Chapter 3
枚舉
Enumerate I

台南女中資訊研究社 38th C++ 進階班課程

TNGS IRC 38th C++ Advanced Course

講師：陳俊安 Colten

什麼是枚舉

- 暴力
- 對，就是暴力
- 而且你可以當作是基本上不用動腦的方式之一

枚舉這個技巧

- 非常難掌握
- 雖然枚舉的概念很簡單
- 但往往我們在寫題目的時候，有時可以枚舉的東西自己完全不會發現
- 掌握枚舉這一個技巧的好處
  - 可以用最無腦的方式拿到 AC，超棒的對吧

枚舉暖身：APCS 2020 10 月 第一題 人力分配

有一個公司有 n 個員工，還有兩個工廠。如果工廠一與工廠二分別有 X_1 與 X_2 個員工，兩個工廠的收益 Y_1, Y_2 分別會是

$$Y_1 = A_1 \times X_1^2 + B_1 \times X_1 + C_1$$

$$Y_2 = A_2 \times X_2^2 + B_2 \times X_2 + C_2$$

請你考慮所有分配員工的方式，找出收益最大的組合，輸出最大收益。

注意，每個員工皆需分配到其中一個工廠。

枚舉暖身：APCS 2020 10 月 第一題 人力分配

- 可以發現 n 的範圍很小，且每一個人一定要分配到其中一個工廠
- 而所有的分配情況是：
- (0,n), (1,n-1), (2,n-2), (3,n-3), … , (n,0)
- 總共會有 n 種可能，因此我們就枚舉第一個工廠分配的人數
- 如果第一個工廠給了 i 個人，那麼第二個工廠就會有 n - i 個人
- 把所有情況的結果都算出來，取最好的答案
- 時間複雜度 O(n)

枚舉暖身：台南女中資研社 37th 期末練習賽 pE. 倒水問題

- 一開始有一杯 N 豪升的水，有兩種操作
  - 把這一杯水倒出 a 毫升
  - 把這一杯水倒出 b 毫升
  - 這兩個操作可以各使用 10 次
- 最後剩下的水不能小於 K 毫升
- 如果最後的水剩下 X，請設計一個程式找出 X - K 的最小可能
- 並輸出你兩個操作各使用了幾次
  - 如果有多種策略可以讓 X - K 最小，輸出第一種操作使用比較多的

枚舉暖身：台南女中資研社 37th 期末練習賽 pE. 倒水問題

- 範圍

測資範圍限制：

- $1 \leq K \leq N \leq 10^4$
- $1 \leq a, b \leq 500$

枚舉暖身：台南女中資研社 37th 期末練習賽 pE. 倒水問題

- 對，我們其實就枚舉這兩個操作要使用各幾次即可
- 只是在算答案的時候記得，如果有多組操作方式都可以讓答案最好
- 那麼要使用第一種操作使用比較多的 (題目要求)

```
12    int now_best = n - k , ans1 = 0, ans2 = 0; // 最壞的情況就是什麼操作都不用
13
14    for(int i=10;i>=0;i--)
15    {
16        for(int j=10;j>=0;j--)
17        {
18            int total = n - a * i - b * j;
19
20            if( total >= k && total - k < now_best ) // 因為我把第一種操作大 ~ 小了，因此如果結果一樣，不會是答案
21            {
22                now_best = total - k;
23                ans1 = i;
24                ans2 = j;
25            }
26        }
27    }
28
29    cout << ans1 << " " << ans2 << " " << now_best << "\n";
```

有限度的枚舉：台南女中資研社 38th 教學上機考 pA. 飲品調配

為了研發出這款飲品，店長在這方面下了非常多的苦工，他希望這個飲品只由 3 種材料來完成，且這三種材料在飲品當中的總數量只會剛好有 N 份。

現在已經確定好了飲品製作的規則，接下來店長想要找到一個完美的比例來讓這款飲品更好喝，每杯飲料被製作出來都會有一個好喝程度，已知如果我們在一杯飲料放入 a 份第一種材料、b 份第二種材料、c 份第三種材料 $(a+b+c=N,\ 0\le a,b,c\le N)$，那麼這杯飲料的好喝程度會是 $2022+|b-c|+ab+bc+c^2-|b^2-a^2|$。

