2022 年
花蓮高中
第一次模擬賽
題解

這不是矩陣乘法

by SorahISA

問題概要

請輸出所有滿足以下方程式且 $0 \le a,b,c,d,e,f,g,h \le 9$ 的解。

$$\begin{bmatrix} a & b \\ c & d \end{bmatrix} \times \begin{bmatrix} e & f \\ g & h \end{bmatrix} = \begin{bmatrix} 10a+e & 10b+f \\ 10c+g & 10d+h \end{bmatrix}$$

0

沒有子任務們

- 子任務 1 (100 分)：無額外限制

满分解

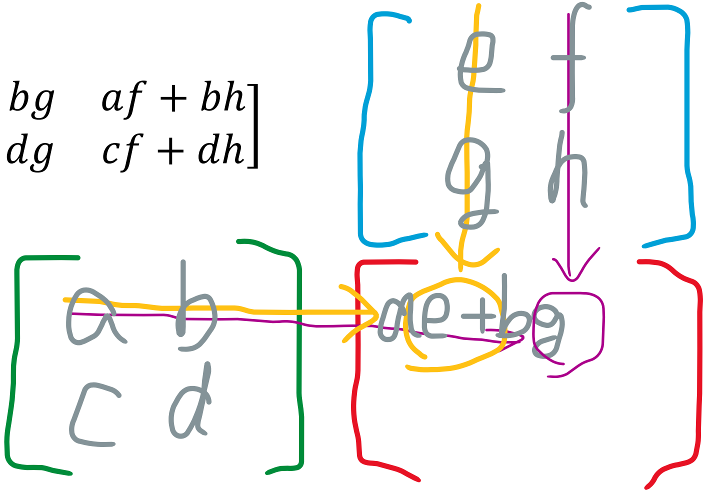
1 滿分解法

希望大家都知道矩陣要怎麼做乘法 OwO

$$\begin{bmatrix} a & b \\ c & d \end{bmatrix} \times \begin{bmatrix} e & f \\ g & h \end{bmatrix} = \begin{bmatrix} ae + bg & af + bh \\ ce + dg & cf + dh \end{bmatrix}$$

我的記法是這樣：

$$A \times B = C$$



最直接的方法就是用 8 層迴圈枚舉 $a \sim h$，再去判斷是不是一組解。

1 滿分解法

希望大家都知道矩陣要怎麼做乘法 OwO

$$\begin{bmatrix} a & b \\ c & d \end{bmatrix} \times \begin{bmatrix} e & f \\ g & h \end{bmatrix} = \begin{bmatrix} ae + bg & af + bh \\ ce + dg & cf + dh \end{bmatrix}$$

我的記法是這樣：

$$\begin{bmatrix} e & f \\ g & h \end{bmatrix}$$

$$A \times B = \begin{bmatrix} a & b \\ c & d \end{bmatrix} \begin{bmatrix} ae+bg & \\ & \end{bmatrix}$$

Accepted

得分：100 分

最直接的方法就是用 8 層迴圈枚舉 $a \sim h$，再去判斷是不是一組解。

1

滿分解法－實作

```cpp
#include <bits/stdc++.h>
using namespace std;
list<int> rng{0, 1, 2, 3, 4, 5, 6, 7, 8, 9};
int main() {
    for (int a : rng) for (int b : rng) for (int c : rng) for (int d : rng) {
        for (int e : rng) for (int f : rng) for (int g : rng) for (int h : rng) {
            if (
                    (a*e + b*g == 10*a + e)
                and (a*f + b*h == 10*b + f)
                and (c*e + d*g == 10*c + g)
                and (c*f + d*h == 10*d + h)
            ) cout << a << b << c << d << e << f << g << h << "\n";
        }
    }
    return 0;
}
```

1 滿分解法之二

如果汝認為本題 8 層迴圈計算時間會超過一秒，也可以在本地算完所有答案再輸出。

這種方法被稱為「本機打表」。

結論

2 結論

本題定位是全場最簡單的題目，但是可能是目前很多人沒學過矩陣乘法，做題率不甚理想。

矩陣乘法在演算法中還算蠻常見的，舉例來説，常見的費式數列 $\{0, 1, 1, 2, 3, 5, 8, 13, \dots\}$ 等線性遞迴式都能被表示成矩陣的乘積

$$\begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix} \times \begin{bmatrix} F_n \\ F_{n-1} \end{bmatrix} = \begin{bmatrix} F_{n+1} \\ F_n \end{bmatrix}$$

而又因為矩陣乘法有結合律，可以透過快速冪計算 $\begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix}^N$ 在 $O(\log N)$ 時間內得到 F_N 的值。

這不是簽到題

by SorahISA

0 問題概要

給一張由英文字母構成、大小為 30×25 的地圖。

汝要找到最近的 M 跟 P，並輸出他們的距離 +2。

此處，(x_1, y_1) 跟 (x_2, y_2) 的距離被定義為：

- 從 (x_1, y_1) 開始，
- 每次可以移動到滿足 $|x_1 - x| \leq 1$ 且$|y_1 - y| \leq 1$ 的 (x, y)，
- 要移動到 (x_2, y_2) 所需的最小步數。

0 子任務們

- 子任務 1 (50 分)：只有一個 P
- 子任務 2 (50 分)：可能有一或多個 P

其它輸入限制：

- 只有一個 M
- 至少有一個 P

满分解

1 滿分解法

三個步驟：找到 M 跟所有 P 的位置、計算每個 P 跟 M 的距離、輸出最小值 +2。

距離的計算方式是 $\max\{|M_x - P_x|, |M_y - P_y|\}$，因為每次移動都可以跟目標在 x 軸跟 y 軸上各接近一格。

1 滿分解法

三個步驟：找到 M 跟所有 P 的位置、計算每個 P 跟 M 的距離、輸出最小值 +2。

Accepted

距離的計算方式是 $\max\{|M_x - P_x|, |M_y - P_y|\}$，因為每次移動都可以跟目標在 x 軸跟 y 軸上各接近一格。

得分：100 分

結論

2 結論

這題難度定位跟「這不是矩陣乘法」相當，都只需要基礎的 for-loop 跟 if-else 的語法知識。

有些人可能看到中「每秒鐘朝八方位移動一格」的敘述就開始寫 BFS 了，有沒有發現其實整張圖都沒有障礙物，可以直線前進？

順帶一題，BFS 的時候要記得將拜訪過的節點設為 visited 才不會超時ㄛ！

這不是排序

by SorahISA

0 問題概要

給定陣列 $A = [A_0, A_1, A_2, \ldots, A_{N-1}]$，請求出將陣列經過如下函式「排序」後的結果。

```cpp
void NotSimpleSort(std::vector<int> &A) {
    int N = A.size();
    for (int i = 0; i < N; ++i)
        for (int j = 0; j < N; ++j)
            if (A[i] ^ A[j]) std::swap(A[i], A[j]);
}
```

