第 3 章　枚举算法

学习要点

- 理解枚举算法的基本思想和设计步骤
- 掌握枚举算法的典型应用范例

【引导问题】 “水仙花数”是一个神奇的 3 位数，其各位数字立方和等于该数本身。例如，153 就是一个水仙花数，因为$153=1^3+5^3+3^3$。怎样找出所有的“水仙花数”？

对于这个趣味数学问题，人们容易想到的解决方案是：对于 100～999 之间的每一个数, 按照水仙花数的条件逐一进行检验，找到一个就输出一个。这种一一列举每个可能值，然后逐一验证的方法被称为**枚举算法**。

尽管计算机没有思维，不能自主解决问题，只能机械地执行指令，但它运算速度快、存储容量大，很多人类手工求解非常困难的问题，计算机却能用最简单的方法进行准确求解。“大道至简”，枚举算法是最朴素和最简单的一种算法思维。


枚举算法-基本原理

3.1　枚举的基本思想

枚举算法，也被称为穷举算法，就是按照问题本身的性质，一一列举出该问题所有可能的解，并在逐一列举的过程中，检验每个可能解是否是问题的真正解，若是，则采纳它，否则抛弃它。在列举的过程中**既不能遗漏也不能重复**。如果有遗漏，则可能造成算法求解结果不正确；如果重复比较多，则将显著降低算法执行效率。

枚举算法是一种充分利用计算机快速计算能力的求解问题方法，是最基本、最朴素的算法思想。枚举算法具有以下三个突出的特点：

① 解的高准确性和全面性。因为枚举算法会检验问题的每一种可能情况，所以只要时间足够，枚举算法求解的答案肯定是正确的（如最优化问题，枚举算法求解出来的解能保证是最优解），还能方便地求解出问题的所有解。

② 实现简单。枚举算法常常通过循环来逐一列举和验证可能解，若用高级程序设计语言来表达，则枚举算法一般由多重 for 循环语句组成，程序逻辑结构清晰简单。

③ 效率提升空间比较大。枚举算法可直接用于求解规模比较小的问题和问题实例，但是当问题规模比较大时，枚举算法的效率通常比较低，需要进一步优化[1]。这是读者需要注意的地方。

1　其实很多算法都可以认为是对枚举算法的一种优化，如回溯法、分支限界算法等。

枚举算法是计算机问题求解最常用的方法之一，常用来解决那些通过公式推导、规则演绎的方法不能解决的问题，也用于解决一些规模比较小的问题。枚举算法求解问题的过程可以分为以下三步（“**枚举三步**”）：

① **确定枚举对象**。枚举对象是枚举算法的关键要素，一般需要若干参数来描述，每个参数包括自身的物理含义和取值范围等要素，这些参数能表征问题及问题解的本质特征。一般地，这些参数之间需要相互独立，而且，参数数目越少，问题解的枚举空间的维度也相应越小；每个参数的取值范围越小，问题解的搜索空间也越小。本质上，确定和刻画枚举对象的过程就是一个数学建模的过程。

② **逐一列举可能解**。根据枚举对象的参数构造循环，一一列举其表达式的每种取值情况。注意：如果枚举对象的参数个数是动态的（如有向图中的路径），那么逐一列举无法通过循环实现，此时可以考虑用递归技术来列举所有对象。如第 7 章中的深度优先搜索和广度优先搜索本质上都是枚举算法。

③ **逐一验证可能解**。根据问题解的要求，一一验证枚举对象表达式的每个取值，如果满足条件，则采纳它，否则抛弃之。

3.2 模糊数字

问题描述：一张单据上有一个 5 位数的编码，因为保管不善，其百位数字已经变得模糊不清。但是知道这个 5 位数是 57 和 67 的倍数。现在要设计一个算法，输出所有满足这些条件的 5 位数，并统计这样的 5 位数的个数。

输入：每行对应一个测试样例，每行包含 4 个数字，依次是万位数、千位数、十位数和个位数；最后一行包含 4 个-1，表示输入结束。

输出：每组测试样例的结果输出占一行，第一个数字表示满足条件的编码个数，后面按升序输出所有满足条件的编码，数字与数字之间用空格隔开。

输入样例：

```
1 9 9 5
-1 -1 -1 -1
```

输出样例：

```
1 19095
```

1．问题分析

首先，确定此问题中的枚举对象。在该 5 位数编码中只有百位数模糊不清，显然百位数字只能是 0～9 中的某个数字，共 10 种取值。因此，百位数字作为枚举对象，用参数 h 描述，$h \in \{0,1,2,3,4,5,6,7,8,9\}$。然后，把数字 h 和问题中已知的其他 4 个数字组成完整的 5 位数编码，假设为 v。最后，验证该 5 位数 v 能否同时被 57 和 67 整除，如果可以，则记录数 v。

容易得到，一个 for 循环就可以遍历和验证百位数 h 的所有可能值，并得到问题的解。

2．算法设计与实现

根据上述分析，本算法可以用一个 for 循环过程实现百位数字的遍历和验证。用

iCount 记录满足条件的 5 位数个数，用数组 iResult 存储满足条件的每个 5 位数，参考代码见程序 3-1。

程序 3-1　模糊数字问题求解程序

```
#include "stdio.h"
int main() {
    int d1, d2, d4, d5, h, iValue=0, iCount = 0;
    int iResult[10];
    scanf("%d%d%d%d", &d5, &d4, &d2, &d1);
    while(d5 !=-1){
        iCount=0;
        for(h=0; h<10; h++){                                    // 枚举对象参数
            iValue=d5*10000+d4*1000+h*100+d2*10+d1;
            if((iValue%57 == 0) && (iValue%67 == 0)){           // 验证
                iResult[iCount] = iValue;
                iCount++;
            }
        }
        printf("%d", iCount);
        for(int i=0; i<iCount; i++) {
            printf(" %d", iResult[i]);
        }
        printf("\r\n");
        scanf("%d%d%d%d", &d5, &d4, &d2, &d1);
    }
    return 0;
}
```

