枚舉 II

Enumerate II

2023 南 11 校聯合寒訓 - 演算法課程

2023.01.31

陳俊安 Colten

主辦單位：

SC ST

NHDK Ten Point Round

協辦單位：

教育部 智慧創新跨域人才培育計畫

成功大學資訊工程學系暨研究所 Department of Computer Science and Information Engineering

贊助單位：

少年圖靈計畫 Young Turing Program

奧義智慧科技 Powered by CyCraft

TEAMT5

DEVCORE

我好像忘記給我的聯絡方式了

- FB: https://www.facebook.com/ColtenOuO/
- IG: colten._.0402
- DC: Colten#0843
- 有任何問題歡迎隨時找我

這一個單元的重點

- 更進階的枚舉題目
- 遞迴枚舉

暖身：Maximum Sum of Products

- 在講遞迴枚舉之前先用一題難度 1600 的題目讓大家暖身
- 順便看看自己對枚舉有所掌握了沒有！
- ~~我們這邊抽一個同學來幫我們翻譯題目~~

You are given two integer arrays a and b of length n.

You can reverse **at most one** subarray (continuous subsegment) of the array a.

Your task is to reverse such a subarray that the sum $\sum_{i=1}^{n} a_i \cdot b_i$ is **maximized**.

→ **Problem tags**

brute force | dp | implementation | math | two pointers | *1600

No tag edit access

暖身：Maximum Sum of Products

- 我們先觀察看看題目給的範圍

Input

The first line contains one integer n ($1 \leq n \leq 5000$).

The second line contains n integers $a_1, a_2, \ldots, a_n$ ($1 \leq a_i \leq 10^7$).

The third line contains n integers $b_1, b_2, \ldots, b_n$ ($1 \leq b_i \leq 10^7$).

暖身：Maximum Sum of Products

- 我們先觀察看看題目給的範圍
- 感覺就是 O(n^2) 之類的

Input

The first line contains one integer n ($1 \leq n \leq 5000$).

The second line contains n integers $a_1, a_2, \ldots, a_n$ ($1 \leq a_i \leq 10^7$).

The third line contains n integers $b_1, b_2, \ldots, b_n$ ($1 \leq b_i \leq 10^7$).

暖身：Maximum Sum of Products

- 我們先想想最暴力的作法，再來想辦法優化
- 枚舉要反轉的區間，並計算答案
- 枚舉區間的時間複雜度 O(n^2)，計算答案 O(n)
- 整體時間複雜度 O(n^3)
- 而我們必須想辦法把時間複雜度降到至少 O(n^2) 量級