0

子任務們

- 子任務 1 (30 分)：$N \leq 500$
- 子任務 2 (70 分)：$N \leq 100\,000$

其它輸入限制：

- 陣列長度 $1 \leq N \leq 100\,000$
- 數值範圍 $0 \leq A_i \leq 2^{31} - 1$

子任務 1

$N \leq 500$

1

子任務 I 的解法

那個函式的複雜度很明顯地是 $O(N^2)$，直接傳上去就可以通過本子題了。

1

子任務 I 的解法

那個函式的複雜度很明顯地是 $O(N^2)$，直接傳上去就可以通過本子題了

Accepted

得分：30 分

1 子任務 1 的慷慨

這是今天最送分的子題，理論上有看題目的都該拿到。

子任務 2

$N \leq 100\,000$

2 子任務 2 的解法

`if (A[i] ^ A[j])` 是什麼？在什麼時候才會成立？

^ 是位元互斥或（xor）運算子，可以當成二進位下不會進位的加法。若兩個數字在某個位元的值相異則 xor 之後該位的值則為 1、相同則為 0。

而 x XOR y 的值會是 true，也就是非零，只有在 $x \neq y$ 的時候。

那麼就可以將 `if (A[i] ^ A[j]) swap(A[i], A[j]);`

改寫成 `if (A[i] != A[j]) swap(A[i], A[j]);` 的形式了！

2 子任務 2 的解法

```
if (A[i] != A[j]) swap(A[i], A[j]);
```

ちょっと待って、不等於的時候都會換，等於的時候換不換又無所謂，那不就可以直接拿掉那個 if 了！？

沒錯，整個 if 判斷式在這裡根本不重要！我們只要知道所有元素兩兩交換後會長怎樣就好了。

這裡提供兩種方法，分別是觀察法以及證明法。

2 子任務 2－觀察法

因為兩兩交換這個操作跟陣列 A 的值無關，所以可以直接將 A 初始化成方便觀察的值，並利用題本上的暴力 code 去做做看各種狀況。

當 $N = 5$、$A = [1, 2, 3, 4, 5]$ 的時候，「排序」完會變成 $[3, 4, 5, 1, 2]$；

當 $N = 10$、$A = [1, 2, \dots, 10]$ 的時候，「排序」完會變成 $[3, 4, \dots, 10, 1, 2]$；

當 $N = 100$、$A = [1, 2, \dots, 100]$ 的時候，「排序」完會變成 $[3, 4, \dots, 100, 1, 2]$；

あ！看起來就是把 A_0 跟 A_1 移到陣列最後面而已嘛。

注意 $N = 1$ 邊界 case。

2

子任務 2 — 觀察法

因為兩兩交換這個操作跟陣列 A 的值無關，所以可以直接將 A 初始化成方便觀察的值，並利用題本上的暴力 code 去做做看各種狀況。

當 $N=5$、$A=[1,2,3,4,5]$ 的時候，「排序」完會變成 $[3,4,5,1,2]$；

當 $N=10$、$A=[1,2,\dots,10]$ 的時候，「排序」完會變成 $[3,4,\dots,10,1,2]$；

當 $N=100$、$A=[1,2,\dots,100]$ 的時候，「排序」完會變成 $[3,4,\dots,100,1,2]$；

Accepted

得分：100 分

あ！看起來就是把 A_0 跟 A_1 移到陣列最後面而已嘛。

注意 $N=1$ 邊界 case。

2 子任務 2－證明法

~~如果汝現在是在考 APCS 觀念題的時候遇到了這題，~~汝要怎麼確定剛剛得出的結論在任何情況下永遠是對的？這就得透過證明了。

對每個 i 分成 $j < i$ 跟 $j > i$ 左右兩種狀況（顯然 $\mathrm{swap}(A_i, A_i)$ 沒有作用）：

當 $j < i$ 時其實就是將 $A_1, A_2, \ldots, A_{i-1}$ 往右移一個位置，並將 A_i 移到第一個位置。

當 $j > i$ 時其實就是將 $A_{i+1}, A_{i+2}, \ldots, A_n$ 往右移一個位置，並將 A_{N-1} 移到第 i 個位置。

來不及做完簡報了 QQ，反正就是分析一下每個位置會被移動到的終點就好了。

結論

3 結論

競賽中常常有些結論不是顯而易見的，但是只要嘗試用程式做幾次暴力並將結果 print 出來，就可以觀察出很簡單的規律了。

比賽時建議多利用觀察法去快速拿分，而練習時請多嘗試去證明為什麼會有這種性質。

送分的子題要好好把握住 >w<

這不是序列操作

by SorahISA

∅

問題概要

給定一個長度為 n 的序列 $a_1, a_2, \dots, a_n$，其元素兩兩相異。汝要在上面做三種操作：

- 1：輸出最小值並把它從序列裡刪掉。
- 2：輸出最大值並把它從序列裡刪掉。
- 3：輸出最小不在 $a_1, a_2, \dots, a_n$ 裡出現的非負整數，並把它加進序列裡。

為了減少輸出量，汝只需要把每 10 個操作的答案加在一起，總共輸出 $\lceil Q / 10 \rceil$ 個數字就好。

0 子任務們

- 子任務 1 (16 分)：$N, Q \leq 1000$
- 子任務 2 (17 分)：只有操作 1 跟 2
- 子任務 3 (12 分)：只有操作 2 跟 3
- 子任務 4 (29 分)：$N = 0$
- 子任務 5 (26 分)：無額外限制

其它輸入限制：

- 初始序列長度 $0 \leq N \leq 200\,000$
- 操作數量 $1 \leq Q \leq 5\,000\,000$
- 數字範圍 $0 \leq a_i \leq 200\,000\,000$

