前言

我們編著這一系列著作的指導思維如下：

1. 程式設計競賽是「透過編寫程式解決問題」的競賽。國際大學生程式設計競賽（International Collegiate Programming Contest，ICPC）和針對中學生的國際資訊奧林匹亞競賽（International Olympiad in Informatics，IOI）在 1980 年代中後期走向成熟，30 多年來，累積了非常大量的試題。這些來自全球各地、凝聚了無數命題者的心血和智慧的試題，不僅可以用於程式設計競賽選手的訓練，而且可以用於教學，以系統、全面地提高學生編寫程式解決問題的能力。

2. 我們認為，評價一個人的專業能力，要看這個人的兩個方面：①知識系統，即他能用哪些知識去解決問題，或者說，他所真正掌握並能應用的知識，而不僅僅是他學過的知識；②思維方式，即他在面對問題（特別是不太標準化的問題）的時候，解決問題的策略是什麼？對於程式設計競賽選手所要求的知識系統，可以概括為 1984 年圖靈獎得主 Niklaus Wirth 提出的著名公式「演算法 + 資料結構 = 程式」，這也是電腦學科知識系統的核心部分。

3. 就本質而言，程式設計是技術，所以，首先牢記學習編寫程式要不斷「Practice, Practice, Practice」！本系列選用程式設計競賽的大量試題，以案例教學的方式進行教學實作並安排學生進行解題訓練。其次，「Practice in a systematic way」。本系列的編寫基於傳統的教學大綱，以系統、全面地提高學生編寫程式解決問題的能力為目標，以程式設計競賽的試題及詳細的解析、帶註解的程式作為實作，在每一章的結束部分提供相關題庫及解題提示，並對大部分試題給出官方的測試資料。

基於上述想法，我們在中國出版了本系列的簡體中文版，在臺灣出版了繁體中文版，在美國由 CRC Press 出版了英文版。

本書第一版是在復旦大學程式設計集訓隊長期活動的基礎上編寫而成的，共分 8 章，主要內容如下：

◆ 第 1 章「求解 Ad Hoc 類型問題的程式編寫實作」：介紹了機制分析法和統計分析法，啟動讀者在沒有經典和模式化演算法可對應的情況下，學會自創簡單的演算法。

◆ 第 2 章「模擬法的程式編寫實作」：啟動讀者按照題意設計數學模型的各種參數，觀察變更這些參數所引起的過程狀態的變化，在此基礎上展開演算法設計。

- ◆ 第 3 章「數論的程式編寫實作」和第 4 章「組合分析的程式編寫實作」：這兩章凸顯了數論和組合分析知識在演算法中的應用。其中，第 3 章圍繞初等數論中的質數運算、求解不定方程和同餘方程、應用積性函數等問題展開實作。第 4 章介紹在編寫程式求解組合類問題時如何計算具有某種特性的物件個數，如何將它們完全列舉出來，如何使用抽屜原理解決存在性問題，如何使用排容原理計算多個聯集的元素數量，如何使用 Pólya 定理對一個問題的各種不同的組合狀態計數。

- ◆ 第 5 章「貪心法的程式編寫實作」和第 6 章「動態規劃方法的程式編寫實作」：在求解具備最佳子結構特徵的問題時，這兩種方法是最常用、最經典的思維方法，但適用場合不同，既有相同點又有區別之處。

- ◆ 第 7 章「高階資料結構的程式編寫實作」：選擇在一般資料結構教材中沒有出現但很有用的一些知識，例如後綴陣列、區段樹、歐拉圖、哈密頓圖、最大獨立集合、割點、橋和雙連通分支等內容展開程式編寫實作。

- ◆ 第 8 章「計算幾何的程式編寫實作」：計算幾何學是演算法系統中一個重要的組成部分，也是先前演算法教材中最薄弱的環節。該章開展點線面運算、掃描線演算法、計算半平面交集、凸包計算和旋轉卡尺演算法等實作。

近來年，根據讀者和學生的回應，我們對本書內容進行了修訂，形成了第二版。我們除了修正第一版中的小錯誤，以及改進一些表述之外，還做了如下較大改進：

對於第 3 章「數論的程式編寫實作」和第 4 章「組合分析的程式編寫實作」的內容和結構，基於數論、組合數學的知識系統，進行全面的加強和改進。其中，第 3 章從質數運算、求解不定方程和同餘方程、特殊的同餘式、積性函數的應用、高斯質數 5 個方面展開實作；而第 4 章從排列的產生、排列和組合的計數、排容原理與鴿籠原理、Pólya 計數公式、產生函數與遞迴關係、快速傅立葉變換（FFT）6 個方面開始實作。對於數論、組合分析所涉及的知識點，都採用程式設計競賽的試題作為實作範例，也就是說，基於數論、組合分析的知識系統，實作範例「魚鱗狀」分佈在各個知識點中。同時，將數學證明能力和編寫程式解決問題能力的訓練相結合，這也是數學類試題的特徵。

對於第 5 章「貪心法的程式編寫實作」和第 6 章「動態規劃方法的程式編寫實作」，則增加了經典問題的實作。在第 5 章中，增加了背包問題、任務排程、區間排程等經典貪心問題的實作；在第 6 章中，則以「背包九講」為基礎，增加 0-1 背包問題的實作。這樣改進的目的，是使讀者能夠更好地體驗貪心和動態規劃的方法。

