



浙江财经大学

Zhejiang University Of Finance & Economics



程序设计实践-枚举

信智学院 陈琰宏





什么是枚举?

- **枚举**又称为**穷举**，是一种很朴素的解题思想。
- 当需要求解的问题存在大量的可能的答案(或中间过程)，而暂时又无法用逻辑方法排除这些可能答案中的大部分时，就**不得不采用逐一检验这些答案的策略**，也就是利用**枚举法**来解题。

例如，在**判断 $m=199$ 是否为素数**时，从素数的定义出发，试图找出2~198范围内能整除 m 的自然数，如果不能找到，则 m 是素数。根据分析，我们将范围缩小到2~14。依次判断2能否整除 m ，判断3能否整除 m ，...，一直判断到14都还没有找到能整除 m 的自然数，因此得出结论： $m=199$ 是素数。这个过程实际上就是一个枚举的过程，**枚举2, 3, 4, 5, ..., 14能否整除 m** 。





1[3309] 求 $x^2 + y^2 = 2000$ 的正整数解。

分析： x 和 y 都是正整数，因此 x 和 y 的取值范围只能是：1、2、...、44，其中44是小于等于 $\sqrt{2000}$ 的最大正整数。对于**在这个范围内的所有 (x, y) 组合**，都去判断一下。也就是**枚举所有的 (x, y) 组合**，判断是否满足 $x^2 + y^2 = 2000$ ，如果满足，则是一组解。即当 x 取1时，考虑 y 取1、2、...、44；然后当 x 取2时，又考虑 y 取1、2、...、44；...；最后当 x 取44时，又考虑 y 取1、2、...、44。整个过程如图4.1所示。在实现时要用到2重循环，从**算法思想的角度看，这个过程就是枚举，即枚举所有的 (x, y) 组合**。

[3309] 求 $x^2 + y^2 = 2000$ 的正整数解。

```
1  #include <stdio.h> // #include <bits/stdc++.h>
2  #include <math.h> // using namespace std;
3  int main( )
4  {
5      int x, y;
6      int m = sqrt(2000); // 循环变量x和y的终值
7      for( x=1; x<=m; x++ ) // x从1枚举到m
8      {
9          for( y=1; y<=m; y++ ) // y也从1枚举到m
10         {
11             if( x*x + y*y == 2000 )
12                 printf( "2000=%d*%d+", x, y );
13         }
14     }
15     return 0;
16 }
```

程序的输出如下：

2000=8*8+44*44

2000=20*20+40*40

2000=40*40+20*20

2000=44*44+8*8

思考：从运行结果可以看出， $(8, 44)$ 这一组解和 $(44, 8)$ 这一组解实际上只是交换了 x 和 y 。如果认为这是同一组解，那么方程就只有两组解： $(4, 44)$ 和 $(20, 40)$ ，该怎样修改程序呢？

Answer:

注意到，以上两组解有个特点： $y \geq x$ ，因此我们在枚举时，只要保证 $y \geq x$ ，则 $(40, 20)$ 和 $(44, 8)$ 这两组解都不会枚举出来，程序的输出就满足要求。



代码实现

```
1  #include <stdio.h>
2  #include <math.h>
3  int main( )
4  {
5      int x, y;
6      int m = sqrt(2000); //循环变量x和y的终值
7      for( x=1; x<=m; x++ ) //x从1枚举到m
8      {
9          for( y=x; y<=m; y++ ) //y也从1枚举到m
10         {
11             if( x*x + y*y == 2000 ) //判断相等必须用“==”
12                 printf( "2000=%d*d+%d*d\n", x, x, y, y );
13         }
14     }
15     return 0;
16 }
```

程序的输出如下：

2000=8*8+44*44

2000=20*20+40*40

在采用枚举方法时，要分析题目的要求，避免输出重复的解。

2 [2019] 开关灯

题目描述：

有 N 盏灯，排成一排。给定每盏灯的初始状态(开或关)，你的任务是计算至少要切换多少盏灯的状态(将开着的灯关掉，或将关掉的灯开起来)，才能使得这 N 盏灯开和关交替出现。

输入描述：

输入文件中包含多个测试数据，每个测试数据占一行。首先是一个整数 N ， $1 \leq N \leq 10000$ ，然后是 N 个整数，表示这 N 盏灯的初始状态，1表示开着的，0表示关着的。测试数据一直到文件尾。