子任務 1

$N, Q \leq 1000$

1 子任務 I 的解法

元素在序列中的位置不重要，可以直接將整個陣列排序。

最小最大值就是 a_1 跟 a_n，都可以在 $O(N)$（用 std::vector 的 erase）到 $O(1)$（用 std::deque 的 pop_front、pop_back）的時間內完成。

陣列的 mex 的求法就是對 $i = 1, 2, \ldots, n$ 檢查：如果 $a_i \neq i - 1$ 就代表陣列中不存在 $i - 1$。

要特別注意可能整個陣列包含了 $0, 1, 2, \ldots, n - 1$ 的所有數字，這時的 mex 值就是 n。

總共要輸出 $\lceil Q / 10 \rceil$ 個數字，正整數除法 $\frac{a}{b}$ 的上取整做法是 (a+b-1)/b。

1 子任務 1 的解法

元素在序列中的位置不重要，可以直接將整個陣列排序。

Accepted

最小最大值就是 a_1 跟 a_n，都可以在 $O(N)$（用 std::vector 的 erase）到 $O(1)$（用 std::deque 的 pop_front、pop_back）的時間內完成。

得分：16 分

陣列的 mex 的求法就是對 $i = 1, 2, \ldots, n$ 檢查：如果 $a_i \neq i - 1$ 就代表陣列中不存在 $i - 1$。

要特別注意可能整個陣列包含了 $0, 1, 2, \ldots, n - 1$ 的所有數字，這時的 mex 值就是 n。

總共要輸出 $\lceil Q / 10 \rceil$ 個數字，正整數除法 $\frac{a}{b}$ 的上取整做法是 `(a+b-1)/b`。

1 子任務 1 的 Wrong Answer

很可惜，所有嘗試寫子任務 1 的 code 要嘛沒有判斷 mex 可能會是 n 的情形，要嘛在判斷輸出多少個數字的時候上取整寫錯導致在 Q 是 10 的倍數時多輸出了一個數字。

mex 的判斷建議寫成一個函式，比較好看（？

```cpp
int mex(vector<int> &vec) {
    /// vec 是排序過的陣列 ///
    /// 其中元素皆 >= 0 且沒有重複元素 ///
    int N = vec.size();
    for (int i = 0; i < N; ++i) {
        if (vec[i] != i) return i;
    }
    return N;
}
```

子任務 2

只有操作 1 跟 2

2 子任務 2 的解法

沒有會加數字的操作 3，只有不斷刪除最小值跟最大值的操作。

那麼就直接開一個 deque 先把數字都排序好，再依據操作來刪掉前後端的數字即可。

2 子任務 2 的解法

沒有會加數字的操作 3，只有不斷刪除最小值跟最大值的操作。

Accepted

那麼就直接開一個 deque 先把數字都排序好，再依據操作來刪掉前後端的數字即可。

得分：16 + 17 分

子任務 3

只有操作 2 跟 3

3 子任務 3 的解法

只有刪除最大值跟加入最小沒出現的數字兩種操作。

如果序列裡的數字是從零開始的連續非負整數們，那刪除 max 跟加入 mex 都會變得很單純，只會動到序列的最後一個數字而已。

$$\max\{0, 1, 2, \ldots, k\} = k$$

$$\operatorname{mex}\{0, 1, 2, \ldots, k\} = k + 1$$

3

子任務 3 的解法

只有刪除最大值跟加入最小沒出現的數字兩種操作。

如果序列裡的數字還不是從零開始的連續非負整數們（有洞），那刪除 max 跟加入 mex 的數字們就還不會有交集。

證明很單純：假設 k 是最大的 $< a_n$ 且沒出現在 a 裡面的數字。那刪除 max 會去把 a_n 刪掉，而加入 mex 會把一個 $\leq k < a_n$ 的數字加進去（補洞）。

於是，可以維護一個變數紀録上一次的 mex 值是多少，只要這個值還小於序列 a 的長度，就代表還有洞要補，而這次的 mex 也一定會比上一次的還要大。

當這個值等於序列 a 的長度時就會變成上一頁的情況。

3 子任務 3 的解法

只有刪除最大值跟加入最小沒出現的數字兩種操作。

Accepted

如果序列裡的數字還不是從零開始的連續非負整數們（有洞），那刪除 max 跟加入 mex 的數字們就還不會有交集。

證明很單純：假設 k 是最大的 $< a_n$ 且沒出現在 a 裡面的數字。那刪除 max 會去把 a_n 刪掉，而加入 mex 會把一個 $\leq k < a_n$ 的數字加進去（補洞）。

得分：33 + 12 分

於是，可以維護一個變數紀錄上一次的 mex 值是多少，只要這個值還小於序列 a 的長度，就代表還有洞要補，而這次的 mex 也一定會比上一次的還要大。

當這個值等於序列 a 的長度時就會變成上一頁的情況。

子任務 4

$N = 0$

4 子任務 4 的解法

現在多了刪最小值的操作，相當於是在後院鑿洞，上個子任務存在的 mex 遞增的性質也被破壞掉了。

不如直接換個思路？

把相鄰的數字「黏」在一起當成一個區間，這樣在算 mex 的時候就可以一次跳過一整段數字。

因為跳過一段數字就會掉進一個洞裡，所以其實跳一次就會直接得到答案了！

刪最小值、刪最大值的操作也仍然只是修改第一個或最後一個區間的端點。

4 子任務 4 的解法

舉個例子：假設依序做 33333111321333 的操作，那麼會長這樣

$$\text{empty} \to [0,0] \to [0,1] \to [0,2] \to [0,3] \to [0,4]$$
$$\to [1,4] \to [2,4] \to [3,4] \to [0,0],[3,4] \to [0,0],[3,3]$$
$$\to [3,3] \to [0,0],[3,3] \to [0,1],[3,3] \to [0,3]$$

如果在刪值時把區間整個刪掉了就移除、查詢 mex 就是看第一個區間包不包含 0：

- 包含 0 時 mex 就會是該區間的右界 +1。
- 不包含 0 時 mex 就會是 0。

實作上需要注意當相鄰兩個區間接觸到彼此就需要將其合併，而這只會發生在前兩個區間。

4 子任務 4 的解法

舉個例子：假設依序做 33333111321333 的操作，那麼會長這樣

$$\text{empty} \to [0,0] \to [0,1] \to [0,2] \to [0,3] \to [0,4]$$
$$\to [1,4] \to [2,4] \to [3,4] \to [0,0],[3,4] \to [0,0],[3,3]$$
$$\to [3,3] \to [0,0],[3,3] \to [0,1],[3,3] \to [0,3]$$