这是一个非常简单的枚举例题，因为枚举对象容易确定和描述，用一个参数即可，而且参数的取值范围非常小。但是并不是所有的情况都这样，下面看一个模糊数字问题的变种。

问题描述：一张单据上有一个 5 位数的编码，因为保管不善，其万位数字和百位数字已经变得模糊不清。但是知道这个 5 位数是 57 和 67 的倍数。现在要设计一个算法，输出所有满足这些条件的 5 位数，并统计这样的 5 位数的个数。

输入：每行对应一个测试样例，每行包含 3 个数字，依次是千位数、十位数和个位数；最后一行包含 3 个−1，表示输入结束。

输出：每组测试样例的结果输出占一行，第一个数字表示满足条件的编码个数，后面按升序输出所有满足条件的编码，数字之间用空格隔开。

输入样例：

```
9 9 5
-1 -1 -1
```

输出样例：

```
1 19095
```

1．问题分析

同样，按照枚举算法的“枚举三步”来分析这个问题。

首先，确定此问题中的枚举对象。在该 5 位数编码中，万位数字和百位数字都模糊不清，显然这两个目标数字都需要考虑，因此枚举对象需要两个描述参数：万位数字 $h_1(1 \leqslant h_1 \leqslant 9)$ 和百位数字 $h_2(0 \leqslant h_2 \leqslant 9)$。注意：$h_1$ 和 h_2 的取值范围不完全一样！整数的最高位不能为零。然后，把数字 h_1 和 h_2 以及其他已知的 3 个数字组成完整的 5 位数编码，假设为 v。最后，验证该 5 位数 v 能否同时被 57 和 67 整除，如果可以，则记录数 v，否则抛弃之。

不难发现，此时需要两层嵌套的 for 循环才能遍历 h_1 和 h_2 的所有可能值。

2．算法设计与实现

根据上述分析，本算法可以用两层嵌套的 for 循环实现万位数字和百位数字的遍历和验证。用 iCount 来记录满足条件的 5 位数个数，用数组 iResult 存储满足条件的每个 5 位数，参考代码见程序 3-2。

程序 3-2　变种模糊数字问题求解程序

```
#include "stdio.h"
int main() {
    int d1, d2, d4, h1, h2, iValue = 0, iCount = 0;
    int iResult[100];
    scanf("%d%d%d", &d4, &d2, &d1);
    while(d4 != -1) {
        iCount = 0;
        for(h1 = 1; h1 < 10; h1++)  {                          // 枚举对象参数
            for(h2 = 0; h2 < 10; h2++) {                       // 枚举对象参数
                iValue=h1*10000+d4*1000+h2*100+d2*10+d1;
                if((iValue%57 == 0) && (iValue%67 == 0)) {     // 验证
                    iResult[iCount] = iValue;
                    iCount++;
                }
            }
        }
        printf("%d", iCount);
        for(int i = 0; i < iCount; i++) {
            printf(" %d", iResult[i]);
        }
        printf("\r\n");
        scanf("%d%d%d", &d4, &d2, &d1);

    }
    return 0;
}
```

思考：如果模糊不清的是两位连续的数字，如千位数和百位数，怎样求解？

3.3　真假银币

问题描述：张三有 12 枚银币，其中有 11 枚真币和 1 枚假币。假币看起来与真币完

全一样，但是它们的重量不一样。遗憾的是，张三不知道假币比真币轻还是重。但是他办公室有一台天平，还有一个聪明的助手。经过精心安排每次的称重，助手保证在称 3 次后确定发现假币。助手想跟张三开一个小小的玩笑，只告诉他每次称重的方案和天平的状态，但是不告诉他哪个是假币，假币比真币轻还是重。请设计一个算法帮张三辨别真假银币。

输入：第一行包含一个正整数，表示测试数据的组数。每组测试数据有 3 行，每行表示一次称重的结果。张三和助手事先把银币标号为 A～L。每次称重的结果用 3 个以空格隔开的字符串表示：天平左边放置的银币标号，天平右边放置的银币标号，以及平衡状态。其中，平衡状态分别用 up、down 和 even 表示，分别表示右端高、右端低和平衡。另外，每次称重天平左右的银币数总是相等的。

输出：每组测试数据的输出占一行，输出假银币的标号，并指明它比真币轻还是重，轻则输出 light，重则输出 heavy。

输入样例：

```
1
ABCD EFGH even
ABCI EFJK up
ABIJ EFGH even
```

输出样例：

```
K light
```

1．问题分析

在此问题中，助手已经设计了正确的称重方案，即保证从 3 组称重数据中能得到唯一的答案。而且答案包含两个要素：假币的标号，表示为 s；假币比真币轻还是重，表示为 b。于是，枚举对象可以表示为一个二元组 (s,b)，且 $s \in \{A,B,\cdots,L\}$，有 12 个可能值；$b \in \{\text{light},\text{heavy},\text{true}\}$，light 表示假币比真币轻，heavy 表示假币比真币重，true 表示银币为真。注意：银币的性质只能是 light、heavy 和 true 中的一种。然后，一一列举每个二元组 (s,b)，共 36 个不同的二元组。依据题意，只有一个二元组 (s,b) 与输入的 3 组称重数据都不矛盾，更具体地说，如果二元组 (s,b) 满足下列条件，则它就是问题的真正解：