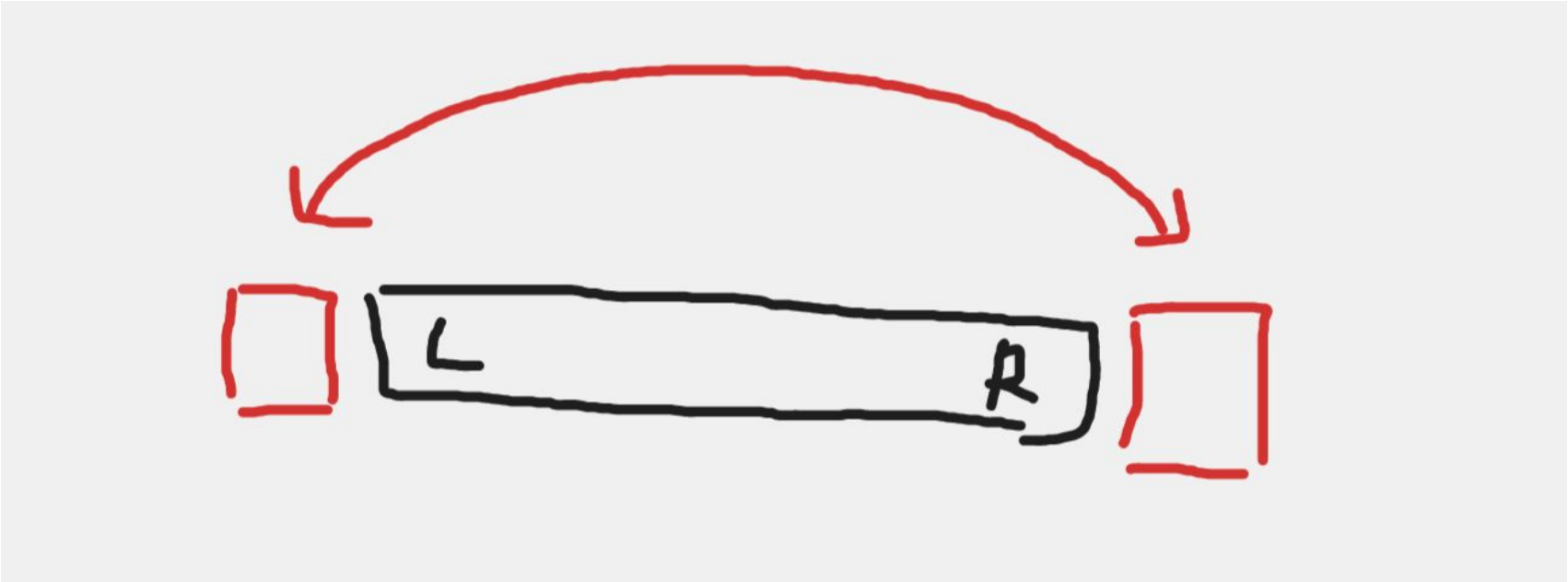
暖身：Maximum Sum of Products

- 如果要枚舉區間的話似乎必須全部枚舉，沒有辦法
- 那計算答案的部分有辦法優化？

暖身：Maximum Sum of Products

- 我們觀察一些特性
- 如果我們翻轉了區間 [L , R], 且答案變成 C
- 那麼當我們翻轉區間 [L - 1 , R + 1] 答案會變成什麼？
  - C + (A_{L-1} * B_{R+1} + A_{R+1} * B_{L-1})
  - 你會發現翻轉 [L - 1 , R + 1] 當中，有一部分的結果會完全跟翻轉 [L , R] 一樣，只是多了 L - 1 , R + 1

暖身：Maximum Sum of Products



暖身：Maximum Sum of Products

- 因此我們可以發現，計算答案的部分是可以下手優化的
- 我們可以一邊枚舉，一邊將該區間的結果計算出來
- 只是我們枚舉的順序必須 從內到外 (~~手順很重要~~)
- 這樣子計算答案的額外時間 O(n) 就直接不見了
- 是一個相當大程度的優化

暖身：Maximum Sum of Products

- 但是我們把要反轉的區間計算出來了，沒被反轉的區間的答案也要！
- 假設 A = [1 , L - 1] , B = [L , R] , C = [R + 1 , n]
  - 這邊指的都是區間和
- 我們還必須計算 A 還有 C，該怎麼辦？

暖身：Maximum Sum of Products

- A，C 都是沒有被反轉的部分
- 我們要找的東西是 區間和
- 所以這個時候~~線段樹~~，哦不是，前綴和 就派上用場了！

暖身：Maximum Sum of Products

- 預先建好一個 b_i = A_1 * B_1 + … + A_i * B_i 的前綴和
- 我們再求 A = [1 , L - 1] , C = [R + 1 , n] 的時候, 就可以 O(1) 計算出答案了
- 整體時間複雜度就變成一個乾淨的 O(n^2)

暖身：Maximum Sum of Products

```
int n;

cin >> n;

vector<int>a(n+1),b(n+1),c(n+1,0);

for(int i=1;i<=n;i++) cin >> a[i];
for(int i=1;i<=n;i++) cin >> b[i];
for(int i=1;i<=n;i++) c[i] = c[i-1] + a[i] * b[i];
```