Accepted

得分：45 + 29 分

如果在刪值時把區間整個刪掉了就移除、查詢 mex 就是看第一個區間包不包含 0：

- 包含 0 時 mex 就會是該區間的右界 +1。
- 不包含 0 時 mex 就會是 0。

實作上需要注意當相鄰兩個區間接觸到彼此就需要將其合併，而這只會發生在前兩個區間。

子任務 5

無額外限制

5 子任務 5 的解法

子任務 4 其實已經把整個解法講完了。

這裡 $N \neq 0$，代表一開始已經有一些區間們在裡面了。

只要好好跟他們打招呼就可以 AC 了！

5

子任務 5 的解法

子任務 4 其實已經把整個解法講完了。

Accepted

這裡 $N \neq 0$，代表一開始已經有一些區間們在裡面了。

只要好好跟他們打招呼就可以 AC 了！

得分：100 分

5

子任務 5 的實作

使用 deque 包 pair 維護區間左右界。

記得要對陣列排序過，並好好打招呼。

更新時也要記得讓鄰居住一起。

```cpp
vector<int> A(N);
for (int &x : A) cin >> x;
sort(begin(A), end(A));

deque<pair<int, int>> itv;
for (int x : A) {
    if (!itv.empty() and x == itv.back().second + 1)
        ++itv.back().second;
    else
        itv.emplace_back(x, x);
}
```

```cpp
for (int i = 0; i < Q; ++i) {
    if (S[i] == '1') {
        /// 輸出並移除最小值 ///
        ans[i/10] += itv[0].first++;
        /// 如果第一個區間變成空的就拿掉 ///
        if (itv[0].first > itv[0].second) itv.pop_front();
    }
    else if (S[i] == '2') {
        /// 輸出並移除最大值 ///
        ans[i/10] += itv.back().second--;
        /// 如果最後一個區間變成空的就拿掉 ///
        if (itv.back().first > itv.back().second) itv.pop_back();
    }
    else if (S[i] == '3') {
        /// 輸出並加入 mex ///
        /// 如果沒有包含 0 就加入 0，並維護可能的區間合併 ///
        if (itv.empty() or itv[0].first > 1) itv.emplace_front(0, 0);
        else if (itv[0].first == 1) itv[0].first = 0;
        /// 否則加入第一個區間的右界 + 1，並維護可能的區間合併 ///
        else {
            ans[i/10] += ++itv[0].second;
            if ((int)itv.size() >= 2 and itv[0].second + 1 == itv[1].first)
                itv.pop_front(), itv[0].first = 0;
        }
    }
}
```

結論

6 結論

這是集合操作，不是序列操作。

要會分析算法的複雜度、以及觀察題目的性質！

在本題中的性質就是 mex 只會出現在從 0 開始的連續一段數字之後，所以加速的做法就是把連續的數字黏起來，做到 $O(1)$ 查詢。

對 `std::vector` 做 `erase` 的複雜度不是 $O(1)$，他會需要把後面的元素一個一個往前搬。

mex 運算在 Game Theory 時會常常用到。

這不是 LCS 問題

by SorahISA

0

問題概要

給汝兩個字串 s、t，請求出任意一組長度為 $\min\{\text{LCS 長度}, k\}$ 的共同子字串（common subsequence）。

子字串的定義是將字串中的零或多個字元刪掉，並將剩下的字元接在一起得到的字串。

特別的，如果汝只輸出長度沒有找到共同子字串，可以得到 30% 的分數。

子任務們

- 子任務 1 (38 分)：$n, m \le 1000$
- 子任務 2 (4 分)：$k = 1$
- 子任務 3 (7 分)：$k \le 2$
- 子任務 4 (51 分)：$n, m \le 100\,000$、$k \le 100$

其它輸入限制：

- 字串長度 $1 \le n, m \le 100\,000$
- $|s| = n$、$|t| = m$
- 輸出長度 $1 \le k \le 100$
- 字元集是 $\{a, b, c, \ldots, z\}$

子任務 1

$n, m \leq 1000$

1 子任務 1 的解法

最長共同子字串（longest common subsequence，LCS）是動態規劃的經典題。

以 $\mathrm{dp}[i][j]$ 表示 $s_{1\ldots i}$ 跟 $t_{1\ldots j}$ 的 LCS 長度，則初始狀態為

$$\mathrm{dp}[0][x] = \mathrm{dp}[x][0] = 0, \qquad \forall x$$

轉移方法為

$$\mathrm{dp}[i][j] = \begin{cases} \max\{\mathrm{dp}[i-1][j], \mathrm{dp}[i][j-1]\}, & s_i \neq t_j \\ \max\{\mathrm{dp}[i-1][j], \mathrm{dp}[i][j-1], \mathrm{dp}[i-1][j-1]+1\}, & s_i = t_j \end{cases}$$

最終就會得到 $\mathrm{LCS}(s,t) = \mathrm{dp}[n][m]$。

注意輸出長度要跟 k 取 min。

子任務 1 的解法

最長共同子字串（longest common subsequence，LCS）是動態規劃的經典題。

(partially) Accepted

以 $\mathrm{dp}[i][j]$ 表示 $s_{1\ldots i}$ 跟 $t_{1\ldots j}$ 的 LCS 長度，則初始狀態為

$$\mathrm{dp}[0][x] = \mathrm{dp}[x][0] = 0, \qquad \forall x$$

得分：11.4 分

轉移方法為

$$\mathrm{dp}[i][j] = \begin{cases} \max\{\mathrm{dp}[i-1][j], \mathrm{dp}[i][j-1]\}, & s_i \neq t_j \\ \max\{\mathrm{dp}[i-1][j], \mathrm{dp}[i][j-1], \mathrm{dp}[i-1][j-1] + 1\}, & s_i = t_j \end{cases}$$

