Chapter 9 線段樹與相關結構（Segment Tree）

●Section 1 概論（Introduction）

線段樹(segment tree, 或 **binary indexed tree**, 或 **Fenwick tree)**是一種處理對於區間的動態修改與詢問的資料結構，也可以不完全的取代平衡二分搜尋樹。在競賽上這個資料結構是十分有用的，一方面是因為線段樹可以有許多的變化，所以其實經常會遇到可以利用線段樹的題目；另一方面，線段樹的實做也是相當簡單的，又可以部分取代相當複雜的平衡二分搜尋樹。底下我們首先先定義一個可以使用線段樹來解決的問題，我們將先解決這個問題，再將其想法推廣到其他問題。

動態 Range Minimum Query(RMQ)問題：

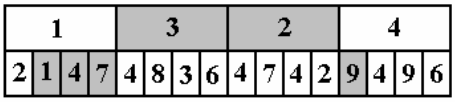
給定一個序列共 n 個數字，現在有兩個操作：一是 $C\ i\ a$，將第 i 個位置的數字改變成 a；另一是 $Q\ i\ j$，詢問區間 $[i,j)$ 內數字的最小值。希望能動態維持這 n 個數字，使得這兩個操作都能在 O(lgn)完成。

§1-1 塊狀鍊表

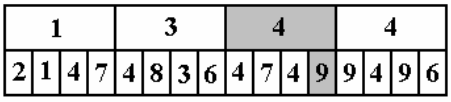
對於這個問題，一個直接的做法是每次修改時直接 O(1)將該位置的數字改變，但這樣子每次詢問就需要 O(n)的時間掃過 $[i,j)$ 內所有數字，不符合我們的要求。我們希望能多維持一些資料，使我們詢問時能夠更加迅速，這時就出現了一個有趣的**塊狀鍊表**的想法：

將這 n 個數字每連續 $\sqrt{n}$ 個包成一段，總共 $\sqrt{n}$ 段，而我們除了維持這 n 個數字的值之外，額外維持**每一段數字的最小值**(如下圖)。這樣我們在詢問 $[i,j)$ 內數字的最小值時，只需要考慮 $[i,j)$ 內每一段的最小值(至多 $\sqrt{n}$ 段)、以及剩下未滿一段的值(前後各至多 $\sqrt{n}$ 個)，這 $3\sqrt{n}$ 個數的最小值即為 $[i,j)$ 的最小值，即可在 O($\sqrt{n}$)時間完成詢問操作。

而對於改變操作，除了改變那個數之外，另一個有可能改變的數字是該數所屬段之最小值，如果該操作僅將該數改小，則可以直接更新；但如果將該數改大，則該段最小值有可能變為該段中任何一數，即需要 O($\sqrt{n}$)時間重新掃過整段求最小值。其實這個兩邊各妥協一些來換取整體效率改善的做法在很多地方都可以看到，而這個結構將原來 O(1)-O(n)的操作改進成了 O($\sqrt{n}$)-O($\sqrt{n}$)，大幅改善了效率。

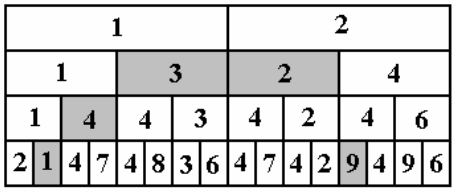


詢問[2,14)時所需考慮的數

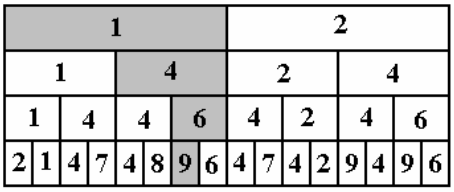


將位置 12 的 2 改成 9，需重算該段最小值

這樣的結構將原本一層的陣列改包成了兩層而加速，使我們想到如果包超過兩層會怎樣呢？如果我們每 $n^{1/3}$ 包成一組共 $n^{2/3}$ 組，每 $n^{1/3}$ 組再包成一段共 $n^{1/3}$ 段，那麼這兩個操作又可加速成 O($n^{1/3}$)-O($n^{1/3}$)。當然我們也不會滿足於這個複雜度，所以當我們一直增加層數，達到 lgn 層時，線段樹就出現了。