暖身：Maximum Sum of Products

```
int ans = c[n];

for(int i=1;i<=n;i++)
{
        int total = 0;

        for(int l=i,r=i+1;l>=1&&r<=n;l--,r++)
        {
                total += a[l] * b[r];
                total += a[r] * b[l];

                ans = max(ans,total+c[l-1]+(c[n]-c[r]));
        }

        total = 0;

        for(int l=i,r=i;l>=1&&r<=n;l--,r++)
        {
                total += a[l] * b[r];
                if( l != r ) total += a[r] * b[l];

                ans = max(ans,total+c[l-1]+(c[n]-c[r]));
        }
}
```

數學教會我們的枚舉：ARC pC. LCM of GCDs

Problem Statement

We have a red bag, a blue bag, and N card packs. Initially, both bags are empty. Each card pack contains two cards with integers written on them. We know that the i-th card pack contains a card with a_i and another with b_i.

For each card pack, we will put one of its cards in the red bag, and the other in the blue bag.

After we put all cards in the bags, let X be the greatest common divisor of all integers written on cards in the red bag. Similarly, let Y be the greatest common divisor of all integers written on cards in the blue bag. Our score will be the least common multiple of X and Y.

Find the maximum possible score.

Constraints

- All values in input are integers.
- $1 \leq N \leq 50$
- $1 \leq a_i, b_i \leq 10^9$

數學教會我們的枚舉：ARC pC. LCM of GCDs

- 現在有兩個袋子，與 N 包卡片，每一包卡片裡有 2 張
- 你現在要將這 N 包卡片放進袋子裡
- 每一包裡的 2 張卡片不能放到同一個袋子
- 假設最後第一個袋子的 GCD 是 A，另一個是 B
- 求出 MAX(LCM(A,B))

Constraints

- All values in input are integers.
- $1 \leq N \leq 50$
- $1 \leq a_i, b_i \leq 10^9$

數學教會我們的枚舉：ARC pC. LCM of GCDs

- 1 ~ 10^9 當中的所有數字，每一個的因數數量基本上不超過 2000 個
- 假設第一包卡片的數字是 X , Y
- 那我們可以確定一件事情
  - 最後袋子的最大公因數一定是 X 的因數跟 Y 的因數

數學教會我們的枚舉：ARC pC. LCM of GCDs

- 因此我們枚舉最後第一個袋子的最大公因數是誰
- 也枚舉最後第二個袋子的最大公因數是誰
- 並看看，我們是否有辦法讓最後的結果成真

數學教會我們的枚舉：ARC pC. LCM of GCDs

- 怎麼檢查呢？
- 如果第一個袋子枚舉的數字是 X，第二個袋子是 Y
- 那麼接下來每一包卡片必須滿足其中一個條件
  - 第一張卡片是 X 的倍數 且 第二張卡片是 Y 的倍數
  - 第一張卡片是 Y 的倍數 且 第二張卡片是 X 的倍數

數學教會我們的枚舉：ARC pC. LCM of GCDs

- 整體時間複雜度 O(第一包卡片的因數數量相乘 * N)
- 最糟糕的情況下 O(2000^2 * N)
- 算是可以在 1 秒內通過了！

位元運算告訴我們的枚舉：CF 735 pB. Cobb

- 這題要馬上聽懂可能真的很難，回家建議再多複習
- 這題是少見的 1700 pB.

You are given n integers $a_1, a_2, \ldots, a_n$ and an integer k. Find the maximum value of $i \cdot j - k \cdot (a_i | a_j)$ over all pairs (i, j) of integers with $1 \le i < j \le n$. Here, $|$ is the bitwise OR operator.

Input

The first line contains a single integer t ($1 \le t \le 10\,000$) — the number of test cases.

The first line of each test case contains two integers n ($2 \le n \le 10^5$) and k ($1 \le k \le \min(n, 100)$).

The second line of each test case contains n integers $a_1, a_2, \ldots, a_n$ ($0 \le a_i \le n$).

It is guaranteed that the sum of n over all test cases doesn't exceed $3 \cdot 10^5$.

Output

For each test case, print a single integer — the maximum possible value of $i \cdot j - k \cdot (a_i | a_j)$.

