2019全國資訊學科能力競賽 解說（NHSPC2019 Editorial）

A.音檔剪輯

時間複雜度 O(n)，記憶體 O(n)

把輸入讀入以後，每一次可以選最右邊且距離不超過 t 的點當下一個剪輯點。

時間複雜度 O(n)，記憶體 O(1)

每次讀入一個數字時，保存下列的變數:

- 上次選的剪輯點
- 上一個數字
- 目前數字

當目前數字跟上次剪輯點距離超過 t 時就可以選上一個數字當作下一個剪輯點，可以做到記憶體 O(1)。

B.完全平方二項係數

步驟1: 式子的導出

- $\frac{m(m-1)}{2} = n^2$
- $m(m-1) = 2n^2$
- $4m^2 - 4m - 8n^2 = 0$ (同乘4)
- $(2m-1)^2 - 2(2n)^2 = 1$ (配方)
- 令 $x = 2m-1, y = 2n$，可得 $x^2 - 2y^2 = 1$

步驟2: 求出 x^2 - 2y^2 = 1 的第 n 組解

首先大前提，我們得需要知道 $x^2 - 2y^2 = 1$ 的正整數解以及 $\binom{m}{2} = n^2$ 的正整數解恰好是一一對應的。這個性質上面步驟一的最後一條式子並不難得出(簡單證明: x 必為奇數，且 x,y 皆為正整數時對應到的 m,n 也都是正整數，反之也是)。有了這個性質的前提，我們可以改求 $x^2 - 2y^2 = 1$ 的第 k 組解來反推 $\binom{m}{2} = n^2$ 的第 k 組解。

由範測輸出的 $(m_1, n_1) = (2, 1)$ 可以得出 $x^2 - 2y^2 = 1$ 的第一組正整數解為 $(x_1, y_1) = (2 \times 2 - 1, 2 \times 2) = (3, 2)$。

至於如何快速求出第 n 組解，這邊提供兩個常見的做法

1. 快速冪: 已知 $x_k + y_k\sqrt{2} = (3 + 2\sqrt{2})^k$，因此重點在於如何求出右式的 n 次方。求法和一般整數時的快速冪差不多，只是每一步驟需要維護兩個數字 - 常數項和 $\sqrt{2}$ 的係數。
2. 矩陣乘法:

$$\begin{bmatrix} 3 & 4 \\ 2 & 3 \end{bmatrix}^{k-1} \times \begin{bmatrix} x_1 \\ y_1 \end{bmatrix} = \begin{bmatrix} x_k \\ y_k \end{bmatrix}$$

上面兩個做法都可以在 O(log n) 的時間求出第 n 項。

步驟3: 反推出 n,m 的解

假設已求出 x_k, y_k，可以用步驟 1 最下面的式子反求:

$m_k = \frac{x_k+1}{2} = (x_k + 1) \times 500000004 (\mod 10^9 + 7)$

$n_k = \frac{y_k}{2} = y_k \times 500000004 (\mod 10^9 + 7)$

C.尋寶之旅

(假設寶石的顏色數量是 m)

子問題1

寶石編號一樣，則這個問題等價於求整棵樹的深度，可以 dfs 或 bfs 在 O(n) 的時間求出。

子問題2

與子問題1相同，針對每種顏色分開去求對應到顏色的最長路，可以在 O(nm) 的時間求出。

子問題3,4

首先，寶石的顏色編號雖然很大，但是可以發現最多不會超過 n 種不同的顏色。所以在一開始可以將所有顏色離散化將顏色的編號 map 到編號 1~n 之間。

再來，可以考慮用下列的 dfs 算法來算出答案:

```
def dfs (node):
  set.insertOne(color[child]) # insert color to multiset
```

```
  for child in childs[node]:
    dfs(child)

  answer = max(answer, set.maxElementCount())
  set.removeOne(color[child]) # remove one color from multiset
```

這裡提供三種 set 的實作方法:

1. 如果用平衡樹或線段樹的話，上面整體複雜度會是 O(n log m)
2. 考慮到 insert 或 remove 操作每次最多只會把某個數字的 count +1 或 -1，如果用 array 紀錄每個元素的 count 的話可以均攤 O(n) 維護 count 最大的元素在哪
3. 可以發現只需要在某個 color insert 的時候才需要更新 answer 的值，因此連最大的元素在哪都不需要維護也可以(i.e. 上面虛擬碼的 set.maxElementCount 可以不用實作)