本書可以用於大學的演算法及相關數學課程的教學和實作，也可以用於程式設計競賽選手的系統訓練。對於本書，我們的使用建議是：書中每章的實作範例可以用於演算法和數學課程的教學、實作和上機作業，以及程式設計競賽選手掌握相關知識點的入門訓練；而每章最後提供的相關題庫中的試題，則可以作為程式設計競賽選手的專項訓練試題，以及學生進一步提高編寫程式能力的練習題。

Chapter 01

求解 Ad Hoc 類型問題的程式編寫實作

Ad Hoc 源自於拉丁語，意思是「為某種目的而特別設定的」。

在程式設計競賽的試題中，有這樣一類試題，解題不能套用現成的演算法，也沒有模式化的求解方法，而是需要程式編寫者自己設計演算法來解答試題，這類型試題被稱作 Ad Hoc 類型的試題，也被稱作雜題。一方面，Ad Hoc 類型試題能夠比較綜合地反映程式編寫者的智慧、知識基礎和創造性思考能力；另一方面，求解 Ad Hoc 類型試題的自創演算法只針對某個問題本身，探索該問題的獨有性質，是一種專為解決某個特定的問題、或完成某項特定的任務而設計的演算法，因此 Ad Hoc 類型試題的求解演算法一般不具備普及意義和可推廣性。

求解 Ad Hoc 類型問題的方法多樣，但從數理分析和思維方式的角度來看，大致可分兩類。

- 機制分析法。採用順向思維方式，從分析內部機制出發，順向推出求解的演算法。
- 統計分析法。採用逆向思維方式，從分析部分解出發，倒著推出求解的演算法。

這兩種方法不是孤立和互相排斥的，在求解 Ad Hoc 類型問題的過程中，既可以根據需要選擇其一，也可以兩者兼用。

1.1 機制分析法的實作範例

所謂機制分析法，就是根據客觀事物的特性，分析其內部的機制，釐清其內在的關係，在適當抽象的條件下，得到可以描述事物屬性的數學工具。

經過數學分析，如果能夠抽象出 Ad Hoc 類型問題的內在規律，則可以採用機制分析法建立數學模型，然後根據模型的原理對應到演算法，編寫程式實作，透過執行演算法得到問題解答，如圖 1-1 所示。

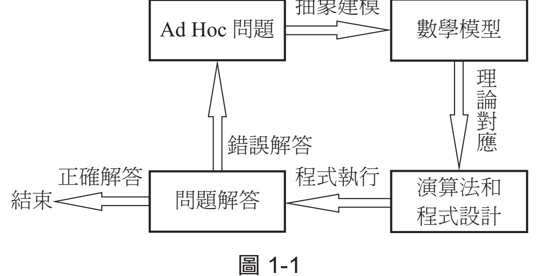

圖 1-1

機制分析法的核心是數學建模，即使用適當的數學思維建立模型，或者提取問題中的有效資訊，用簡明的方式表達其規律。需要注意以下幾點：

（1）選擇的模型必須盡量呈現問題的本質特徵。但這並不意味著模型越複雜越好，累贅的資訊會影響演算法效率。

（2）模型的建立不是一個一蹴而成的過程，而是要經過反覆檢驗和修改，在實作中不斷完善。

（3）數學模型通常有嚴格的格式，但程式編寫形式可不拘一格。

機制分析法是一個複雜的資料抽象過程。我們要善於透視問題的本質，尋找突破口，進而選擇適當的模型。模型的建構過程可以幫助我們認識問題，不同的模型從不同的角度反映問題，可以引發不同的思路，發揮引導發散思維的作用。但認識問題的最終目的是解決問題，模型的固有性質雖然可以幫我們建立演算法，其優劣亦可透過時空複雜度等指標來分析和衡量，但最終還是以程式的執行結果為標準。所以模型不是一成不變的，同樣要透過各種技術不斷最佳化。模型的產生雖然是人腦思維的產物，但它仍然是客觀事物在人腦中的反映。所以要培養良好的建模能力，還必須在平時學習中累積豐富的知識和經驗。

下面提供兩個機制分析法的實作範例。

1.1.1 ► Factstone Benchmark

2010 年，Amtel 已開發出 128 位元處理器的電腦；到 2020 年，它將開發出 256 位元電腦；以此類推，Amtel 實行每 10 年就將晶片字組長度翻一番的戰略（之前，Amtel 於 2000 年開發了 64 位元電腦；1990 年，開發了 32 位元電腦；1980 年，開發了 16 位元電腦；1970 年，開發了 8 位元電腦；1960 年首先開發了 4 位元電腦）。

Amtel 使用新的標準檢查等級 ——Factstone—— 來宣傳其新處理器大大提高的能力。Factstone 等級被定義為這樣的最大整數 n，即 $n!$ 可以表示為一個電腦的字的不帶正負號的整數（比如 1960 年的 4 位元電腦可表示為 $3!=6$，而不能表示為 $4!=24$）。

提供一個年份 y 且 $1960 \le y \le 2160$，Amtel 最近發行的晶片的 Factstone 等級是什麼？

輸入

輸入提供若干測試案例。每個測試案例一行，提供年份 y。在最後一個測試案例後，即在最後一行提供 0。

輸出

對於每個測試案例，輸出一行，提供 Factstone 等級。

範例輸入	範例輸出
1960 1981 0	3 8

試題來源：Waterloo local 2005.09.24
線上測試：POJ 2661，ZOJ 2545，UVA 10916

❖ 試題解析

對於提供的年份，首先，求出當年 Amtel 處理器的字組大小；然後，計算出最大的 n 值，使得 $n!$ 成為一個符合字組的大小的不帶正負號的整數。

1960 年，處理器的字組的大小是 4 位元，以後每 10 年字組的大小翻一番。由此可以推出，在 Y 年處理器的字組的位元數為 $K=2^{2+\left\lfloor\frac{Y-1960}{10}\right\rfloor}$，$K$ 位二進位數的最大不帶正負號的整數是 2^K-1。如果 $n!$ 是不大於 2^K-1 的最大正整數，則 n 為 Y 年晶片的 Factstone 等級。計算方法有兩種：

