Contents

1 p1 一筆畫問題

給定 N 個點，我們希望能幫這些點排個順序，使得要用一筆畫依序經過它們的話，「射線」的次數不可能低於 $N-1$。

這題的所有子題除了暴力和奇怪特殊解以外沒有其他用意，因此接下來會直接描述最後一個子題的做法。

1.1 如何判斷能否少於 $N-1$

雖然最後的構造看起來與這件事無關，但是知道 checker 如何找出最佳解或許有助於想到並且證明最後的構造。

我們證明以下的 greedy 演算法是對的：

- 對每個點 P_i 維護一個角度集合 A_i，代表嘗試從 P_i 走到 P_{i+1} 時可以使用任何通過 P_i 且幅角為 $\alpha \in A_i$ 的射線，但如果幅角不在這個集合內，就必須花費一塊錢。一開始 $A_1 = [0, 2\pi)$，對於每個 $i = 2, 3, \ldots, N$，能不需要花錢，就不花錢，然後適當地從 A_i 轉移到 A_{i+1}，盡量保留所有可用的角度，最後花的錢就是最佳解。

證明.
對於一個一筆畫結果 SOL，我們可以將點分成幾個段落，同個斷落的點都是由同一條射線負責，特別的，如果一個點同時是一個射線的終點和下一個射線的起點，則它屬於前者的段落。顯然的，在任何最佳解裡，一個射線對應的段落不會是空的。

令我們 greedy 演算法的結果為 GREEDY，而隨便一個最佳解為 OPT。我們只須證明可以逐步地將 OPT 的段落分法變成 GREEDY 的分法，並且途中段落數量保持不變。

假設在 $\mathrm{OPT}, \mathrm{GREEDY}$ 中 $P_1, P_2, \ldots, P_x$ 所屬的段落長相一樣，但 P_{x+1} 不一樣。

- *Case 1.* 在 GREEDY 中，P_x, P_{x+1} 屬於同個段落
  假設在 OPT 中，P_{x+1} 的段落對應的射線是 R，我們可以直接將 OPT 中 $P_1, P_2, \ldots, P_{x+1}$ 替換成 GREEDY 的，然後到 P_{x+1} 時，用 R 繼續往下跑。這樣只是把 P_{x+1} 從原本的段落挪到前一個段落，整體來說段落數量不會變多，而新的 OPT 很明顯是一個合法的解。
- *Case 2.* 在 GREEDY 中，P_x, P_{x+1} 屬於不同段落
  我們證明這個情況不可能發生。
  首先因為 P_1, P_2 在 GREEDY 中必屬同個段落，$x \geq 2$ 必須成立。

令 $t < x$ 為最大的正整數使得 P_t, P_{t+1} 在 GREEDY 中屬於同個段落。因為 $t < x$ 所以 P_t, P_{t+1} 在 OPT 中也屬於同個段落。不只如此，我們還知道 $P_{t+2}, \ldots, P_x$ 在 GREEDY 和 OPT 中，都兩兩不屬於同個段落。

假設 OPT 中，$P_{t+2}, \ldots, P_x$ 所屬段落的射線為 $R_{t+2}, \ldots, R_x$。因為 P_t, P_{t+1} 屬於同一個段落，我們可以確定 $A_{t+1} = \{\arg \overrightarrow{P_t P_{t+1}}\}$，而且在 GREEDY 和 OPT 上都是用 $\overrightarrow{P_t P_{t+1}}$ 抵達 P_{t+1}。

不難發現，$\arg R_i \notin A_{i-1}, \arg R_i \in A_i, \forall t+2 \le i \le x$。這是因為從 $t+1$ 開始，A_i 維護了所有接下來所有可能省錢的射線。注意到如果中途有機會用同一條射線經過兩個以上的點的話，GREEDY 應該要選擇這麼做，但並沒有發生這件事，因此 A_i 只能盡可能維護所有接下來能使用的射線。

由於 $\arg R_x \in A_x$ 且 R_x 至少會依序通過 P_x, P_{x+1}，GREEDY 應該會選擇 R_x 並將 P_x, P_{x+1} 才對，矛盾。因此這個情況不會發生。

由以上討論可知我們總是能一步步將 OPT 的段落分法變成 GREEDY 的分法且途中段落數不變。 □

1.2 滿分構造

這裡提供一個能通過整題的簡單構造：先將所有點排序，然後反覆取頭尾即可。

沒錯，就是這麼簡單。

那為什麼他是對的呢？這部分讀者可以自行證明看看，證明過程大概如下：

不失一般性最後 $P_i = (X_i, Y_i)$。考慮 checker 用的 greedy 演算法，可以證明對於所有 $i > 1$，如果 i 是偶數，則 $A_i \subseteq (-\pi/2, \pi/2]$，否則 $A_i \subseteq (\pi/2, 3\pi/2]$，注意區間是半開半閉。感性一點來說，一個點可行的角度範圍永遠會在左半邊不然就是右半邊。

1.3 Remarks

本題靈感來源是 NWERC 2017 Problem C，原題就是要寫出這題的 checker。

本題存在很多不同的做法，定位上是最簡單題。

2 p2 苦無出鹿

題目的大意是有兩種顏色的點、跟兩種顏色的弧，數有多少種可能可以用同色點一對一配對、不相交相連就能分開。

我們這裡用原本題目 IOICamp 2023 Day4 pE — 孤獨搖滾的顏色，也就是點是 R B 、弧是 Y P 。

2.1 子任務 1

要如何驗證一組狀況是不是解呢？我們會想要把 Y B P 跟 P B Y 的狀況相連，因為在這個狀況 B 一定要連起來，否則兩部份不可能分的開。同樣，Y R P 跟 P R Y 也需要相連，而另外四種狀況 Y B Y 、Y R Y 、P B P 、P R P 就不用理會，因為他們相連只會讓狀況更複雜，對於同色弦切開完全沒有幫助。

所以，根據 Y 以及 P 的狀況，可以化成一些需要相連的兩種 R 以及兩種 B 。如果 P R Y — Y R P 連線互撞，那可以直接交換就好，所以問題是能不能在 P R Y — Y R P 、P B Y — Y B P 不互撞的時候連起來。

觀察到從任意一個弦把圓「剪開」化成序列，其實不會改變解的狀況，所以這邊就可以直接用原本攤開的狀況 greedy 的使用 stack 做匹配。

在子任務 1 中可以直接枚舉並檢查，在 $O(4^N \cdot N)$ 的時間內做完。

2.2 子任務 2, 3, 4

嘗試做一些等價條件的觀察，有解的情況必定是這個環狀字串中 P R Y 與 Y R P 出現的次數相同，因為他們必定要一對一匹配。

神奇的是，如果這件事情有發生，那就一定有解！這個理由是因為，考慮任意一個 P R Y ，往下順時針找到第一個 Y R P ，並將他們連線。在這些連線結束後，因為切開的弦上初始、結尾都是相同的，所以說中間必定沒有任何需要連線的紅點，不僅如此，當顏色發生切換的時候，中間一定是藍色的點！因為開始與結束的顏色相同，所以切換一定發生偶數次，這也代表區間中 P B Y 與 Y B P 的出現次數也是相同的，就可以照同樣的方式相連。

所以其中一個動態規劃的方法就是先填好前兩個字元，然後 DP 的使用兩個維度：填到哪個位置、以及 P R Y 與 Y R P 出現的次數差值，就可以紀錄方法數進行動態規劃數出所有的可能。

如果這邊實作得宜，常數健康可以通過 $N \leq 10^4$ 的子任務。

2.3 子任務 5

子任務 5 是設計給原範圍差不多等價的狀況，據稱原題作者的方法是使用矩陣並使用多項式操作加速，不過在這裡我們不會討論這個作法，因為直接一頭處理對於滿分幫助不大。

以下我們會討論滿分作法，大致上可能與這個子任務的作法相似，只是不需要矩陣。

2.4 子任務 6

一個非常簡單的等價條件：P R Y 與 Y R P 出現的次數相同會等價 P R 與 R P 的出現次數相同，這是因為唯一在這兩者之外會同時貢獻的只有 P R P ，對差值沒有任何影響。

這代表著對於原先的序列，連續的 ？ 並不會互相干擾！如果原先序列是空的，那我們直接計算答案，這個答案是

$$2 \cdot \binom{2N}{N}$$

證明從略。

如果原先序列至少有一個不是 ？ 的字元，那我們可以透過這些字元將序列切成若干份：每一份都是從某個字元開始，有一連串的 ？，然後結束在某個字元。我們的目標是數有多少種把 ？ 放好的狀況，使得 P R 的出現次數與 R P 相同。

注意到因為這個出現次數的差距跟序列的相同，所以我們可以用一個線性長度的多項式去紀錄每一種差值的方法數，所以整體作法分成兩步：

- 算出每一塊的多項式
- 全部乘起來

2.4.1 算出每一塊的多項式

直接算出多項式可能會需要很複雜的數學，不過其實這題是有簡單規律的，我們先用兩個多項式分別表達，因為同時只會在乎 R, B 或 P, Y 兩組的其中一組。我們讓多項式的第 i 項的係數是「P R 的出現次數減去 R P 出現的次數是 i，的方法數有幾種」，因為這個定義項的編號可以是負的，不過這就只是為了方便，實作上只要往正的位移足夠的量就行。

當 R, B 要轉移給 P, Y 的時候，因為 R P 會減去 1，所以發生的事情是

$$\begin{cases} P \leftarrow x^{-1}R + B \\ Y \leftarrow R + B \end{cases}$$

當 P, Y 要轉移給 R, B 的時候，因為 P R 會加上 1，所以發生的事情是

$$\begin{cases} R \leftarrow xP + Y \\ B \leftarrow P + Y \end{cases}$$

這時候你會發現一件很酷的事情：如果把 R, B 跟 P, Y 兩個多項式都寫成穿插的樣子，那你會發現

$$[x^0]R \quad [x^0]B$$

$$[x^{-1}]P \quad [x^0]Y \quad [x^0]P$$

$$[x^{-1}]B \quad [x^0]R \quad [x^0]B \quad [x^1]R$$

$$[x^{-1}]Y \quad [x^{-1}]P \quad [x^0]Y \quad [x^0]P \quad [x^1]Y$$

$$[x^{-1}]R \quad [x^{-1}]B \quad [x^0]R \quad [x^0]B \quad [x^1]R \quad [x^1]B$$

$$[x^{-2}]P \quad [x^{-1}]Y \quad [x^{-1}]P \quad [x^0]Y \quad [x^0]P \quad [x^1]Y \quad [x^1]P$$

這個東西巧妙的其實是一個巴斯卡三角形的形狀！所以說，只需要維護各項在三角形上的位置就可以好好計算。透過預處理，所有的係數都可以在線性於 ? 長度的時間內計算完畢。

2.4.2 全部乘起來

在這個部份，要如何好好的將所有的多項式乘起來呢？因為兩個多項式乘起來是花 $O(\ell \log \ell)$ 的時間，既然這個 cost 很接近線性，不如就用霍夫曼編碼（Huffman Encoding）的方式，每次拿兩個最小的多項式乘起來就好了！

所以這部份有著 NTT 模板應該是輕而易舉的。

不過值得一提的是因為子任務 6 只是對作法做出常數等級的優化，所以在某些情況下你可能需要去找一份很快的 NTT 才可以通過。

2.5 後話

這題其實在 The 2nd Universal Cup. Stage 4: Taipei，出題者在準備這一題的時候好像忘記測試當時的 topcoder，所以似乎是有機會過的。不過比起原先的作法，我覺得實作乾淨許多，常數少了約 4 倍，跑起來的速度也是相當快。

3 p3 網際網鹿

先重新描述一次問題和我們接下來會使用的用詞：給一張 n 點 m 邊的無向簡單圖，每個節點要被塗成三種顏色的其中一種，每種顏色至少要有一個節點，在塗完顏色之後，如果一條邊的兩端點同色，那麼這條邊就會消失，剩下來的圖中不能有環。

在接下來畫的圖中，我們會把三種顏色的點分別塗成紅藍綠色，分別為顏色 $1, 2, 3$，未知顏色的會是灰色，最後會留下的邊會被畫成實線，不會留下的邊會被畫成虛線。總之目標是所有實線必須是一個森林。

先做做原題，原題最單純的作法應該是直接拿一個生成樹，然後利用非樹邊數量不多的特性，但因為後面我們的邊數會更多，所以這作法行不太通。在 twpca 的題解裡面，有提到存在 $O(n)$ 的構造演算法直接構造三個顏色數量分別是 $1, 1, n-2$ 的解（就把大小 1 的那個顏色當作顏色 $1, 2$ 好了），這要怎麼做呢？對於一個 $1, 1, n-2$ 的塗法，會被留下的邊只有顏色 1,2 那兩個點的所有鄰邊，因此如果要能構成環，這個環上肯定得要 3 個顏色都出現。

假設前兩個顏色的點分別是 x, y，一個 $1, 1, n-2$ 的解合法的**充要**條件是這樣：

- 無三角形：x, y 相鄰的話，他們沒有任何共用鄰點，也就是 (x, y) 這條邊不在任何三角形上。
- 無四邊形：x, y 不相鄰的話，他們共用的鄰點數量至多一個。

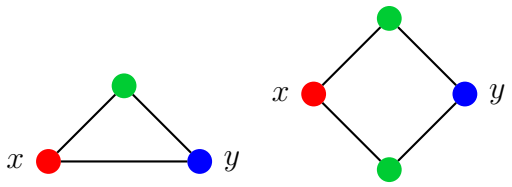


Figure 1: 左圖是違反無三角形的解，右圖是違反無四邊形的解

因此，一張圖**沒有** $1, 1, n-2$ 的解的條件是，對於任意兩個相異點 x, y，都滿足：

- 三角形條件：x, y 相鄰的話，它們至少有共用一個鄰點，也就是 (x, y) 這條邊在一個三角形上。
- 四邊形條件：x, y 不相鄰的話，它們至少有共用兩個鄰點。

只要兩個節點 x, y 不滿足上述條件，那把它們分別塗成顏色 $1, 2$，就是一個 $1, 1, n-2$ 的解。

3.1 判定條件與判定樹

定義 3.1. 判定條件
對於一個節點 r 的判定條件為「是否**不**存在一個 $1, 1, n-2$ 的解，使得 r 是顏色 1 或 2」。

整張圖沒有 $1, 1, n-2$ 的解若且唯若對於任意節點的判定條件都被滿足。不過，要檢查對所有節點的判定條件，其實不太容易，所以我們先來看看要怎麼檢查對一個點 r 的判定條件，以及滿足判定條件時會發生什麼事情。

考慮對 r 而言的 BFS 生成樹，首先，要滿足三角形和四邊形條件，代表任意一個節點 x 都要跟 r 至少共一個鄰點，因此 x 和 r 的距離 ≤ 2，BFS 生成樹只有三層，第一層是 r，第二層是 r 的所有鄰點，剩下都在第三層。

