] = lazyChg[vertex] * (r - middle);
```

```
        lazyChg[vertex] = 0;
```


```
        lazyAdd[vertex] = 0;
```

```
    }
```

```
}
```



```
void SegmentTree::updateAdd(int vertex, int ql, int qr, int l, int r, int addVal)
{
    if(ql == l && qr == r){//當前區間為欲更新區間，直接更新後結束
        lazyAdd[vertex] += addVal;
        maxVal[vertex] += addVal;
        minVal[vertex] += addVal;
        sumVal[vertex] += addVal * (r - l + 1);
        return ;
    }
    push_down(vertex, l, r);//向下傳遞lazy tag
    int middle = (l + r) / 2;
    if(qr <= middle){/*如果欲查詢的區間範圍完全在前半段*/
        updateAdd(vertex * 2 + 1, ql, qr, l, middle, addVal);
    }else if(ql > middle){/*如果欲查詢的區間範圍完全在後半段*/
        updateAdd(vertex * 2 + 2, ql, qr, middle + 1, r, addVal);
    }else{
        /*如果欲查詢的範圍橫跨左右半段*/
        updateAdd(vertex * 2 + 1, ql, middle, l, middle, addVal);
        updateAdd(vertex * 2 + 2, middle + 1, qr, middle + 1, r, addVal);
    }
    //向上合併子節點的結果
    push_up(vertex);
}
```

(4) 區間、單點賦值：先加後改VS先改後加

- ➡ Note:兩種順序沒有絕對，可依需要選擇不同的優先序。
- ➡ Note:除了加值和賦值，還可以依其他需求使用懶人標記的概念
- ➡ Note:當同時維護多組懶人標記，切記理清資料的相依性。