- 方法 1：直接求不大於 2^K-1 的最大正整數 $n!$，這種方法極容易溢位且速度慢。
- 方法 2：採用對數計算，即根據 $\log_2 n!=\log_2 n+\log_2(n-1)+\cdots+\log_2 1\le\log_2(2^K-1)<K$，計算 n。

顯然，方法 2 的效率要比方法 1 的效率高。演算法實作如下：

計算 Y 年字組的位元數 K，累加 $\log_2 i$（i 從 1 出發，每次加 1），直到數字超過 K 為止。此時，$i-1$ 即為 Factstone 等級。

❖ 參考程式

```
01  #include <stdio.h>
02  #include <math.h>
03  int y,Y,i,j,m;                // 年份為 y
04  double f,w;                   // y 年字組的位元數為 w，log2i 的累加值為 f
05  main(){
06      while (1 == scanf("%d",&y) && y){     // 輸入年份 y
07          w = log(4);           // 按照每 10 年字組的大小翻一番的規律，計算 y 年字組的位元數 w
08          for (Y=1960; Y<=y; Y+=10){
09             w *= 2;
10            }
11      i = 1;                    // 累加 log2i（每次 i 加 1），直到數字超過 w
12          f = 0;
13          while (f < w) {
14              f += log((double)++i);
15          }
16          printf("%d\n",i-1) ;  // 輸出 Factstone 等級
17      }
18      if (y) printf("fishy ending %d\n",y);
19  }
```

1.1.2 ► Bridge

n 個人要在晚上過橋，任何時候最多兩人一組過橋，每組要有一支手電筒。在這 n 個人中只有一支手電筒可以用，因此要安排以某種往返的方式來返還手電筒，使得更多的人可以過橋。

碁峰 www.gotop.com.tw

每個人的過橋速度不同，每組的速度由速度較慢的成員所決定。請確定一個策略，使得 n 個人用最少的時間過橋。

輸入

輸入的第一行提供 n，接下來的 n 行提供每個人的過橋時間，不會超過 1000 人，且沒有人的過橋時間超過 100 秒。

輸出

輸出的第一行提供所有 n 個人過橋的總秒數，接下來的若干行提供實作策略。每行包含一個或兩個整數，表示組成一組過橋的一個人或兩個人（每個人用其在輸入中提供的過橋所用的時間來標示。雖然許多人有相同的過橋時間，但即使有混淆，對結果也沒有影響）。這裡要注意的是過橋也有方向性，因為要返還手電筒讓更多的人通過。如果用時最少的策略有多個，則任意一個都可以。

範例輸入	範例輸出
4 1 2 5 10	17 1 2 1 5 10 2 1 2

試題來源： Waterloo local 2000.09.30

線上測試： POJ 2573，ZOJ 1877，UVA 10037

❖ 試題解析

分析本題，可以得出一個簡單的邏輯：要使得 n 個人用最少時間過橋，慢的成員必須藉助快的成員傳遞手電筒。

由於一次過橋最多兩人且手電筒需要往返傳遞，因此以兩個人過橋為一個分析單位計算過橋時間。為了方便，我們用 n 個人的過橋時間表示這 n 個人。我們按過橋時間遞增的順序排序 n 個成員。設目前序列為

A 是最快的人，B 是次快的人，A 和 B 是序列首部的兩個元素

a 是最慢的人，b 是次慢的人，a 和 b 是序列尾部的兩個元素

讓 a 和 b 用最少時間過橋，有兩種過橋方案。

方案 1：用最快的成員 A 傳遞手電筒，幫助 a 和 b 過橋。

如果帶一個最慢的成員 a，則所用的時間是 $a+A$（a 表示最快和最慢的兩個成員 A 和 a 到對岸所需的時間，而 A 是最快的成員傳回所需的時間）。顯然，A 帶 a 和 b 過橋所用的時間是 $2*A+a+b$。

方案 2：用最快的成員 A 和次快的成員 B 傳遞手電筒幫助 a 和 b 過橋。

步驟 1：A 和 B 到對岸，所用時間為 B；
步驟 2：A 返回，將手電筒給最慢的 a 和 b，所用時間為 A；
步驟 3：a 和 b 到對岸，所用時間為 a；到對岸後，他們將手電筒交給 B；
步驟 4：B 需要返回原來的岸邊，因為要交還手電筒，所需時間為 B。

所以，需要的總時間為 $2*B+A+a$。

顯然，a 和 b 要用最少時間過橋，只能藉助 A 或者 A 和 B 傳遞手電筒過橋，其他方法都會增加過河時間。至於哪一種過橋方式更有效，計算並比較一下就行了。

如果 $2*A+a+b<2*B+A+a$，則採用方案 1，即用最快的成員 A 傳遞手電筒；否則採用方案 2，即用最快的成員 A 和次快的成員 B 傳遞手電筒（$2*A+a+b<2*B+A+a$ 等同於 $b+A<2*B$）。