2 [2019] 开关灯

输出描述：

对每个测试数据，输出占一行，表示至少需要切换状态的灯的数目。

样例输入：

```
9 1 0 0 1 1 1 0 1 0
3 1 0 1
```

样例输出：

```
3
0
```

注解：

第1个测试数据表示有9盏灯，初始时，这9盏灯的状态为：1 0 0 1 1 1 0 1 0，至少需要切换3盏灯(第1、2、5盏灯)的状态才能使得这9盏灯开和关的状态交替出现：0 1 0 1 0 1 0 1 0。



分析

分析：本题可以采取不同的枚举思路求解。

第一种枚举思路： N 盏灯，每盏灯都有两种状态：1和0， N 盏灯共有 2^N 种状态，从000...0到111...1。可以**枚举这 2^N 种状态**，每种状态都判断一下是否是开和关交替出现，如果是则还要记录从初始状态转换到该状态需要切换的灯的数目。但这种枚举策略势必要花费很多时间，因为 N 最大可以取到10000，而 2^{10000} 的数量级是 10^{3010} 。

第二种枚举思路：要使得 N 盏灯开和关交替出现，实际上只有两种可能：**奇数位置上的灯是开着的**，和**偶数位置上的灯上是开着的**，如图4.2所示。只要分别计算从 N 盏灯的初始状态出发，切换到这两种状态所需要切换灯的数目，取**较小者**即可。



分析

第二种枚举思路：要使得N盏灯开和关交替出现，实际上只有两种可能：**奇数位置上的灯是开着的**，和**偶数位置上的灯上是开着的**，如图4所示。只要分别计算从N盏灯的初始状态出发，切换到这两种状态所需要切换灯的数目，取**较小者**即可。

第1个测试数据对应的初始状态如图所示，将这9盏灯切换到奇数位置上的灯是开着的，需要切换6盏灯；切换到偶数位置上的灯是开着的，需要切换3盏灯；因此至少需要切换3盏灯。

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

(a) 初始状态

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

(b) 奇数位置上的灯开着

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

(c) 偶数位置上的灯开着



分析

第三种枚举思路：注意到上述分析中 $3+6=9$ ，9就是灯的数目 N 。稍加分析就可以得到结论：**如果将 N 盏灯调整成奇数位置上的灯是开着的，需要调整灯的数目为 $numo$ ，则将这 N 盏灯调整为偶数位置上的灯是开着的，需要调整灯的数目 $nume = N - numo$ 。**因此只要判断 $numo$ 是否小于 $N/2$ ，如果是则取 $numo$ ，否则取 $N - numo$ 。这种思路**只要枚举一种状态**即可。

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

(a) 初始状态

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

(b) 奇数位置上的灯开着

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

(c) 偶数位置上的灯开着

切换状态



代码实现

```
1  #include <stdio.h>
2  int main( )
3  {
4      int N, i;    //N表示灯的数目
5      int numo;   //调整成奇数位置上的灯是开着的需要调整灯的数目
6      int lights[10001];
7      while( scanf( "%d", &N )!=EOF )
8      {
9          numo = 0;
10         for( i=1; i<=N; i++ ) scanf( "%d", &lights[i] ); //读入初始状态
11         //将N盏灯的转换调整为奇数位置上为1, 偶数位置上为0
12         for( i=1; i<=N; i++ )
13         {
14             if( i%2==1 && lights[i]==0) //奇数位置为0, 需要调整
15                 numo++;
16             if( i%2==0 && lights[i]==1) //偶数位置为1, 需要调整
17                 numo++;
18         }
19         printf( "%d\n";, numo<=N/2 ? numo : N-numo );//"="不能去掉
20     }
21     return 0;
22 }
```



填空

```
1  #include <stdio.h>
2  int main( )
3  {
4      int N, i;    //N表示灯的数目
5      int numo;   //调整成奇数位置上的灯是开着的需要调整灯的数目
6      int lights[10001];
7      while( scanf( "%d", &N )!=EOF )
8      {
9          numo = 0;
10         for( i=1; i<=N; i++ ) scanf( "%d", &lights[i] ); //读入初始状态
11         //将N盏灯的转换调整为奇数位置上为1, 偶数位置上为0
12         for( i=1; i<=N; i++ )
13         {
14
15
16
17
18         }
19         printf( "%d\n";;, numo<=N/2 ? numo : N-numo );//""不能去掉
20     }
21     return 0;
22 }
```