詢問[2,14)時所需考慮的數



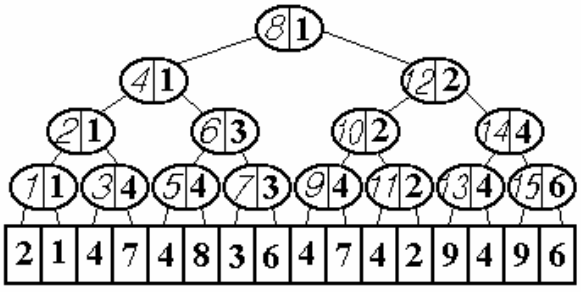
將位置 7 的 2 改成 3，需重新計算的值

觀察線段樹的結構，我們可以發現它的一個性質：每一組都是恰好包著另外兩組，也就是說如果我們把每一組當成一個節點，那麼它的結構就是一棵**完全二元樹**。而每次改變一個數字時，所有包含它的段落(即在二元樹中所有祖先)共 *lgn* 個要重新計算(在其兩個孩子中取較小的，僅需常數時間)，故改變的時間複雜度是 O(*lgn*)。而在詢問操作中每層要考慮的至多兩段(前後各一段，想想看，為什麼？)，故僅需考慮 2*lgn* 段，其複雜度亦為 O(*lgn*)，即這個資料結構符合我們問題的要求。

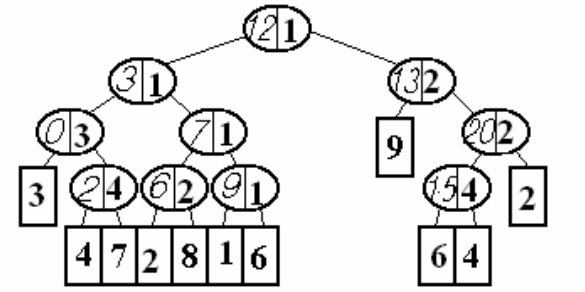
§1-2 二元搜尋樹

線段樹的基本結構如前述是一棵二元樹，這不禁讓我們想把它跟一個我們之前學過的結構：**二元搜尋樹**做一個比較。其實線段樹的想法跟二分搜尋樹是有所關聯的，我們考慮一棵鍵值為 1~*n*-1 之二元搜尋樹(左下圖)，而每個節點另外再存一個 *min* 值紀錄該節點之下子樹中所有數字(不是鍵值)的最小值，且每個葉子的兩個孩子沒有鍵值(即原來二元搜尋樹之 *nil* 指標)，其 *min* 值為我們所要維持的陣列對應位置的值。

則這個結構的意義與線段樹是相同的，每次改變時只要重新計算該葉子到根路徑上所有節點的 *min* 值；詢問操作時由根節點開始考慮，如果該詢問區間完整覆蓋了某一棵子樹，則可直接考慮該子樹之 *min* 值，否則就往兩個子節點遞迴下去，則這兩個操作的複雜度都是 O(*h*)(*h* 是該二元搜尋樹的深度)，值得注意的是這兩個操作並沒有要求這棵二元搜尋樹是完全平衡的，也就是說我們可以使用任意的一棵二元搜尋樹來維持一樣的性質(右下圖)。則這個結構支持二元搜尋樹的相關操作(插入、刪除、旋轉等)，以及改變與詢問這兩個操作。如果我們對二元搜尋樹做適當的平衡，使 *h*=O(*lgn*)，則所有操作都能在 O(*lgn*)完成。



圖中斜體字為二元搜尋樹之鍵值，
粗體字為該子樹對應數值之最小值



一個任意的二元搜尋樹的例子

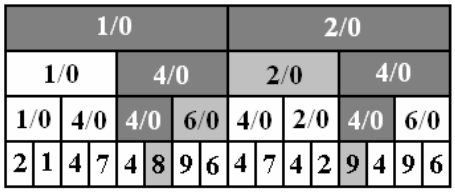
值得注意的是：**任意可以用線段樹達成的操作都可以用平衡二元搜尋樹達成**；而在某些方面，線段樹也可以拿來替代二元搜尋樹。如果我們確定所有二元搜尋樹操作的數字範圍在 1~*w* 之間，那麼我們可以對於 1~*w* 這 *w* 個數建立線段樹紀錄每個數是否出現，那麼我們可以以 *lgw* 的時間完成二元搜尋樹的各項操作，某種程度的取代平衡二元搜尋樹(*lgn* 完成各項操作，*n* 為元素個數)。也可以將線段樹視為一種**利用小範圍整數特性而預先平衡**的二元搜尋樹，這些關係值得大家在對這些結構更加熟悉後再細細體會。

§1-3 改變操作的一個推廣

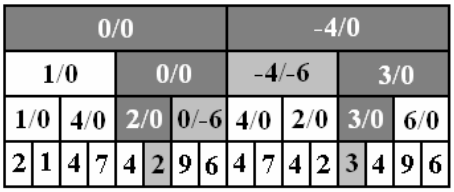
上兩節我們看到了線段樹的兩個基本操作：改變一個數或著詢問某一段的最小值。其實一般來講線段樹的所支援的操作是更廣泛的，還包含**改變一段數字**這個操作。而這個操作有兩種可能：(a)將一段數字增減一個數、(b)將一段數字全部改成一個數。底下就這兩種操作分別討論(不考慮同時出現這兩個操作)。