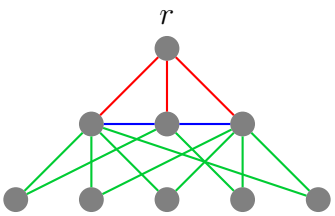


Figure 2: 判定樹

我們來討論一下如果節點 r 的判定條件是被滿足的，這張圖會有哪些邊：

- 紅色邊：r 跟鄰點們之間的邊，所有 r 的鄰邊都是這一種。
- 藍色邊：第二層節點之間的邊。第二層節點要跟 r 滿足三角形條件，也就是所有 r 跟第二層節點的邊都要在三角形上，這個三角形就只能是由兩個紅色邊和一個藍色邊構成，因此第二層的每個點都要有至少一條相鄰藍色邊。
- 綠色邊：第三層節點和第二層節點之間的邊。注意第三層節點不和 r 直接相鄰，因此要滿足四邊形條件，也就是跟 r 共至少兩個鄰點，而這些鄰點只能在第二層，因此每個第三層節點，都至少有兩條相鄰綠色邊。

所有這三種邊構成判定樹（雖然它其實不一定是樹），一個判定樹是**合法判定樹**代表我們可以用其中的邊知道 r 的判定條件是滿足的，也就是：每個第二層節點至少有一條相鄰藍色邊（沒有的話，該節點與 r 構成 $1, 1, n-2$ 的解）、每個第三層節點有兩條相鄰綠色邊（沒有的話，該節點與 r 構成 $1, 1, n-2$ 的解）。

定義 3.2. minimal 合法判定樹
如果一個判定樹是合法的，並且其中有一條邊被拔掉後，這個圖還是合法判定樹，那我們就把它拔掉，拔到不能拔為止，剩下來的我們就稱之為 minimal 合法判定樹。minimal

合法判定樹的所有第三層節點都有恰好兩條綠色鄰邊。

3.2 $m \leq 2n-4$

首先先注意到一件事情是 $m \leq 2n-4$ 的時候，整張圖的最小度數是 $\lfloor 2m/n \rfloor \leq 3$。

如果最小度數是 1 那就很快樂，如果你挑一個度數為 1 的節點當 r 做判定樹，那第二層節點就沒有藍色鄰邊，第三層節點也不可能有兩條綠色鄰邊，拿 r 跟隨便一個點塗成顏色 $1,2$ 就好了。

接下來我們討論最小度數是 2，同樣我們挑度數為 2 的節點當 r 做判定樹，這個判定樹如果合法的話會有幾條邊呢？紅色邊有 2 條，藍色邊有 1 條（在兩個第二層節點之間），綠色邊至少要有 $2(n-3)$ 條，加總就是 $2+1+2(n-3) = 2n-3 > 2n-4$，所以在 $m \leq 2n-4$ 時，這個判定樹不可能合法，只要看哪個判定樹條件沒被滿足，就能找到解了。

然後是最小度數為 3，同樣挑度數為 3 的節點當 r，判定樹合法的話，有 3 條紅色邊，至少 2 條藍色邊，和至少 $2(n-4)$ 條綠色邊，總共是 $3+2+2(n-4) = 2n-3$，也超過 $2n-4$ 了，所以就一樣看哪個條件不被滿足就好。

3.3 $m \leq 2n-3$

注意到 $\deg(r) = 2$ 的時候，合法判定樹只有一種長相：

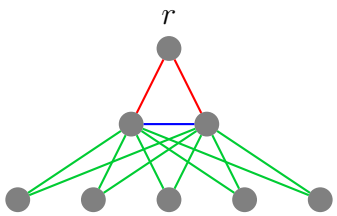


Figure 3: $\deg(r) = 2$ 合法判定樹

看一下就會發現，光是判定樹上的邊，就足夠讓我們確定不可能存在 $1, 1, n-2$ 的解，因此最小度數為 2 時，一旦 r 的判定條件被滿足，我們就可以直接確定沒有 $1, 1, n-2$ 的解了。其實這張圖是 $K_{2,n-2}$ 然後左側兩個點之間加一條邊，我們可以更進一步證明，只要包含這種子圖，那就完全無解。令左側那兩個點叫作 x_1, x_2，那麼：

- 如果 x_1, x_2 同色，假設這個顏色叫 1，那麼另一側要有至少一個顏色 2 和至少一個顏色 3 節點，然後就會出現環。
- 如果 x_1, x_2 異色，假設 x_1 是顏色 1，x_2 是顏色 2，那另一側要有至少一個顏色 3 節點，然後就會出現環。

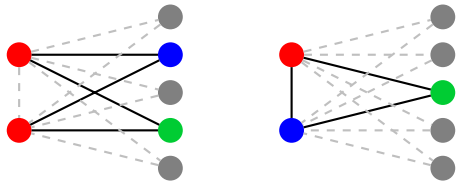


Figure 4: $K_{2,n-2}$ 左側加一條邊，左側兩個點同色和異色的 case

因此，最小度數為 2 時，只要對一個度數為 2 的點 r，檢查對 r 的判定條件是否滿足，不滿足就找到解了，滿足的話輸出 -1。

接下來是最小度數為 3 時，注意到剛剛算的 $2n-3$ 條邊其實是 minimal 合法判定樹的邊數，實際上的邊可能更多：在 minimal 合法判定樹上，所有第三層節點的度數都恰好是 2，但這個 case 的前提是最小度數為 3，因此我們知道每個第三層節點至少要有一條不在 minimal 合法判定樹上的鄰邊，只要第三層節點有至少一個，那整張圖的邊就至少要有 $2n-2$ 條，所以我們知道 $n \geq 5$ 時，是不可能滿足對 r 的判定條件的，至於 $n < 5$，要最小度數為 3 就只能是 K_4，而 K_4 的邊數是 $6 = 2 \times 4 - 2$，又是 $2n-2$。

結論是，$m \leq 2n-3$ 時，如果最小度數為 2，那滿足對 r 判定條件就是無解，而最小度數為 3 時，不可能滿足對 r 的判定條件。

3.4 $m \leq 2n-1$

最小度數 ≤ 2 的我們都做完了。最小度數為 3 的時候，為了要讓第三層節點的度數都至少有 3，只要第三層節點夠多，就會需要比 minimal 合法判定樹多更多邊。所以一個直接的作法就是 n 小的時候直接暴力枚舉所有塗色方法，$n \leq 8$ 就夠了，n 大的時候不可能滿足判定條件。

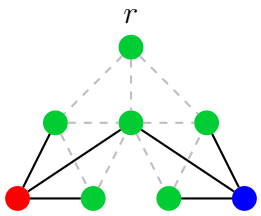


Figure 5: 最小度數為 3、滿足對度數 3 的 r 的判定條件，但有解。$n = 8$，$m = 15$，$m = 2n - 1$。

3.5 如果 $m \geq 2n$ 則 $m \leq 2.5n - 6$

最小度數是 3 的時候，第三層節點到底要多至少幾條邊？minimal 合法判定樹上的第三層節點每個度數都是 2，總度數至少要再多 $n - 4$，所以總邊數至少要有 $2n - 3 + \lceil 0.5(n-4) \rceil \geq 2n - 3 + 0.5(n-4) = 2.5n - 5$，因此 $m \leq 2.5n - 6$ 的時候是不可能滿足判定條件的。

注意到 $m \leq 2.5n - 6$ 時最小度數可能會是 4，和前面一樣，我們算一下以某個度數 4 節點為根的合法判定樹至少要有幾條邊。紅色邊有 4 條，藍色邊至少要有 2 條（根度數是 d 時就至少要有 $d/2$ 條），綠色邊至少有 $2(n-5)$ 條，為了讓第三層節點的度數至少 4，在 minimal 合法判定樹之外至少要再有 $2(n-5)/2 = n - 5$ 條，總共是 $4+2+2(n-5)+n-5 = 3n-9$。$3n-9 > 2.5n-6 \iff n > 6$，所以在 $n > 6$ 時不可能滿足判定條件。最小度數是 4 的時候，邊數至少要是 $2n$，而 $n \leq 6$ 的時候 $2n > 2.5n - 6$，所以不可能出現 $n \leq 6$ 且最小度數是 4.

結論是，$m \leq 2.5n - 6$ 時最小度數無論是 3 還是 4 都還是不可能滿足判定條件。但是 $m \leq 2n - 1$ 的時候有可能 $m > 2.5n - 6$，還是要記得暴力。

3.6 如果 $m \geq 2n$ 且 $m > 2.5n - 6$ 則 $m \leq 3n - 10$

最小度數是 4 的狀況剛剛做完了。$m \leq 3n - 10$ 會另外多出最小度數是 5 的狀況，用一樣的算法，紅色邊有 5 條，藍色邊有至少 3 條，綠色邊有 $2(n-6)$ 條，第三層的額外邊有 $3(n-6)/2 = 1.5n - 9$ 條，總共是 $5 + 3 + 2n - 12 + 1.5n - 9 = 3.5n - 13$ 條，$3.5n - 13 > 3n - 10 \iff n > 6$，所以 $n > 6$ 的時候都不會滿足判定條件，最小度數是 5 時，邊數至少要是 $2.5n$，而 $n \leq 6$ 時 $2.5n > 3n - 10$，所以不會出現 $n \leq 6$ 而最小度數是 5.

剩下來的唯一問題是，最小度數是 3 但 $m > 2.5n - 6$ 時，有可能會滿足判定條件。

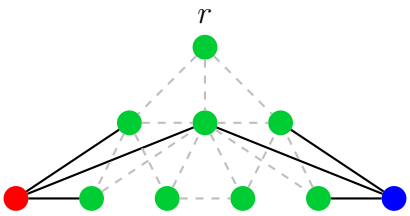


Figure 6: 最小度數為 3、滿足對度數 3 的 r 的判定條件，但有 $1, 1, n-2$ 的解，同時 $\max(2n, 2.5n-6) < m \leq 3n-10$。($n = 10, m = 20$)

如果我們退一步，想辦法直接枚舉 $1, 1, n-2$ 的解，找到的話就找到了，找不到的話，在最小度數為 2 時，沒有 $1, 1, n-2$ 的解就真的沒有解，會不會最小度數是 3 的時候也是？照著這個想法，我們假設不存在 $1, 1, n-2$ 的解，但是存在不是 $1, 1, n-2$ 的解，來看看會發生什麼事。

同樣考慮以某個度數 3 的點 r 為根的判定樹，第二層節點我們叫作 x_1, x_2, x_3，注意到藍色邊之中必定包含一條鏈（根節點度數 3 的合法判定樹中，藍色邊至少要有 2 個，所以只會是一條鏈或一個三角形）串起這三個點，假設這個路徑上的順序就是 x_1, x_2, x_3，也就是 x_1, x_2 間有邊，x_2, x_3 間有邊。

我們先來討論一下 r, x_1, x_2, x_3 這四個點的顏色。

r, x_1, x_2, x_3 之中不可以有三種顏色，因為只看它們四個點的話，一定有個子圖是 $K_{2,2}$ 左側加一條邊（最小度數 2 時會爛掉的那個形狀），所以它們四個點的 induced subgraph 是沒有解的。因此，這四個點之中頂多出現兩種顏色。

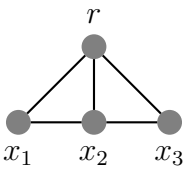


Figure 7: 這個形狀是不會有解的。

r, x_1, x_2, x_3 也不可以全部都是同樣顏色。假設它們都是顏色 1，第三層節點就要有至少三個點是顏色 2 或 3，我們把這兩個顏色統稱叫特殊色，我們把第三層節點分成三類，根據它們相鄰的第二層節點分類，記作 $(1,2), (1,3), (2,3), (1,2,3)$，像是 $(1,2)$ 就代表跟 x_1, x_2 相鄰。$(1,2,3)$ 那個類型是不可以有特殊色的，否則第三層再有其他任何特殊色節點都會造成環。

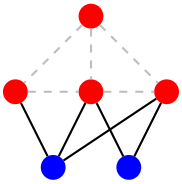


Figure 8: 如果有 $(1, 2, 3)$ 類型的節點是特殊顏色，其它第三層的特殊顏色點一定會跟它形成環。

剩下三類，同一個類型的點最多只能有一個特殊色，否則也會有環。

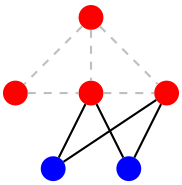


Figure 9: 同一類的節點有兩個是特殊色就會有環。

既然要有至少三個特殊色節點，就只能是每個類型各一個，還是會出現環。

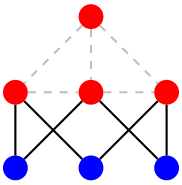


Figure 10: 三個類型各一個特殊色點會有環。

所以，上面那四個點會有恰兩種顏色，假設就是 $1, 2$，第三層至少要有一個節點是顏色 3。x_2 至少要跟 x_1, x_3 其中一個同色，如果 x_2 跟它們兩個都不同色的話，就會出現一條路徑 x_1, x_2, x_3，第三層的任何一個顏色 3 節點都會造成環。不失一般性，假設 x_2 是跟 x_1 同色，都是顏色 1。x_3 不可以也是顏色 1，否則 r 就得是顏色 2，會讓 x_1, x_2, x_3 連通，發生一樣的問題。x_3 是顏色 2 的話，那 r 不能是顏色 2，否則會有一條路徑是 x_1, r, x_2, x_3，還是有一樣的問題。

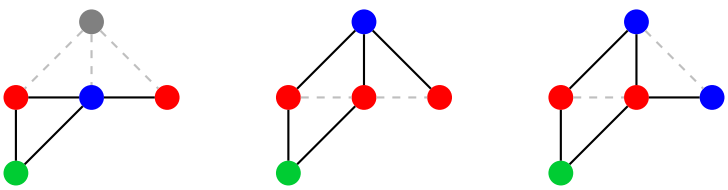


Figure 11: 如果上面那四個點之間的邊就會讓 x_1, x_2, x_3 連通，那第三層的顏色 3 節點一定會造成環。

剩下的唯一一種可能性，就是 r, x_1, x_2 是顏色 1，x_3 是顏色 2。第三層必須至少有一個顏色 3 的節點，首先，因為 x_2, x_3 之間有一條邊，它不可以跟 x_2, x_3 都有邊，因此顏色 3 點的類型只能是 $(1, 2)$ 或 $(1, 3)$。

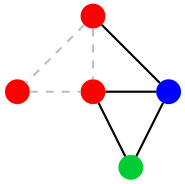


Figure 12: 顏色 3 節點的類型不可以是 $(2, 3)$ 或 $(1, 2, 3)$。

假設有個顏色 3 的點叫作 y，因為 r, x_1, x_2, x_3, y 會造成 x_1, x_2, x_3 連通，要是還有其他顏色 3 節點，它跟 x_1, x_2, x_3 其中兩個相鄰，肯定會出現環。因此顏色 3 的節點只能有一個。

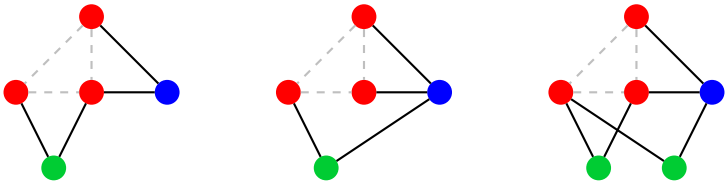


Figure 13: 左邊兩個圖是 y 可能的兩種類型，右圖表示了有第二個顏色 3 節點就會出現環。

對於一個第三層顏色 2 節點，它不可以跟 x_1, x_2 都相鄰，所以它的類型只能是 $(1, 3)$ 或 $(2, 3)$。由此可知，任何 y 的第三層鄰點都不可以是顏色 2，因為 y 跟它之間有邊、它跟 x_1 或 x_2 有邊，而 y 和 x_1, x_2 經由 y, x_1, x_2, x_3 連通。