- 在称重结果为 even 的天平两边都没有出现 s。
- 如果 $b=\text{light}$，则在称量结果为 up 的天平右边肯定出现 s，而在称量结果为 down 的天平左边一定出现 s。
- 如果 $b=\text{heavy}$，则在称量结果为 up 的天平左边肯定出现 s，而在称量结果为 down 的天平右边一定出现 s。

2．算法设计与实现

具体实现时需要注意两点：

（1）选择合适的算法

对每个银币 s 逐一验证：

- s 比真币轻的猜测是否成立？猜测成立，则进行输出。
- s 比真币重的猜测是否成立？猜测成立则进行输出。
- 如果上述两种猜测都不成立，则表明 s 是真币，继续验证下一个银币。

（2）选择合适的数据结构

以字符串数组存储每次称量的结果。每次称量时，天平左右两边最多有 6 枚银币，因此，字符串的长度需要为 7，最后 1 位存储字符串的结束符“\0”，便于程序中使用字符串操作函数。实现代码见程序 3-3。

程序 3-3　真假银币问题求解程序

```
#include "stdio.h"
#include "string.h"

char left[3][7],right[3][7],results[3][7];
bool isHeavy(char);
bool isLight(char);
int main() {
    char curChar;
    int n, i;
    scanf("%d",&n);
    while(n > 0) {
        for(i = 0; i < 3; i++)
            scanf("%s%s%s", left[i], right[i], results[i]);
        for(curChar = 'A'; curChar <= 'L'; curChar++) {
            if(isLight(curChar)) {                          // 判断结果为轻
                printf("%c %s\r\n", curChar, "light");
                break;
            }
            if(isHeavy(curChar)) {                          // 判断结果为重
                printf("%c %s\r\n",curChar,"heavy");
                break;
            }
        }
        n--;
    }
    return 0;
}
```

判断当前字符 curChar 为假币，且比真币轻的代码见程序 3-4。

程序 3-4　判断银币为轻

```
bool isLight(char curChar) {
    for(int i = 0; i < 3; i++) {
        switch (results[i][0]) {
            case 'u':  if(strchr(right[i], curChar) == NULL)
                           return false;
            case 'e':  if((strchr(right[i],curChar) !=NULL) || (strchr(left[i],curChar) !=NULL))
                           return false;
            case 'd':  if(strchr(left[i], curChar) == NULL)
                           return false;
        }
    }
```

```
        return true;
    }
```

判断当前字符 curChar 为假币，且比真币重的代码见程序 3-5。

程序 3-5　判断银币为重

```
bool isHeavy(char curChar) {
    for(int i = 0; i < 3; i++) {
        switch (results[i][0]) {
            case 'u':  if(strchr(left[i], curChar) == NULL)
                           return false;
            case 'e':  if((strchr(right[i],curChar)!=NULL)||(strchr(left[i],curChar)!=NULL))
                           return false;
            case 'd':  if(strchr(right[i], curChar) == NULL)
                           return false;
        }
    }
    return true;
}
```

3.4　*m* 钱 *n* 鸡

问题描述：我国古代数学家张丘建在《算经》中出了一道题“鸡翁一，值钱五；鸡母一，值钱三；鸡雏三，值钱一。百钱买百鸡，问鸡翁、鸡母、鸡雏各几何？”现在假定各鸡种的价格不变，拥有的钱数为 *m*，需要购买的鸡数为 *n*，试求出所有可能的购买方案总数。

输入：每行对应一个测试样例，每行包含 2 个数字，分别为 *m* 和 *n*；最后一行包含 2 个−1，表示输入结束。

输出：每组测试样例的结果输出占一行，输出可能购买方案总数。

输入样例：

```
100 100
-1 -1
```

输出样例：

```
4
```

1．问题分析

显然，此问题的任何一个购买方案都会包含鸡翁、鸡母、鸡雏的数目，假设分别用 *n*1、*n*2、*n*3 表示，且 $0 \leqslant n1$，$n2$，$n3 \leqslant n$ 于是枚举对象可直接表示为一个三元组 $(n1, n2, n3)$；然后，一一列举每个三元组，并判断表达式 $n1+n2+n3=n$、$5n_1+3n_2+\frac{1}{3}n_3=m$ 是否成立，如果成立，则该三元组是一个可行的购买方案。

2．算法设计与实现

根据上述分析，本算法采用三层嵌套的 for 循环实现，见程序 3-6。

程序 3-6　*m* 钱 *n* 鸡问题求解程序一

```
#include "stdio.h"
int main() {
    int N, M, n1, n2, n3, iCount = 0;
    scanf("%d %d", &N, &M);
    while(N != -1) {
        iCount = 0;
        for(n1 = 0; n1 < N; n1++)
            for(n2 = 0; n2 < N; n2++)
                for(n3 = 0; n3 < N; n3++)                          // 枚举对象三元组
                    if((n1+n2+n3 == N) && (5*n1+3*n2+n3/3.0 == M)) // 验证，是 3.0，不是 3
                        iCount++;
        printf("%d\r\n", iCount);
        scanf("%d%d", &N, &M);
    }
    return 0;
}
```

按照上述思路求解，算法实现需要 3 层嵌套的 for 循环，共 n^3 次验证，算法复杂度为 O(n^3)。有兴趣的同学可试验输入 n=m=10000 时，程序 3-6 的执行时间是多少？