如果我們想要支援詢問某段最小值與將一段數字增減一個數這兩個操作，假設我們直接使用先前線段樹的結構，那麼可以在 O(*lgn*)詢問某段最小值；但是當我們將某一段數字增減時，我們必須要改變所有有被這段蓋到的線段樹節點所存的值，總共有 O(*n*)段，即我們需要至少 θ(*n*)的時間才能維持這個結構，不符合我們的要求。

我們希望能夠不要改那麼多東西，如果我們改變某一個節點的 *chg* 值可以讓他所有的子孫的值增加 *chg*，那麼我們就不需要去改變它的所有子孫，則我們要改變的就只有(1)被它完整覆蓋的極大段落(即詢問該段時會考慮到的段落，至多 O(*lgn*)個) (2)所有(1)的祖先(亦只有 O(*lgn*)個，想想看為什麼？)。其中(1)的部分要改變 *chg* 值；而(2)則是要改變 *min* 值(如下圖)。



每個節點上數字為 *min/chg*，
改變[6,14)時，淺灰為(1)，深灰為(2)



將[6,14)所有數字減 6 的結果

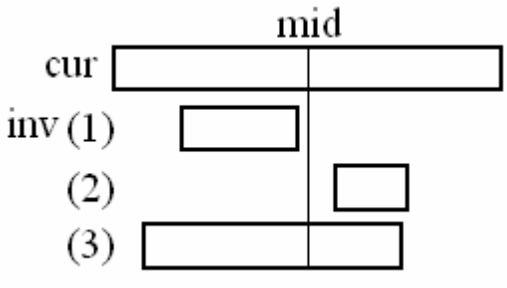
那在這樣的維持下，要怎麼完成詢問操作呢？詢問操作中我們依然對同樣的 O(*lgn*)段求最小值，只是要注意的是這裡的最小值不一定是真正的值，因為有可能在它的某個祖先中，其 *chg* 值不為 0，即所有它的子孫的值都要改變 *chg*。故我們在求最小值時，要另外計算其**所有祖先的 *chg* 值總和**，在加到該段 *min* 值上才是該段實際上的 *min* 值，再對這 O(*lgn*)個實際 *min* 值求最小即可，亦能在 O(*lgn*)時間完成，即解決了(a)部分的問題。

對於(b)部分的問題，其實解決方法也跟(a)類似，我們可以同樣的多紀錄一個 *same* 值，如果 *same≠NULL* 則代表該節點以下所有值均為 *same*，則我們一樣的可以改變(1)的 *same* 值及(2)的 *min* 值。但是這樣會有一個問題：如果我們先將[1,5)改成 1，再將[1,2)改成 2，但這時候我們節點[1,5)的 *same* 值會是 1，即我們會以為[1,5)以下全都是 1，就發生錯誤了。

解決這個錯誤的方法如下，當我們要在一個 *same=a≠NULL* 的區間下改變值的時候(即(2)中有一個區間的 *same* 值不是 *NULL* 時)，那我們就先把它的 *same* 值改成 *NULL* 並把它的兩個孩子的 *same* 值改成 *a*，即將這個區間**拆成兩半**，再繼續改變的動作。這樣子的話就可以維持沒有某兩個 *same* 值非 *NULL* 之節點其一為另一個的祖先，即不會有先前這種矛盾的狀況。

●Section 2 線段樹的實作(Implementation)

由於 segment tree 是一棵二元樹，且其深度不會太深，故我們可以用與先前的 heap 相似的存法：開一個夠大的陣列 S(若要維持 n 個元素，則其大小至少要開 2^k 使 $2^k \geq 2n$，想想看為什麼？)，則可將根(區間[1,n+1))存在 $S[1]$，$S[i]$(區間[L,R))的兩個孩子(區間$[L, \frac{L+R}{2})$、$[\frac{L+R}{2}, R)$)分別存在 $S[2i]$、$S[2i+1]$，用遞回的方式完成這兩個操作。提供簡易的虛擬碼如下(對於上一節的(a))：