店長想要知道在 3 種材料總數量為 N 的情況下，飲料的好喝程度的最大值是多少，請你設計一個程式幫忙計算出來吧！

別忘記了！這 3 種材料的分配方式都必須符合 $a+b+c=N,\ 0\le a,b,c\le N$ 且 a,b,c 都必須是非負整數。

測資範圍限制

- $1\le N\le 5000$

有限度的枚舉：台南女中資研社 38th 教學上機考 pA. 飲品調配

- 如果我們枚舉三種材料個要使用幾次
- 時間複雜度為 O(N^3)
- N^3 = 5000 * 5000 * 5000 > 10^9
- 肯定吃一個大大的 TLE

有限度的枚舉：台南女中資研社 38th 教學上機考 pA. 飲品調配

- 題目有說 a + b + c = N, 這意味著你只要知道 a,b 的數量，就可以知道 c 的數量
- 因此我們不需要枚舉 a,b,c，只要枚舉 a,b 就可以了
- 如此一來時間複雜度就變成了 O(N^2)

```
 7 signed main(void)
 8 {
 9     int n;
10     cin >> n;
11
12     for(int a=0;a<n;a++)
13     {
14         for(int b=0;b<n;b++)
15         {
16             int c = n - ( a + b );
17             if( c < 0 ) break;
18
19             // do something
20         }
21     }
22
23     return 0;
24 }
```

有限度的枚舉：Codeforces Round #702 pC. Sum of Cubes

C. Sum of Cubes

time limit per test: 2 seconds
memory limit per test: 256 megabytes
input: standard input
output: standard output

You are given a positive integer x. Check whether the number x is representable as the sum of the cubes of two positive integers.

Formally, you need to check if there are two integers a and b ($1 \le a, b$) such that $a^3 + b^3 = x$.

For example, if $x = 35$, then the numbers $a = 2$ and $b = 3$ are suitable ($2^3 + 3^3 = 8 + 27 = 35$). If $x = 4$, then no pair of numbers a and b is suitable.

Input

The first line contains one integer t ($1 \le t \le 100$) — the number of test cases. Then t test cases follow.

Each test case contains one integer x ($1 \le x \le 10^{12}$).

Please note, that the input for some test cases won't fit into 32-bit integer type, so you should use at least 64-bit integer type in your programming language.

Output

For each test case, output on a separate line:

- "YES" if x is representable as the sum of the cubes of two positive integers.
- "NO" otherwise.

You can output "YES" and "NO" in any case (for example, the strings yEs, yes, Yes and YES will be recognized as positive).

有限度的枚舉：Codeforces Round #702 pC. Sum of Cubes

- 如果要使 a^3 + b^3 = x，那麼如果 a^3 = i，那 b^3 = x - i
- 因此我們可以去枚舉 a^3 是多少，並看看 x - a^3 是否屬於 完全立方數
- 看看是否為完全立方數可以善用 cbrt () 函數，開立方根

經典問題：列出 x 的所有因數（x <= 10^9）

- 在還沒有學過之前，相信大家都直接枚舉 1 ~ x 所有數字
- 但現在會 TLE 了，我們該怎麼處理？
- 這其實也是有限度的枚舉哦！

經典問題：列出 x 的所有因數（x <= 10^9）

- 如果 a 是 x 的因數，那麼表示 x / a 也是 x 的因數
- 因此我們在枚舉比較小的數字的時候，其實就也可以把大的因數計算出來
  - 2 是 10 的因數，我們可以順便抓出 10 / 2 = 5 這一個因數

經典問題：列出 x 的所有因數（x <= 10^9）

- 假設 a * b = x，且保證 a <= b
- 我們枚舉 a，那麼我們需要枚舉到什麼程度？
  - 根號 x
  - 如果 a > 根號 x，a 在之前一定已經被我們計算出來了
  - 根號 x * 根號 x = x
- 如此一來時間複雜度就會是 O(根號 x)