位元運算告訴我們的枚舉：CF 735 pB. Cobb

- 先講結論，結論是我們只要枚舉 n - 2k ~ n 的部分
- 怎麼證明？
- 希望等一下不會有人睡著

位元運算告訴我們的枚舉：CF 735 pB. Cobb

- a_i OR a_j < 2n
- 我們必須先觀察出這一個特點
- 0 <= a_i <= n，如果 a_i OR a_j = 2n
- 我們假設 a_i = n，那麼如果 a_i OR a_j = 2n
- 相當於 a_i << 1（a_i 向左位移 1 位）
  - 也就是 a_i 的二進位最右邊多出了一個 0

位元運算告訴我們的枚舉：CF 735 pB. Cobb

- 但是 n 我們可以保證會是所有數字轉成二進位後位數最多的一個了
- 我們即使把 n OR n，結果的二進位也不會多出一位
- 因此我們可以確定 a_i OR $a_j < 2n$

位元運算告訴我們的枚舉：CF 735 pB. Cobb

- 接下來我們觀察一下式子
- $i * j - k * (a_i \text{ OR } a_j)$
- $a_i \text{ OR } a_j < 2n$ 這一件事情我們已經知道了
- 因此我們可以得到一個關鍵
  - 如果我們要最大化答案，對 $i * j$ 動手是最有效的

位元運算告訴我們的枚舉：CF 735 pB. Cobb

- 既然對 i * j 動手最有效，那假設我們選 i = n，j = n - 1
- 然後我們帶進去式子後會變成
- n * (n - 1) - k * (a_n OR a_{n-1})
- 接下來我們把 a_n OR a_{n-1} 替換成 2n
  - 因為 a_n OR a_{n-1} < 2n，為了方便我們假設是 2n
  - 詳細為什麼是 2n 我們後面一點會解釋

位元運算告訴我們的枚舉：CF 735 pB. Cobb

- 式子會變成 $n * (n - 1) - 2nk = (n^2 - n) - 2nk$
- 接下來我們提出一個有理想的目標
- 我們的目標是找到一組 (i, j) 使代入式子後比我們原本選擇的 $(n, n - 1)$ 還要大
- 轉成式子後會變成
- $i * j - k * (a_i \text{ OR } a_j) > (n^2 - n) - 2nk$

位元運算告訴我們的枚舉：CF 735 pB. Cobb

- $i * j - k * (a_i \text{ OR } a_j) > (n^2 - n) - 2nk$
- 大家都說 有夢最美 對吧？
- 那我們假設 a_i OR a_j 很剛好的 = 0，我們完全不損失
  - 右側的式子會讓 a_n OR a_{n-1} = 2n 是因為為了假設我們在損失最大的情況下能找到一組更好的解 (我們原本的目標就設定是要找到一組比（n，n - 1）好的解)
- 式子就變成 $i * j > (n^2 - n) - 2nk$

位元運算告訴我們的枚舉：CF 735 pB. Cobb

- $i * j > (n^2 - n) - 2nk$
- 接下來我們試著代入一些數字
- 假設 $i = n$
- 式子會變成 $n * j > (n^2 - n) - 2nk$
- 化簡一下會變成 $j > n - 2k - 1$
- 也就是說如果 $i = n$ 時, j 一定至少要 $> n - 2k - 1$ 才有機會比我們選 $i = n$, $j = n - 1$ 還要好

位元運算告訴我們的枚舉：CF 735 pB. Cobb

- 綜合以上我們就可以得知
- 我們所選擇的 i,j 一定會 >= n - 2k
- 當你 i 選 n 這麼大的數字了 j 都至少要 n - 2k 才行
- 我們推倒的時候有特別讓 狀態最佳化
  - 把左側 OR 的結果預設 0，右側預設 2n
  - 就是為了看看在最理想的結果，我們有沒有辦法找到一個比選 i = n，j = n - 1 更好的答案