改變及詢問的 case1~3

```
SEG_UPDATE (NODE pos, INTERVAL cur, INTERVAL inv, INT value)
1  if inv isn't inside cur                          ▷ 例外處理，非必要
2      then return
3  if inv = cur
4      then pos.min  ← pos.min+ value
5           pos.chg  ← pos.chg+ value
6           return
7  INT mid  ← ( cur.left + cur.right ) / 2
8  if inv.right  ≤  mid                             ▷ Case 1
9      Seg_Update( pos.leftchild   , {cur.left, mid}     , {inv.left, inv.right} , value )
10 else if inv.right  ≥  mid                        ▷ Case 2
11     Seg_Update( pos.rightchild , {mid, cur.right}  , {inv.left, inv.right} , value )
12 else                                             ▷ Case 3
13     Seg_Update( pos.leftchild   , {cur.left, mid}     , {inv.left, mid}       , value )
14     Seg_Update( pos.rightchild , {mid, cur.right}  , {mid, inv.right}      , value )
15 pos.min  ← min( pos.leftchild.min, pos.rightchild.min) + pos.chg
16 return
```

```
SEG_ QUERY (NODE pos, INTERVAL cur, INTERVAL inv, INT tot)
1  if inv isn't inside cur                          ▷ 例外處理，非必要
2      then return  ∞
3  if inv = cur
4      then return pos.min
5  INT mid  ← ( cur.left + cur.right ) / 2, ret  ← ∞
6  if inv.right  ≤  mid                             ▷ Case 1
7      ret  ← Seg_Query( pos.leftchild   , {cur.left, mid}     , {inv.left, inv.right} , tot+pos.chg )
8  else if inv.right  ≥  mid                        ▷ Case 2
9      ret  ← Seg_Query( pos.rightchild , {mid, cur.right}  , {inv.left, inv.right} , tot+pos.chg )
10 else                                             ▷ Case 3
11     ret  ← Seg_Query( pos.leftchild   , {cur.left, mid}     , {inv.left, inv.right} , tot+pos.chg )
12     ret <?= Seg_Query( pos.rightchild , {mid, cur.right}  , {inv.left, inv.right} , tot+pos.chg )
13 return ret
```

§2-1 動態建立線段樹

有時候我們的數字範圍 w 太大，使我們的時空皆無法負荷該大小的整棵線段樹，經常我們可以使用離散化的技巧，將數字範圍縮減至所有操作中牽涉到的數字數量，即可解決。但是偶爾會有題目給的數字無法離散化的狀況(即無法在操作前先得知所有操作的資訊)，那要怎麼辦呢?

這時我們可以使用動態配置的方式來建立線段樹，即只有當 Update 操作改到某個節點的值時才動態配置出該節點，這時候就必須要實際儲存每個節點的左子節點及右子節點，而不能直接使用陣列索引值的 $2i$ 及 $2i+1$。詢問時如果問到了尚未被配置的節點，則其及其底下的所有節點的值必均尚未改變，即可直接用初始值算出對於詢問的回答。此時所需的總時間及空間複雜度皆為 O($nlgw$) (n 為操作數)。

*** 線段樹相關例題：**

題一《線段問題》
給定平面上的一些線段，問一條 $x=a$ 的垂直線 L 會跟幾條線段相交？

題二《垂直線段相交問題》
給定很多垂直線段以及水平線段，請問有幾個交點？

題三《矩形覆蓋面積計算》(TIOJ1224)
平面上有 n 個矩形(n≦100000)，其邊都與座標軸垂直或平行，請問其覆蓋的總面積。

題四《矩形覆蓋周長計算》
平面上有 n 個矩形，其邊都與座標軸垂直或平行，請問其覆蓋區域的總周長。

題五《約瑟問題》(TIOJ1382)
有 n 個人圍成圓圈(n≦100000)，另外給一個序列 A，從編號 1 開始數，第 i 次數 A_i 個，被數到的人就離開圓圈並向後繼續數，請求出 n 個人離開圓圈的順序。

題六《Count color》(PKUOJ2777)
給定一條線段[1,L]，有兩種操作：將[a,b]塗上 c 色、詢問[a,b]有幾種顏色，請依序回答所有詢問(L≦100000，顏色數≦30，操作數≦100000)。

題七《The Agency》(TIOJ1272)
給定一棵有根樹，一開始所有點都是白色的，有兩種操作：將以一個節點 i 為根的子樹全部變色、詢問某個節點的顏色。請依序回答所有詢問。(點數, 操作數≦100000)

題八《A Simple Problem with Integers》(PKUOJ3468)
給定一排 n 個數字，有兩種操作：把第 i 個到第 j 個數加 c、詢問第 i 個數到第 j 個數總和。(n, 操作數≦100000)

題九《Stars in Your Window》(PKUOJ2482)
平面上有 n 個星星(n≦10000)，給定一個長 w 寬 h 的框框，框框的邊必須與座標軸平行，請問框框在不同位置可以圍住星星數量的最大值。

題十《動態連續和問題》
給定一排 n 個數字，請實作兩個操作：將第 i 個到第 j 個數全部改成 C、詢問這 n 個數字的最大連續和。

題十一《我很忙》(TIOJ1408)
一個很忙的條件是在時間$[A,B]$內至少有 C 分鐘是忙的，給定 n 個這樣的條件，至少要忙碌多久才能使所有條件都被滿足？(所有數字≦100,000)