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

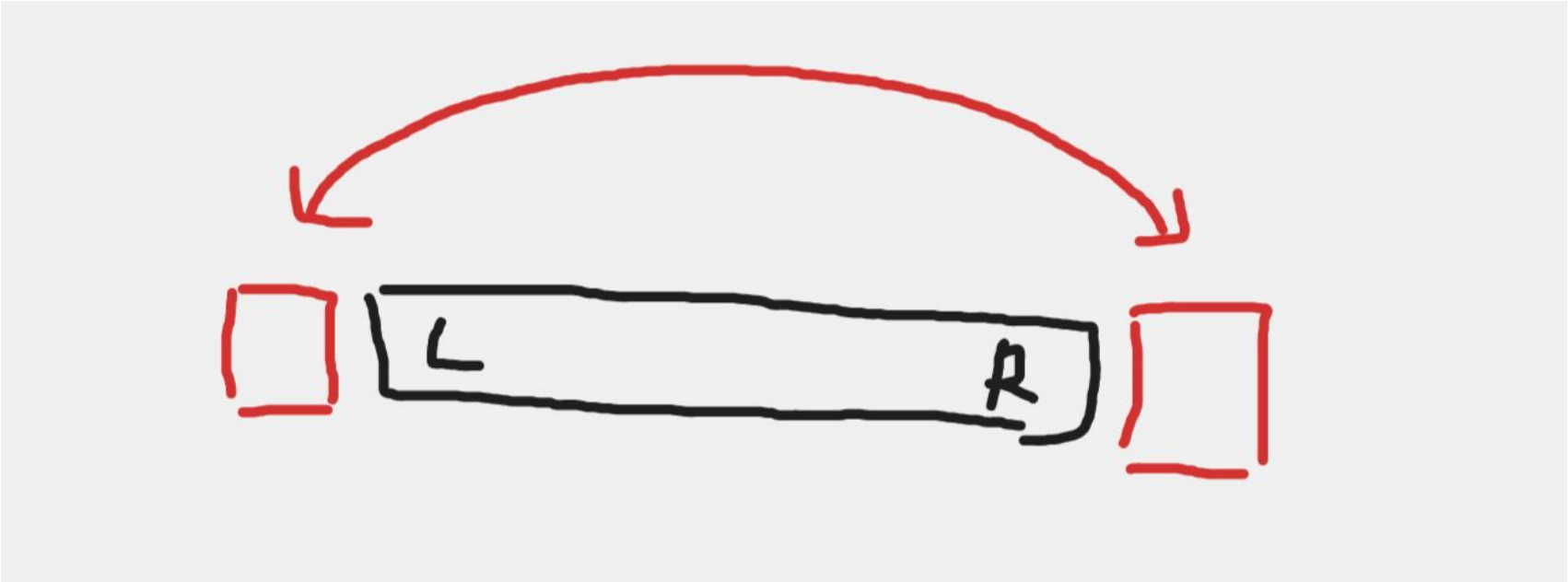
暖身：Maximum Sum of Products

- 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)
- 算是可以在時限內通過了！

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：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";
```

枚舉關鍵點：奇數偶數全排列（Hard）

PROBLEMS SUBMIT CODE MY SUBMISSIONS STATUS STANDINGS ADM. EDIT CUSTOM INVOCATION

H2. 奇數偶數全排列 (Odd and even permutation)【Hard Version】

time limit per test: 1 second
memory limit per test: 256 megabytes
input: standard input
output: standard output

Easy 與 Hard 的差別在於分數的計算方式，因此請都詳細閱讀兩種版本的題目

Colten 很喜歡只有 3 個數字的序列，除此之外，他還會對這種序列再細分，並幫這些序列打分數。

- 如果這個序列的奇數剛好有 2 個則分數加 1 分。
- 如果這個序列的偶數剛好有 2 個則分數加 2 分。
- 如果這個序列的奇偶互不相臨則分數加 5 分。
- 如果這個序列第 i $(1 \leq i \leq 2)$ 個位置是奇數，那麼如果第 $i+1$ 個位置也是奇數，則加 5 分。
- 如果這個序列第 i $(1 \leq i \leq 2)$ 個位置是偶數，那麼如果第 $i+1$ 個位置也是偶數，則加 5 分。
- 如果這個序列第 1 個位置的數字加上第 3 個位置的數字大於第 2 個位置的數字的話則加 5 分。
- 如果這個序列的任意兩項相加，都會大於序列中的所有數字的話，則分數加 7 分。

為了得到分數越高的序列，Koying 決定幫 Colten 改變序列的順序，因此他可以對序列做數次操作 (也可以完全不做任何操作)。

Koying 每次操作可以任意選擇兩個位置，並將這兩個位置的數字交換位置。

現在 Colten 想知道 Koying 排出的序列最高的分數可能是多少。

因此請你設計一個程式，告訴 Colten 答案吧！

Input

只有一筆資料。

輸入只有一行，依序輸入 3 個正整數，依序代表序列原本由左到右排列的三個數字。

保證序列裡的數字不超過 1000。

NHDK Ten Point Round Group

Public

Manager

Ten Point Round #12 (Div. 3)

Finished

Judgem. status: **100%** (0+1779=1779)

Contest Manager

→ About Time Scaling

This contest uses time limits scaling policy (depending on a programming language). The system automatically adjusts time limits by the following multipliers for some languages. Despite scaling (adjustment), the time limit cannot be more than 30 seconds. Read the details by the link.

→ Virtual participation

Virtual contest is a way to take part in past contest, as close as possible to participation on time. It is supported only ICPC mode for virtual contests. If you've seen these problems, a virtual contest is

例題 7：奇數偶數全排列（Hard）

- 關鍵點在於數字的排列方式，而不是我們要怎麼使用最佳策略
- 所以我們就只要把所有排列組合枚舉一次就可以了