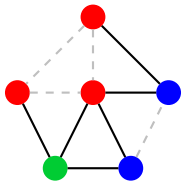


Figure 14: y 不能跟第三層顏色 2 節點相鄰。

因此，任何 y 的第三層鄰點，都只能是顏色 1。別忘了我們現在假設不存在 $1, 1, n-2$ 的解，因此 y 和 x_3 得滿足三角形和四邊形條件。對 y 的類型分一下 case：如果 y 的類型是 $(1, 2)$，那它跟 x_3 要滿足四邊形條件，它們已知共一個鄰點 x_2 了，第二個鄰點只能是 x_3 或第三層節點，是第三層節點的話，那個點只能是顏色 1，所以會出現環。如果是

$(1,3)$，那它跟 x_3 要滿足三角形條件，要共一個鄰點，這個鄰點也只能是 x_1 或第三層節點，都只能是顏色 1，所以也會出現環。

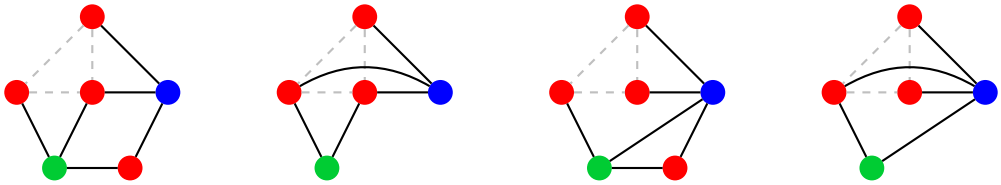


Figure 15: 左邊兩個是類型 $(1,2)$ 的狀況，右邊兩個是類型 $(1,3)$ 的狀況。

因此，在最小度數是 3 時，只要沒有 $1,1,n-2$ 的解，就無解。

3.7 總結

總結一下作法：令 r 是某個度數最小的節點，

- 如果 $\deg(r) \leq 1$，那麼 r 和任意一個點可以當作 $1,1,n-2$ 的解。
- 如果 $\deg(r) = 2$，那麼檢查 r 的判定條件，不滿足就找到解了，滿足的話輸出 -1。
- 如果 $\deg(r) = 3$，暴力檢查有沒有 $1,1,n-2$ 的解，沒有的話輸出 -1。
- 如果 $\deg(r) \geq 4$，那麼不可能滿足判定條件，檢查對 r 的判定條件可以找到 $1,1,n-2$ 解。

唯一剩下的問題是怎麼在 $\deg(r) = 3$ 的時候暴力檢查有沒有 $1,1,n-2$ 的解。一種想法是用 bitset 暴力搞，先用 bitset 存好鄰接矩陣的每個 row，枚舉一個節點 x，開兩個 bitset b_1, b_2 代表已知有共一個鄰點的節點和已知有共兩個鄰點的節點，跟 x 相鄰的點我們直接當成它先有一個共鄰點，所以 b_1 的初始值是 $adj[x]$，接下來枚舉 x 的鄰點 y，所有 $adj[y]$ 裡的點會跟 x 多共一個鄰點，拿去更新 b_1, b_2，最後看 b_2 裡是 0 的節點，那個點跟 x 就是 $1,1,n-2$ 的解。

不過這樣做非常暴力又要很多 bitset 運算，可能不容易過，以下提供一個常數較小的作法。

首先我們可以檢查 r 和它 3 個鄰點的判定條件，都滿足的話我們就可以放心讓它們都是有 $n-2$ 個的那種顏色（假設是顏色 3）。由於對 r 的判定條件是滿足的，我們知道 r 為根的判定樹上，第三層節點都和至少兩個第二層節點相鄰，因此兩個第三層節點，在第二層至少共一個鄰點。注意到 $(1,2,3)$ 類型的第三層節點，和其他任何第三層節點，都會在第二層共兩個鄰點，因此這個類型是不能塗顏色 $1,2$ 的。至於其他類型，同類型的點也會在第二層共兩個鄰點，因此顏色 $1,2$ 的那兩個點必須屬於不同類，而不同類的點

在第二層共恰一個鄰點，因此我們想要找到兩個點 x, y，滿足它們的類型不同、不相鄰，也在第三層不共鄰點。

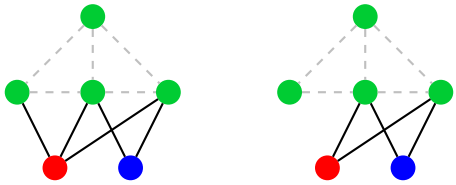


Figure 16: $(1, 2, 3)$ 類型的點不能是顏色 $1, 2$，顏色 $1, 2$ 的兩個點不能是同個類型。

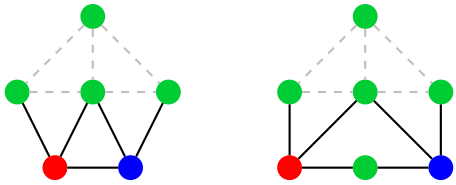


Figure 17: 顏色 $1, 2$ 的兩個點不能相鄰，不能在第三層共鄰點

先處理好三個類型節點的 mask，然後枚舉 x，再枚舉它的第三層鄰點 z，把 $adj[z]$ or 起來，再跟 x 類型與 $(1, 2, 3)$ 類型的 mask or 起來，把 r, x_1, x_2, x_3, x 都設成 1，對應 bit 是 1 的節點就代表它不可以被選作 y，如果有節點在這個結果裡是 0 那它就是 y 了。複雜度是 $O(nm/64)$。

3.8 後話

這是一個沒有 $1, 1, n-2$ 解但是有解的圖：

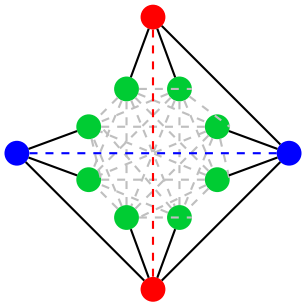


Figure 18: 中間的綠色是一個 K_8，兩個藍點間有邊，兩個紅點間有邊。

可以看到中間那個 K_8 製造很多邊，但我也不知道上面那坨判 case 方法這樣子推推推可以推到 m 是多少，畢竟這題已經噁到我不想推了。

4 p4 飢腸鹿鹿

4.1 子任務 1

注意到風味值的範圍不大，可以先將數字按照風味值分類，並照滿足度大到小排好。

一個初步的想法是若我們枚舉哪些風味值選了奇數個，哪些選了偶數個。先將要選奇數個的風味值都選一個，接著可以一直貪心選擇兩個相同風味值且滿足度總和最大的鹿角對，就這樣一直取直到選到 K 個就結束操作。

接著我們先將所有鹿角對都按照總風味值大到小排好，並稱瓶頸值 g 為選擇的鹿角對中最後面的一對的編號（若不存在則 $g = -1$）。可以發現為於此瓶頸值前面的鹿角對都會選。

接著我們考慮枚舉這個瓶頸值 g，那對於某一種風味值 f，假設有 y_f 個鹿角對在 g 之前，那這個風味值不是選擇 y_f 個鹿角，就是選擇 $y_f + 1$ 個鹿角。

接下來就可以 dp 了。定義 $dp[i][j][k]$ 代表考慮前 i 種風味值，當前選的鹿角數量比下界（也就是 $\sum y_f$）多了 j，且目前風味值 xor 為 k 的最大滿足度總和。因為 i, j, k 的維度都只有 C，可以在 $O(C^3)$ 的時間內算完 dp。

最後再觀察到最多只有 C 種瓶頸值可以讓選擇的的鹿角數量包含 K，所以套用前面的作法就作完了，整體複雜度為 $O(C^4 + N \log N)$。

4.2 子任務 2

以下我會介紹兩個作法，第一個作法是由子任務 1 優化得來的，第二個作法則是與前面完全不相干的作法。

4.2.1 作法一

事實上前面枚舉「瓶頸值」再作 dp 的事情有點多餘，注意到每次更新瓶頸值後，需要考慮的鹿角只會多一個，因此我們其實只會有 $2C$ 個鹿角是不確定會不會被選擇的，其他都可以確定會或不會。

在經過前面子題一的作法得出這 $2C$ 個鹿角後，我們再次考慮 dp：$dp[i][j][k]$ 代表考

慮前 i 個鹿角，目前已選擇 j 個且目前風味值 XOR 為 k 的最大滿足度總和。我們可以在 $O(C^3)$ 的時間蓋出 dp 表並計算出答案，總複雜度就是 $O(C^3 + N \log N)$。

4.2.2 作法二

假設滿足度前 K 大的鹿角編號為 $S = \{S_1, S_2, \ldots, S_K\}$，我們接下來會證明對於每個 x，若存在一組解則最優解與 S 不會相差太多。更具體來說，我們會證明以下 Lemma：

引理 4.1.
若存在至少一組選 K 個鹿角的方法使風味值 XOR 為 x，則存在一組最優解 $T = \{T_1, T_2, \ldots, T_K\}$ 滿足 $|T \setminus S| \leq \lceil \log_2 (C+1) \rceil$。

證明.
令 T 為其中一組風味值 XOR 為 x 的解，且 $S \setminus T = \{A_1, A_2, \ldots, A_{|A|}\}$、$T \setminus S = \{B_1, B_2, \ldots, B_{|B|}\}$ 與 $v_i = f_{A_i} \oplus f_{B_i}$，並假設 $|A| = |B| > \lceil \log_2 (C+1) \rceil$。由於 v_i 有超過 $2^{\lceil \log_2 (C+1) \rceil} \geq C+1$ 種集合，根據鴿籠原理，必存在一個非空子集 $\{i_1, i_2, \ldots, i_m\}$ 滿足 $v_{i_1} \oplus v_{i_2} \oplus \ldots \oplus v_{i_m} = 0$。接著令 $T' = (T \cup \{A_{i_1}, A_{i_2}, \ldots, A_{i_m}\}) \setminus \{B_{i_1}, B_{i_2}, \ldots, B_{i_m}\}$，可以發現 T' 的風味值依然為 x，且由於 A_i 的滿足度不會比 B_j 低，T' 的總滿足度一定不會比 T 低。所以可以一直透過此方法得出一個與 S 更接近的解，直到 $|T \setminus S| \leq \lceil \log_2 (C+1) \rceil$。 □

有了以上 Lemma，我們就有了以下做法：首先將所有鹿角按照滿足度排序，並稱前 K 大的為大鹿角，剩下的為小鹿角。我們再跑以下兩種 dp：$dp1[i][j][k]$ 代表考慮前 i 個大鹿角，目前選了 j 個且風味值 XOR 為 k 的最小滿足度總和；$dp2[i][j][k]$ 代表考慮前 i 個小鹿角，目前選了 j 個且風味值 XOR 為 k 的最大滿足度總和。跑完兩種 dp 後只要再枚舉 $|T \setminus S|$ 的值與 $T \setminus S$ 跟 $S \setminus T$ 的風味值 XOR 即可算出答案。注意到 j 的維度只要跑到 $\lceil \log_2 (C+1) \rceil$ 且 k 的維度有 C。

再進一步觀察可以發現對於每種風味值 f，我只在乎前 $\lceil \log_2 (C+1) \rceil$ 小滿足度的大鹿角與前 $\lceil \log_2 (C+1) \rceil$ 大滿足度的小鹿角，也因此 i 的維度其實只有 $2C\lceil \log_2 (C+1) \rceil$，這樣就可以得到一個 $O(C^2 \log^2 C + N \log N)$ 複雜度的作法。

4.3 後話

跟 source 說的一樣，這題是從 2024 台清交程式競賽的其中一題改編的，當時只有子任務 1，後來陸續搞出了兩個可以解決更高限制的作法也就讓這題誕生了。這裡也特別感

謝當時的隊伍 PixelCat14MathGoat 提出了作法二。

另外事實上把作法一與作法二合併的話可以得到一個 $O(C^2 \log C + N \log N)$ 的解，但只差一個 $\log C$ 應該很難區分，而且出題者發現的時候已經太晚了就沒有多加子題了。不過我懷疑這題還存在複雜度更好的解，只是到目前還沒有想到，歡迎各位試著想想看 XD。

5 p5 高速公鹿

5.1 子任務 1

直接使用 Dijkstra 等方法 $\mathcal{O}((N+M)\log(N+M))$ 回答一個詢問。

5.2 子任務 2

原題的限制。原題有一個基本上一樣的討論加上掃描線還是線段樹的作法，但印象中題解說要用到時間各相異，讓我覺得想推銷一下正解的作法。

5.3 子任務 3

因為各個詢問之間是分開的，我們分開討論三種詢問。有一件 trivial 的事情是我們可以假設我們不會重複走一個邊兩次。

1. 先來處理看上去最難的第二種詢問，也就是拔掉一條邊的詢問。
   我們可以構建一張沒有權重的有向圖，每個頂點對應到原圖的頂點以及一個時間。那麼，每個原本題目裡面的邊就等於是把 (u_i, s_i) 和 $v_i, t_i)$ 這兩個新圖中的頂點連一條有向邊，而在原圖當中的「原地等待」行為可以被當成就是從 (v, t) 走到 $(v, t+1)$，每個原圖中的頂點就會在新圖中引出一條鏈。那麼，假設沒有任何修改的話，答案就等於是看終點那個頂點的鏈上哪個時間開始，是從起點的頭走得到的。也可以說，我們把時間這一維加入頂點中，使得原圖「某個時間能抵達某個頂點」這件事變成在新圖上的 reachability 的問題。
   這種看法有什麼幫助呢？Dominator tree 正好是一個很經典的工具，可以處理「拔掉一個頂點」的固定起點任意終點 reachability 的問題。具體來說，假設起點是 s、終點是 t，那麼想看「拔掉頂點 x 之後 s 是否可以走到 t」就等於是在看 Dominator tree 上 x 是不是 t 的祖先，如果是的話就表示從 s 走到 t 一定要經過 x，拔掉 x 之後就走不到 t 了。然而會發現按照前一段的轉換方法我們要處理的是「拔掉一條邊」的 reachability 問題，但其實問題也不大，我們把每條邊中間再嵌入一個新頂點就可以當作是拔掉那一個頂點了。
   「快速檢查樹上是否是祖先」有很多種寫法，最簡單的應該是檢查 dfs 進出時間是否是相互包含的關係，可以 $\mathcal{O}(1)$ 回答，見 Euler tour [1] 。那麼只要二分搜終點那條鏈

[1] https://usaco.guide/gold/tree-euler?lang=cpp

到哪裡才 reachable 就可以回答最原本的問題了。注意我們不能真的對 $1 \leq t < 10^9$ 建出 (v, t) 到 $(v, t+1)$ 的邊，必須每個頂點只保存關鍵的時間點再把他們連起來，這樣才能讓新圖的大小是 $\mathcal{O}(N+M)$。