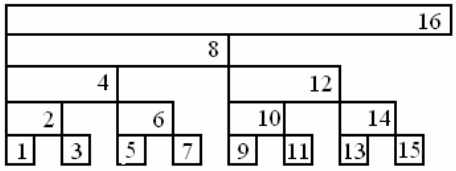
題十二《Sails》(IOI07, TIOJ1343)
一艘船共有 n 個船桅，每個船桅有高度 h 及所要掛的船帆數 w，每個船帆的高度為 1 且必須掛在船桅的整數高度上，同一個船桅的每個高度只能掛一面帆。定義船的 *inefficiency* 是不同船桅間的同高度船帆數總和，給定所有船桅的 h 跟 w，請問最小的 *inefficiency* 是多少？(所有數字≦100,000)

題十三《Salesman》(IOI09)
一個銷售員住在河流旁的位置 H，他想坐船去參加許多會議，每個會議有舉行時間 t 跟河流旁的位置 r、以及參加所的的利益 p (60%的測資中 t 不重複，r 一定不重複)。而在河流往上游開船每公里需要 U 的油錢，往下則需要 D。他希望從 H 開始，參加某些會議之後回到 H，請問最多能賺多少錢 (需扣除所花油錢)？(所有數字不超過 500001)

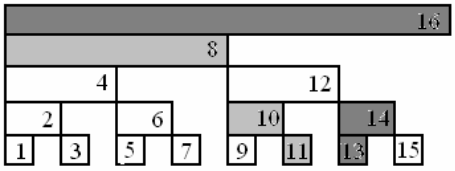
●Section 3 線段樹的變化 (Variation)

§3-1 樹狀數組

樹狀數組是一種對於線段樹的簡化，如果說對於某棵線段樹的操作滿足以下兩個性質：(1)其改變操作皆是改變一個值，(2)其詢問操作皆是詢問一個**從頭開始的區間**$[1,i]$(若該線段樹所詢問的性質具有可排容性，例如求區間的和，則可由$[1,j-1]$及$[1,i-1]$的詢問結果推出$[i,j]$的結果，故此情況可支援任意區間的詢問)。則可將其刪去一半的節點簡化為樹狀數組(如下圖)，只需維持一半的區間，則可以發現此時所維持的**所有區間之後端點皆相異**，故我們可以以後端點的位置來表示一個區間，則可直接以一個索引值為 1~n 的陣列來維持所有區間的結果。



一個樹狀數組所需維持的區間示意圖



淺灰色為詢問[1,11]時所需考慮的值
深灰色為改變 13 之值時所需改變之值

則如果 k 是奇數，那編號 k 所維持的範圍即$(k-1,k]$(注意這裡為了便利改變了區間的開閉)；而如果 k 除 4 的餘數是 2，則其維持的範圍為$(k-2,k]$。不難歸納出對於任意的 k，其所維持的範圍是 $(k-2^i,k]$，其中 i 是最小的 i *使得* $2^i \mid k$ 且 2^{i+1} 不是 k 的因數，也就是 k 從**最低位數來的第一個非 0 位**，稱為 ***k* 的 lowbit**。我們有一個很有趣的計算這個值的方法：$\mathbf{2^i=k\&(-k)}$ (想想看為什麼？)，也就是可以在 O(1)算出其 lowbit 值。

既然我們可以算出 lowbit 值，那我們要怎樣完成樹狀數組的兩個操作呢？對於[1,i]的詢問操作，其回答即(i-lowbit(i),i]區間所維護的值加上[1,i-lowbit(i)]的詢問結果，重複操作即可在 O(lgn)內完成回答。而對於改變操作，我們需要改變所有包含 i 的區間，經過一些觀察後我們可以發現其所有要改變的區間是編號 $i_0=i$, $i_1=i_0+\text{lowbit}(i_0)$, ..., $i_{k+1}=i_k+\text{lowbit}(i_k)$, ... (想想看為什麼？)，又編號超過 n 的區間不需維持，則亦可在 O(lgn)內完成，提供虛擬碼如下：

```
UPDATE (INT pos, INT value)
1  ▷ 將位於 pos 之值增減 value
2  while pos ≦ n
3      then seg[pos] ← seg[pos]+ value
4           pos      ← pos + (pos&(-pos))
5  return
```

```
QUERY (INT end)
1  ▷ 詢問[1,end]之和
2  INT ret ← 0
3  while end > 0
4      then ret  ← ret + seg[end]
5           end  ← end - (end&(-end))
6  return ret
```