我們每次幫助目前最慢和次慢的兩個成員過橋（$n-=2$），累計每個最佳過橋方案的時間。最後，產生兩種可能的情況：

- 對岸剩下 2 個隊員（$n==2$），全部過橋，即累計時間 B；
- 對岸剩下 3 個隊員（$n==3$），用最快的成員傳遞手電筒，幫助最慢的成員過橋，然後與次慢的成員一起過橋，即累計時間 $a+A+b$。

❖ 參考程式

```
01  #include<iostream>
02  #include<algorithm>
03  #include<cstdio>
04  #include<cstring>
05  #include<cstdlib>
06  #include<cmath>
07  #include<string>
08  using namespace std;
09  int n,i,j,k,a[111111];            // 人數為 n，每個人的速度儲存於序列 a[]
10  int ans=0;                        // 初始化 n 個人過橋的總時間
11  int main ( ) {
12      scanf("%d",&n);               // 輸入每個人的速度
13      for(i=1;i<=n;i++)scanf("%d",a+i);
14      if(n==1){                     // 輸出 1 個人的過橋方案
15          printf("%d\n%d\n",a[1],a[1]);return 0;
16      }
17      int nn=n;
18      sort(a+1,a+n+1);              // 按照速度遞增順序排序
19      while(n>3){                   // 統計 n 個人過橋的總時間
20        if(a[1]+a[n-1]<2*a[2]){     // 累計用 a[1] 傳遞手電筒幫助最慢 2 個成員過橋所需的時間
21            ans+=a[n]+a[1]*2+a[n-1];
22        }else{                  // 累計用 a[1]a[2] 傳遞手電筒幫助最慢 2 個成員過橋所需的時間
23              ans+=a[2]+a[1]+a[2]+a[n];
24          }
25          n-=2;                     // 兩個最慢的成員過橋
26      }
27      if(n==2)ans+=a[2];            // 對岸剩下 2 個成員，累計其過橋的時間
28      else  ans+=a[1]+a[2]+a[3];    // 對岸剩下 3 個成員，累計其過橋的時間
```


碁峰 www.gotop.com.tw

```
29      printf("%d\n",ans);             // 輸出 n 個人過橋的總時間
30      n=nn;
31      while(n>3){                     // 輸出每組人過橋所用的時間
32          if(a[1]+a[n-1]<2*a[2])      // 輸出用 a[1] 傳遞手電筒的過橋方案
33              printf("%d %d\n%d\n%d %d\n%d\n",a[1],a[n],a[1],a[1],a[n-1],a[1]);
34          else                        // 輸出用 a[1] 和 a[2] 傳遞手電筒的過橋方案
35              printf("%d %d\n%d\n%d %d\n%d\n",a[1],a[2],a[1],a[n-1],a[n],a[2]);
36          n-=2;                       // 兩個最慢的成員過橋
37      }
38      if(n==2)printf("%d %d\n",a[1],a[2]);     // 剩下 2 個隊員過橋，輸出過橋方案
39      else                                     // 剩下 3 個隊員過橋，輸出過橋方案
40          printf("%d %d\n%d\n%d %d\n",a[1],a[3],a[1],a[1],a[2]);
41      return 0;
42  }
```

1.2 統計分析法的實作範例

在一時得不到事物的特徵機制的情況下，我們可先透過手算或程式編寫等方法測試得到一些資料，即問題的部分解，再利用數理統計知識對資料進行處理，從而得到最終的數學模型。

圖 1-2 提供了統計分析法的大致流程：先從 Ad Hoc 問題的原型出發，透過手工或簡單的程式得到問題的部分解，即解集 A；然後運用數理統計方法透過部分解，得到問題原型的主要屬性（大部分屬性是規律性的東西），從而建立數學模型，然後透過演算法設計和程式編寫得到問題的全部解，即全解集 I。這裡需要注意的是：

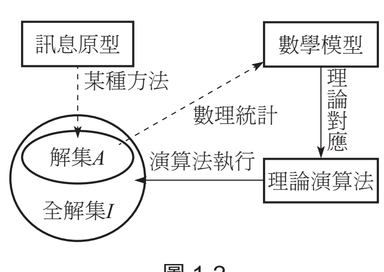

圖 1-2

- ◆ 因為有時候根本無法求出問題的部分解，或者無法用數理統計知識分析部分解，所以求部分解的過程和對部分解進行數理統計的過程畫的是虛線，表示不是每個資訊原型都能用統計分析法建模。
- ◆ 所有模型對應的演算法是將盲目搜尋排除在外的。因為盲目搜尋是從全集 I 出發求解集 A 的，這違背了建模的目的。我們所討論的統計分析法，是在對全解集 I 的子集 A 進行數理統計的基礎上建立數學模型，所以盲目搜尋不屬於統計分析法的範疇。
- ◆ 一般來說，我們可先採用機制分析法進行分析，如果機制分析進行不下去，再考慮使用統計分析法。當然，如果問題容易找到部分簡單解，我們亦可優先考慮統計分析法。事實上，機制分析所得出的某些結論，往往可被有效地運用於統計分析法；而統計分析法得出的某些規律，最終需要透過機制分析驗證其準確性。所以，它們彼此並不是孤立的，我們在建模的時候完全可以兩者兼用。

1.2.1 ▸ Ants

一支螞蟻軍隊在長度為 l 公分的橫竿上走，每隻螞蟻的速度恒定且為 1 公分 / 秒。當一隻行走的螞蟻到達橫竿終點的時候，它就立即掉了下去；當兩隻螞蟻相遇的時候，它們就調轉頭，並開始往相反的方向走。我們知道螞蟻在橫竿上原來的位置，但不知道螞蟻行走的方向。請計算所有螞蟻從橫竿上掉下去的最早可能時間和最晚可能時間。