2. 接著是第三種詢問，新增一條邊。這是最簡單的一種詢問。順帶一提題解的剩餘部份應該都跟原題一樣了。
   - 假設最佳解沒有用到新增的那條邊：
     那麼就是原圖本來的答案，直接最一開始算出來就可以了。
   - 假設最佳解有用到新增的那條邊：
     先不管新增的邊只考慮原圖，我們可以先以 Dijkstra 一次原本的圖，預處理每個頂點 v 最早要到什麼時間才能抵達，令這個時間叫 a_v。接著，考慮處理第二種詢問時提到的那個新圖。我們一樣可以用一次多點源 Dijkstra（這樣邊的方向要反過來）或是一次 dfs 預處理出 $b_{v,t}$，代表新圖上從 (v, t) 開始走，可以抵達 (N, t') 的最小 t' 是多少（N 是終點）。
     令新增的邊是 (u', v', s', t')。那麼，首先要確認我們趕得上這條邊，也就是說要確認是否可以在時間 s' 前抵達 u'，即 $a_{u'} \leq s'$。接著，如果可以趕上，那麼抵達終點最早的時間就會是只考慮原圖，且在時間 t' 從 v' 出發的話抵達終點最早的時間，即 $b_{v',t'}$。
3. 最後是第一種詢問。
   - 假設最佳解沒有用到修改的那條邊：那就等於是第二種詢問。
   - 假設最佳解有用到修改的那條邊：
     讓人意外的是，這可以直接當作第三種詢問的第二個 case，也就是「新增了一條邊且一定要經過這條邊」來做。這是因為修改前後 u_i, v_i 並沒有被改變，因此若在對應的最佳解當中如果同時用到修改後的邊以及修改前的邊，我們一定可以去掉修改前的邊且得到一樣好的解。

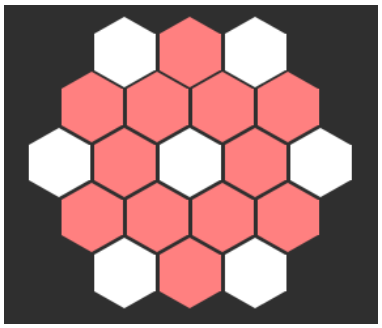
     證明.
     首先，如果同時用到了修改前和修改後的邊的話，修改前和修改後的時段一定不會是互相包含的關係。假設先走到修改前的邊，那直接在第一次走到 u_i 的時候在 u_i 原地等待就可以避免使用到修改前的邊了。假設先走到修改後的邊，之後又用到修改前的邊。那麼不如在第一次走到 v_i 的時候在 v_i 原地等待。 □

時間複雜度的部份，預處理時需要排序那些關鍵的時間以及蓋 Dominator tree、Euler tour，每個詢問可以在 $\mathcal{O}(1)$ 次二分搜回答，總共應該是 $\mathcal{O}((N+M+Q)\log(M))$。

note: 事實上本題可以強制在線回答三種詢問，因為即使新增或修改抵達時間為 t，要繼續搭乘下去抵達終點也必須等到下一個「關鍵時間」，也就是在原圖中相關的那些邊的出發時間，除非是直接抵達終點，在那種情況則可以直接答案跟 t 取 min。

6 p6 花田一鹿

先講結論：除了 $n = 3$ 且空格在下圖紅色時無解，其他情況都有解。



這應該不難猜，但問題是要如何好好構。以下我們會用數學歸納法的方式，慢慢將問題歸約到 n 足夠小的問題，再用簡單的方法解決。

6.1 子任務 1

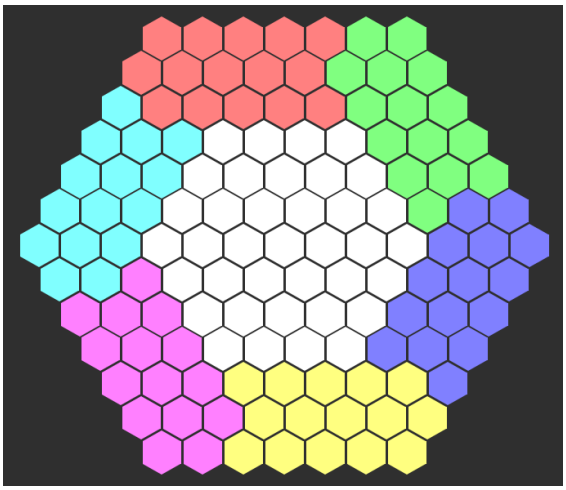
有兩種解法：第一是直接爆搜，第二是耐心判完所有 case。爆搜可以一直考慮最左上角的田地被怎樣的盆栽覆蓋，再跳過不可能被填滿的情況即可。如果發現爆搜跑太慢的話也可以本地將所有答案打表下來。

6.2 子任務 2

這筆子任務其實就是 NPSC 的題目，網路上也查的到 題解 [2] 。

簡單來說，我們可以將田地大小為 n 的情況歸約到田地大小 $n - 3$。如下圖：

[2] https://www.youtube.com/watch?v=q-nhNEpS3k0&list=PLjXPYRCTSLlThPuS1rybUKLNobnJvKHoy&index=7



之後只要特判 $n = 2$ 與 $n = 3$ 的情況即可。

6.3 後續步驟

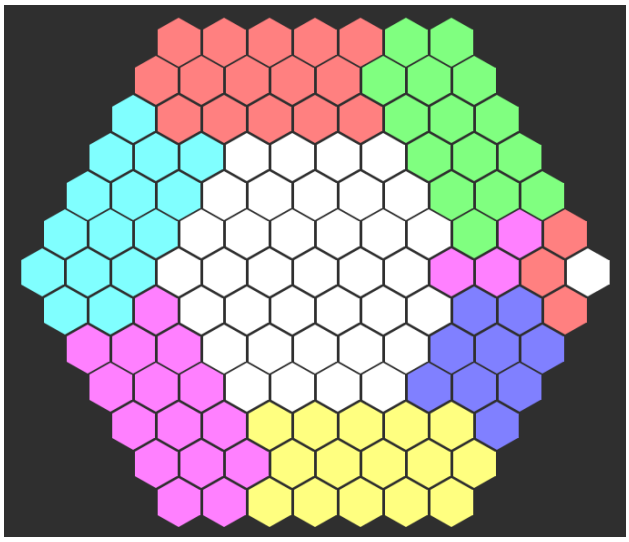
透過爆搜可以發現在 $4 \leq n \leq 6$ 的情況下，任意一個空格都有解。而子任務 2 也帶給我們一個方向：給定任意田地大小 $n > 6$，只要我們可以歸約到田地大小 $n - 3$ 的問題，那我們就可以遞迴構造完所有解。具體來說，我們考慮以下流程：

- 若 $n \leq 6$，直接用爆搜或判 case 做完。
- 否則想辦法將原問題歸約成 $n - 3$ 的問題並遞迴。因為對任意 $4 \leq n \leq 6$ 都有解，所以不會歸約到無解的問題。

因此，以下我們的目標主要為透過構造來完成歸約。

6.4 子任務 3

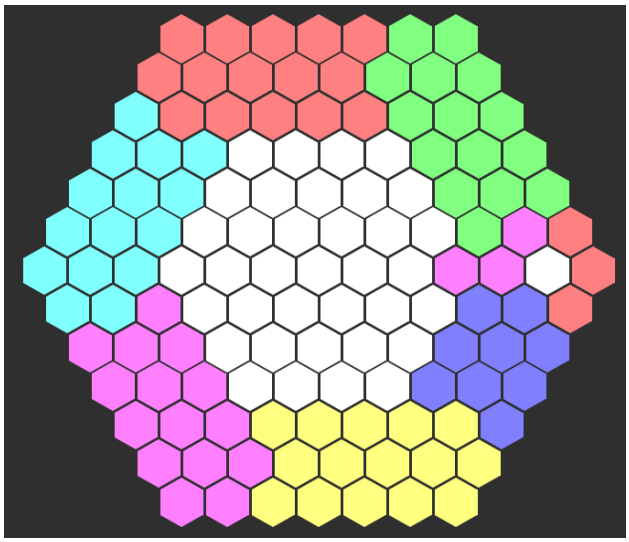
考慮以下構造：



這樣就能順利歸約。

6.5 子任務 4

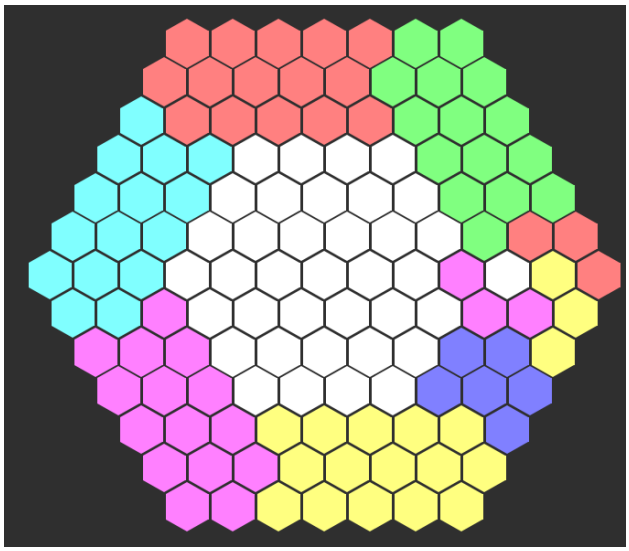
考慮以下構造：



這樣就能順利歸約。

6.6 子任務 5

若 $x = n - 3$，考慮以下構造：



否則使用子任務 2 的構造法即可順利歸約。

6.7 子任務 6

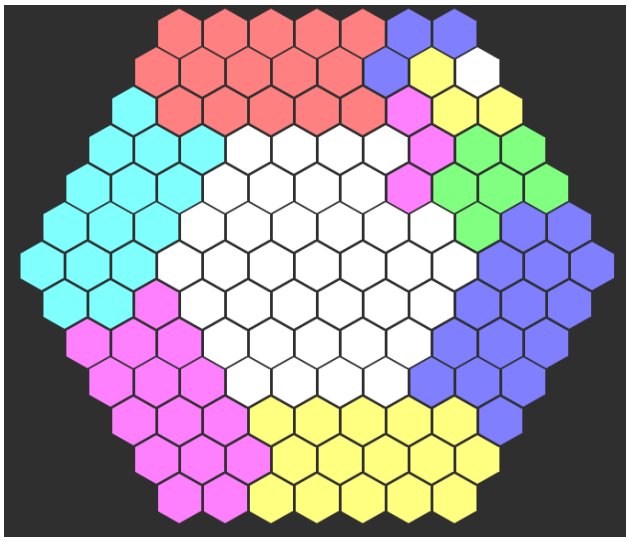
繼續構造！從現在開始我們會只考慮 $y > 0$ 的 case。

6.7.1 $y = 1$

將田地照兩點鐘方向翻轉，會變成 $y = n - 2$ 的情況。

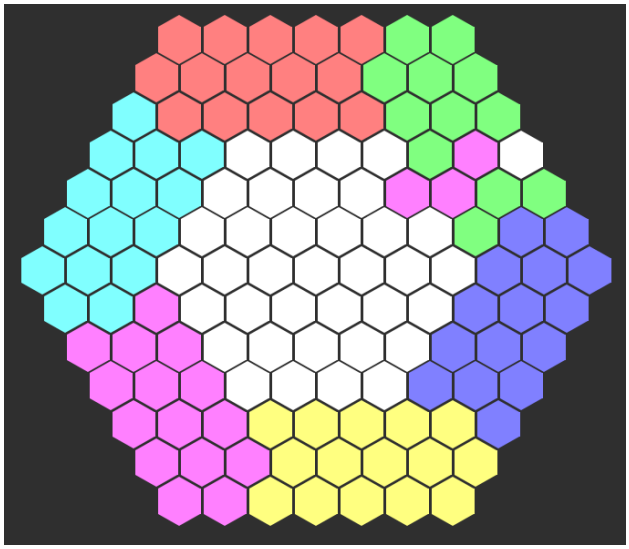
6.7.2 $y = n - 2$

考慮以下構造：



6.7.3 其他情況

考慮以下構造：



這樣就能順利歸約。

6.8 子任務 7

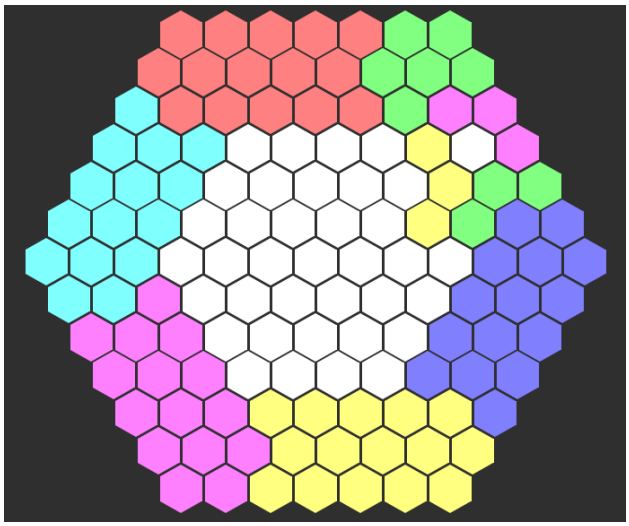
持續構造！

6.8.1 $y = 1$

將田地照兩點鐘方向翻轉，會變成 $y = n - 3$ 的情況。

6.8.2 其他情況

考慮以下構造：

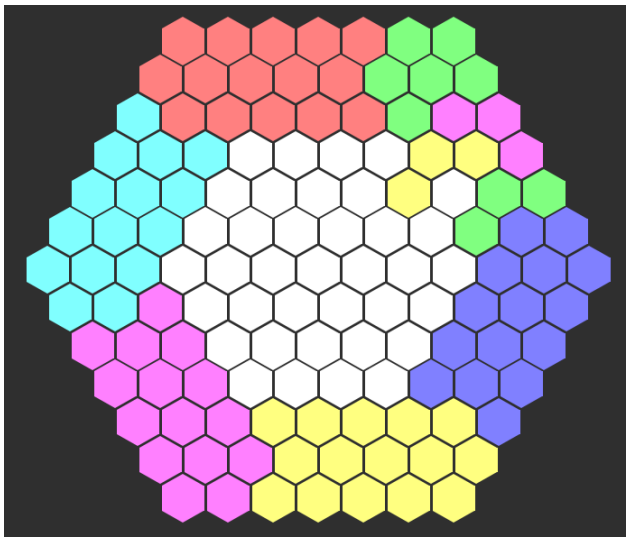


這樣就能順利歸約。

6.9 子任務 8

為了方便前面子任務的空格不是在原點就是在第一象限，我們要先記得把空格轉到第一象限。同時我們先將 $x + y < n - 3$ 用子任務 2 的方法完成歸約，因此以下只考慮 $x + y = n - 3$。