3 [1311] 歌德巴赫猜想

1742年，德国数学家哥德巴赫(Goldbach)提出了著名的哥德巴赫猜想(Goldbach Conjecture)：任何一个不小于4的偶数可以表示为两个素数之和。这个猜想至今都没有完全被证明是正确的。但是，对于一个大于或等于5的奇数，有的可以表示成两个素数之和，有的则不能。**给定一个大于或等于5的奇数，判断是否能分解成两个素数之和。**

输入输入文件包含多个测试数据，每个测试数据占一行，为一个正整数 m ， m 为奇数，且不小于5，不大于32767。测试数据一直到文件尾。

输出对每个测试数据，如果 m 能分解成两个素数之和，输出yes，否则输出no。

样例输入

21

75

99

113

样例输出

yes

yes

yes

no

分析：简单题，因为是奇数，而最小的素数是2，但在本题中，只要判断m-2是否是素数就可以。

```
1  #include<bits/stdc++.h>
2  using namespace std;
3  int prime(int x){
4      for(int i=2;i<=sqrt(x);i++)
5          if(x%i==0)return 0;
6      return 1;
7  }
8  int main(){
9      int n;
10 while(cin>>n){
11     if(prime(n-2))
12         cout<<"yes"<<endl;
13     else
14         cout<<"no"<<endl;
15 }
```



4[2020] 歌德巴赫猜想2

歌德巴赫猜想：对任何一个不小于4的偶数，总是可以分解成一个素数对之和， $n = p_1 + p_2$ ， p_1 和 p_2 都是素数。

这个猜想至今都还没有被证明是正确的或是错误的。没有人知道这个猜想到底是否正确。然而，对于一个给定的偶数，我们可以去找这样的素数对。本题的任务是编写一个程序实现，**对于一个给定的偶数，输出满足条件的素数对的个数。**

注意，在本题中，对两个素数 p_1 和 p_2 ， (p_1, p_2) 和 (p_2, p_1) 是同一个素数对。



4[2020] 歌德巴赫猜想2

输入描述：

输入文件包含多个测试数据，每个测试数据占一行，为一个整数，并且假定这个整数是偶数，且不小于4，小于 2^{15} 。输入文件的最后一行为0，表示输入结束。

输出描述：

对输入文件中的每个偶数(除最后的0外)，输出满足条件的素数对的个数。

样例输入：

6
10
12
0

样例输出：

1
2
1

注解：

10有两种分解方式：“3+7”和“5+5”。



代码

```
1  #include<bits/stdc++.h>
2  using namespace std;
3  int isPrime(int x){
4      for(int i=2;i*i<=x;i++)
5          if(x%i==0)return 0;
6      return 1;
7  }
8  int main(){//2020 哥德巴赫猜想
9      int n,k;
10     while(scanf("%d",&n)!=EOF){
11         k=0;
12         if(n==0)break;
13         for(int i=2;i<=n/2;i++){
14             if(isPrime(i)&&isPrime(n-i))
15                 k++;
16         }
17         printf("%d\n",k);
18     }
19
20     return 0;
21 }
```

填空

```
1 #include<bits/stdc++.h>
2 using namespace std;
3  int isPrime(int x){
4     
5
6
7 }
8  int main(){//2020 哥德巴赫猜想
9     int n,k;
10  while scanf("%d",&n)!=EOF){
11     k=0;
12     if(n==0)break;
13     
14
15
16
17     printf("%d\n",k);
18 }
19     return 0;
20 }
```

5 [6860 2975]火柴棒等式

[问题描述]

给出 n 根火柴棒，可以拼出多少个形如" $A+B=C$ "的等式？

等式中的 A 、 B 、 C 是用火柴棒拼出的整数（若该数非零，则最高位不能是 0）。用火柴棒拼数字 0~9 的拼法如图 9.7-1 所示。



需要注意以下几点：

- (1) 加号与等号各自需要两根火柴棒。
- (2) 如果 $A \neq B$ ，则 $A+B=C$ 与 $B+A=C$ 视为不同的等式（ A 、 B 、 C 均大于或等于 0）。
- (3) n 根火柴棒必须全部用上（ $n \leq 24$ ）。

[输入样例]14

[输出样例]2

[样例说明]