輸入

輸入的第一行提供一個整數，表示測試案例個數。每個測試案例首先提供兩個整數：橫竿的長度（以公分為單位）和在橫竿上的螞蟻的數量 n。接下來提供 n 個整數，表示每隻螞蟻在橫竿上從左端測量過來的位置，沒有特定的次序。所有輸入資料不大於 1,000,000，資料之間用空格分隔。

輸出

對於輸入的每個測試案例，輸出用一個空格分隔的兩個數，第一個數是所有的螞蟻掉下橫竿的最早可能的時間（如果它們的行走方向選擇合適），第二個數是所有的螞蟻掉下橫竿的最晚可能的時間。

範例輸入	範例輸出
2 10 3 2 6 7 214 7 11 12 7 13 176 23 191	4 8 38 207

試題來源： Waterloo local 2004.09.19

線上測試： POJ 1852，ZOJ 2376，UVA 10714

❖ 試題解析

螞蟻數的上限為 1,000,000，爬行方式會達到 $2^{1000000}$ 種，這是一個天文數字，因此不可能逐一列舉螞蟻的爬行方式。

我們先研究螞蟻少的時候的一些情況，如圖 1-3 所示。

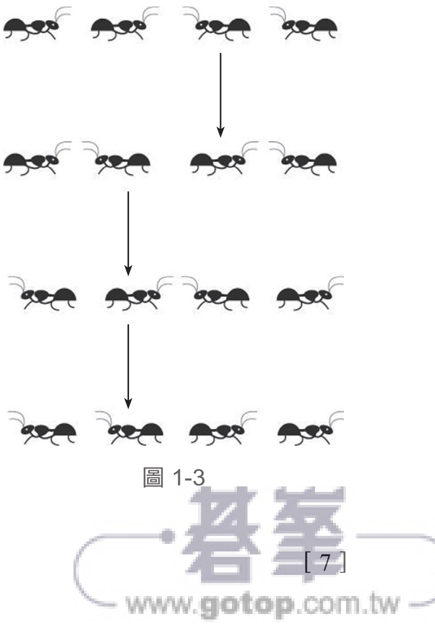

圖 1-3

顯然，螞蟻越多，變化越多，情況越複雜。而解題的瓶頸就是螞蟻相遇的情況。假如我們拘泥於「對於相遇如何處理」這個細節，將陷入無從著手的境地。

假如出現這樣一種情況：螞蟻永遠不會相遇（即所有向左走的螞蟻都在向右走的螞蟻的左邊），那麼很容易找出 O(n) 的演算法。

讓我們回過頭觀察前面提供的例子。我們發現螞蟻在相遇前為「　」，在相遇後就變成了「　」，這就相當於忽略了「相遇」這一事件。也就是說，我們可

碁峰 www.gotop.com.tw

以假設這些螞蟻即使相遇了也不理睬對方而繼續走自己的路。對於問題來說，所有的螞蟻都是一樣的，並無相異之處，因此這個假設當然是合理的。這樣，每隻螞蟻掉落所用的時間就只有兩個取值：一個是向左走用的時間，一個是向右走用的時間。全部掉落的最早時間就是每隻螞蟻儘快掉落用時的最大值，因為這些螞蟻互不干擾。同理，全部掉落的最遲時間就是每隻螞蟻盡量慢掉落用時的最大值。由此得出演算法，設 l_i 為第 i 隻螞蟻在橫竿上從左端過來測量的位置（$1 \le i \le n$）；little_i 為第 i 隻螞蟻掉下橫竿的最早時間，little 為 n 隻螞蟻掉下橫竿的最早時間；big_i 為第 i 隻螞蟻掉下橫竿的最晚時間，big 為 n 隻螞蟻掉下橫竿的最晚時間。則 $\text{little}_i = \min\{l_i, L-l_i\}$，$\text{big}_i = \max\{l_i, L-l_i\}$，$1 \le i \le n$；$\text{little} = \max\{\text{little}_i | 1 \le i \le n\}$，$\text{big} = \max\{\text{big}_i | 1 \le i \le n\}$。

本題從最簡單的情況入手，透過分析發現所有螞蟻的等價性，將「相遇後轉向」轉變為「相遇後互不干擾」，從而簡化了問題，輕而易舉地計算出答案。

❖ 參考程式

```
01  #include <stdio.h>
02  int c,big,little,L,i,j,k,n;         // 測試案例數為 c；big、little 為最晚時間和最早時間；
03                                      // 橫竿長度為 L；竿上的螞蟻數為 n
04  main( ){
05      scanf("%d",&c);                 // 輸入測試案例數
06      while (c-- && (2 == scanf("%d%d",&L,&n))) {     // 輸入橫竿長度和橫竿上的螞蟻數
07          big = little = 0;           // 最晚時間和最早時間初始化
08          for (i=0;i<n;i++) {         // 輸入每隻螞蟻的測量位置
09              scanf("%d",&k);
10              if (k > big) big = k;         // 根據 k 的左方長度和右方長度調整最晚時間
11              if (L-k > big) big = L-k;
12              if (k > L-k) k = L-k;         // 由 k 左、右方長度的最小值調整最早時間
13              if (k > little) little = k;
14          }
15          printf("%d %d\n",little,big);    // 輸出所有螞蟻掉下橫竿的最早時間和最晚時間
16      }
17      if (c != -1) printf("missing input\n");
18  }
```