考慮以下構造：

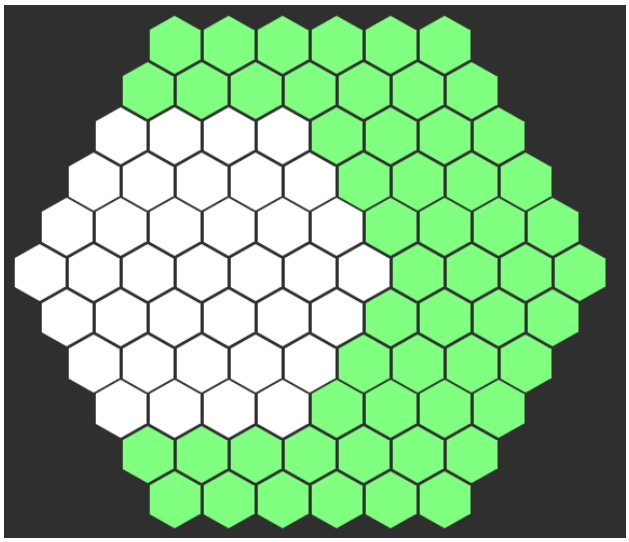


這樣就完成所有歸約了。

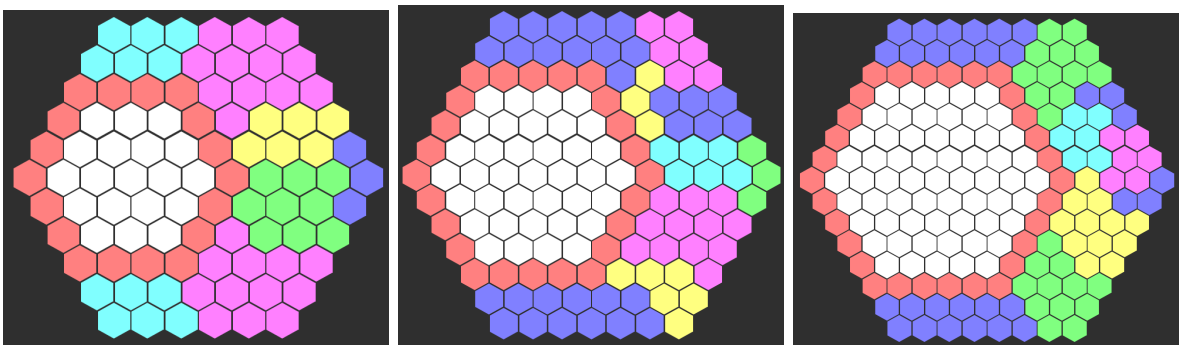
6.10 另解

以下提供一個無視幾乎所有子任務的解，感謝驗題者 Fysty 提出。

注意到對任何 $n \geq 6$，我們都可以填滿下圖綠色區域：



詳細可以按照 n 除 3 的餘數分為三種 case：



而我們可以透過旋轉田地來完成歸約，讓田地大小 -2。當 $n \leq 5$ 時，就可以使用子任務一的方法完成。

7 p7 對號鹿座

本題改編自 IOI 2018 Day 1 p2 - Seats，該題的精美程度在當年讓選手們眼睛為之一亮，在前國手推廣後，已在現代臺灣資料結構題的解題方法中成為經典，是一道極具啟發性的好題目。

當然，本題除了 Seats 本身給的思路外，還需要進一步的觀察才可以完成整道題目。在接下來的章節中，本文會盡可能的沿著子任務來從頭引導出整道題目的做法，使讀者不用上網先回顧 Seats 的解法再閱讀本文。

7.1 子任務 1

這是一筆看起來想讓暴力通過的子任務，旨在讓參賽者理解題意。當然，本題所謂的「暴力子任務」也是得先了解何謂「曼哈頓凸」的圖形，才方便做判斷。

如果想要判斷所有學生最多是被分成幾個梯次進場的，那想像上就是將題目轉換成以下形式：

- 一開始，所有 $N \times M$ 的格子都是白色的。
- 當某一個梯次總共 X 位學生入場後，編號 $1 \sim X$ 的學生入座，此時我們把這些學生坐的位置塗上黑色。
- 那麼，這些黑色格子必須形成曼哈頓凸的圖形。

也就是說，對於 $X = 1 \sim N \times M$，我們希望檢查塗黑 $1 \sim X$ 後到底是不是曼哈頓凸的圖形，而滿足其為曼哈頓凸圖形的 X 數量即是答案，畢竟如果圖形都滿足了，理當要選進來當成一個梯次，才能讓梯次最多。

因此現在的問題就是「到底如何檢查圖形為曼哈頓凸」，複習一下曼哈頓凸的定義：

定義 7.1. 曼哈頓凸
一個二維表格上的連通塊滿足「曼哈頓凸」的性質若且唯若連通塊中的任兩個點它們在連通塊中的最短距離即為兩個點之間的曼哈頓距離。

在子任務 1 中，如果對於每一筆詢問，需要窮舉 NM 種不同的 X，每一種 X 又想按照定義花 $O((NM)^2)$ 窮舉兩個點，每次也得花 $O(NM)$ 計算最短路徑，這樣可是得花上 $O((NM)^4)$ 才能處理一筆詢問，顯然再乘上 Q 後這個的複雜度是完全無法接受的，

我們先來看看曼哈頓凸有什麼樣的性質，如果我們有一種至少能在 $O((NM)^2)$ 的時

間檢查是否為曼哈頓凸的方法，時間複雜度就可以優化至 $O((NM)^3Q)$，會比較有機會通過。

事實上有如下的引理：

引理 7.1.
一個二維表格上的黑色區塊滿足「曼哈頓凸」的性質若且唯若：

- 所有黑色區塊四方位連通。
- 對於包含黑色區塊的每個橫列（row），其內部的黑色區塊為一連續區間。
- 對於包含黑色區塊的每個直行（column），其內部的黑色區塊為一連續區間。

證明.
我們先證明以上引理的必要性。首先連通是最基本的條件，在連通的情況下，若有任何橫列或直行內的黑色區塊形成兩個以上的連續區間，那麼對於相鄰的兩個黑色區間，他們的端點之間無法透過該橫列或直行內部互相來往，但若要走出曼哈頓距離長度的路徑，這一段路是唯一可能的路徑，因此形成連續區間是必要條件。

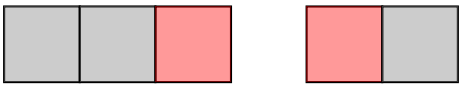


Figure 19: 一個橫列有兩個連續區間時，他們的端點之間將無法透過直接連線走出曼哈頓距離長度的路徑。

接著我們證明引理的充分性。對於任兩個區塊內的格子，由於他們是連通的，考慮他們的任何一條最短路徑，若不失一般性假設該兩點為「左下右上」的形式，則該路徑只要不斷的向右、向上，就可以形成一長度為曼哈頓距離的路徑。

我們假設沒有這樣的路徑存在，那個這個最短路徑肯定執行過向左或向下的操作，這裡不失一般性假設該路徑在格子 A 執行了向左的操作，由於最終該路徑必須抵達右上的位置，所以肯定存在與 A 同一直列的另一格子 B 被該路徑經過，此時因為同一直行內的黑色區塊需為連續區間，這表示 A 和 B 中間的格子也必須為黑色格子，那麼我們便可以直接將該路徑內部從 A 和 B 之間往返的這一段路徑用他們的直接連線連起來，即獲得更短的路徑，造成矛盾。因此，形成連續區間是充分條件。

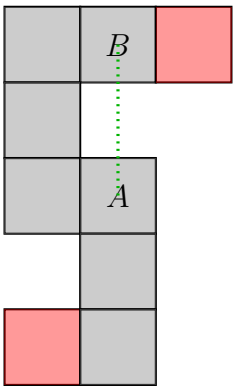


Figure 20: 對於任何一條在兩個紅色格子之間行走卻比曼哈頓距離還長的路徑，肯定能找到拐出如同上圖中 A 到 B 的空隙，並利用同一橫列或直行的黑色格子皆連續的性質將其連線起來。

□

既然我們都有引理 7.1 了，那麼檢查一個圖形是否為曼哈頓凸，就照著該引理的敘述檢驗就好，相信檢查連通、區塊為連續區間都是在 $O(NM)$ 時間內能輕鬆辦到的事。因此，總共的時間複雜度甚至可以到 $O((NM)^2Q)$，完全足以在時限內通過。

7.2 子任務 2

在 $N = 1$ 的情況下，圖形退化成一橫列，而當然，曼哈頓凸的圖形也自然退化成一條黑色的連續區間。

不過為了應付交換操作，我們勢必得想到一個良好處理一筆詢問的方式，才有辦法在每次詢問更新後能快速回答出所求。

回憶一下我們子任務 1 在做的事情：對於每一筆詢問，窮舉學生編號 X，對於每一種 X 都花 $O(NM)$ 的時間檢查曼哈頓凸的性質。

不覺得每次都花 $O(NM)$ 很虧嗎？想像我們窮舉 X 的過程是從 $1 \sim N \times M$ 窮舉上去，那這樣其實圖像化看的話，就是在一格一格的把格子按照編號由小到大塗黑而已。如果每塗黑一格都能 $O(1)$ 檢查性質是否還滿足，感覺上就會比較有優化下去的可能性。

不過要一口氣處理曼哈頓凸的圖形還是稍顯複雜，這裡我們就來先思考簡單的情況，也就是子任務 2 的「只是判斷是否為一個區間」的狀況。

對於一段黑色區塊，要判斷他們是否是連續區間，一個很直接的想法是維護最左、最右的黑色區塊位置，如此一來，只要判斷他們之間的距離是否為當前總個數 -1 就好。不過這個方法真的適合用來處理後續的交換操作嗎？要知道，每次交換操作我們需要回答的是「考慮所有 X 後的情況」，如果要一口氣維護「對於所有 X，是否兩端的區塊距離為 $X-1$」的話，看起來非常困難，我們勢必得換一種判斷方法。

這裡，我們用上的是 IOI 2018 Day 1 p2 Seats 厲害的想法：

引理 7.2.
對於一段長度為 M 的白色橫列，在上面塗上一些黑色區塊之後，可以用下列的方式判斷其是否為一段連續區間：

- 將橫列的兩側各延伸出一格白色格子，則考慮所有相鄰的格子對，顏色為「一黑一白」的格子對必恰有兩對。

進一步地：

- 當黑色區塊形成的圖形不為連續區間時，顏色為「一黑一白」的格子必定超過兩對。

證明.
對於每一段連續的黑色格子，其左右兩端皆為白色格子，這會直接製造兩對「一黑一白」的格子對。因此，引理直接成立。 □

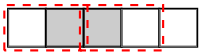


Figure 21: 連續的黑色區間會有恰好兩對「一黑一白」的格子對。

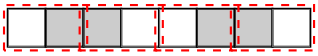


Figure 22: 超過一段的連續黑色區間會有超過兩對「一黑一白」的格子對。

引理 7.2 能帶給我們什麼好處呢？不是只是有另一種在多塗黑一個格子後 $O(1)$ 的判斷方式嗎？Seats 厲害的地方就在於，他將這個「塗黑格子」的過程看成 $1 \sim N \times M$ 個時間點，進一步得到以下觀察：

- 對於一對相鄰的格子對，他們的顏色組成的時間點為一段連續區間。

更準確地說，假設一對相鄰的格子對內的數字為 $l < r$，那麼在時間點 $[l, r)$ 之間，這對格子都是一黑一白的！當然，對於兩端點外加的白色格子，我們可以讓他們的編號為 $N \times M + 1$，就可以讓兩端點有相同的性質。

這讓我們有一種非常酷炫的方式可以計算一筆詢問的所求：

- 對於每一對相鄰的格子對，將他們形成一黑一白的時間區間拿出來進行「區間加值」。
- 則，考慮所有時間點，那些「被加恰好兩次」的時間點個數即為所求。

這種「透過小區塊的影響量辨識整個圖形」的技巧，從被 IOI 2019 的臺灣國手們稱作「pattern 維護法」，也就是將這個「相鄰格子對」稱作 pattern，透過這些 pattern 就能夠知道整個圖形是否有我們要的性質了。

所以這有多酷？看看每次交換兩個編號後會發生什麼事？

- 至多只有四對相鄰格子對的時間區間會被更改。

因此，對於每次交換操作，我們只需要更改這四對格子的時間區間，也就是進行至多八次的「區間加減值」操作，再查詢有幾個 2 就好了！

不過等等，「區間加減值、詢問有幾個 2」真的能做嗎？別忘了引理 7.2 的附加敘述：

- 當黑色區塊形成的圖形不為連續區間時，顏色為「一黑一白」的格子必定超過兩對。

也就是說，

- 其他形成非連續區間的時間點都會滿足他們「被加超過兩次」。

因此，我們要維護的 2 一定會是整段時間點的最小值（畢竟時間點 1 肯定是合法的），而「最小值個數」正是線段樹能好好維護的資訊！這也是為什麼這招「pattern 維護法」會在當年讓一些競賽選手感到如此佩服的原因：原來只要找到足夠小巧的 pattern，讓其有「修改單一格子後，影響到 pattern 數量夠少」，就能夠只處理那些被影響到的 pattern，達成整體上快速的維護！

最後做個整理

- 我們需要一棵支援「區間加減值、詢問最小值個數」的線段樹。
- 在一開始，對於每一對相鄰的格子對（包含兩側多延伸的兩個格子，其編號皆為 $N \times M + 1$），假設他們的編號包含 $l < r$，則對區間 $[l, r)$ 進行區間加值。
- 對於每次交換操作，會影響到至多四對相鄰格子的區間，妥善處理後進行至多八次的「區間加減值」操作。
- 每次要回答詢問的答案時，直接回報線段樹的最小值個數，該值即為當下的答案。

整體複雜度為 $O((NM + Q)\log(NM))$，非常良好。若妥善預處理則可以進一步達到 $O(NM + Q\log(NM))$，但並非必要。

7.3 子任務 3

現在我們要來維護曼哈頓凸的性質了，而根據前面的說法，就是要找到屬於曼哈頓凸圖形的「pattern」，使得修改單一格子後，pattern 的影響量夠小，才能夠做出快速的維護。回顧一下引理 7.1，一個二維表格上的黑色區塊滿足「曼哈頓凸」的性質若且唯若：

- 所有黑色區塊四方位連通。
- 對於包含黑色區塊的每個橫列（row），其內部的黑色區塊為一連續區間。
- 對於包含黑色區塊的每個直行（column），其內部的黑色區塊為一連續區間。

感覺非常簡單？對於每個橫列和直行，既然他們都要是連續區間，我們就都對他們維護子任務 2 做的事情。只是要注意，所有的橫列和直行的線段樹都必須要共用，這是因為我們得仰賴線段樹的最小值個數才能得到答案，另一方面，反正只要是不連續的時間點，其被加的次數就會大於 2，也因此，對於某個特定的時間點 X，假設當前有 R 個橫列包含黑色區塊、有 C 個直行包含黑色區塊，那其為曼哈頓凸的條件應該會是「該時間點被加恰好 $2(R+C)$ 次」才對。

恩……？怎麼感覺哪裡怪怪的？我們來一一看看問題在哪。

7.3.1 無法得知 R 與 C 的確切值

如果我們照樣維護最小值個數的話，由於時間點 1 的被加的值肯定是 4，這讓所有時間點被加的次數也只能是 4 才會被我們算到，而當 $R+C$ 變大時，這顯然會讓我們的答案錯誤。

不過既然每個有黑色區塊的直行和橫列都會貢獻 2 的加值，我們不如方便一點，讓那些「還沒有黑色區塊」的直行和橫列也在他們「還沒有黑色區塊的時間區間」無條件給予 2 的加值。如此一來，多了 $N+M$ 個區間加值，但我們讓符合曼哈頓凸性質的加值量被固定提升到了 $2(N+M)$，此時便能直接查詢最小值個數來得到所求。