這三種作法都可以快速得出答案，做法 3 的實作比較簡單並且沒有 log，所以在這邊提供作法 3 的虛擬碼:

```
def dfs (node):
  set.insertOne(color[child]) # insert color to multiset
  answer = max(answer, set.count(color[child]))

  for child in childs[node]:
    dfs(child)

  set.removeOne(color[child]) # remove one color from multiset
```

整體的複雜度

1. 離散化: O(n) (因為編號順序不影響，可以不需要排序原本顏色編號)
2. dfs: O(n+m) = O(n)

D.獵人與斯芬克斯

一維版本

考慮這一題的一維版本: 給一數列，如何求出最大又不超過 K 的連續和?

這裡提供一個基於前綴和 O(n log n) 的做法:

- 令 F_i = 數列前 i 個數字的和，$F_0 = 0$。則數列任意區間和 [l,r] 都可以寫成兩個數字相減: $F_r - F_{l-1}$
- 從左往右枚舉右界 r，接下來要求出 l 使得 $F_r - F_l \le K$ 且最大
  - 等價求最小且 $\ge F_r - K$ 的 F_l

- 上面的所求可以用 set 維護並用 lower_bound 在 O(log n) 求出，或者要用線段樹或 BIT 離散化也可以

以上可以得到一個在 O(n log n) 時間求出此問題一維版本的作法。

二維版本

考慮一個 x 座標範圍固定在 x_1, x_2 之間的子矩陣，這時候最佳的 y 座標範圍 y_1, y_2 就可以套用上面一維版本的方法做出。

由於 x 上下界共有 $O(m^2)$ 種可能，每次沒舉完上下界需要花 O(n log n) 的時間檢查，時間共為 $O(m^2 n \log n)$。

E.水槽

- 求出 lh[i] = i 左邊下一個比 i 高的板子
- 同樣 rh[i] = i 右邊下一個比 i 高的板子
- lh, rh 可以用 stack 在 O(n) 時間求出

再來定義 next(l,r) 為:

- 若 l.height < r.height: [lh[l], r]
- 否則為 [l, rh[r]]
- 簡單來講，next(l,r) 是當 [l,r] 區間會被水淹滿時，下一個會被淹滿的區間

接下來可以用遞回求出:

- 對於一個注水點 s，[s,s] 為起點找出最大的區間 [l,r] 滿足
  - 目前的水夠填滿 [l,r]
  - 但是不夠填滿 next(l,r)
- 先把 [l,r] 區間的水填到某個高度
- 再遞迴下去填 next(l,r) - [l,r] 的區間

可以在均攤 O(n) 的時間遞迴求出。

F.翻轉與框架區間

觀察

以下面為例，假設紅色為要反轉的區間

2	1	3	5	7	8	6	4	9	10

反轉以後，得到新的框架區間

2	1	3	5	4	6	8	7	9	10

從上面的新框架區間，反回考慮原本翻轉前的位置:

2	1	3	5	7	8	6	4	9	10

以上面為例子，框架區間 (3,6) 可以看成 3,6 兩個數字(即框架區間的最大及最小的數字)往右延伸的兩個區間。再更一般一點，可以得出下列的結論:

對於某次翻轉後得到的新的框架區間 S(x,y)，必定滿足:

1. x,y 是區間最大及最小的數字(順序可以反過來)
2. 考慮新的框架區間在原數列上的位置，只有兩種可能
    - x, y 兩數字往右延伸的兩個 disjoint 區間的連集
    - x, y 兩數字往左延伸的兩個 disjoint 區間的連集

一般的結論

這裡令兩個變數 R 與 S。R 代表某次翻轉的區間，S 代表翻轉過後所生成的某個框架區間(在翻轉後數列的位置)。可以發現 S 與 R 的關係只有下面幾種可能

1. R 與 S 相交，且互不為子區間 (上面例子的情況)
2. R 是 S 的子區間，且 S 恰好包含 R 的左端點或右端點之一

其他的情形都可以發現 S 即使不需要翻轉也是一個框架區間，並不是新增的矛盾，可以不用考慮。由 S 翻轉後的位置去考慮翻轉前的狀況

(紅=區間R，藍=區間S, M=最大or最小數字的位置)

Case 1:

翻轉後:

		M			M				

翻轉前:

		M				M			

Case 2:

翻轉後:

		M			M				

翻轉前:

		M		M					

上面兩個 case 只考慮 R 在 S 右邊的情況，若反過來考慮，則可以得到與上面觀察相當的結論。

O(n^3) 算法

由上面的結論，可以發現只要枚舉下面三個變數

1. 區間最小的數字 i
2. 區間最大的數字 j
3. 兩數字向左或向右延伸

枚舉完方向以後，可以以兩數字為起點 greedy 向左或向右延伸到第一個不在 [i,j] 區間的數字為止。每次枚舉完可以在 O(n) 的時間檢查，時間複雜度為 $O(n^3)$

O(n^2) 算法

(做法應該很多種，這裡僅提供一位驗題者的作法做為參考)

可以多 DP 數個表格:

- mni[i][j]: 原數列上區間 [i,j] 最小的數字
- mxi[i][j]: 原數列上區間 [i,j] 最大的數字
- mnp[i][j]: [i,j] 之間的所有數字裡，最小的座標(原數列的index)
- mxp[i][j]: [i,j] 之間的所有數字裡，最大的座標

延伸原本 $O(n^3)$ 的算法，當枚舉完 i,j 考慮往右或往左延伸到哪時，可以把 mnp[i][j] 或 mxp[i][j] 表格的值拿出來輔助，做出 O(1) 的判斷。

G.隔離採礦

O(n^2) DP

若定義 dp[i] = 最右邊選採礦點 i 且最大的價值總和，可以得到轉移式

$$dp[i] = \max_{j<i且i,j中間有更高的礦井} dp[j] + v[i]$$

對於每個 i，可以由大到小枚舉 j，並維護 i 到 j 之間最高的山 k。這樣狀態一共有 O(n) 個，轉移 O(n)，複雜度 $O(n^2)$。

O(n log n)

與上面一樣從左到右算出 dp[i]，多維護幾個 DP 和幾棵樹來加速轉移:

1. dp1[h]: 高度 = h 的所有座標 x 裡面，最大的 dp[x]
2. dp2[h]: 對於 i 以前某個高度 = h 的油井，左邊所有高度小於 h 油井最大的 dp 值
3. 對於上述兩個表格都用樹維護，使得可以做區間查詢最大值與單點更新

算法如下:

```
for i in range(1,n+1):
  dp[i] = dp2.queryRangeMax(h[i]+1, MAX) + value[i] # h=高度, value=價值, queryRangeMax(l,r)
  dp1.update(h[i], dp[i]) # update(index, value)
  dp2.update(h[i], dp1.queryRangeMax(MIN, h[i]-1))
```

每次求 DP 值和更新都花 O(log n)，複雜度 O(n log n)。

O(n)

高度都不相等的情況

考慮高度數列「從某個數字往左看的遞增序列」，例如數列 9,7,8,2,4,3,6,5 ，從 3 開始往左會遇到的遞增數列為右邊紅色的部分 9,7,8,2,4,3,6,5。

注意到下面兩件事情:

1. 對於每個 i，上面的遞增數列可以用 stack 在均攤 O(n) 的時間維護
2. 考慮上面 $O(n^2)$ 的算法，紅色部分其實代表不能轉移的 j，而黑色則代表可以轉移的 j

可以開一個 stack 維護這個往左遞增的數列，並另外開一個變數維護 stack pop 出來座標的最大 dp 值。可以每次轉移均攤 O(1)。

高度有相等的情況

上面的做法還是可以做，但要小心很多邊界情況，例如:

1. 當一個高度被從 stack pop 出來的時候，要一次更新一個區間同樣高度的 dp 值
2. 當要算 dp[i] 時，需要判斷 stack 裡面有沒有 h[i]，有跟沒有轉移的情況會不一樣

H.保全公司

結論

這題的可以觀察的結論非常多，這邊選一些重要的結論列出來。

為了方便討論，y=2 的點簡稱為「上」，y=0 的點簡稱為「中」，y=-2 的點簡稱為「下」。若一個三角形由兩個 y=2 及一個 y=-2 的三角形組成，簡稱為「上上下」。