樹狀數組大幅的簡化了線段樹，儘管其用途較為狹隘，仍不失為一個有趣的想法。

§3-2 線段樹上的二分搜尋

如果要對於線段樹或樹狀數組所維持的性質作二分搜尋，譬如求第一個[1,i]的和超過 K 的 i(前提是區間中所有數都是正數，如此才能二分搜尋)。直接作的方法是在二分搜的過程中，每次都花 O(lgn)去詢問線段樹該區間的值，則總時間複雜度 O(lg^2n)。但是其實有一個有趣的加速方法，以下以樹狀數組為例來介紹這種方法：

我們不使用原來的二分搜尋順序，而使用基於樹狀數組結構的順序來做詢問：我們首先詢問[1,2^j]的值，其中 $j=\lfloor \lg n \rfloor$，這次詢問只需要 O(1)的時間(因為 2^j-lowbit(2^j)=0，即 2^i 在樹狀數組的第一層)；如果其值不超過 K，則 i 必定發生在 2^j 之後，則問題變成在後半段找第一個 i 使得[2^j+1,i]之和超過 K-Sum[1~2^j]，又可注意到樹狀數組後半段之結構亦是一個樹狀數組，故可以重複進行下去；如果其值超過 K，則 i 必定在 2^j 之前，故問題變為在前半段中找 i 使其和超過 K，而前半段的結構亦是一個樹狀數組，亦可繼續遞迴。如此每一步都只要花 O(1)的時間，故可以在 O(lgn)的時間內求出答案。

這個方法的重點是將區間二分搜的順序改成了**由樹狀數組的根開始往下走的順序**，則每一次詢問的值皆可由之前的詢問加上該節點的值在 O(1)求出。同樣的方法可以應用在線段樹或著一般的二元搜尋樹上，相當實用。

題一《啊嘶起啦集合》(TIOJ1305)
實作這三個操作：在集合中插入一個數字、在集合中刪除一個數字、找出集合中第 k 小的數。(操作數≦100000)

題二《Hiring》(IOI09) (其實這題可以不用這份講義的東西啦 XD)
有 n 個員工來應徵工作(n≦500000)，每個人有其最低薪資 S 及其能力等級 Q。為了公平起見，所有被雇用的員工的薪資(必超過其最低薪資)必須跟其能力等級成正比。你現在有 W 的錢，希望在能雇到最多員工的狀態下花最少的錢，則要雇哪些人呢？

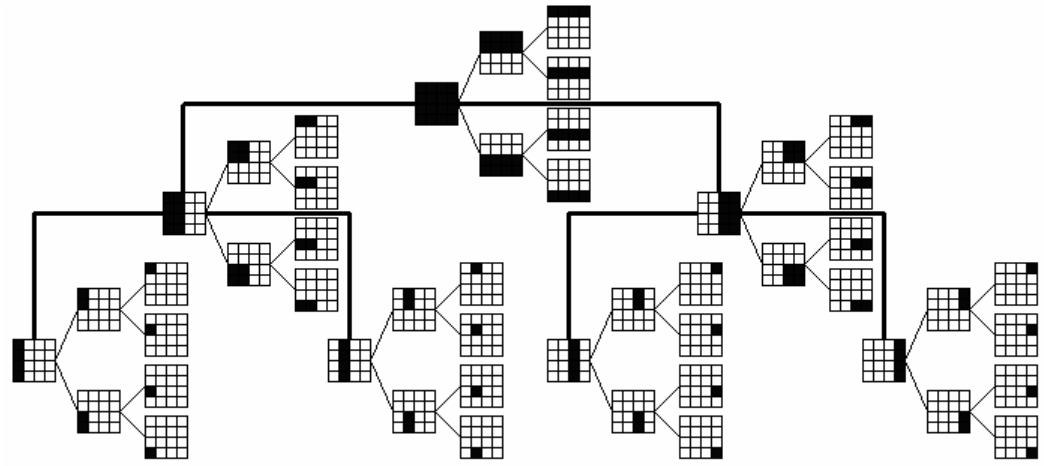
§3-3 高維線段樹

有時候我們要處理的詢問或改變不僅限於區間，而是一個二維上的長方形$[a,b)\times[c,d)$，或是更高維度，這時候我們就會需要推廣到高維的線段樹或樹狀數組。底下討論二維的情形(更高維度方法類似，但不常見)，這裡先介紹如何處理比較簡單的問題(即在兩個操作的其中一個中，每次牽涉到的是一個位置而不是一個區間)：