7.3.2 別忘了圖形要連通！

這也是我們卡在子任務 3 的最大瓶頸，圖形要連通。既然 $Q=0$，那這裡我們當然能先用一個併查集維護連通性就好，待滿分解的章節我們再來揭曉如何處理這個情況。

至此，我們便能夠在 $O((NM)\log(NM))$ 的時間通過子任務 3。不過其實如果只是要維護區間是否連續，回到最原始的「一一把格子塗黑」的做法當然是可以直接 $O(1)$ 維護

的，這樣也當然能在 $O(NM\alpha(NM))$ 的時間內通過子任務 3，不過這個做法不好延伸到滿分解，因此我們不以該做法當成子任務 3 的主要做法。

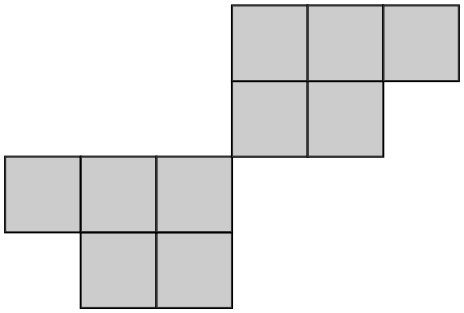


Figure 23: 一個即使所有行列內的黑色區塊都為連續區間，依然不為曼哈頓凸的例子。

7.4 滿分解

結合子任務 2 和 3 的想法，我們發現對於每次交換操作，都只會影響到至多

- 八對相鄰格子的一黑一白時間區間。
- 兩個橫列的沒黑色區塊時間區間。
- 兩個直行的沒黑色區塊時間區間。

也因此，我們只需要至多二十四個區間加減值就能完成維護。但這樣當然沒有解決問題，因為最困難的問題就是：這 X 個格子到底有沒有連通？

要知道，要一口氣對於所有 X 維護是否連通是非常困難的，如果把問題想成在一般圖上面的黑點的話，乍看之下並沒有什麼好做法。

所以，「圖形是 $N \times M$ 表格上的一些黑色格子」這個事實就變得相當重要了。有什麼性質是在這個條件下、又跟連通塊數量有關的呢？沒錯，就是大家小時候或多或少都有聽過的「平面圖歐拉公式」

引理 7.3. 平面圖歐拉公式
對於一張平面圖，考慮其頂點個數 V、邊數 E、面數 F 和連通塊數量 C，有恆等式：

$$V - E + F = C + 1$$

這裡就不提供該公式的證明了，有興趣的讀者可以去 維基百科的條目 [3] 上做進一步的了解。

[3] https://en.wikipedia.org/wiki/Planar_graph#Euler's_formula

要怎麼把這個公式應用到這題上呢？一個錯誤的想法是把格子和格子之間的交點當成平面圖上的點，下圖的左邊顯示了一個會導致出錯的範例，其理由是因為這會讓斜角相鄰的區塊被當成同一個連通塊，造成檢驗失敗。因此，正確的做法是將每一個格子當成一個點，並以實際上四方位的邊來當成圖上的邊，如下圖的右邊所示。

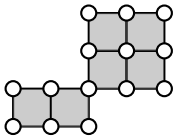


Figure 24: 失敗的平面圖對應方法，共有 14 個點、19 條邊和 7 個面，計算後僅得到一個連通塊。

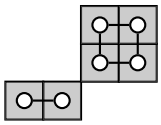


Figure 25: 正確的平面圖對應方法，共有 6 個點、5 條邊和 2 個面，計算後得到兩個連通塊。

不過面要怎麼定義呢？這樣看起來我們似乎不知道怎麼維護圍成一大圈的面，好像又回到原點了。不過別急，我們先來看看如果要使用歐拉公式的話，具體應該要怎麼應用到我們的資料結構上。

如果想要計算連通塊個數，那具體的公式應為

$$C = V - E + F - 1$$

此時，我們知道

- 針對點，對於每個格子，我們可以知道他被塗黑的時間點，也就是編號 x。那麼這個格子會在時間 $[x, N \times M]$ 之間讓點的數量增加 1。
- 針對邊，對於每對相鄰格子，我們可以知道兩個格子都被塗黑的時間點，也就是他們包含的最大編號 x。那麼這對格子會在時間 $[x, N \times M]$ 之間讓邊的數量增加 1。

點跟邊的維護看起來都不會造成問題，因為對於單一格子的修改，也只會造成五個這些時間點區間的修改而已。

最關鍵的面呢？我們不妨大膽的做下列的簡化：

- 我們只考慮當 2×2 方格全被塗黑時，由這四個方格的點所圍出的面。

如果用這樣去想，那維護面的個數就簡單了：考慮四個格子內的最大編號 x，那麼在時間 $[x, N \times M]$ 之間，這四個格子會讓面的數量增加 1。

這時，如果把點、邊和面的時間區間都丟進線段樹內，那造成的貢獻就會是 C，注意到這裡少 1 是因為在平面圖對面的定義下，整個外圍也是一個面，而我們並沒有把這個面的值加進線段樹內。我們要的情形是 $C = 1$，也因此，只要在加值的總量為

$2(N+M)+1$ 時，就會是一個合法的曼哈頓凸圖形！反過來說，如果連通塊變多，歐拉公式的貢獻也會比 2 還大，這就會讓總加值量變大，進而迴避最小值個數的統計。

可是不對啊？我們只考慮 2×2 格子所形成的面的話，那如果遇到更大的面不就少算了嗎？少算會造成加值量變小，如果變小的量恰好讓加值量變成最小值、甚至更小，這樣怎麼可能是對的？

很巧的是，這其實不會出現問題！讓我們來看看以下引理：

引理 7.4.
如果我們說一個 2×2 格子形成的面之面積為一，那麼對於任何面積大於 1 的面，按照我們之前執行過的所有加值操作處理後，他實際上會影響的總加值量至少為 4。進一步地，這些加值量與預期要影響曼哈頓凸的總加值量（撇除掉面的加值量）無關。

證明.
那麼對於任何面積大於 1 的面，在這個面內部肯定存在一個未被塗黑的白色格子，考慮這個白色格子在其所在的橫列左右延伸，此時兩側都一定會分別撞到各一個黑色格子，進而形成兩對一黑一白的相鄰對；同理，在其所在的直行上下延伸後，兩側也都會撞到各一個黑色格子，進而形成兩對一黑一白的相鄰對。

因此，這四對相鄰對在我們的加值操作都會增加至少 4 的非必要增加量。 □

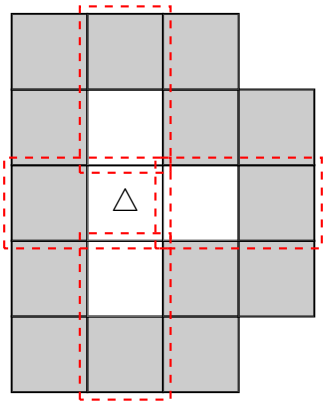


Figure 26: 如果有一個面的面積大於一，則其中間的白色格子，以打三角形的格子為例，往上下左右延伸一定會遇到各一對一黑一白的相鄰對。

有了引理 7.4，對於一個面積大於 1 的面，我們想像上可以讓他從這個加值量搶 1 過來，當作是維護連通塊數量用的值，至於剩下的增加量，就會讓該時間點的總加值量變大，使得該時間點掉出最小值的個數統計量，避免我們的答案變大！聽起來很詭異，但他確實有在運作。

最後，我們來分析看看對於每次交換操作會影響到幾個時間區間。

- 八對相鄰格子的一黑一白時間區間。
- 兩個橫列的沒黑色區塊時間區間。
- 兩個直行的沒黑色區塊時間區間。
- 兩個點的時間區間。
- 八個邊的時間區間。
- 八個面的時間區間。

總共至多六十次區間加減值操作，儘管總時間複雜度依然是 $O((NM+Q)\log(NM))$，常數也還是非常的巨大。不過，在 $N \times M \leq 10^5$、時限六秒的條件下，足以通過了！

7.5 維護連通性的另解

覺得使用歐拉公式太複雜了嗎？在眾多選手的捧場下，筆者從選手的解答發現其實有更簡潔的 pattern 可以維護曼哈頓凸的性質。

我們再次參考 IOI 2018 Day 1 p2 Seats 的想法，原本在 Seats 的滿分解要維護的圖形是矩形，而要維護矩形的話，在原題官方解答所使用的 pattern 是這樣的：

- 我們在 $N \times M$ 表格的外圈加上一圈白色格子後，考慮每個 2×2 方格，則黑色區塊為矩形若且唯若「三白一黑」的組合有恰好四個。

若想在本題應用同樣的思路，可以試著畫出幾個曼哈頓凸的圖形，相信很快就會發現以下幾個無法只靠「三白一黑」驗證的例子：

Figure 27: 失敗例子一。

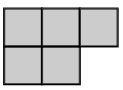


Figure 28: 失敗例子二。

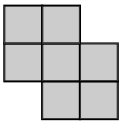


Figure 29: 失敗例子三。

不過這幾個例子，我們似乎可以稍作修改 pattern 的維護方式來得到新的猜想：

- 考慮對於每個 2×2 方格，維護「三白一黑」減去「三黑一白」的組合數。

此時，下方的引理便因此誕生：

引理 7.5.
對於任意圖形，只要對於所有包含黑色區塊橫列，其黑色區塊在橫列內部皆形成連續區間，則其「三白一黑」減去「三黑一白」的組合數至少為 4。

更進一步地，若該黑色區塊連通，其「三白一黑」減去「三黑一白」的組合數恰為 4。

證明.
對於任兩個相鄰的橫列，我們討論這兩列的兩個黑色區間討論的五種可能情況，並計算這兩列內對「三白一黑」減去「三黑一白」組合數的貢獻：

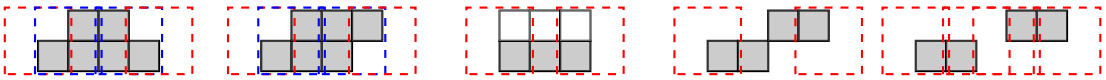


Figure 30: 五種黑色區間對的情況。

- Case 1：其中一個橫列的黑色區間完全包含另一個黑色區間。
  可發現其貢獻為 0。
- Case 2：兩個橫列的黑色區間有交集，但沒有包含關係。
  可發現其貢獻為 0。
- Case 3：其中一個橫列沒有黑色區塊。
  可發現其貢獻為 2。
- Case 4：兩個橫列的黑色區間沒有交集，且距離為一。
  可發現其貢獻為 2。
- Case 5：兩個橫列的黑色區間沒有交集，且距離超過一。
  可發現其貢獻為 4。

總和上面五種情況，計算所有相鄰橫列之總和後，可得只要存在任意黑色格子，其組合數至少為 4。進一步地，對於一個連通的圖形，其貢獻只來自最上方和最下方的橫列，因此組合數恰為 4。

□

有了引理 7.5，我們可以再做進一步觀察。注意到我們僅維護「每一行、每一列黑色連續」會失敗的原因在於圖形可能會不連通，但這個不連通的條件稍微更強一點，也就是

- 對於兩個不連通的圖形，他們的行和列不會有交集。

因為有交集的話，就會被「同一行同一列不連續」判掉。

這也表示，對於兩個曼哈頓凸的圖形，如果要拼在一起又能躲過我們原先維護的資訊的話，他們最靠近的極限也只能有兩格黑色格子共享一個角落而已——這表示，兩個曼哈頓凸的圖形接在一起，頂多只能對消對方的一塊「三白一黑 2×2 方格」！換句話說，兩個曼哈頓凸的圖形行列不相交地接在一起，一定會讓「三白一黑」減去「三黑一白」的組合數超過 4。

因此我們便得到結論：只要在維護好「每一行、每一列黑色連續」的前提下，多維護「三白一黑」減去「三黑一白」的組合數，就可以得到整張圖形是曼哈頓凸的加值量恰為 $2(N+M)+4$、不是的情況只會更大的結論。而維護「三白一黑」減去「三黑一白」的組合數，對於一次交換操作，只會多影響到十六個時間區間，比歐拉公式的時間區間少了兩個，常數略勝。

這個做法有趣的地方在於，如果只單看三白一黑減去三黑一白個數為 4 的圖形的話，筆者並沒有發現什麼簡單的敘述方式可以概括這些圖形，例如以下的例子就是一個相減為 4 卻無法一言蔽之的圖形：

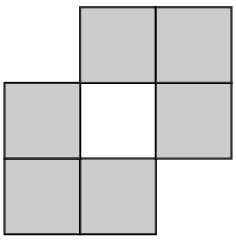


Figure 31: 長相奇怪，卻擁有恰為 4 的「三白一黑」減「三黑一白」組合數

這同時也表示他與歐拉公式相比還是少維護了一些資訊，例如若我們將原題升級為「每一階段結束可能是一至兩個曼哈頓凸連通塊」的話，這個做法大概就行不通了。

7.6 後話

這題是筆者在第一年 IOICamp 在課程中引入 pattern 這個主題時 (2020 年) 就有的點子，但由於它的實作量龐大，加上 pattern 這個概念早已因為 IOICamp 而流行，這題的價值也只剩下一個跟歐拉公式的結合，變得不太好放進其他比賽，才會出現在 NTUCPC Challenge。不過歐拉公式可以直接應用在 pattern 上面真的是一個很有趣的事實，希望能給沒有聽說過該做法的讀者一點感到驚喜。

另外，本題題解是是耗時（a.k.a. 拖延）最久的題解，筆者深感抱歉，感謝讀者們長久以來的等待。

8 p8 無鹿可退

先重新描述一下問題：我們有一張 N 點 M 邊的圖，每一條邊都有一個可以建造的時間區間，我們的目標是選出一個邊的子集，使得我們可以安排它們的建造時間（同時只能建一條邊，一條邊要建造一天的時間），並且最小化「需要再多加幾條邊才能讓圖連通」，很明顯地這個「需要多加幾條邊」就是 $N-$ 已有的連通塊數量，所以目標其實是最大化連通塊數量。既然如此，要是我們的邊集不是一個森林，那我們可以取生成森林，反正還是可以安排時間，連通塊數量也不會變少，因此我們的答案總是可以取一個森林，接下來就當成我們要取一個**有盡量多條邊的森林**。

8.1 Scheduling

子題 2 和子題 3 分別是「怎麼取都可以安排時間」和「怎麼取都是森林」，其實子題 2 根本就只要全部輸出 1 就會過，連找生成樹都不用，畢竟題目本身沒叫你最少化邊數，而子題 3 是個經典問題：給你 M 項工作，每個工作可以做的時間都是一個區間，一個工作要花一天完成，你同時只能做一項工作，求你最多可以做幾項工作。這個問題我們接下來稱之為 scheduling。

一個 greedy 的作法是：掃描線，從第一天開始看，在每一天我們要決定現在要做什麼工作，而這個工作就是「目前可以做、還沒做的工作之中，deadline 最早的那一個」，這樣可以 $O(M \log M)$ 做完。