如果深入分析这个问题，可以发现鸡翁的价格为 5，显然购买的鸡翁的数目不可能超过 m/5；同理，购买的鸡母的数目不可能超过 m/3。因此，n_1 和 n_2 的取值范围可以进一步缩小，即 $0 \leqslant n_1 \leqslant m/5$，$0 \leqslant n_2 \leqslant m/3$。另外，当 n_1 和 n_2 的值确定以后，n_3 的值就唯一确定，即 $n_3 = n - n_1 - n_2$，而不需枚举其他值。也就是说，n_1、n_2、n_3 不是相互独立的，而是线性相关的。依上所述，可以把 3 层嵌套的 for 循环改进为 2 层嵌套循环，算法复杂度减少为 O(n^2)。参考代码见程序 3-7。

程序 3-7　*m* 钱 *n* 鸡问题求解程序二

```
#include "stdio.h"
#include "stdafx.h"
int main() {
    int N, M, n1, n2, n3, iCount = 0;
    scanf("%d %d", &N, &M);
    while(N != -1) {
        iCount = 0;
        for(n1 = 0; n1 <= M/5; n1++) {
            for(n2 = 0; n2 <= M/3; n2++) {              // 枚举对象二元组
                n3 = N - n1 - n2;
                if(5*n1 + 3*n2 + n3/3.0 == M)           // 验证，注意是 3.0，不是 3
                    iCount++;
                }
            }
        }
        printf("%d\r\n", iCount);
        scanf("%d%d", &N, &M);
    }
    return 0;
}
```

3.5 数字配对

问题描述：输入一个长度为 n 的数组 A 和一个正整数 k，从数组中选择两个数，记为 $a[i]$和 $a[j]$，使其和是 k 的倍数。设计一个算法计算有多少种不同选择方法。

假设给定数组元素 a_1=1，a_2=2，a_3=2，(a_1, a_2)和(a_2, a_1)被认为是同一种选法，但是(a_1, a_2)和(a_1, a_3)被认为是不同的选法。

枚举优化-数组配对

输入：

第一行有两个正整数 n 和 k，n≤1000000，k≤1000；第二行有 n 个正整数，每个数的大小不超过 10^9。

输出：

输出符合要求的选择方法数目。

输入样例：

```
5 6
1 2 3 4 5
```

输出样例：

```
2
```

1．问题分析

根据题意，我们可以直接枚举所有的选择组合。每种符合要求的选择组合可以表示为一个二元组 (a_i, a_j)，i 和 j 表示对应元素在数组中的下标，取值范围为 0<i<n，i<j<n。然后一一列举每个二元组 (a_i, a_j)，并且判断 $(a_i + a_j)\,\%\,k$ 的结果是否等于 0，如果等于，则增加一个选法，否则抛弃之。

2．算法设计与实现

根据上述分析，本算法可以用两层嵌套的 for 循环实现，见程序 3-8。

程序 3-8　数字配对问题求解算法一

```
#include<iostream>
#include "string.h"
using namespace std;
#define  LL  long long
int n, k;
int a[1000011];  //全局变量
int main(){
     memset(a,0,sizeof(a));
     cin>>n>>k;
     LL ans=0;
     for(int i=0; i<n; i++)
        cin>>a[i];
     for(int i=0; i<n; i++)
        for(int j=i+1; j<n; j++)
           if ((a[i]+a[j])%k==0)
              ans++;
     cout<<ans<<endl;
```

```
    return 0;
}
```

上述算法思路清晰，实现简便，但是其时间复杂度为 $O(n^2)$。当输入数组规模比较大时，算法执行时间增长会非常快。有没有更加有效的算法呢？

在算法一中，枚举对象是由数组元素构成的二元组，规模比较大。根据同余性质（即 $(a_i + a_j) \% k = 0$ 等价于 $(a_i \% k + a_j \% k) \% k = 0$），我们可以把余数相同的数组元素合并为一个子集，然后**把子集作为枚举对象**。假设子集 $b_t = \{a_i \mid a_i \% k = t\}(0 \leqslant t < k)$，则枚举对象设定为新的二元组 (b_i, b_j)，其中 i 和 j 表示子集对应的余数，取值范围为 $0 \leqslant i, j < k$。然后一一列举每个二元组 (b_i, b_j)，并且判断 $(i+j) \% k$ 的结果是否等于 0，如果相等，则增加的选法数为 $|b_i| \times |b_j|$（$|\cdot|$ 表示集合中元素的个数），否则抛弃之。有两种情况需要特殊考虑：$t = 0$ 或 $t = k/2$ 时，(b_t, b_t) 可以构造合适的二元组，其数目为 $|b_t| \times (|b_t| - 1) / 2$。

根据上面的分析，我们可以得到优化后的算法，参考代码见程序 3-9。

程序 3-9　数字配对问题求解算法二

```
#include<iostream>
#include "string.h"
using namespace std;
#define       LL        long long
int n, k;
int a[1000011];                              // 全局变量

void main() {
    memset(a, 0, sizeof(a));
    cin>>n>>k;
    for(int i=0; i < n; i++) {
        int t;
        cin>>t;
        a[t%k]++; //复用数组 a
    }
    work();
}
void work() {
    LL ans = 0;
    for(int i = 0; i < k; i++) {
        int j = (k-i)%k;
        if(j < i)
            break;
        else if(j == i)
            ans += 1LL*a[i]*(a[i]-1)/2;
        else
            ans += 1LL*a[i]*a[j];
    }
    cout<<ans<<endl;
}
```