最終就會得到 $\mathrm{LCS}(s, t) = \mathrm{dp}[n][m]$。

注意輸出長度要跟 k 取 min。

1 子任務 I 的解法

最長共同子字串（longest common subsequence，LCS）是動態規劃的經典題。

以 $\mathrm{dp}[i][j]$ 表示 $s_{1\ldots i}$ 跟 $t_{1\ldots j}$ 的 LCS 長度，則初始狀態為

$$\mathrm{dp}[0][x] = \mathrm{dp}[x][0] = 0, \qquad \forall x$$

轉移方法為

$$\mathrm{dp}[i][j] = \begin{cases} \max\{\mathrm{dp}[i-1][j], \mathrm{dp}[i][j-1]\}, & s_i \neq t_j \\ \max\{\mathrm{dp}[i-1][j], \mathrm{dp}[i][j-1], \mathrm{dp}[i-1][j-1]+1\}, & s_i = t_j \end{cases}$$

最終就會得到 $\mathrm{LCS}(s,t) = \mathrm{dp}[n][m]$，輸出答案時就從 $\mathrm{dp}[n][m]$ 不斷回溯直到答案變成 0。

注意輸出長度要跟 k 取 min。

1 子任務 1 的解法

最長共同子字串（longest common subsequence，LCS）是動態規劃的經典題。

Accepted

以 $\mathrm{dp}[i][j]$ 表示 $s_{1\dots i}$ 跟 $t_{1\dots j}$ 的 LCS 長度，則初始狀態為

$$\mathrm{dp}[0][x] = \mathrm{dp}[x][0] = 0, \qquad \forall x$$

得分：38 分

轉移方法為

$$\mathrm{dp}[i][j] = \begin{cases} \max\{\mathrm{dp}[i-1][j], \mathrm{dp}[i][j-1]\}, & s_i \neq t_j \\ \max\{\mathrm{dp}[i-1][j], \mathrm{dp}[i][j-1], \mathrm{dp}[i-1][j-1]+1\}, & s_i = t_j \end{cases}$$

最終就會得到 $\mathrm{LCS}(s,t) = \mathrm{dp}[n][m]$，輸出答案時就從 $\mathrm{dp}[n][m]$ 不斷回溯直到答案變成 0。

注意輸出長度要跟 k 取 min。

子任務 2

$k = 1$

2 子任務 2 的解法

s 跟 t 有沒有長度為 1 的共同子字串？

s 跟 t 有沒有長度為 1 的共同字元？

有沒有一個字元同時在 s 跟 t 裡出現？

字元總共就 26 種，用一個 bool 陣列去紀録那些有出現過就好了。

2

子任務 2 的解法

s 跟 t 有沒有長度為 1 的共同子字串？

s 跟 t 有沒有長度為 1 的共同字元？

有沒有一個字元同時在 s 跟 t 裡出現？

字元總共就 26 種，用一個 bool 陣列去紀錄那些有出現過就好了。

Accepted

得分：38 + 4 分

子任務 3

$k \leq 2$

3 子任務 3 的解法

因為 k 很小，所以可以枚舉 $O(|\Sigma|^k)$ 種可能的答案（Σ 是字元集）做檢查。

如果要檢查字串 a 是不是字串 s 的子字串，可以透過 greedy 來匹配。

➢ 只要遇到就直接配，因為不配只會讓狀態更糟。

於是，只要枚舉所有 $|\Sigma|^2 + |\Sigma| = 602$ 種可能的 LCS，分別去判斷它是不是 s 跟 t 的子字串。

如果它同時是 s 跟 t 的子字串，則它就是一組共同子字串。

3 子任務 3 的解法

因為 k 很小，所以可以枚舉 $O(|\Sigma|^k)$ 種可能的答案（Σ 是字元集）做檢查。

Accepted

如果要檢查字串 a 是不是字串 s 的子字串，可以透過 greedy 來匹配。

➢ 只要遇到就直接配，因為不配只會讓狀態更糟。

得分：38 + 11 分

於是，只要枚舉所有 $|\Sigma|^2 + |\Sigma| = 602$ 種可能的 LCS，分別去判斷它是不是 s 跟 t 的子字串。

如果它同時是 s 跟 t 的子字串，則它就是一組共同子字串。

子任務 4

$n, m \leq 100\,000$、$k \leq 100$

4 子任務 4 的解法

從剛剛的子任務 3，我們發現可以枚舉字串來檢查它會不會是 s 跟 t 的共同子字串。

於是也可以考慮使用 DP 來解決這個問題，令 $\mathrm{dp}[i][\ell]=j$ 代表 $s_{1\dots i}$ 的所有長度為 ℓ 的子字串中，最少只會匹配到 $t_{1\dots j}$。初始狀態為

$$\mathrm{dp}[0][x]=\mathrm{dp}[y][0]=0,\qquad \forall x\in[0,k],y\in[0,n]$$

轉移方法為

$$\mathrm{dp}[i][\ell]=\min\{\mathrm{dp}[i-1][\ell],\text{最小的 } p>\mathrm{dp}[i-1][\ell-1] \text{ 滿足 } t_p=s_i\}$$

跟一般的 LCS 概念相同，分成最後一個字元「相同」或「不同」兩種狀態。

轉移後半的部分可以預處理字串 t「位置 p 以後第一次出現字元 c 的位置」，
就能做到 $O(1)$ 查詢。

4

子任務 4 的解法

從剛剛的子任務 3，我們發現可以枚舉字串來檢查它會不會是 s 跟 t 的共同子字串。

於是也可以考慮使用 DP 來解決這個問題，令 $\mathrm{dp}[i][\ell]$ 代表 $s_{1\ldots i}$ 的所有長度為 ℓ 的子字串中，最少只會匹配到 $t_{1\ldots j}$。初始狀態為

$$\mathrm{dp}[0][x] = \mathrm{dp}[y][0] = 0, \qquad \forall x \in [0,k], y \in [0,n]$$

轉移方法為

$$\mathrm{dp}[i][\ell] = \min\{\mathrm{dp}[i-1][\ell], \text{最小的 } p > \mathrm{dp}[i-1][\ell-1] \text{ 滿足 } t_p = s_i\}$$

跟一般的 LCS 概念相同，分成最後一個字元「相同」或「不同」兩種狀態。

轉移後半的部分可以預處理字串 t「位置 p 以後第一次出現字元 c 的位置」，就能做到 $O(1)$ 查詢。

Accepted (partially)

得分：34.3 + 30 分

4 子任務 4 的解法

$$\mathrm{dp}[i][\ell] = \min\{\mathrm{dp}[i-1][\ell], \text{最小的 } p > \mathrm{dp}[i-1][\ell-1] \text{ 滿足 } t_p = s_i\}$$