位元運算告訴我們的枚舉：CF 735 pB. Cobb

- 以上就是這一題為什麼可以只枚舉 n - 2k ~ n
- 比賽的時候當然沒這麼多時間讓你證明
- 因此最關鍵的地方是可以發現範圍很特別 k <= min(n,100)
- 且很明顯要讓答案變大的重點在於 i * j 的部分

遞迴枚舉

- 遞迴只應天上有，凡人應當用迴圈
- 顧名思義，就是用遞迴來枚舉
- 這很像廢話，~~但我沒辦法~~

遞迴枚舉：Gunjyo 與骰子 II

Output: standard output

本題為 Gunjyo 與骰子 II 的簡單版本，簡單與困難之間的差異在於輸入的方式，請詳細閱讀題目敘述。

Gunjyo 學姊身為大家的學姊所以她有 n 個骰子，每個骰子都長的一樣，點數只會有三種可能，分別為 a_1, a_2, a_3。

接下來 Gunjyo 會一次骰這 n 個骰子，她想要知道所有的骰子總和能湊出 $\leq x$ 的組合會有幾種。

特別注意的是，不同顆骰子被視為不同的組合。

Input

只有一組資料。

第一行輸入一個四個整數 n, a_1, a_2, a_3, x。

測資範圍限制

- $1 \leq n \leq 18$
- $1 \leq a_1, a_2, a_3 \leq 1000$
- $1 \leq x \leq 18000$

Output

輸出一個整數，表示答案。

遞迴枚舉：Gunjyo 與骰子 II

- 你應該不會想要寫 18 層迴圈對吧
- 所以這個時候高級版的迴圈 遞迴 就派上用場了
- 我們利用遞迴把每一個骰子的狀態都枚舉出來

```
16
17 void solve(int idx,int sum,int x) // 目前在枚舉第 idx + 1 個骰子 , 總和為 sum , 題目要求 ≤ x 的數量
18 {
19     if( idx == n )
20     {
21         if( sum ≤ x ) ans++;
22
23         return;
24     }
25
26     for(int i=0;i<3;i++) solve(idx+1,sum+a[i],x); // 枚舉骰出來的點數
27 }
```

```
33     int x;
34     cin >> n >> a[0] >> a[1] >> a[2] >> x;
35
36     solve(0,0,x);
37
38     cout << ans << "\n";
```

遞迴枚舉：Dreamoon and MRT

- 現在 Dreamoon 位於某一個起點
- 他會做 M 次左右移動，只會移動到數線上有標點的位置
- 每一次移動只會往左或往右
- 求出這一條數線上最少有標幾個點
- M <= 25

遞迴枚舉：Dreamoon and MRT

- 每一次移動只會往左邊或右邊
- 因此我們當然就是枚舉 Dreamoon 每一次是往左還往右
- 把所有情況枚舉出來，並且一邊枚舉一邊計算這樣子數線上至少會需要標幾個點，Dreamoon 才能這樣移動
- 枚舉下來取最小的標點數量當作答案

遞迴枚舉：Dreamoon and MRT

- 我們可以額外開一個陣列紀錄數線上該點的位置有沒有標記

```
7 int visited[5000005];
8
9 void dfs(int now,int index,int station)
```

遞迴枚舉：Dreamoon and MRT

- 發現移動結束了，停止遞迴

```
11      if( index == m )
12      {
13          ans = min(ans,station);
14
15          return;
16      }
```

遞迴枚舉：Dreamoon and MRT

- 枚舉往右邊移動

```
18      if( visited[now+d[index]] == 1 ) dfs(now+d[index],index+1,station); // 已經有站
19      else // 沒有站
20      {
21          visited[now+d[index]] = 1; // 標記成有站
22
23          dfs(now+d[index],index+1,station+1);
24
25          visited[now+d[index]] = 0; // 還原成沒有站
26      }
```

遞迴枚舉：Dreamoon and MRT

- 枚舉往左邊移動留給大家自己發揮！
- 要特別注意的是一開始的起點位置必須設定一個大一點的數字
- 否則一直往左邊走可能會讓點的位置 < 0 導致陣列 overflow