算法二的时间复杂度为 $O(n)$，主要的时间开销在于把输入数组中的元素归并到相应同余子集中，枚举同余子集 (b_i,b_j) 的时间开销为 $O(k)$，一般 k 的值非常小。

3.6 绳子切割

问题描述：有 n（$n \leqslant 1000$）条绳子，它们的长度分别为 l_i（$l_i \leqslant 1000000000000$）。如果从它们中切割出 k（$k \leqslant 1000000$）条长度相同的绳子，那么这 k 条绳子每条最长能有多长？答案保留到小数点后 2 位。

输入：包括 2 行，第一行输入 n 和 k，第二行分别输入 n 条绳子的长度。

输出：切割出的绳子的最大长度值。

输入样例：

```
4 11
8.02 7.43 4.57 5.39
```

输出样例：

```
2.00
```

提示：每条绳子分别可以得到 4 条、3 条、2 条、2 条，共 11 条。

1．问题分析

根据题意，我们可以直接把切割出的绳子的长度 l 作为枚举对象，其取值范围为 $0.00 \leqslant l \leqslant 1000000000000.00$。然后以 0.01 为步长从大到小一一列举每个值，如果满足 $\sum_{i=1}^{n}\lfloor l_i / l \rfloor \geqslant k$，则当前值是答案，否则继续列举并验证。

2．算法设计与实现

根据上述分析，本算法可以用一层循环实现，见程序 3-10。

程序 3-10　绳子切割问题求解算法一

```
#include<iostream>
#include<iomanip>
using namespace std;

int  n, k;
double l[1000011];
int main() {
    cin>>n>>k;
    for (int i=0; i < n; i++)
        cin>>l[i];
    for (double L = 1000000000000; L >= 0; L -= 0.01) {
        double sum=0;
        for (int i=0; i < n; i++)
            sum += ll(l[i]/L);
        if (sum >= k) {
            cout<<fixed<<setprecision(2);
            cout<<L<<endl;
```

```
            break;
        }
    }
    return 0;
}
```

上述算法的时间复杂度为线性的，但是执行效率取决于最长的绳子的长度。有没有更优的算法呢？

算法一使用的是朴素的枚举方法，也就是按照给定步长一一列举所有可能的长度值 l。但是我们可以优化列举的过程，改善算法的时间复杂度。根据题意，枚举对象的下界 $l_{\mathrm{d}}=0.00$，上界 $l_{\mathrm{u}}=1000000000000.00$，其中值可表示为 $\mathrm{mid}=(l_{\mathrm{d}}+l_{\mathrm{u}})/2$。我们先验证 mid，如果满足 $\sum_{i=1}^{n}\lfloor l_i/\mathrm{mid}\rfloor\geqslant k$，则可以切割出 k 条长度等于 mid 的绳子，则比 mid 小的值肯定也满足切割要求，但不是最优解，因此不用考虑，即 l 的下界更新为 $l_{\mathrm{d}}=\mathrm{mid}$；否则，不能切割出 k 条长度等于 mid 的绳子，则比 mid 大的值肯定也不满足，因此 l 的上界更新为 $l_{\mathrm{u}}=\mathrm{mid}-0.01$。这个过程可以以此类推，直到 $l_{\mathrm{d}}=l_{\mathrm{u}}$，则算法终止。这种优化的枚举算法称之为**二分法**。

根据上面的分析，我们可以得到优化后的算法，参考代码见程序 3-11。

程序 3-11　绳子切割问题求解算法二

```
#include<iostream>
#include<iomanip>
using namespace std;

int n, k;
double l[1000011];
int main() {
    cin>>n>>k;
    for (int i=0; i < n; i++)
        cin>>l[i];
    double ld=0;
    double lu=1000000000000;
    while (lu - ld >0.01) {
        double mid = (ld+lu) / 2;
        double sum=0;
        for (int i=0; i < n; i++)
            sum += LL(l[i] / mid);
        if (sum < k)
            lu = mid;
        else
            ld = mid;
    }
    cout<<fixed<<setprecision(2);
    cout<<ld<<endl;
    return 0;
}
```

一般地，能用二分法求解的问题可以归纳为以下 2 类问题原型：

①“求满足某个条件 $C(x)$的**最小 x**”，其中 $C(x)$满足：如果任意 x 满足 $C(x)$，则所有的 $x' \geqslant x$ 也满足 $C(x')$。

②“求满足某个条件 $C(x)$的**最大 x**”，其中 $C(x)$满足：如果任意 x 满足 $C(x)$，则所有的 $x' \leqslant x$ 也满足 $C(x')$。

3.7 石头距离

问题描述：有一条河，河中间有一些石头，石头的数量和相邻两颗石头之间的距离已知。现在可以移除一些石头，假设最多可以移除 m 颗石头（注意：首尾两颗石头不可以移除，且假定所有的石头都处于同一条直线），问最多移除 m 颗石头后，相邻两颗石头之间的**最小距离的最大值**是多少？

输入：多组输入（不超过 20 组数据，读入以 EOF 结尾），每组第一行输入两个数字：n（$2 \leqslant n \leqslant 30$）为石头的个数，$m$（$0 \leqslant m \leqslant n-2$）为可移除的石头数目。随后 n−1 个正整数表示顺序相邻两块石头的距离 d（$d \leqslant 1000$）。