跟一般的 LCS 不同，LCS 的轉移來源很單純，在回溯時只要看左方、上方、左上方三者。

而這個版本的轉移來源，特別是後半，也要額外記録才能回溯。

不過一般的 DP 紀録的是轉移位置，這裡跟 p 取 $\min$ 不是一個好的 $\mathrm{dp}[*][*]$ 位置？

那麼就換一種紀録方法，根據轉移式我們已經知道 $\mathrm{dp}[n][k] = p$ 時 t_p 一定有被匹配到。

那就用 t_p 去找上一個匹配到的 s_i，並遞迴到 $\mathrm{dp}[i][k-1]$。

跟轉移時類似，需要對 s 預處理「位置 p 以前最後一次出現字元 c 的位置」。

子任務 4 的解法

$$\mathrm{dp}[i][\ell] = \min\{\mathrm{dp}[i-1][\ell], \text{最小的 } p > \mathrm{dp}[i-1][\ell-1] \text{ 滿足 } t_p = s_i\}$$

Accepted

跟一般的 LCS 不同，LCS 的轉移來源很單純，在回溯時只要看左方、上方、左上方三者。

而這個版本的轉移來源，特別是後半，也要額外記録才能回溯。

不過一般的 DP 紀録的是轉移位置，這裡跟 p 取 min 不是一個好的 $\mathrm{dp}[*][*]$ 位置？

得分：100 分

那麼就換一種紀録方法，根據轉移式我們已經知道 $\mathrm{dp}[n][k] = p$ 時 t_p 一定有被匹配到。

那就用 t_p 去找上一個匹配到的 s_i，並遞迴到 $\mathrm{dp}[i][k-1]$。

跟轉移時類似，需要對 s 預處理「位置 p 以前最後一次出現字元 c 的位置」。

結論

5 結論

實作細節可以再想想，並試著寫寫看。

有些子題常常可以給汝帶來整題的想法，如果題目沒有子題或是它子題亂切，汝也可以試著自己切有特殊性質的子題出來（當然這種情況就不用真的把 code 寫出來）。

學會利用子題是高中比賽時最好要具備的能力。

有時候可以透過看範圍限制來大概猜到題目要求的複雜度，像這題 $O(nm)$ 顯然太大，比較合理的就是諸如 $O\big((n+m)\cdot k\big)$ 這種感覺的範圍。

當然，隨便亂猜複雜度也不好，特別是對於變數多的題目非常容易猜錯，而通常這比亂 claim greedy 做法的後果還要糟，還請在嘗試的時候拿捏一下程度。

這不是二分搜

by SorahISA

0 問題概要

本題又名「隨機化二分搜」。

簡單來說，就是給一個排列 $A_1, A_2, \ldots, A_N$，要汝找出有多少組 (L, R, x) 滿足：

對區間 $A_{L\ldots R}$ 做二分搜時，無論二分搜戳的位置是那個都要能找到 x。

汝要輸出 N 個數字，分別代表區間長度為 $k = 1, 2, \ldots, N$ 的解數量之和。

∅

子任務們

- 子任務 1 (1 分)：$N \le 3$
- 子任務 2 (10 分)：$N \le 15$
- 子任務 3 (14 分)：$N \le 60$
- 子任務 4 (17 分)：$N \le 600$
- 子任務 5 (22 分)：$N \le 5000$
- 子任務 6 (36 分)：$N \le 1\,000\,000$

其它輸入限制：

- 陣列長度 $1 \le N \le 1\,000\,000$
- $a_1, a_2, \ldots, a_N$ 是 $1, 2, \ldots, N$ 的排列

子任務

$N \leq 3$

1 子任務 1 的解法

總共有 $1! + 2! + 3! = 9$ 種可能，每一種的狀態也不多，可以手解。

1

子任務 1 的解法

總共有 $1! + 2! + 3! = 9$ 種可能，每一種的狀態也不多，可以手解。

Accepted

得分：1 分

子任務 2

$N \leq 15$

2

子任務 2 的解法

根據題意來進行暴力遞迴。對所有 (L, R, x) 都詢問一遍並統計答案即可。

```cpp
int check(vector<int> &A, int L, int R, int x) {
    if (L > R) return 0;
    int flag = 1;
    for (int p = L; p <= R; ++p) {
        if (A[p] == x) flag &= 1;
        if (A[p] >  x) flag &= check(A, L, p-1, x);
        if (A[p] <  x) flag &= check(A, p+1, R, x);
    }
    return flag;
}
```

2

子任務 2 的解法

根據題意來進行暴力遞迴。對所有 (L, R, x) 都詢問一遍並統計答案即可

```
int check(vector<int> &A, int L, int R, int x) {
    if (L > R) return 0;
    int flag = 1;
    for (int p = L; p <= R; ++p) {
        if (A[p] == x) flag &= 1;
        if (A[p] >  x) flag &= check(A, L, p-1, x);
        if (A[p] <  x) flag &= check(A, p+1, R, x);
    }
    return flag;
}
```

Accepted

得分：11 分

子任務 3

$N \leq 60$

3

子任務 3 的解法一

對剛剛的暴力做點優化。

其實傳入函式的狀態只有 $O(N^3)$ 種，
加上每次要往 $O(N)$ 個狀態遞迴下去。

可以用個 std::map 包 std::tuple
來儲存狀態，當已經處理過這個狀態
就直接回傳答案。

複雜度大致上是 $O(N^4 \log N)$。

```
map<tuple<int, int, int>, int> _check;

int check(vector<int> &A, int L, int R, int x) {
    if (_check.count({L, R, x})) return _check[{L, R, x}];
    if (L > R) return 0;
    int flag = 1;
    for (int p = L; p <= R; ++p) {
        if (A[p] == x) flag &= 1;
        if (A[p] >  x) flag &= check(A, L, p-1, x);
        if (A[p] <  x) flag &= check(A, p+1, R, x);
    }
    return _check[{L, R, x}] = flag;
}
```