1.2.2 ▸ Matches Game

有一個簡單的遊戲，在這個遊戲中，有若干堆火柴和兩個玩家。兩個玩家一輪一輪地玩。在每一輪中，一個玩家可以選擇一個堆，並從該堆取走任意根火柴（當然，取走火柴的數量不可能為 0，也不可能大於所選的火柴的數量）。如果在一個玩家取了火柴後沒有火柴留下，那麼這個玩家就贏了。假設兩個玩家非常聰明，請你告訴大家先玩的玩家是否可以贏。

輸入

輸入由若干行組成，每行一個測試案例。每行開始首先提供整數 M（$1 \le M \le 20$），表示火柴堆的堆數；然後提供 M 個不超過 10,000,000 的正整數，表示每個火柴堆的火柴數量。

輸出

對每個測試案例，如果是先手的玩家贏，則在一行中輸出 "Yes"；否則輸出 "No"。

範例輸入	範例輸出
2 45 45 3 3 6 9	No Yes

試題來源：POJ Monthly, readchild

線上測試：POJ 2234

❖ 試題解析

本題是一個 Nimm 博弈問題。遊戲的各種情況分析如下。

情況 1：如果遊戲開始時只有一堆火柴，則走先手的玩家取走這一堆的所有火柴而獲勝。

情況 2：如果遊戲開始時有兩堆火柴，且這兩堆火柴的數量分別為 N_1 和 N_2。

- ◆ 如果 $N_1 \neq N_2$，則走先手的玩家先從大堆火柴中取走一些火柴，使得兩堆火柴數量相等；然後，走後手的玩家每次從一堆火柴裡取走一些火柴後，走先手的玩家就從另一堆火柴裡取相同數量的火柴；最終走先手的玩家獲勝。
- ◆ 如果 $N_1 = N_2$，每次在走先手的玩家從一堆火柴中取走一些火柴之後，走後手的玩家就從另一堆火柴裡取相同數量的火柴；最終走後手的玩家獲勝。

情況 3：遊戲開始時有多於兩堆的火柴。

每個自然數都能夠表示成一個二進位數字。例如，$57_{(10)} = 111001_{(2)}$，即 $57_{(10)} = 2^5 + 2^4 + 2^3 + 2^0$。57 根火柴組成的一堆可以視為 4 個小堆：$2^5$ 根火柴組成一堆，2^4 根火柴組成一堆，2^3 根火柴組成一堆，2^0 根火柴組成一堆。

設遊戲開始時有 k 堆火柴，這 k 堆火柴中的火柴數量分別是 $N_1, N_2, \cdots, N_k$，N_i 可以表示為一個 $s+1$ 位二進位數字，即 $N_i = n_{is} \cdots n_{i1}n_{i0}$，$n_{ij}$ 是一個二進位，$0 \le j \le s$，$1 \le i \le k$。如果二進位數字的位數小於 $s+1$，在前面加 0。

如果 $n_{10} + n_{20} + \cdots + n_{k0}$ 是偶數，$n_{11} + n_{21} + \cdots + n_{k1}$ 是偶數……$n_{1s} + n_{2s} + \cdots + n_{ks}$ 是偶數，即 n_{10} XOR n_{20} XOR…XOR n_{k0} 是 0，n_{11} XOR n_{21} XOR…XOR n_{k1} 是 0……n_{1s} XOR n_{2s} XOR…XOR n_{ks} 是 0，則稱遊戲的狀態是平衡的；否則，遊戲的狀態是非平衡的。如果一個玩家面對一個非平衡的狀態，他可以從某一堆火柴中取走一些火柴，使得遊戲狀態變成平衡的狀態；而如果一個玩家面對一個平衡的狀態，那麼無論他採取怎樣的策略，遊戲狀態都將變為非平衡狀態。遊戲的最終狀態是所有的二進位數字為 0，也就是說，遊戲的最終狀態是平衡的。所以，獲勝的策略（Bouton 定理）如下。

如果遊戲的初始狀態是非平衡的，則走先手的玩家會贏；如果遊戲的初始狀態是平衡的，則走後手的玩家會贏。

例如，有 4 堆火柴，分別有 7、9、12 和 15 根火柴。7、9、12 和 15 可以表示為二進位數字 0111、1001、1100 和 1111，如下表所示。

堆中的火柴數	$2^3=8$	$2^2=4$	$2^1=2$	$2^0=1$
7	0	1	1	1
9	1	0	0	1
12	1	1	0	0
15	1	1	1	1
	奇數	奇數	偶數	奇數

遊戲的初始狀態是非平衡的，走先手的玩家從一堆火柴中取出一些火柴，使得遊戲的狀態變成平衡的狀態。有多種選擇，例如，走先手的玩家從 12 根一堆的火柴中取出 11 根火柴，遊戲的狀態就變成平衡的狀態，如下表所示。

堆中的火柴數	$2^3=8$	$2^2=4$	$2^1=2$	$2^0=1$
7	0	1	1	1
9	1	0	0	1
12⇒1	0	0	0	1
15	1	1	1	1

走先手的玩家從一堆火柴中取出一些火柴，使得遊戲的狀態變成平衡的狀態的方法是，選擇表中的一行（某一堆火柴），並在這一行的奇數列翻轉二進位的值。在翻轉了這些值之後，在這一行裡，火柴的數量就少於初始的火柴數量。走先手的玩家從相關的堆中取走的火柴數量是初始火柴數量和目前火柴數量的差。然後，走後手的玩家在平衡的狀態下取火柴，狀態就會變成非平衡狀態。接下來，無論走後手的玩家取走多少根火柴，走先手的玩家都使得狀態平衡。這一過程一直重複，直到走後手的玩家最後一次在平衡狀態下取走一些火柴，然後走先手的玩家取走所有剩餘的火柴。