```
void SegmentTree::updateChg(int vertex, int ql, int qr, int l, int r, int chgVal)
{
    if(ql == l && qr == r){//當前區間為欲更新區間，直接更新後結束
        lazyChg[vertex] = chgVal;
        maxVal[vertex] = chgVal;
        minVal[vertex] = chgVal;
        sumVal[vertex] = chgVal * (r - l + 1);
        return ;
    }
    push_down(vertex, l, r);//向下傳遞lazy tag
    int middle = (l + r) / 2;
    if(qr <= middle){/*如果欲查詢的區間範圍完全在前半段*/
        updateChg(vertex * 2 + 1, ql, qr, l, middle, chgVal);
    }else if(ql > middle){/*如果欲查詢的區間範圍完全在後半段*/
        updateChg(vertex * 2 + 2, ql, qr, middle + 1, r, chgVal);
    }else{
        /*如果欲查詢的範圍橫跨左右半段*/
        updateChg(vertex * 2 + 1, ql, middle, l, middle, chgVal);
        updateChg(vertex * 2 + 2, middle + 1, qr, middle + 1, r, chgVal);
    }
    //向上合併子節點的結果
    push_up(vertex);
}
```

複雜度分析

- (1) 建樹空間複雜度- $O(N)$, N for nodes
- Why?
- $1+2+4+8+\dots+2^{\lceil \log N \rceil} < 2^{\lceil \log N \rceil+1} < 4N$



- ➡ (2) 建樹時間複雜度- $O(N)$, N for nodes

- ➡ (3) 單點、區間查詢時間複雜度- $O(\log N)$

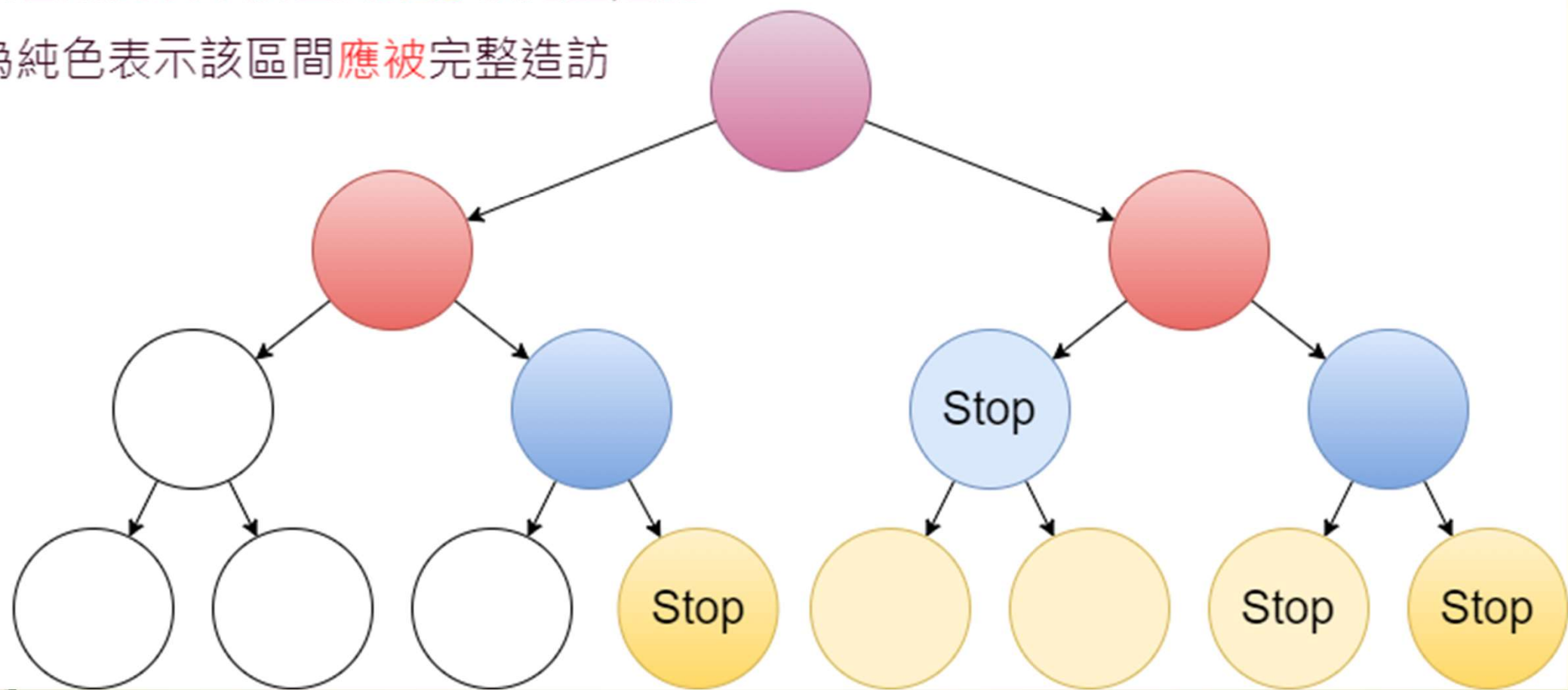
- ➡ Why?


- ➡ 仔細觀察查詢過程，發現每次最多分支四層
因此最多造訪 $4\log N$ 個點(Hint:區間連續)


真的嗎?假如有五個點的話會怎樣?

節點顏色漸變表示該區間未必被完整造訪

節點為純色表示該區間應被完整造訪



- 
- ➡ (3) 單點、區間修改時間複雜度- $O(\log N)$
 - ➡ 單點修改-樹高約為 $\log N$