- 關於解的三角形
  - 不會三個點 y 座標一樣
  - 不會三個點 x 座標皆為正或皆為負
    - 每個象限都有 n 個點，不會有選到最後沒有 x 座標為正/負的點可以選的情況
    - 因此若三個點 x 座標皆為正，把其中一個點往 x- 方向移面積只會變大不會變小
  - 每個三角形從每個象限裡最多選一個點
    - 因為每個象限都恰好有 n 個點，所以每個三角形都可以從任意象限挑點，不會有最後出現某個象限沒點挑的現象。
- 考慮下面幾種三角形的形狀
  - 上上下: 這種情況，下面的點無所謂因為不會決定面積可以留到最後選任一剩下的點即可。
    - 假設上面兩個點 x 座標分別為 x1, x2，面積為 |x1 - x2|*2
    - 可以知道 x1, x2 必定會一正一負，因此又可以寫成 2|x1| + 2|x2|
  - 下下上: 與上上下一樣
  - 上中下
    - 假設上、中、下的 x 座標分別為 x1, x2, x3，可得面積 = 2*|avg(x1,x3) - x2|。其中 avg= 兩數字平均
    - 再考慮上面的公式，可以發現面積最大化的時候只有兩種情況:
      a. x1,x3 為負，x2 為正
      b. x1,x3 為正，x2 為負
    - 可得到結論
      - 上下兩點的 x1,x3 同正負，並和中間點的 x2 不同正負
      - 面積可簡化為: |x1| + |x3| + 2|x2|
  - 中中上(or 中中下)

    - 對於中間的兩 x 座標 x1, x2 必定不同正負，面積可以寫成 |x1| + |x2|
    - 可以跟上中下對比，可以發現這種三角形是不必要考慮的，因為 x1 或 x2 有一點可以往上或往下移變成上中下的三角形
        - 證明: 2|x1| ≥ |x1| + |x2| or 2|x2| ≥ |x1| + |x2| 一定會發生
        - 若 2|x1| ≥ |x1| + |x2|，把 x2 的點移到上面或下面，答案不會比較差。也不會有點不夠選的情況

到這裡為止，我們可以把所有三角形的面積寫成 x 座標的絕對值相加，並且可以得到一些簡單的結論:

- 要考慮的三角形形狀只有「上上下」、「下下上」及「上中下」三種
- 上中下需要考慮兩種 case:
    - 中間 x<0，上下 x>0
    - 中間 x>0，上下 x<0
- greedy 順序:
    - 對於一個上面的點，若此點絕對值越大，則應該要優先給上上下的三角形，再給上中下，最後在給上中下
        - 因為座標=>面積在上中下的公式是乘以一倍，在上上下是乘以兩倍
    - 所以我們會得到一個 greedy 選點的順序
        - a. 上面、下面 x 座標絕對值最大的點應該優先給上上下、下下上的三角形選
        - a. 接著再分給上中下的三角形

O(n^3) 作法

枚舉以下數字

- a: 上上下的三角形個數
- b: 下下上的三角形個數
- c: 上中下，且上下 x<0 的個數
- d: 上中下，且上下 x>0 的個數 = n-a-b-c

算法:

1. 先將上面 x 座標最大以及最小的 a 個數字分給上上下的三角形
2. 再將下面 x 座標最大以及最小的 b 個數字分給下下上的三角形
3. 取中間 x 座標最大的 c 個點，以及上、下剩下 x 座標最小的 c 個點
4. 取中間 x 座標最小的 d 個點，以及上、下剩下 x 座標最大的 d 個點

上面的算法，若分別維護上、中、下三個 x 座標的前綴和，可以在 O(1) 得到三角形的面積。由於 d 的值可以用 n 減另外三個變數算出，枚舉量為 $O(n^3)$。

O(n^2 log n) 作法

上述做法當枚舉為 a,b 以後，觀察 type C(上中下，且上下 x<0) 與 type D(上中下，且上下 x>0) 的三角形。

可以發現，C三角形和D三角形選的點絕對會互斥。可以看成是在所有 C 和 D 三角形的連集裡選最大的 n-a-b 個。並且一定會滿足性值:

- 最大未選的 C 三角形面積 ≤ 最小已選的 D 三角形
- 最大未選的 D 三角形面積 ≤ 最小已選的 C 三角形

枚舉 a,b 以後，可以用上面兩個條件來二分搜 c,d 的值，可以做到 O(n^2 log n)。

O(n^2) 作法

考慮上面兩個做法，在 a 固定，b 遞增 1 的情況，這時候 c,d 只會有幾種變化:

1. d==0, c -= 1
2. c==0, d -= 1
3. c,d > 0，則算出最小的 C/D 三角形。若最小的三角形是 C type，則 c -= 1，否則 d -= 1

可以在 $O(n^2)$ 的時間維護。