同理，遊戲的初始狀態是平衡狀態時，走後手的玩家會贏。

所以，本題演算法如下。

N 堆火柴表示為 *N* 個二進位數字。如果初始狀態是非平衡的，走先手的玩家贏；否則，走後手的玩家會贏。

❖ 參考程式

```
01  # include <cstdio>
02  # include <cstring>
03  # include <cstdlib>
04  # include <iostream>
05  # include <string>
06  # include <cmath>
07  # include <algorithm>
08  using namespace std;
```

```
09  int main(){
10      int n;
11      while(~scanf("%d",&n)){              // 輸入火柴堆數
12          int a=0,b;                       // 結果 a 初始化，目前堆的火柴數為 b
13          for(int i=0;i<n;i++){            // 輸入每堆火柴的數量
14              scanf("%d",&b);
15              a^=b;                        // 互斥目前堆的火柴數
16          }
17          printf("%s\n",a?"Yes":"No");    // 若 a 出現非平衡位，則走先手的玩家贏；
18                                           // 若 a 的所有位平衡，則走後手的玩家贏
19      }
20      return 0;
21  }
```

1.3 相關題庫

1.3.1 ▸ WERTYU

一種常見的打字錯誤是將手放在鍵盤上正確位置的右邊。因此造成將 Q 輸入為 W、將 J 輸入為 K，等等。請對以這種方式鍵入的訊息進行解碼。

輸入

輸入包含若干行文字。每一行包含數字、空格、大寫字母（除 Q、A、Z 之外），或如圖 1-4 中所示的標點符號（除倒引號 (`) 之外），用單字標記的鍵（Tab、BackSpace、Control 等）不在輸入中。

圖 1-4

輸出

對於輸入的每個字母或標點符號，用圖 1-4 所示的鍵盤上左邊的鍵的內容來替代。輸入中的空格也顯示在輸出中。

範例輸入	範例輸出
O S, GOMR YPFSU/	I AM FINE TODAY.

試題來源： Waterloo local 2001.01.27

線上測試： POJ 2538，ZOJ 1884，UVA 10082

提示

先根據圖 1-4 中的鍵盤，離線提供轉換表，儲存每個鍵對應的左側鍵（註：根據題意，單字鍵（Tab 鍵、BackSpace 鍵、Shift 鍵等）以及每一行最左邊的鍵（Q、A、Z）不在轉換表中。此外所有字母都是大寫的）。以後每輸入一個字母或標點符號，直接輸出轉換表中對應的左側鍵。

1.3.2 ▸ Soundex

Soundex 編碼是將根據它們的拼寫聽起來相同的單字歸類在一起。例如，can 和 khawn、con 和 gone 在 Soundex 編碼下是相等的。

Soundex 編碼涉及將每個單字轉換成一連串的數字，其中每個數字代表一個字母：

1 表示 B、F、P 或 V
2 表示 C、G、J、K、Q、S、X 或 Z
3 表示 D 或 T
4 表示 L
5 表示 M 或 N
6 表示 R

字母 A、E、I、O、U、H、W 和 Y 在 Soundex 編碼中不被表示，並且如果存在連續的字母，這些字母是用相同的數字表示的，那麼這些字母就僅用一個數字來表示。具有相同 Soundex 編碼的單字被認為是相等的。

輸入

輸入的每一行提供一個單字，全大寫，少於 20 個字母。

輸出

對每行輸入，輸出一行，提供 Soundex 編碼。

範例輸入	範例輸出
KHAWN PFISTER BOBBY	25 1236 11

試題來源：Waterloo local 1999.09.25

線上測試：POJ 2608，ZOJ 1858，UVA 10260

提示

由左到右將單字的每個字母轉化為對應數字並略去重複數字，就得到單字的 Soundex 編碼。

1.3.3 ▸ Mine Sweeper

踩地雷（Mine Sweeper）是一個在 $n \times n$ 的網格上玩的遊戲。在網格中隱藏了 m 枚地雷，每一枚地雷在網格上不同的方格中。玩家不中斷點擊網格上的方格。如果有地雷的方格被觸發，則地雷爆炸，玩家就輸掉了遊戲；如果一個沒有地雷的方格被觸發了，就出現 0 ～ 8 之間的整數，表示包含地雷的相鄰方格和對角相鄰方格的數目。圖 1-5 提供了玩該遊戲的部分連續截圖。

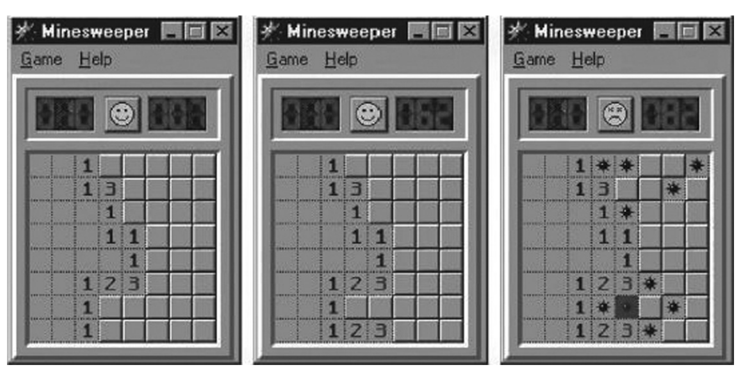

圖 1-5

在這裡，n 為 8，m 為 10，空白方格表示整數 0，凸起的方格表示該方格還未被觸發，類似星號的圖片則代表地雷。圖 1-5 中最左邊的圖表示這個遊戲開始玩了一會兒的情況。到中間的圖，玩家點擊了兩個方格，玩家每次都選擇了一個安全的方格。再到最右邊的圖，玩家就沒有那麼幸運了，他選擇了一個有地雷的方格，因此輸了遊戲。如果玩家繼續觸發安全的方格，直到只有 m 個包含地雷的方格沒有被觸發，則玩家獲勝。