- 把計算分數的東西寫成一個 Function 會比較乾淨一點

練習提示：Atcoder Beginner Contest 100 pD. Patisserie ABC

Atcoder Beginner Contest 100 pD. Patisserie ABC

給你三個長度為 n $(1 \leq n \leq 1000)$ 的序列 x, y, z，選擇 m $(0 \leq m \leq n)$ 個正整數 t_i，找出 $|\sum_{i=1}^{m} x_{t_i}| + |\sum_{i=1}^{m} y_{t_i}| + |\sum_{i=1}^{m} z_{t_i}|$ 的最大值

（提示：絕對值前的正負跟每個元素帶給的貢獻度）

全排列枚舉

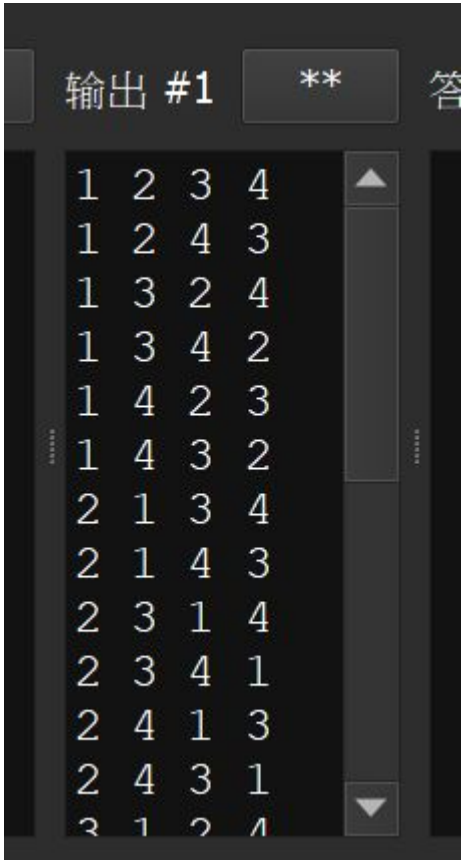
- 把所有排列的情況都枚舉出來，選出最好的答案
- 更直白的來說的話，就是每一種排列都試試看
- 其實剛剛介紹到的 奇數偶數全排列（Hard）也算是一種全排列枚舉

使用遞迴的方式把所有排列都列出來

```cpp
void solve(int n)
{
    if( n == 4 )
    {
        for(int i=1;i≤4;i++) cout ≪ ans[i] ≪ " ";
        cout ≪ "\n";
        return;
    }

    for(int i=1;i≤4;i++)
    {
        if( used[i] == 0 ) // 當前這組還沒有用過 i 這個數字
        {
            used[i] = 1; // 第 ( n + 1 ) 個數字我們用 i 試試看，把 i 標記成我們用過了
            ans[n+1] = i;
            solve(n+1);
            used[i] = 0; // 當前這一個位置要試試看擺其他數字了，所以現在 i 變成我們還沒用過了
        }
    }
}
signed main()
{
    solve(0);

    return 0 ;
}
```



八皇后問題：Chessboard and Queens

Time limit: 1.00 s **Memory limit:** 512 MB

Your task is to place eight queens on a chessboard so that no two queens are attacking each other. As an additional challenge, each square is either free or reserved, and you can only place queens on the free squares. However, the reserved squares do not prevent queens from attacking each other.

How many possible ways are there to place the queens?

Input

The input has eight lines, and each of them has eight characters. Each square i
either free (.) or reserved (*).

Output

Print one integer: the number of ways you can place the queens.

Example

Input:
```
........
........
..*.....
........
........
....**.
...*....
........
```

Output:
```
65
```

八皇后問題：Chessboard and Queens

- 八皇后的規則是 直排、橫排、斜排 都不行有皇后
- 我們先處理橫排的情況
  - 很明顯如果有 n 個皇后要放，那橫排的每一排一定都有一個皇后

八皇后問題：Chessboard and Queens

- 那直排也是一樣的道理
  - 每一個直排也一定都要剛好有一個皇后
  - 因此我們枚舉皇后的方式可以變成
    - 枚舉每一橫排的皇后要放在哪一個直排上 (全排列枚舉)
    - 每一次枚舉後，再看看每一個皇后的斜排有沒有卡到
    - 沒有卡到就 ans + 1