输出：每组输出一行结果，表示最大值。

输入样例：

```
5 2
4 1 3 2
```

输出样例：

```
5
```

提示：此时移走第 2 颗和第 4 颗石头。

1．问题分析

根据题意，移除的石头排列 $(S_1, S_2, \cdots, S_{m-1}, S_m)$ 设定为枚举对象，其中，$S_1 \in \{2, \cdots, n-1\}$，$S_i \in \{S_{i-1}, +1, \cdots, n-1\}$ $(1 < i \leqslant m)$。但是参数 m 是动态的，列举枚举对象不太方便。我们也可以设计不同的模型来表示枚举对象，即 n−2 维 0-1 向量 $b_2, \cdots, b_{n-1}$，其中 $b_i \in \{0,1\}$ $(2 \leqslant i \leqslant n-1)$表示第 i 块石头的状态，1 表示移除，0 则表示保留。此模型可以用递归方法来枚举，见程序 3-12。

程序 3-12　石头距离问题求解算法一

```
#include<iostream>
#include<iomanip>
#include<algorithm>
using namespace std;
#define        LL    long long

int n, m;
int p[1011];
int main() {
```

```
    cin>>n>>m;
    int nowpos=p[0]=0;
    for (int i=1; i<n; i++) {
        int t;
        cin>>t;
        nowpos+=t;
        p[i]=nowpos;
    }
    int maxans=0;
    for (int s=0; s<(1<<n); s++) {            // 此处 n 必须小于 31，否则会超出 int 范围
        int removeNum=0;
        for (int i=0; i<n; i++)
            if ((s>>i) & 1)
                removeNum++;
        if (removeNum != m)
            continue;
        int lastPos=-1;
        int minDis=1e9;
        for (int i=0; i<n; i++) {
            if ((s>>i) & 1)
                continue;
            if (lastPos != -1)
                minDis = min(minDis, p[i] - lastPos);
            lastPos=p[i];
        }
        maxans = max(maxans, minDis);
    }
    cout<<maxans<<endl;
    return 0;
}
```

算法一本质上枚举了所有的 $n-2$ 维 0-1 向量，共 2^{n-2} 个，因此该算法的时间复杂度是指数级别。

这个问题还可以应用二分法直接枚举最小距离。首先，这个问题可以表述为以下模型：求满足条件 $C(d)$ 的最大值 d，其中 $C(d)$:= 最多移除 m 块石头后**最近**两块石头的距离不小于 d。为了求解方便，我们可以把条件 $C(d)$进一步简化，即得到新模型：求满足条件 $C(d)$ 的最大值 d，其中 $C(d)$:= 最多移除 m 块石头后任意相邻两块石头的距离不小于 d。显然，条件 $C(d)$满足单调性，即如果任意 x 满足 $C(x)$，则所有的 $x' \leqslant x$ 也满足 $C(x')$。因此，此问题可以采用二分法快速枚举距离 d。

2．算法设计与实现

首先，设计一个判定算法 $C(d)$，判定最多移除 m 块石头后任意相邻两块石头的距离大于 d。应用贪心思想，$C(d)$的判定程序可以实现如下。

程序 3-13　$C(d)$的判定算法

```
int Validate(int d) {
```

```
    int k = m;                                  // 可以移除的石头数目
    int st = 1;                                 // 表示最初的石头
    for(int en = 2; en <= n;;) {                // 表示结尾的石头
        int disCur = dis[en] - dis[st];         // st 与 en 石头的间距
        while(disCur < d) {                     // 此时可以移除第 en 个石头
            k--;                                // 移除当前石头
            en++;                               // 考虑下一个石头
            if (k < 0)
                return 0;                       // 移除石头大于 m
            if (en > n) {                       // 特殊情况
                if (st == 1)                    // 第 1 块与第 n 块石头间的距离小于 d
                    return 0;
                else
                    return 1;                   // 可以把 st 移除与前一个区间合并满足条件
            }
            disCur = dis[en] - dis[st];
        }
        st=en;                                  // 更新起点
        en++;
    }
    return 1;
}
```

显然，算法 Validate 的时间复杂度是线性的，为 $O(n)$。基于此判定算法，我们应用二分法思想快速枚举最小距离，见程序 3-14。

程序 3-14　石头距离问题求解算法二

```
int a[1005], dis[1005]={0};
int n, m;
int main(){
    while(~scanf("%d%d", &n, &m)){
        for(int i=2; i <= n; i++) {          // a[i]表示石头 i-1 到 i 的间距，下标从 1 开始
            scanf("%d", &a[i]);
            dis[i] = a[i] + dis[i-1];        // sum[i]表示石头 1 到 i 的间距
        }
        int lb = 0, ub = 1000*1000+5;        // 距离上界
        while(lb<ub) {
            int mid = (lb+ub) / 2;
            if(Validate(mid))
                lb = mid+1;                  // 更新下界
            else
                ub = mid-1;                  // 更新上界
        }
        printf("%d\n", lb);
    }
    return 0;
}
```

在上述二分枚举方法中，函数 Validate( )的时间复杂度为 $O(n)$，其调用的次数为 $\log_2 L$，其中 L 是一个常数，表示可能的最大距离。综合起来，二分枚举算法的时间复杂度为多项式级别，远远优于算法一。

在上述例子中，枚举对象相对比较简单，可以用循环列举所有的对象。需要特别指出的是，并不是所有的枚举算法都能用循环实现。第 7 章介绍的搜索技术本质上是一种带优化的枚举策略，其枚举过程用深度优先搜索和广度优先搜索等方法实现。