請編寫一個程式，輸入遊戲進行的資訊，輸出相關的網格。

輸入

輸入的第一行提供一個正整數 n（$n \leq 10$）。接下來的 n 行描述地雷的位置，每行用 n 個字元表示一行的內容：句點表示方格沒有地雷，而星號代表這個方格有地雷。然後的 n 行每行提供 n 個字元：被觸發的位置用 x 標示，未被觸發的位置用句點標示，範例輸入對應於圖 1-5 中間的圖。

輸出

輸出提供網格，每個方格被填入適當的值。如果被觸發的方格沒有地雷，則提供 0 ～ 8 之間的值；如果有一個地雷被觸發，則所有有地雷的方格位置都用星號標示。所有其他的方格都用句點標示。

範例輸入	範例輸出
8 ...**..* ......*. ....*...	001..... 0013.... 0001.... 00011... 00001... 00123...

碁峰 www.gotop.com.tw

範例輸入	範例輸出
.....*.. ...**.*. .....*.. xxx..... xxxx.... xxxx.... xxxxx... xxxxx... xxxxx... xxx..... xxxxx...	001..... 00123...

試題來源：Waterloo local 1999.10.02

線上測試：POJ 2612，ZOJ 1862，UVA 10279

提示

試題提供了地雷矩陣 $g[i][j]$ 和觸發情況矩陣 try[i][j]（$1 \le i, j \le n$），要求計算和輸出網格。

首先判斷是否有地雷被觸發，即是否存在（try[i][j]=='x' && $g[i][j]$== '*'）的格子（i, j），設定地雷被觸發標誌

$$\text{mc} = \begin{cases} \text{'*'} & \text{地雷被觸發} \\ \text{'.'} & \text{地雷沒有被觸發} \end{cases}$$

然後從左向右計算和輸出每個位置（i, j）的網格狀態（$1 \le i, j \le n$）：

- 若（i, j）被觸發，但沒有地雷（try[i][j]=='x' && $g[i][j]$=='.'），則統計（i, j）的 8 個相鄰方格中有地雷的位置數 x，x 被填入（i, j）。
- 否則（即 try[i][j]=='.' ‖ $g[i][j]$=='*'），如果（i, j）有地雷，則 mc 被填入（i, j）；如果（i, j）沒有地雷，則 '.' 被填入（i, j）。

1.3.4 ▸ Tic Tac Toe

井字遊戲（Tic Tac Toe）是一個在 3×3 的網格上玩的遊戲。一個玩家 X 開始將一個 'X' 放置在一個未被占據的網格位置上，然後另外一個玩家 O 則將一個 'O' 放置在一個未被占據的網格位置上。'X' 和 'O' 就這樣被交替地放置，直到所有的網格被占滿，或者有一個玩家的符號在網格中占據了一整行（垂直、水平或對角）。

開始的時候，用 9 個點表示為空的井字遊戲，在任何時候放 'X' 或放 'O' 都會被放置在適當的位置上。圖 1-6 說明了從開始到結束井字遊戲的下棋步驟，最終玩家 X 獲勝。

...	X..	X.O	X.O	X.O	X.O	X.O	X.O
...	...	...	...	.O.	.O.	OO.	OO.
...	...	...	..X	..X	X.X	X.X	XXX

圖 1-6

請編寫一個程式，輸入網格，確定其是不是有效的井字遊戲的一個步驟。也就是說，透過一系列的步驟在遊戲的開始到結束之間產生這一網格。

輸入

輸入的第一行提供 N，表示測試案例的數目。然後提供 $4N-1$ 行，說明 N 個用空行分隔的網格圖。

輸出

對於每個測試案例，在一行中輸出 "yes" 或 "no"，表示該網格圖是否是有效的井字遊戲的一個步驟。

範例輸入	範例輸出
2 X.O OO. XXX O.X XX. OOO	yes no

試題來源： Waterloo local 2002.09.21

線上測試： POJ 2361，ZOJ 1908，UVA 10363

提示

由於玩家 X 先走且輪流執子，因此若網格圖為有效的井字遊戲的一個步驟，一定同時呈現下述特徵：

- 'O' 的數目一定小於等於 'X' 的數目；
- 如果 'X' 的數目比 'O' 多 1 個，那麼不可能是玩家 O 贏了井字遊戲；
- 如果 'X' 的數目和 'O' 的數目相等，則不可能是玩家 X 贏了井字遊戲。

網格圖為無效的井字遊戲的一個步驟，至少呈現下述 5 個特徵之一：

- 'O' 的個數大於 'X' 的個數；
- 'X' 的個數至少比 'O' 的個數多 2；
- 已經判出玩家 O 和玩家 X 同時贏；
- 已經判出玩家 O 贏，但 'O' 的個數與 'X' 的個數不等；
- 已經判出玩家 X 贏，但雙方棋子個數相同。

否則網格圖為有效的井字遊戲的一個步驟。