證明一下正確性，令 $f(E, d)$ 代表「E 是所有工作的集合，但我們只有第 d 天和之後可以工作，至少可以做幾項工作」。令 d' 是第 d 以後第一天有工作可以做的日期，那麼 $f(E, d) = f(E, d')$，所以我們不失一般性假設第 d 天是有工作可以做的，在第 d 天可以做的工作之中，令 deadline（就是 t_i）最小的那個工作是 k（有多個就任取），我們假設有一組最佳解 E^* 之中，在第 d 天做的工作不是 k，那麼：

- 如果 E^* 中沒有做 k，那直接把第 d 天改成做 k。
- 否則，如果 E^* 中第 d 天沒有做事，那把 k 改到這天來做。
- 否則，假設第 d 天做的工作是 j，我們知道 $s_j \le b_j = d < b_k \le t_k \le t_j$，其中 b_j, b_k 分別是工作 j, k 在 E^* 中被建造的日期，因此我們可以直接把它們的建造日期交換。

以上三種狀況都不會讓有做的工作數變少，因此肯定有一組最佳解，在第 d 天是蓋 k 的，在此之後這個問題就變成了 $f(E \setminus \{k\}, d+1)$，考慮對 E 的大小數歸就可以證明正確性。

8.2 Matroid

接下來我們都會使用 matroid intersection，先幫大家複習一下 matroid。

定義 8.1. Matroid
令 E 為一個有限集合，我們稱一個二元組 $M = (E, \mathcal{I})$ 是一個擬陣（matroid），其中 $\mathcal{I} \subseteq 2^E$ 是一個**非空**集合，若滿足以下兩個條件：

- 遺傳特性（hereditary property）：若 $I \in \mathcal{I}$ 且 $I' \subsetneq I$，則 $I' \in \mathcal{I}$。
- 擴充特性（augmentation property）：對於 $I_1, I_2 \in \mathcal{I}$ 滿足 $|I_1| < |I_2|$，存在 $e \in I_2 \setminus I_1$ 使得 $I_1 \cup \{e\} \in \mathcal{I}$。

$\mathcal{I}$ 中的集合 I 被稱作獨立集。

在這題裡面，matroid 的元素集合 E 就是題目給的邊集，每個邊除了是圖上的一條邊以外，也是一項工作，我們的目標是選一個 E 的子集 I，滿足 S 在圖上是森林，且它們在 scheduling 問題中是可以排程的。這兩個條件各自是一個 matroid，我們叫作 $M_1 = (E, \mathcal{I}_1)$ 和 $M_2 = (E, \mathcal{I}_2)$，前者是常見圖擬陣（graphic matroid，就是 $I \in \mathcal{I}_1$ 若 I 在圖上是一個森林），後者我們接下來叫它 scheduling matroid，$I \in \mathcal{I}_2$ 若 I 在 scheduling 問題中可以排程。

引理 8.1.
Scheduling matroid $M_2 = (E, \mathcal{I}_2)$ 確實是 matroid。

證明.
檢查一下定義：

- Hereditary property：如果 I 可以排程，那它的子集 I' 當然也可以排程。
- Augmentation property：對於兩個可以排程的集合 I_1, I_2 滿足 $|I_1| < |I_2|$，考慮它們的各自一組排程的方法，I_2 中一定有一些邊 e 的建造時間，I_1 在那個時間是沒有做事的，要是 I_1 也有蓋 e，那就可以把 I_1 的 e 直接移到那個時間點蓋。
  因此，存在各自一種排程的方法，使得對於所有 I_1, I_2 的共同元素，在 I_2 建造的時候，同時 I_1 是有在做事的。如果有一條邊 e 在 I_2 裡建造的同時，I_1 沒有做事，那麼我們知道 $e \notin I_1$，而且我們可以直接把 e 加進 I_1 裡，就在這個時間點蓋，得出 $I_1 \cup \{e\} \in \mathcal{I}_2$。因為 $|I_2| > |I_1|$，所以 e 肯定存在。

□

目標就是找到最大的 $I \in \mathcal{I}_1 \cap \mathcal{I}_2$，所以就是要做 graphic matroid 和 scheduling

matroid 的 intersection。

為了讓接下來的敘述清楚一點，我們也複習一下擬陣交：

定義 8.2. 交換圖
對於兩個 matroid $M_1 = (E, \mathcal{I}_1), M_2 = (E, \mathcal{I}_2)$ 以及一個在它們交集中的獨立集 $I \in \mathcal{I}_1 \cap \mathcal{I}_2$ 的交換圖 $D_{M_1,M_2}(I)$ 是一張**有向**二分圖，左側的節點對應到 I 中的每個元素，右側的節點則對應到不在 I 中的每個元素。對於 $x \in I,\ y \in E \setminus I$，有 $x \to y$ 這條邊若 $(I \setminus \{x\}) \cup \{y\} \in \mathcal{I}_1$，有 $y \to x$ 這條邊若 $(I \setminus \{x\}) \cup \{y\} \in \mathcal{I}_2$，反正就是把 x 從 I 拿掉，再放進 y，如果在 M_1 獨立就有往右的邊，在 M_2 獨立就有往左的邊。

擬陣交的方法是：

1. 一開始令 $I = \varnothing$。
2. 令 X_1, X_2 為 $E \setminus I$ 的子集，在其中的元素代表加進 I 後，分別會在 M_1 與在 M_2 獨立。所以，X_1 隨時都是當下的 $\{e \in E \setminus I \mid I \cup \{e\} \in \mathcal{I}_1\}$、$X_2$ 隨時都是當下的 $\{e \in E \setminus I \mid I \cup \{e\} \in \mathcal{I}_2\}$，先計算一次 X_1, X_2。
3. 當 $X_1 \neq \varnothing \wedge X_2 \neq \varnothing$ 時，不斷重複：
   (1) 如果，存在 $e \in X_1 \cap X_2$，令 $I \leftarrow I \cup \{e\}$。
   (2) 否則，建交換圖 $D_{M_1,M_2}(I)$，找到一條從 X_1 到 X_2 的最短路徑 P。如果沒有路徑，就直接結束，不然的話，$I \leftarrow I \oplus P$。
   (3) 重算一次 X_1, X_2。
4. 最後的 I 就是答案。

8.3 $O(N^3 M \log N)$

我們都知道怎麼檢查一個集合 $S \subseteq E$ 在 M_1, M_2 分別是不是獨立了，檢查在 M_1 是否獨立要 $O(|S|)$ 的時間，檢查在 M_2 是否獨立要 $O(|S| \log |S|)$ 的時間，實際上 I 的大小最多就是 $N-1$，所以在做 matroid intersection 的時候，要檢查的 $|S|$ 最多就是 N，也就是說要檢查 e 是否在 X_1, X_2 裡或檢查交換圖上一條邊是否存在，需要 $O(N \log N)$ 的時間。硬是建一次交換圖需要 $O(NM) \times O(N \log N) = O(N^2 M \log N)$ 的時間，重算一次 X_1, X_2 要 $O(M) \times O(N \log N) = O(NM \log N)$ 的時間，3. 那個迴圈會跑最多 N 次，總時間複雜度是 $O(N^3 M \log N)$。

這個複雜度預期會過到 subtask 4。有一個偷吃步技巧是，其實你可以不用一開始就把交換圖整個建出來，而是在 BFS 找最短路徑的時候，想要走一條邊時（如果對面的點已經走過了那就不用檢查）才檢查有沒有那條邊存在，這個想法待會還會用到。這樣做可

以過到 subtask 5，我不知道這樣要怎麼卡掉也不知道複雜度究竟會不會其實是好的。

8.4 $O(N^2(M + N \log N))$

剛才做法的瓶頸是在需要建 N 次交換圖，建一次交換圖需要 $O(N^2 M \log N)$ 的時間，接下來我們要做的事情就是想辦法把這個時間降到 $O(N(M + N \log N))$。在建交換圖的時候，我們會嘗試去找「對於每個 $x \in I$，有哪些 y，滿足 $(I \setminus \{x\}) \cup \{y\} \in \mathcal{I}_1$ 或 $\mathcal{I}_2$」。對於 $I' \in \mathcal{I}_1 \cap \mathcal{I}_2$ 令 $g_i(I') = \{y \in E \setminus I' \mid I' \cup \{y\} \in \mathcal{I}_i\}$（像 X_1, X_2 其實就是 $g_1(I), g_2(I)$），我們就是想要找所有的 $g_1(I \setminus \{x\})$ 和 $g_2(I \setminus \{x\})$。如果我們有辦法在 $O(M + N \log N)$ 的時間算一個 $g_1(I')$ 和 $g_2(I')$，那我們就成功了。

找 $g_1(I')$ 很簡單，先把連通塊都找好，$y \in E \setminus I'$ 的兩端在不同連通塊那就在 $g_1(I')$ 裡，$O(N)$ 就做完了。

找 $g_2(I')$ 就麻煩一點了。考慮我們用 greedy 作法找出一個 I' 的 scheduling，然後，想像一下如果我們現在把一個 $y \notin I'$ 加進來，按照 greedy 作法，y 會在某個時間點 d_1 被蓋，在此之前蓋的所有東西，都和還沒加入之前的一模一樣，而原本在 d_1 蓋的邊，則會被「擠掉」，拿到之後的某個時間點 d_2 蓋，而那個時間點本來蓋的邊，也會被「擠掉」，拿到更之後的某個時間點 d_3 蓋……，直到擠到一個本來沒蓋邊的時間為止，或者有人被擠掉之後就蓋不了了。要是 y 加入之後還能蓋完所有邊，那我們就知道是有一連串的時間點 $d_1, d_2, \ldots, d_k$，y 可以在 d_1 蓋、本來在 d_1 蓋的邊可以在 d_2 蓋、本來在 d_2 蓋的邊可以在 d_3 蓋、……，最後 d_k 是一個本來沒有蓋東西的時間點，而只要存在這樣的一連串時間點，我們也知道 y 加入 I' 後還是獨立集。

令 $dp[d] =$ 要是本來時間 d 蓋的邊擠掉了，那還有沒有辦法全部蓋完，假設本來時間 d 蓋的邊可以建造的時段是 $[s, t]$，那 $dp[d] = \bigvee_{d' \in (s,t]} dp[d']$（注意因為 greedy 最佳子結構，我們並不在乎實際上 greedy 會把那條邊拿去什麼時候蓋，只要蓋得了就好），d 本來就沒蓋東西的話那就 $dp[d] = T$，而如果 y 可以建造的時間是 $[s_y, t_y]$，那能不能把它加進 I' 就是 $\bigvee_{d' \in [s_y, t_y]} dp[d']$。先做一次 greedy 之後，把時間離散化再倒著 DP 搭配個後綴和，就可以在 $O(M + N \log N)$ 的時間算好一個 $g_2(I')$，$O(N \log N)$ 的部分是來自做 greedy 和離散化。總時間複雜度是 $O(N^2(M + N \log N))$。

一些比較複雜的處理方法（例如真的去找擠掉之後會被擠到哪個時間點，當轉移點然後 DP[4]）可能會弄出 $O(N^2(N + M) \log N)$ 或 $O(N^2(N + M) \log(N + M))$ 之類的複雜度，實際上因為我做原題的時候比較笨，所以 subtask 5 的預期複雜度是這個，後來才想到可以這麼簡單地處理並得到不錯的複雜度。subtask 6 的預期複雜度是 $O(N^2(N + M))$，任

[4]Greedy 就是唯一轉移點的 DP

何有 log 的複雜度可能會被卡，但好好壓常應該有機會過。

8.5 $O(N^2(N+M))$

剛才在括號裡的 N 多帶 $\log N$ 是因為我們在同一輪之中，找了 $|I|+1$ 次的 $g_2(I')$（多一次是找 $X_2 = g_2(I)$），而每次找 $g_2(I')$ 我們都先用一次 greedy 重找 scheduling，不過注意到後面 $|I|$ 次，I' 都是某個 $I \setminus \{x\}$，所以要是我們有辦法先找出 I 的 scheduling，再想辦法把 x 拔掉，就可以不用帶那個 log。

同樣考慮 greedy 做出來的解會怎麼改變，剛剛加入一個 y 的時候是會把別的東西「擠掉」，那麼把一個 x 拔掉，就是會有東西「補過來」，假設 x 本來建造的時間是 d，那麼後面可能會有一個邊被搬來 d 蓋，這條邊其實就是「d 之後最早蓋的，可以在 d 蓋的邊」，而這條邊被搬過來後，同樣會再有一個邊來填補它的空位，直到之後再也找不到能填空位的邊為止。要找到來填補空位的邊非常簡單，只要暴力往後掃就好了，遇到能補空位的就停下來，因為下一個要填的空位剛好就是這次掃到停止的地方，所以這整個過程總共是 $O(N)$ 的。

這樣一來，時間複雜度就是 $O(N(N \log N + N(N+M))) = O(N^2(N+M))$。可以過到 subtask 6。

8.6 $O(N(N+M)\log N)$

什麼？居然可以有 $o(N^2M)$ 的複雜度！剛才我們一直需要 $\Omega(N^2M)$ 的時間是因為，我們一直想要先建出整張交換圖，但我們要對交換圖做的事情其實只有找 X_1 到 X_2 的最短路徑，也就是以 X_1 為起點 BFS，而 BFS 生成樹上只有 $O(N+M)$ 條邊而已，要是我們有辦法精準地找出 BFS 生成樹上的邊，就不需要建整張交換圖。方法是，當我們走到一個點 v 的時候，我們要去找「有哪些 unvisited 的點，有 v 到它的出邊」，可以在 $\tilde{O}$(數量) 的時間做到就成功了。

因為本來的方向比較難做，所以我們把所有邊都轉方向，因此現在左往右的邊是 scheduling matroid 的交換，右往左的邊是 graphic matroid 的交換，目標是從 X_2 走到 X_1，我們想做的事情有：

1. 在走到 $y \notin I$ 時，要找有哪些 unvisited $x \in I$，滿足 $(I \setminus \{x\}) \cup \{y\}$ 在圖上沒有環。
2. 在走到 $x \in I$ 時，要找有哪些 unvisited $y \notin I$，滿足 $(I \setminus \{x\}) \cup \{y\}$ 可以 scheduling。

先做第一個，I 本來是個森林，我們要知道的是「加上 y 之後，拔掉哪些 x 可以讓圖沒有環」，要是 y 加上去之後還是森林，那它就跟全部 x 有邊（但這代表 y 在 X_1 也就是我們的終點集合，所以 y 的出邊其實不重要），不然的話，y 加上去之後會出現一個環，這個環上的任何 x 拔掉後都會讓環消失，我們也一定得拔環上的邊，在 I 的森林上，y 兩端點之間的路徑上的邊，就是所有我們要找的 x，別忘了我們的時間必須是 $\tilde{O}(\text{unvisted } x \text{ 的數量})$，搭配 DSU 就可以做到 $O(\text{數量}\alpha(N))$。

第二個要做的事情是，「拔掉 x 之後有哪些 y 可以加進來」，前面我們做這件事的方法是找拔 x 的 scheduling 後重新 DP 一次，但我們現在頂多只能在每一輪開頭做一次 I 的 scheduling，其實我們不用那麼執著於特定的 x，畢竟我們現在是在 BFS，我們只要知道「有哪些 y 滿足，拔掉 visited 的 x 之中的任何一個，y 就可以加進來」，也就是在 $d \in [s_y, t_y]$ 中，有任何一個 $dp[d]$，在拔掉任何一個 visited x 之後可以變成 true，那 y 就是好的。