對於樹狀數組，改變必僅限於一個位置。不難想像我們的處理方式如下：建立一個二維陣列 S，其中 $S[i][j]$維護$(i\text{-lowbit}(i), i]\times(j\text{-lowbit}(j), j]$區塊之值。則對於改變某個位置$(i,j)$之值時，設 $i_0,i_1,\ldots$及 $j_0,j_1,\ldots$之記號如上節，則所有需要改的區塊為 $S[i_0][j_0], S[i_0][j_1], \ldots, S[i_1][j_0], S[i_1][j_1], \ldots, \ldots$等共 $O(lg^2n)$個；詢問之處理方法類似，亦可由 $O(lg^2n)$個區塊中推出$[1,i]\times[1,j]$之值。這兩個操作可以簡易的用兩層迴圈來實現，複雜度皆為 $O(lg^2n)$。(註：此處所有 lg^2n 皆代表$(lgn)^2$，非 $lglgn$)

二維樹狀數組的此種處理方式類似**樹套樹**的想法，即利用一棵樹狀數組代表所有列(即二維陣列的第一維)，且每個節點階是一棵樹狀數組，代表將該點所紀錄的各列對於行建立樹狀數組。將同樣的想法使用到二維線段樹，即對於第一維建立線段樹，而線段樹的每個節點都是一棵線段樹來紀錄第二維的資訊(如下圖)。

對於二維線段樹的兩個操作(詢問或修改$[a,b)\times[c,d)$)，其實作方法即用之前的方法先在第一維對$[a,b)$作操作，而在操作過程中若要詢問或修改一個節點(其實是一棵第二維的線段樹)的值時，即在該樹對$[c,d)$用 $O(lgn)$作相應的詢問或修改即可，則第一維將對至多 $O(lgn)$個節點作詢問或修改，即總複雜度 $O(lg^2n)$。



那麼為什麼我們要限制說其中一項操作必須只針對一個位置呢？這是因為在二維線段樹所維持的所有區塊中可能會有**兩個互不包含且彼此相交的區塊 *I*, *J***，這在一維的狀況下不會發生。如果我們允許兩個操作皆牽涉到區塊，那麼我們就無法確認 I 中所存的資訊會不會在詢問 J 時造成影響，則我們就不可以在改變 I 時只將改變的資訊存在 I，還要存在所有跟 I 相交的區塊，這種區塊可能有相當多。

其實對於這個一般的問題講師個人不知道一個完整的答案(搞不好其實是不能用二維線段樹作 XD)，底下對於一個特殊的狀況提出一種做法：

問題《Tetris 3D》(POI13th)

在 $n*m$ 的網格上進行 k 次操作($n,m\leq 1000$, $k\leq 20000$)，其中每次操作是把一個長方體丟在一個位置(給高度及所丟位置，位置必定在網格上且邊必與座標軸平行)，如果撞到之前丟的物體即停止移動(不會有任何轉動)，詢問說在 k 次操作結束後，所有方塊的最高點的高度為何？

我們使用一棵二維線段樹來紀錄，其中每次操作即在該矩形區域求高度最大值 x，並且將該矩形區域之高度全部改成 $x+h$，h 為長方體之高。最後輸出整塊的高度最大值即可。那麼這個問題有什麼可供利用的性質呢？這裡所用到的特殊性質是**高度只會越來越高**。

對於每個二維線段樹的節點，我們紀錄四個值：$v_{00},v_{01},v_{10},v_{11}$，其中兩個足標分別代表了兩軸，1 跟 0 則代表了該軸是否被完全覆蓋(後詳)。接著是詢問的操作，詢問時如果你所要詢問的區塊完整覆蓋了目前的區塊，則考慮目前區塊的 v_{11}(兩軸皆完全覆蓋)；若 x 軸完全覆蓋而 y 軸部分覆蓋，則考慮其 v_{10}；其餘兩種狀況類似。而詢問的做法即從 x 軸的根開始走，如果$[a,b]$完整覆蓋了目前的節點所對應的區間，對該節點 y 軸之線段樹對$[c,d]$作詢問 1，不須對 x 軸繼續遞迴；否則則先對該節點 y 軸之線段樹對$[c,d]$作詢問 0，接著對 x 軸遞迴下去。而詢問 i(0 或 1)的做法亦是從該節點的根開始走，如果$[c,d]$完整覆蓋了目前區間，則以 v_{i1} 更新回答；否則則以 v_{i0} 更新回答並遞迴下去。則所有考慮的區塊即詢問$[a,b]$與詢問$[c,d]$之所走過的區間相乘，故兩維各需詢問 $O(lgn)$個區間，相乘共 $O(lg^2n)$個區塊，即在 $O(lg^2n)$時間內完成回答。

對於改變操作所有需更新的區塊與詢問操作相同，僅需用目前的值將所有需更新的區塊的某些值作更新，只是更新的值不同(若改變區塊完整覆蓋目前區塊，則四個都要改；若 x 軸完整覆蓋 y 軸部分覆蓋，則更新 v_{01},v_{11}；若 x 軸部分覆蓋 y 軸完整覆蓋，則更新 v_{10},v_{11}；若皆部分覆蓋，則只更新 v_{11})。亦可在 $O(lg^2n)$時間內完成。