习 题 3

3-1　模糊数字

问题描述：一张单据上有一个 8 位数的编码，因为保管不善，只有第 7、3、2 位数字清楚。但是知道这个 8 位数是 57 和 67 的倍数。现在要设计一个算法，求出所有满足这些条件的 8 位数的个数。

输入：每行对应一组测试样例，每行包含 3 个数字，依次是第 7、3、2 位数字；最后一行包含 3 个 0，表示输入结束。

输出：满足条件的 8 位数的个数，每组测试数据输出占一行。

输入样例：

```
9 9 5
```

输出样例：

```
24
```

3-2　完美立方

问题描述：$a^3=b^3+c^3+d^3$ 为完美立方等式，如 $12^3=6^3+8^3+10^3$。编写一个程序，对任给定正整数 N（$N\leqslant 100$），寻找所有的四元组 (a,b,c,d)，使得 $a^3=b^3+c^3+d^3$，其中 $1<a,b,c,d\leqslant N$。

输入：每行对应一个测试数据，包含一个正整数 N（$N\leqslant 100$）；最后一行包含一个 0，表示输入结束。

输出：满足完美立方等式的四元组数目，每组测试数据输出占一行。

输入样例：

```
24
```

输出样例：

```
7
```

3-3　整数近似

问题描述：Forth 语言不支持浮点运算，它的作者 Chuck Moore 认为浮点运算太慢，而且在很多情况下能用两个整数的商来代替。如在计算半径为 R 的圆面积时，他建议用公式 $R\times R\times 355/113$ 代替，而且计算结果精确度还比较高。$355/113\approx 3.141593$，它与圆周率π的绝对误差不超过 2×10^{-7}。任意给定一个浮点数 A 和整数 L，请设计算法求出两个正整数 N 和 D（$1\leqslant N,D\leqslant L$），使得绝对误差 $\left|A-N/D\right|$ 最小。

输入：每行对应一组测试数据，每行包含浮点数 A（$0.1\leqslant A<10$）和整数 L

（$1 \leqslant L \leqslant 100000$）。

输出：正整数 N 和 D，用空格隔开，每组测试数据输出占一行。

输入样例：

```
3.14159265358979 10000
```

输出样例：

```
355 113
```

3-4　字符串位置

问题描述：给出一个很长的数字串 S：12345678910111213 14…，它由所有的自然数从小到大依次排列得到。任意给出一个数字串 S_1（其长度不大于 4），求出 S_1 在 S 中第一次出现的位置。

输入：每行对应一个测试样例，输入字符串 S_1。

输出：该字符串出现的第一个位置，每组测试样例输出一行。

输入样例：

```
101
2132
```

输出样例：

```
10
529
```

3-5　方程求解

问题描述：形如 $ax^3+bx^2+cx+d=0$ 的一个一元三次方程。给出该方程中各项的系数（a，b，c，d 均为实数），并约定该方程存在 3 个不同实根（根的范围为-100～100），且根与根之差的绝对值大于或等于 1。要求由小到大依次在同一行输出这 3 个实根（根与根之间留有空格），并精确到小数点后 2 位。

提示：方程 $f(x)=0$，若存在 2 个数 x_1 和 x_2，且 $x_1 < x_2$，$f(x_1) \times f(x_2) < 0$，则在 (x_1, x_2) 之间一定有一个根。

输入：每行对应一组测试样例，输入系数 a、b、c 和 d。

输出：由小到大依次输出 3 个实根，用空格隔开，每组测试样例输出一行。

输入样例：

```
-1 6 -11 6
```

输出样例：

```
1.00 2.00 3.00
```

3-6　除法问题

问题描述：输入正整数 n，按从小到大的顺序输出所有形如 $abcde/fghij=n$ 的表达式，其中 a～j 恰好为数字 0～9 的一个排列，$2 \leqslant n \leqslant 79$。

输入：正整数 n。

输出：所有满足要求的表达式，每个表达式占一行。

输入样例：

```
62
```

输出样例：

```
79546/01283=62
94736/01528=62
```

3-7　分数拆分

问题描述：输入正整数 k，找到所有的正整数 $x \geqslant y$，使得 $\frac{1}{k}=\frac{1}{x}+\frac{1}{y}$。

输入：正整数 k。

输出：所有满足要求的表达式，每个表达式占一行。

输入样例：

```
2
```

输出样例：

```
1/2=1/6+1/3
1/2=1/4+1/4
```

3-8　玻璃球的移动

问题描述：有 3 个桶用来装回收的玻璃瓶，玻璃瓶的颜色有三种：棕色（Brown）、绿色（Green）、透明色（Clear）。已知每个桶中的玻璃瓶的颜色及数量，现在要搬移桶子里的玻璃瓶，使得最后每个桶中都只有单一颜色的玻璃瓶，以方便回收。任务是算出最少搬移的瓶子数。假设每个桶子的容量无限大，并且总共搬移的瓶子数不会超过 2^{31}。

输入：每个输入样例占一行，每行有 9 个整数：前 3 个代表第 1 个桶中棕色、绿色、透明色的瓶子数。接下来的 3 个数代表第 2 个中棕色、绿色、透明色的瓶子数。最后的 3 少数代表第 3 个桶中棕色、绿色、透明色的瓶子数。

例如，10 15 20 30 12 8 15 8 31 表示有 20 个 Clear 色的玻璃瓶在第 1 个桶中，12 个 Green 色的玻璃瓶在第 2 个桶中，15 个 Brown 色的玻璃瓶在第 3 个桶中。

输出：对每个测试样例，输出 3 个桶中最后存放之玻璃瓶颜色，以及最小搬移的瓶子数。以 G、B、C 分别代表绿色（Green）、棕色（Brown）、透明色（Clear）。