3 子任務 3 的解法一

對剛剛的暴力做點優化

Accepted

其實傳入函式的狀態只有 $O(N^3)$ 種，
加上每次要往 $O(N)$ 個狀態遞迴下去。

得分：25 分

可以用個 std::map 包 std::tuple
來儲存狀態，當已經處理過這個狀態
就直接回傳答案。

複雜度大致上是 $O(N^4 \log N)$。

```
map<tuple<int, int, int>, int> _check;

int check(vector<int> &A, int L, int R, int x) {
    if (_check.count({L, R, x})) return _check[{L, R, x}];
    if (L > R) return 0;
    int flag = 1;
    for (int p = L; p <= R; ++p) {
        if (A[p] == x) flag &= 1;
        if (A[p] >  x) flag &= check(A, L, p-1, x);
        if (A[p] <  x) flag &= check(A, p+1, R, x);
    }
    return _check[{L, R, x}] = flag;
}
```

3 子任務 3 的解法二

暴力活不久，還是來想看看題目的性質ㄅ。

首先是「詢問 (L, R, x) 可以變成詢問有多少個 $p \in [L, R]$ 的 $\left(L, R, A_p\right)$」。

再來是，觀察到「戳一個位置並根據他跟 x 的大小來決定往左或右遞迴」這件事，
代表著「x 左邊的數字一定要比 x 更小、右邊則要更大」。

證明可以使用反證法：如果戳到左邊更大或是右邊更小那搜尋的區間就會不再包含 x。

$$[3, 2, 1, 4, 6, 5]$$

像是上述區間中，只有 4 滿足左邊更小右邊更大的性質。

3 子任務 3 的解法二

也就是，題目變成

$$\text{有多少組 } (L, R, p) \text{ 滿足 } p \in [L, R] \text{ 且}$$

$$\text{對於 } L \le x < p \text{ 都有 } A_x < A_p\text{、}$$

$$\text{對於 } p < y \le R \text{ 都有 } A_p < A_y\text{。}$$

可以使用四層迴圈暴力做掉。

子任務 3 的解法二

也就是，題目變成

Accepted

有多少組 (L,R,p) 滿足 $p \in [L,R]$ 且

對於 $L \le x < p$ 都有 $A_x < A_p$、

對於 $p < y \le R$ 都有 $A_p < A_y$。

得分：25 分

可以使用四層迴圈暴力做掉。

子任務 4

$N \leq 600$

4 子任務 4 的解法

有多少組 (L,R,p) 滿足 $p \in [L,R]$ 且 $A_x < A_p < A_y$。

看著範圍的 $N \le 600$，大致上可以想到我們需要一個大約是 $O(N^3)$ 的解法。

考慮能 $O(1)$ 找出一組 (L,R,p) 是不是好的的做法。

對於左界 L，他對 p 是好的左界若且唯若對於所有 $L \le x < p$ 都有 $A_x < A_p$，右界亦同。

於是可以找出對每個 p 最左端的左界 L' 以及最右端的右界 R'，
最後枚舉 $O(N^3)$ 種狀態統計答案。

4

子任務 4 的解法

有多少組 (L,R,p) 滿足 $p \in [L,R]$ 且 $A_x < A_p < A_y$。

Accepted

看著範圍的 $N \le 600$，大致上可以想到我們需要一個大約是 $O(N^3)$ 的解法。

考慮能 $O(1)$ 找出一組 (L,R,p) 是不是好的的做法。

得分：42 分

對於左界 L，他對 p 是好的左界若且唯若對於所有 $L \le x < p$ 都有 $A_x < A_p$，右界亦同。

於是可以找出對每個 p 最左端的左界 L' 以及最右端的右界 R'，
最後枚舉 $O(N^3)$ 種狀態統計答案。

4 子任務 4 的實作

右邊是找最左端的 L' 以及最右端的 R'。

下面是枚舉 (L, R, p) 計算答案。

```
for (int i = 0; i < N; ++i) {
    l[i] = r[i] = i;
    while (l[i] - 1 >= 0 and A[l[i] - 1] < A[i]) --l[i];
    while (r[i] + 1 <  N and A[r[i] + 1] > A[i]) ++r[i];
}
```

```
for (int len = 1; len <= N; ++len) {
    int ans = 0;
    for (int L = 0, R = len-1; R < N; ++L, ++R) {
        for (int p = L; p <= R; ++p) ans += (l[p] <= L and R <= r[p]);
    }
    cout << ans << " \n"[len == N];
}
```

子任務 5

$N \leq 5000$

子任務 5 的解法

再加速！

不要枚舉狀態了，改成枚舉中點 p。

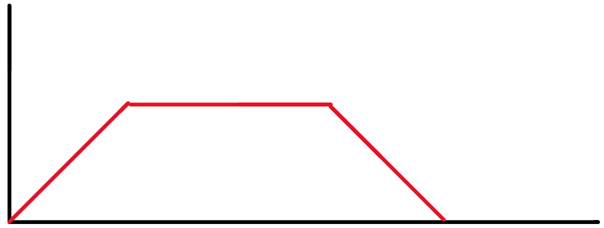
在上一個子任務時，我們已經求出對 p 的最佳左右界了，這代表我們已經知道

對於 $L' \leq L \leq p$ 且 $p \leq R \leq R'$，區間 $[L, R]$ 都是好的

那我們就不要再花 $O(N^2)$ 的時間枚舉區間了，直接計算這裡到底有幾個長度分別為 $1, 2, \ldots, k$ 的區間就好！

5 子任務 5 的解法

令 L_p 跟 R_p 分別代表往左及往右的最大長度（包含自己）。



不失公正性的假設 $L_p \leq R_p$，那 p 會：

- 往區間長度是 $1, 2, \ldots, L_p$ 的分別貢獻 $1, 2, \ldots, L_p$ 次；
- 往區間長度 $L_p + 1, L_p + 2, \ldots, R_p - 1$ 的貢獻 L_p 次；
- 往區間長度 $R_p, R_p + 1, \ldots, R_p + L_p - 1$ 的分別貢獻 $L_p, L_p - 1, \ldots, 1$ 次。

可以在 $O(N)$ 次操作内計算完成！

5

子任務 5 的解法

令 L_p 跟 R_p 分別代表往左及往右的最大長度（包含自己）。

Accepted

不失公正性的假設 $L_p \le R_p$，那 p 會：

➢ 往區間長度是 $1,2,\dots,L_p$ 的分別貢獻 $1,2,\dots,L_p$ 次；

➢ 往區間長度 $L_p+1, L_p+2, \dots, R_p-1$ 的貢獻 L_p 次；

得分：64 分

➢ 往區間長度 $R_p, R_p+1, \dots, R_p+L_p-1$ 的分別貢獻 $L_p, L_p-1, \dots, 1$ 次。

可以在 $O(N)$ 次操作內計算完成！

5

子任務 5 的實作

3-4 行在計算 L_p 跟 R_p，6-8 行在統計區間數量。

```
1 for (int i = 0; i < N; ++i) {
2     int L = 0, R = 0;
3     while (i-L >= 0 and A[i] >= A[i-L]) ++L;
4     while (i+R <  N and A[i] <= A[i+R]) ++R;
5     if (L > R) swap(L, R);
6     for (int j = 1; j < L;   ++j) ans[j] += j;
7     for (int j = L; j < R;   ++j) ans[j] += L;
8     for (int j = R; j < L+R; ++j) ans[j] += L+R-j;
9 }
```