這個方法中所用到的主要關鍵是對一個節點紀錄的那四個值，利用那四個值來分辨之前的改變對詢問區塊的影響(如果改變區塊 K(將其內之值全改成 w)完整覆蓋目前區塊 I，那麼詢問區塊 J 只要跟 I 有交集，其回答就至少 w(因為之後只會變大不會變小)；如果 K 在 x 軸方向完全覆蓋 I 而在 y 軸上部分覆蓋，那如果 J 在 y 軸上完全覆蓋 I 而 x 軸部分覆蓋，J,K 必有交集，即回答至少 w，其餘兩種狀況類似)。另一個關鍵是每個值都只會越來越大，因為過程中可能某個值已經全部被其他值取代，但在線段樹中還是存有這個值(例如一個同一個數的大矩形被很多個小矩形蓋過)，這時候在詢問時還是會考慮到這個值，只是一定還會考慮到其他更大的值，才能維持算法的正確性。

*** 二維線段樹相關例題：**

題一《保加利亞的俄羅斯娃娃》(TIOJ1266)
給定一個 $n*n$ 的數字表(n≦1000)，請求其二維 LIS 長度(即最長的數列，使其兩個座標及所對應的數值皆(非嚴格)遞增)。

題二《壞掉的小畫家 The broken painter》
小畫家壞掉了，現在只能拿來畫矩形(與座標軸平行)，而且每畫出一個矩形時其內部的顏色都將變成其互補色(白變黑黑變白)，請實作兩個操作：畫矩形、詢問(x,y)顏色。

題三《Mobile phones》(PKUOJ1195)
給一個一開始數字全為 0 的 $S*S$ 表格，實作兩個操作：改變一個數字、詢問一個矩形內的數字和。(S≦1024, 操作數≦60002)

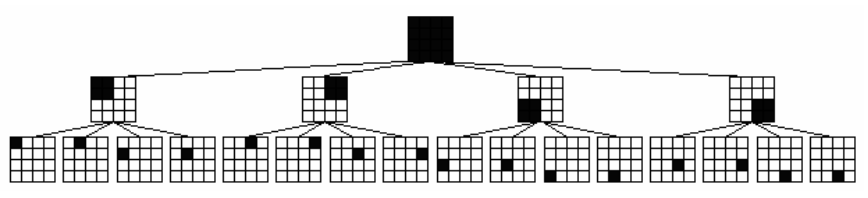
題四《Census》(UVA11297)
給一個 $n*n$ 的表格，實作兩個操作：改變一個數字、詢問一個矩形內的最大值及最小值。(n≦500, 操作數≦40000)

題五《卡嗯問題(誤)》(TOI09 2! 二模第三題)
有 n 個依序出現的卡嗯想排成一列隊伍，每個卡嗯有上嗯值 a_i 跟下嗯值 b_i，兩個值可以互相交換。每次一個卡嗯出現時，他可以選擇(1)跑去電人(2)排在隊伍最前面(3)排在隊伍最後面。而這個隊伍必須要滿足嚴格遞增的關係，即前面卡嗯的 a_i 要小於後面卡嗯的 a_j 且前面卡嗯的 b_i 要小於後面卡嗯的 b_j，問卡嗯們最多可以排多長？(n≦1000000)

§3-4 四分樹

既然二維線段樹的問題是會有互不包含且彼此相交的區塊，那我們如果不要維持那麼多的區塊來避免這個情況，會發生什麼事呢？於是就有了四分樹這個結構。

四分樹的結構如左下圖所示，每個節點有四個子節點，其區塊即目前區塊分成四塊，直到無法分解(葉節點)為止。這樣它就可以維持一維線段樹的性質：對於一個區塊，所有完全包含它的區塊皆是它祖先；反之所有完全被它包含的區塊皆是它子孫。有了這個性質之後就可以實現上一節中所無法解決的，對任意區塊的改變與詢問了。



關於改變與詢問一個區塊的操作，我們就一樣可以沿著四分數由根節點開始走，每次將我們的詢問區塊最多切成四塊並遞回下去，右上圖為一個分割的例子。

但在這種切割方法下，四邊經常會出現很多碎塊。仔細分析分割塊數之後會發現很不幸的，這種分割方法所產生的塊數是 $\theta(n)$的，即**在四分樹上的兩個操作的複雜度是 $\theta(n)$**，並非二維線段樹的 $O(lg^2 n)$，所需時間高了不少。但與樸素的實作比起來(兩個操作皆 $\theta(n^2)$)或是 n 棵線段樹(兩個操作皆 $\theta(nlgn)$)，仍是一種改進。