例如，BCG30 代表最后搬移的结果是第 1 个桶中的玻璃瓶颜色为 Brown，第 2 个桶中的玻璃瓶颜色为 Clear，第 3 个桶中的玻璃瓶颜色为 Green，并且总共搬移了 30 个玻璃瓶。

如果最小搬移瓶子数有一组以上的组合，请输出字典顺序最小的那组答案。

输入样例：

```
1 2 3 4 5 6 7 8 9
5 10 5 20 10 5 10 20 10
```

输出样例：

```
BCG30
CBG50
```

3-9　素数环

问题描述：输入正整数 n，把整数 $1,2,3,\cdots,n$ 组成一个环，使得相邻两个整数之和均为素数；输出时从整数 1 开始逆时针排列；同一个环应恰好输出一次。

输入：正整数 n，n<17。

输出：整数环序列，从 1 开始逆时针排列。

输入样例：

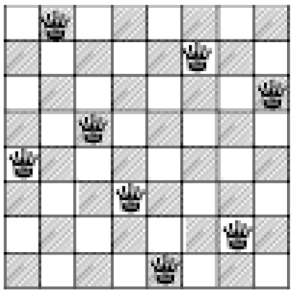
```
6
```

输出样例：

```
1 4 3 2 5 6
1 6 5 2 3 4
```

3-10　有障碍物的 *n* 皇后问题

问题描述：相信 *n* 皇后问题对每个研究递归的人来讲都不陌生，这个问题是在一个 *n*×*n* 大小的棋盘上摆 *n* 个皇后，让它们不会互相攻击到。为了让这个问题更难，我们设计了一些障碍物在棋盘上，这些点上不能放皇后，但是这些障碍物并不会防止皇后被攻击。



在传统的 8 皇后问题中，旋转与映射被视为不同解法，因此我们有 92 种可能的方式来放置皇后。

输入：输入的数据最多包含 10 个测试样例，每个测试样例的第一行会有一个整数 *n*（3<*n*<15）；接下来的 *n* 行代表棋盘数据，'.'代表空的盘格，'*'代表放有障碍物的盘格；0 代表输入结束。

输出：对每个测试样例，输出这是第几个 case 以及这个 case 有几种可能的放置方式。

输入样例：

```
8
........
........
........
........
........
........
........
........
4
.*..
....
....
....
0
```

输出样例：

```
Case 1: 92
Case 2: 1
```

3-11　跳舞的数字

问题描述：数字喜欢跳舞。有一天，1、2、3、4、5、6、7 和 8 排成一列要来跳舞。每次一个“公”数字可以邀请“母”数字和他跳舞，或者一个“母”数字可以邀请“公”数字和她跳舞，前提是他们的和必须是质数。在每次跳舞前，仅有一个数字来到他想要邀请的对象的左边或右边。

为了简便说明，我们定义一个数字“公”或“母”是根据正负来表示。例如，4 代表公的，−7 代表母的。假设原来数字的排列为{1, 2, 4, 5, 6, −7, −3, 8}。假如−3 想要和 4 跳舞，她可以来到 4 的左边，变成{1, 2, −3, 4, 5, 6, −7, 8}。或者她可以来到 4 的右边，变成{1, 2, 4, −3, 5, 6, −7, 8}。注意：−3 不能和 5 跳舞，因为他们的和（不管正负号）3+5=8 不是质数。2 也不能和 5 跳舞，因为他们都是公的。

给出数字一开始排列的顺序，请找出最小跳舞的次数，使得最后排列的顺序是递增的（不管正负号）。

输入：输入含有多组测试数据（最多 20 组）；每组测试数据一列，含有绝对值 1～8 这 8 个整数的某种排列；最后一列仅有一个 0，代表输入结束。

输出：对每组测试数据，输出这是第几组测试数据，以及最小跳舞的次数，使得最后排列的顺序是递增的。假如不可能，输出−1。

输入样例：

```
1 2 4 5 6 -7 -3 8
1 2 3 4 5 6 7 8
1 2 3 5 -4 6 7 8
1 2 3 5 4 6 7 8
2 -8 -4 5 6 7 3 -1
0
```

输出样例：

```
Case 1: 1
Case 2: 0
Case 3: 1
Case 4: -1
Case 5: 3
```

3-12　KTV 的组合

问题描述：最近有一首三人合唱的歌很流行，你与朋友共 9 个人一同到 KTV 欢唱，一人只能唱一次，也就是将 9 个人分成 3 组，一组 3 人，每人刚好都被分派到一个组别。

但是有些人并不喜欢与另一些人搭档，而有些组合的效果并不好听，所以我们对所有可能的三人组合打分数，请找出 9 人最高的分组分数总和。

输入：输入最多有 1000 组测试数据，每组数据的第一行有一个整数 $n(0<n<81)$ 表示所有可能的组合总数，接下来有 n 行，每行有 4 个整数表示一种组合，4 个整数分别为 a, b, c, s $(1\leqslant a<b<c\leqslant 9, 0<s<10000)$，表示$(a, b, c)$这三人的组合的分数为 s。n=0 表示测试结束。

输出：请对每组测试数据输出其数据编号及最高的分数，若不存在任意一组可能的组合，则输出−1。

输入样例：

```
3
1 2 3 1
4 5 6 2
7 8 9 3
4
1 2 3 1
1 4 5 2
1 6 7 3
1 8 9 4
0
```

输出样例：

```
Case 1: 6
Case 2: -1
```