子任務 6

$N \leq 1\,000\,000$

6 子任務 6 的解法

套用子任務 5 的做法，不過在查詢合法左右界以及統計答案這兩點都還需要加速。

6 子任務 6 的解法

套用子任務 5 的做法，不過在查詢合法左右界以及統計答案這兩點都還需要加速。

「查詢最小的左界使區間內數字皆 $< A_p$」可以變成「查詢 p 左邊第一個大於他的位置」。

「查詢最大的右界使區間內數字皆 $> A_p$」可以變成「查詢 p 右邊第一個小於他的位置」。

這是單調隊列的經典題目，可以使用 stack 來維護，一次求出所有值的複雜度是 $O(N)$。

6 子任務 6 的解法

套用子任務 5 的做法，不過在查詢合法左右界以及統計答案這兩點都還需要加速。

如果是單純的區間加值，汝可能會用差分跟前綴和來解決。

不過在這裡要加的不是一個常數，而是一個有斜率的函數，可以怎麼做？

其實，差分跟前綴和一樣可以解決這個問題！

6 子任務 6 的解法

套用子任務 5 的做法，不過在查詢合法左右界以及統計答案這兩點都還需要加速。

學過物理應該都對加速度、速度、位置不陌生ㄅ，差分跟前綴和可以用這種方式來理解。

把 ans_ℓ 當成在時間 ℓ 的位置，那麼：

- 區間 $[L, R]$ 加 x 就是讓時間 L 的速度加上 x、R 減去 x、最後做積分（前綴和）。
- 區間 $[L, R]$ 加 $a(p - L) \cdot x$ 就是讓時間 L 的加速度加上 a、R 減去 a、做一次積分（前綴和）得到速度之後再將時間 R 的速度減去 x、最後做積分（前綴和）。

剛剛的三段區間加值就可以轉換為幾次單點修改跟兩遍前綴和（積分）了！

子任務 6 的解法

套用子任務 5 的做法，不過在查詢合法左右界以及統計答案這兩點都還需要加速。

Accepted

學過物理應該都對加速度、速度、位置不陌生ㄅ，差分跟前綴和可以用這種方式來理解。

把 ans_ℓ 當成在時間 ℓ 的位置，那麼：

得分：100 分

- 區間 $[L,R]$ 加 x 就是讓時間 L 的速度加上 x、R 減去 x、最後做積分（前綴和）。
- 區間 $[L,R]$ 加 $a(p-L)\cdot x$ 就是讓時間 L 的加速度加上 a、R 減去 a、做一次積分（前綴和）得到速度之後再將時間 R 的速度減去 x、最後做積分（前綴和）。

剛剛的三段區間加值就可以轉換為幾次單點修改跟兩遍前綴和（積分）了！

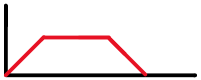
6 子任務 6 的實作

```cpp
vector<int> diff(N+3, 0);

for (int i = 0; i < N; ++i) {
    if (L[i] > R[i]) swap(L[i], R[i]);
    ++diff[1];
    --diff[L[i] + 1];
    --diff[R[i] + 1];
    ++diff[L[i] + R[i] + 1];
}

for (int i = 1; i < N+3; ++i) diff[i] += diff[i-1];
for (int i = 1; i < N+3; ++i) diff[i] += diff[i-1];

for (int i = 1; i <= N; ++i) {
    cout << diff[i] << " \n"[i == N];
}
```

```cpp
stack<int> stk;
for (int i = 0; i < N; ++i) {
    while (!stk.empty() and A[stk.top()] < A[i]) stk.pop();
    if (stk.empty()) L[i] = i + 1;
    else             L[i] = i - stk.top();
    stk.emplace(i);
}
```

上面是單調隊列找 L_p 的做法，
R_p 就是反著做而已。

左邊是三次區間修改最後變成的樣子，
因為三段區間是連著的，
所以有些項剛好可以抵銷。



結論

7 結論

這題好難ㄛ QQ。

希望汝學到的知識：

- 差分、前綴和
- 單調隊列

這兩項也都是競賽中常出現的題型，差分前綴和還可以更進一層延生出支援修改的，名為 BIT（或 Fenwick Tree）的資料結構。

單調隊列也時常運用於 DP 的優化上。

賽後討論

I 比賽結果

我認為應該要拿到的子任務們：

A. 11 分暴力實作

B. 38 分裸 LCS + 4 分判斷相同字元

C. 100 分矩陣乘法觀念題

D. 100 分迴圈

E. 16 分暴力 + 17 分排序

F. 30 分送分、100 分寫 code 觀察

A	B	C	D	E	F	0x07
42	0	100	100	0	100	342
0	0	100	100	0	30	230
0	0	0	100	0	100	200
0	0	0	100	0	100	200
25	0	0	100	0	30	155
0	0	0	100	17	0	117
0	0	0	0	0	0	0

總共是 11+42+100+100+33+100=386 分

II 題目分析及定位（上）

A. 單調隊列 + 二次差分前綴和。

- ➢ 難題，能在全國賽拿到前三等獎者應該要能解出。

B. 動態規劃

- ➢ 中等題，為經典題的變形，能在全國賽拿到前四等獎者應該要能解出。

C. 矩陣乘法觀念。

- ➢ 水題，能進校代表隊都應該要能解出。

II 題目分析及定位（下）

D. for + if-else 語法題。

- 水題，能進校代表隊都應該要能解出。

E. STL 資料結構題

- 中等題，能進全國賽者應該要能解出。

F. 觀察題。

- 中偏易，希望能點出「用程式輔助觀察」的重要性。

ＩＩ 題目分析及定位

感覺我的定位好不準ㄛ QQ

感謝各位的聆聽