 - ➡ 區間修改-只先標記要修改的區間，實作方式類似於區間查詢。實際呼叫查詢函式等時，再請他們順便修改區間。因此複雜度同區間查詢為 $O(\log N)$



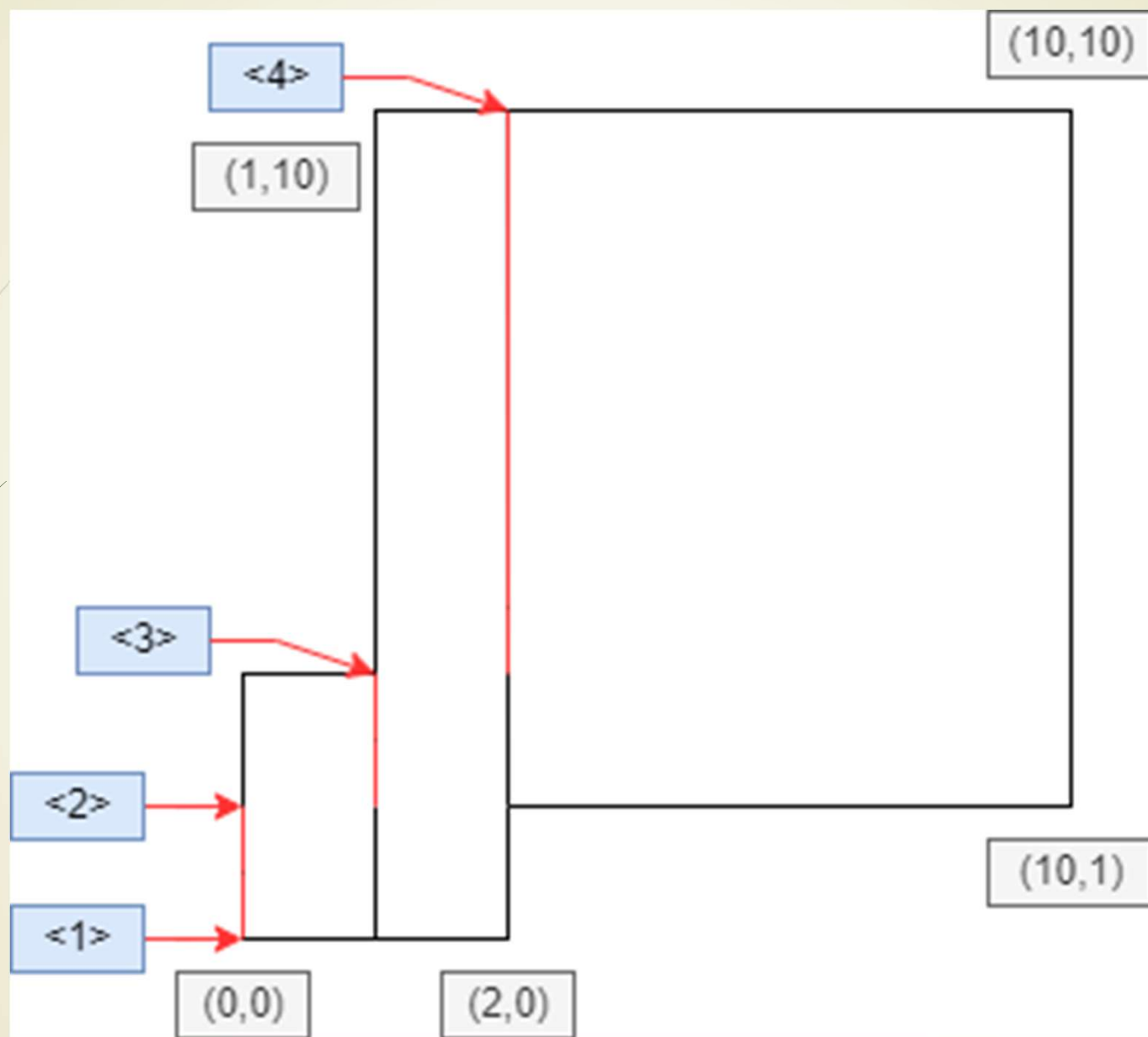
其他可被線段樹處理的問題？

- 要求：可分治性！
- 舉例：LCM、GCD
 - 矩形覆蓋面積問題
 - 牆面染色問題等等

範例2. 矩形面積覆蓋問題

Q：給定 N 個在平面上的矩形，求他們彼此之間的覆蓋面積(來源)

- 關鍵字：掃描線、離散化
- 在此只講概念
- 代碼





更多應用和變化

➤ (1) 線段樹和二分搜尋法

例：查詢 $[1, N]$ 之間第一個不小於 K 的數

Hint：利用線段數的特性

➤ (2) 高維線段樹-樹包樹

可維護二維平面，乃至更高維度等

查詢複雜度 $O((\log N)^D)$, D for 維度

Reference :

- <https://hackmd.io/@wiwiho/cp-note/%2F%40wiwiho%2FCPN-segment-tree>
- <https://zh.wikipedia.org/zh-tw/%E7%B7%9A%E6%AE%B5%E6%A8%B9>
- https://cp-algorithms.com/data_structures/segment_tree.html#find-the-smallest-number-greater-or-equal-to-a-specified-number-acceleration-with-fractional-cascading
- <https://zerojudge.tw/ShowProblem?problemid=d539>
- <https://66lemon66.blogspot.com/2021/01/zerojudge-d539-max-c.html>
- <https://tioj.ck.tp.edu.tw/problems/1224>
- <http://cbdcoding.blogspot.com/2015/02/tioj-1224hoj-12.html>
- <https://zhuanlan.zhihu.com/p/103616664>
- <http://pisces.ck.tp.edu.tw/~peng/index.php?action=showfile&file=f220f2d6b33ae091978ebf59d2af5908bc8190b51>
- https://www.youtube.com/watch?v=GLuT4zKzdic&t=1279s&ab_channel=sprout-tw
- https://www.youtube.com/watch?v=5DAIKf61xLs&ab_channel=sprout-tw



感謝聆聽!