如果本來 $dp[d]$ 就是 true，那不管拔了什麼它肯定還是 true，不過拔掉一個 x 之後，可能有些本來是 false 的會變成 true。首先，拔掉 x 會造成 scheduling 改變，假設拔掉 x 後會遞補它空位的是 $c(x)$，所以會有一連串的人 $x, c(x), c(c(x)), c(c(c(x))), \ldots$ 被移動，在所有它們本來的建造時間 d，$dp[d]$ 都會變成 true，然後可能也會有其他 $dp[d]$ 被連帶影響。

怎麼處理這些連帶影響？重新描述一次我們現在面對的問題，我們有一個 visited x 的集合，我們要能往它裡面加入一個新的 x，與此同時，我們要維護以下兩種事件：

1. 有一個本來是 false 的 $dp[d]$ 變成 true 了。假設 i 是本來在第 d 天蓋的邊，$dp[d]$ 變成 true 就是 $dp[d+1..t_i]$ 有一個變成 true 的那一刻。
2. 有一個 y 變成好的，也就是 $dp[s_y...t_y]$ 有一個變成 true。

所以觸發事件的條件都是同一個形式：某個區間裡的 $dp[d]$ 有任何一個變成 true，然後我們會做的操作就只有把某個 $dp[d]$ 變成 true，第一種事件會導致把事件發生的 $dp[d]$ 設成 true，還有加新的 visited x 就是把 $x, c(x), c(c(x)), \ldots$ 的建造時間的 $dp[d]$ 變成 true，如果一個 x' 在 $x, c(x), c(c(x)), \ldots$ 之中，我們就說它發生了「新 x 造成的修改」，但注意到同個 x' 可能會遇到很多次新 x 造成的修改，不過只要它遇到過一次，所有 $c(x'), c(c(x'))$ 就會都被修改過了，因此要是 $x, c(x), c(c(x)), \ldots$ 中遇到一個已經有「新 x 造成的修改」過的人就可以停下來，這樣「新 x 造成的修改」次數就會是 $O(N)$，而 $dp[d]$ 變成 true 也只會發生在本來有蓋東西的時間點，數量同樣是 $O(N)$。總的來說，我們會有 $O(N)$ 個把某個 $dp[d]$ 變成 true 的操作，然後我們要去處理被觸發的事件。

到這裡作法就已經非常明確了：對時間 d 開線段樹（重要的時間點只有本來有蓋東

西的時間點，可以先離散化讓數量只有 $O(N)$ 個），對於會被區間 $[l, r]$ 觸發的事件，對線段樹區間修改 $[l, r]$，在終止節點上存這個事件，然後每次修改 $dp[d]$ 時，對線段樹單點修改 d，觸發路徑上節點存的所有事件，已經觸發過就略過。

這樣一來，加一個事件或做一個修改需要花費的時間就是 $O(\log N)$，總共的事件數量是 $O(N + M)$，一輪的時間複雜度是 $O((N + M)\alpha(N) + (N + M)\log N) = O((N + M)\log N)$，總時間複雜度就是 $O(N(N + M)\log N)$。

9 p9 鹿不敷出

原始目標為最小化剩餘的物品數量，以下為了較方便說明，我們將目標改為最大化被分配進福袋的物品數量。同時我們稱角型印章盒為物品 A，鹿角娃娃為物品 B，並不失一般性假設 $A \geq B$，同時令 $C = \max(N, M, A, B)$。

9.1 簡化問題

首先令 $G = \gcd(A, B)$，若 K 不是 G 的倍數的話答案顯然為 0，否則可以先將 A, B, K 都除上 G，之後就可以假設 $\gcd(A, B) = 1$。

引理 9.1.
滿足 $0 \leq X < B, 0 \leq Y < A, K - (AX + BY) \equiv 0 \pmod{AB}$ 的 X, Y 是唯一的。

證明.
假設存在兩組相異的 $(X, Y), (X', Y')$ 滿足 $(AX + BY) \equiv (AX' + BY') \pmod{AB}$，可得 $A(X - X') \equiv B(Y - Y') \pmod{AB}$。由於 $\gcd(A, B) = 1$，此式只有在 $A(X - X') \equiv 0 \pmod{AB}$ 時才會滿足，因此可得 $X \equiv X' \pmod{B}$ 與 $Y \equiv Y' \pmod{A}$。又由於 $0 \leq X, X' < B, 0 \leq Y, Y' < A$，$X = X', Y = Y'$，與假設矛盾。 □

而由 擴展歐幾里得演算法 [5] 可以知道 X, Y 的存在性，我們先用此演算法找到 X, Y，之後令 $Z = \frac{K - AX - BY}{AB}$，我們可以寫下 $K = ZAB + XA + YB$。也就是說若我們定義「一組物品 A」為 B 個物品 A，「一組物品 B」為 A 個物品 B，一個福袋會包括 X 個物品 A、Y 個物品 B 與 Z **組**物品（哪個種類皆可）。由此也可以先判掉一個簡單的 case：若 $Z < 0$ 則答案為 0。

9.2 子任務 1

我們嘗試枚舉福袋的數量，定義 $f(\ell)$ 為若我們製作 ℓ 個福袋的情況下，最多能有幾個物品被放進福袋裡。首先，要能夠製作 ℓ 個福袋需滿足以下三個條件：

- $X\ell \leq N$
- $Y\ell \leq M$
- $\left\lfloor \frac{N - X\ell}{B} \right\rfloor + \left\lfloor \frac{M - Y\ell}{A} \right\rfloor \geq Z\ell$

[5] https://cp-algorithms.com/algebra/extended-euclid-algorithm.html

不滿足的話則 $f(\ell) = 0$。滿足的話我們總共需要 $Z\ell$ 組物品，且因為 $A \geq B$，優先選物品 B 的組會較優。由此我們可以推出：

$$f(\ell) = \min\left(\left\lfloor \frac{M - Y\ell}{A} \right\rfloor, Z\ell\right) \times (A - B) + BZ\ell + \ell(X + Y)$$

而我們只要嘗試計算 $f(1), f(2), \ldots, f(N + M)$ 的值並取 max 即可。每筆測資可以在 $O(C + \log C)$ 內做完。

9.3 子任務 2

在這筆子任務中，我們會分為兩個 case 討論：$A \leq \sqrt{C}$ 與 $A > \sqrt{C}$。

9.3.1 case 1

我們會沿續子任務 1 的作法並嘗試觀察更多性質。首先觀察能製作出 ℓ 個福袋的三個條件，可以發現這些在 ℓ 越大的時候都會越難成立，所以我們可以先二分搜出最大滿足三個條件的 ℓ，並稱它為 ℓ_{mx}。以下我們再分為兩種 case 討論函數 $f(\ell)$。

- $\left\lfloor \frac{M-Y\ell}{A} \right\rfloor \geq Z\ell$：我們可以先二分搜出滿足此條件的最大 ℓ，並稱它為 ℓ_{mn}。顯然可以發現 $f(\ell_{mn})$ 即為這種 case 的最大值。
- $\left\lfloor \frac{M-Y\ell}{A} \right\rfloor < Z\ell$：注意到

  $$f(\ell + A) - f(\ell) = -Y(A - B) + A(BZ + X + Y) = ABZ + AX + BY > 0$$

  所以在這種情況中，我們只要找到前 A 大的 ℓ 值並計算它們的 $f(\ell)$ 最大值即可。複雜度為 $O(A + \log C) = O(\sqrt{C} + \log C)$。

9.3.2 case 2

我們嘗試枚舉使用的物品 B 的組數，可以發現能放儘量多個福袋越好。而我們可以用 greedy 的方式在 $O(1)$ 算完最多可以用幾個福袋，算完後再代入函數 f 即可。因為物品 B 的組數 $< \sqrt{C}$，總複雜度為 $O(\sqrt{C} + \log C)$。

兩種 case 都可以在 $O(\sqrt{C} + \log C)$ 內做完，並能夠通過此子任務。注意在做擴展歐幾里得時就算使用 64-bit integer 還是可能會 overflow，可以使用 128-bit integer 等方法來避免。

9.4 子任務 3

接下來我們會嘗試優化子任務 2 case 1 的作法，用更好的複雜度解決 $\lfloor \frac{M-Y\ell}{A} \rfloor < Z\ell$ 的 case。

定義 $g(\ell) = \lfloor \frac{M-Y\ell}{A} \rfloor$，我們首先用二分搜找出最大的 ℓ' 使 $g(\ell') > g(\ell)$。接著我會證明兩個 Lemma，說明這種 case 的答案其實就是 $\max(f(\ell'), f(\ell_{mx}))$：

引理 9.2.
$\forall \ell' < \ell < \ell_{mx}, f(\ell) < f(\ell_{mx})$

證明.
對於所有滿足條件的 ℓ，$g(\ell) = g(\ell_{mx})$，所以 $f(\ell) < f(\ell_{mx})$。 □

引理 9.3.
$\forall \ell_{mn} \le \ell < \ell', f(\ell) < f(\ell')$

證明.
若 $g(\ell) = g(\ell')$，顯然可知 $f(\ell) < f(\ell')$，因此以下假設 $g(\ell) < g(\ell')$。我們再令 $h(\ell) = (M - Y\ell) \bmod A$。注意到 $Y(\ell' - \ell) = (g(\ell) - g(\ell'))A + (h(\ell) - h(\ell'))$。而且若 $h(\ell') \ge Y$，則 $g(\ell' + 1) = g(\ell')$，與 ℓ' 為最大滿足 $g(\ell') > g(\ell_{mx})$ 的數矛盾，所以 $h(\ell') < Y$。因此，我們可得：

$$Y(\ell' - \ell) + Y \ge (g(\ell) - g(\ell'))A$$

最後：

$$\begin{aligned} f(\ell) - f(\ell') &= (g(\ell) - g(\ell'))(A - B) - (\ell' - \ell)(ZB + X + Y) \\ &\le Y - B(g(\ell) - g(\ell')) - (\ell' - \ell)(ZB + X) \le Y - B < 0 \end{aligned}$$

□

總結來說，我們先用二分搜找出 ℓ_{mx}, ℓ_{mn}，再找出 ℓ'，最後輸出這三者的函數最大值即可，每筆測資可以在 $O(\log C)$ 的時間內做完。實作需要大數除法、取餘等操作，用 python 就可以解決一切煩惱。

9.5 後話

這是出題者不小心在團練中讀錯 這題 [6] 的產物，因為兩題其實差有點多，所以就沒放 source 了，其實把第一筆範例測資拿去搜尋還是可以搜到，雖然應該沒有用就是了。

[6] https://codeforces.com/gym/105143/problem/G

也感謝我的隊友 BurnedChicken 與 syl123456 提出了這題的子任務 2 的作法。

子任務 3 的產生純粹是個意外，第一天的時候看著一些人用著奇怪的解快要唬爛過全部的測資，我就有點擔心測資不夠強開始想要生更強的測資，然後想著想著發現我想構的很多個峰的測資好像沒辦法構出來，再仔細證明一下就發現更好的解了。也感謝 warner1129 同意收回他原先已經獲得的分數讓這個子題能夠正常的出現（不過過了幾個小時他還是拿滿了 orz）

不過這題的子任務 1, 2 測資其實還是沒很強，出題者想了很久還是不知道有沒有什麼好 pattern 能夠卡類似「對 $f(\ell)$ 直接三分搜，搜到範圍 ≤ 50000 之後暴力找 max」這種解，因為 $f(\ell)$ 的性質真的太好。對拍是能夠找到測資但只要把三分搜改成四分搜等細微操作就有很高機會可以繞過了。這個解也是為什麼最後子任務 3 是把 N, M, A, B, K 調大，而不是調大 T，反正大數什麼的用個 python 就能解決了對吧（?）

10 p10 壅壅鹿鹿

10.1 子任務 1

事實上這筆子任務可以直接查詢 YTP 的題解，然後就可以現場學習作法，連結 [7] 在這。這也是為什麼這筆子任務只有 1 分的原因。

詳細作法是考慮一個經典計算多邊形面積的方法：測量師公式 [8] 。只要將多邊形的頂點按照逆時針方向排好，假如說排成 $p_0, p_1, \ldots, p_{n-1}$，那多邊形面積就是

$$\frac{1}{2}\sum_{i=0}^{n-1} p_i \times p_{i+1}$$

where $p_n = p_0$，$a \times b$ 代表點 a 與點 b 的外積。

接著我們考慮把式子拆開來。考慮枚舉任兩點 a, b，並計算他們作為凸包上逆時針順序上的相鄰兩點的機率是多少。這其實相當單純：若有其他被選到的點在 $\overrightarrow{ab}$ 的半平面下方，那 ab 一定不會是凸包上相鄰的點。所以只要計算位於 $\overrightarrow{ab}$ 半平面下方的點數量就好了。這個可以很輕鬆的在 $O(N)$ 時間內算完，加上前面的枚舉總複雜度 $O(N^3)$。

10.2 子任務 2

我們嘗試優化前面的解。我們考慮枚舉點 a，之後我們想像有一條很長的直線通過點 a 並開始旋轉，旋轉的過程只要遇到一個新的點，就把他加入或移除半平面裡，這樣就可以用較好的複雜度算完每個 $\overrightarrow{ab}$ 的半平面點數量。具體的實作方法可以先以 a 為中心將其他點照極角排序好，之後就跟著那個順序掃過一遍即可。一個點 a 可以在 $O(N \log N)$ 時間內算完，加上枚舉總複雜度為 $O(N^2 \log N)$。

10.3 子任務 3

這題可以做到更好的複雜度有一個重點：這題只要輸出浮點數就好了！所以考慮前面的作法，我們先定一個足夠小的常數 K，並考慮所有半平面點數量 $\leq K$ 的 $\overrightarrow{ab}$ 點對。稍微估計一下會發現 $K = 80$ 就已經足夠「接近」精準的答案了。

[7] https://www.youtube.com/watch?v=VhCWWgCZbAo
[8] https://en.wikipedia.org/wiki/Shoelace_formula

再來 稍微查一些資料 [9] 與 稍微查一些 paper [10] 可以知道：

- 給定平面上任意 N 個點，恰有 K 個點的半平面數量有 $O(NK^{1/3})$ 種。
- 我們可以在 $O(NK(K^{1/3}+\log N))$ 的時間內得出這種所有的半平面。

因此我們可以在 $O(NK(K^{1/3}+\log N))$ 時間內完成這題，$K=80$ 要通過應該是綽綽有餘。

10.4 後話

出題者其實不會實作論文裡提到的方法，因此參考了 這份 code [11] 並稍微修改了一下。感謝這筆 submission 的主人 WiwiHo。

另外因為出題者也不知道怎麼好好生 50000 個點且沒有三點共線的測資，所以這題的測資都是從之前的題目偷來的，像是 這題 [12] 與 這題 [13] ，因此這題最後一筆子任務的測資可能沒有很強。

題外話，這題的測資輸出是用 $O(N^2\log N)$ 的解並花了兩天跑出來的。

[9] https://en.wikipedia.org/wiki/K-set_(geometry)
[10] https://dl.acm.org/doi/pdf/10.1145/160985.160994
[11] https://qoj.ac/submission/177422
[12] https://tioj.ck.tp.edu.tw/problems/1631
[13] https://qoj.ac/contest/1045/problem/5083