两个等式分别为： $0+1=1$ 和 $1+0=1$ 。

分析

首先，预处理每个数字（0~9）需要用几根火柴棒，存储在数组 f 中。然后，穷举 a 和 b ，算出它们的和 c ，再判断下列约束条件是否成立： $f(a) + f(b) + f(c) = n - 4$ （ $+$ 和 $=$ 占据了4根火柴棒）。现在的问题是： a 和 b 的范围有多大？可以发现尽量用数字 1 拼成的数比较大，分析可知最多不会超过 1111。程序实现时，分别用三个循环语句预处理好所有两位数、三位数、四位数构成所需要的火柴棒数量。

代码

```
1  #include <iostream>
2  using namespace std;
3  int fun(int x)    //用来计算一个数所需要的火柴棍总数
4  {
5      int num=0;    //用来计数变量
6      int f[10]={6,2,5,5,4,5,6,3,7,6};
7      //f[]记录用一个数组记录0~9数字所需的火柴棍数
8      while(x/10!=0)
9      {            // x除以10不等于0的话,说明该数至少有两位
10         num+=f[x%10];    //加上该位火柴棍数
11         x=x/10;
12     }
13     num+=f[x];        //加上最高位的火柴棍数
14     return num;
15 }
```

代码

```
17 int main()  
18 {  
19     int a,b,c,m,sum=0;  
20     cin>>m;           //火柴棍总个数  
21     for(a=0;a<=1111;a++) //开始枚举  
22     {  
23         for(b=0;b<=1111;b++)  
24         {  
25             c=a+b;  
26             if(fun(a)+fun(b)+fun(c)==m-4) //去掉+和=  
27                 sum++;  
28         }  
29     }  
30     cout<<sum<<endl;  
31     return 0;  
32 }
```

思考?

```
1  #include <iostream>
2  using namespace std;
3  int fun(int x) //用来计算一个数所需要的火柴棍总数
4  {
5      int num=0; //用来计数变量
6      int f[10]={6,2,5,5,4,5,6,3,7,6}; //用一个数组记录0~9数字
7      while(x!=0) // x除以10不等于0的话,说明该数至少有两位
8      {
9          num+=f[x%10]; //加上该位火柴棍数
10         x=x/10;
11     }
12     // num+=f[x]; //加上最高位的火柴棍数
13     return num;
14 }
15 int main()
16 {
17     int a,b,c,m,sum=0;
18     cin>>m; //火柴棍总个数
19     for(a=0;a<=1111;a++) //开始枚举
20     {
21         for(b=0;b<=1111;b++)
22         {
23             c=a+b;
24             if(fun(a)+fun(b)+fun(c)==m-4) //去掉+和=
25             {
26                 cout<<a<<" "<<b<<" "<<c<<endl;
27                 sum++;
28             }
29         }
30     }
31     cout<<sum<<endl;
32     return 0;
33 }
```

```
14
0 2 2
0 3 3
0 5 5
0 17 17
0 71 71
2 0 2
3 0 3
5 0 5
17 0 17
71 0 71
10
```


填空

```
1 #include <iostream>
2 using namespace std;
3 int fun(int x) //用来计算一个数所需要的火柴棍总数
4 {
5     int num=0; //用来计数变量
6     int f[10]={6,2,5,5,4,5,6,3,7,6};
7     while(____1____)
8     {
9         num+=____2____; //加上该位火柴棍数
10        x=____3____;
11    }
12    num+=f[x]; //加上最高位的
13    return ____4____;
14 }

16 int main()
17 {
18     int a,b,c,m,sum=0;
19     cin>>m; //火柴棍总个数
20     for(a=0;a<=1111;a++) //开始枚举
21     {
22         for(b=0;b<=1111;b++)
23         {
24             c=a+b;
25             if(____5____) //去掉+和=
26                 sum++;
27         }
28     }
29     cout<<sum<<endl;
30     return 0;
31 }
```

6 [6861] 奶牛碑文

小伟暑假期间到大草原旅游，在一块石头上发现了一些有趣的碑文。碑文似乎是一个神秘古老的语言，只包括三个大写字母 C、O 和 W。尽管小伟看不懂，但是令他高兴的是，C、O、W 的顺序形式构成了一句他最喜欢的奶牛单词 "COW"。现在，他想知道有多少次 COW 出现在文本中。

如果 COW 内穿插了其他字符，只要 COW 字符出现在正确的顺序，小伟也不介意。甚至，他也不介意出现不同的 COW 共享一些字母。例如，CWOW 出现了 1 次 COW，CCOW 算出现了 2 次 COW，CCOOWW 算出现了 8 次 COW。

[输入格式]

第 1 行为 1 个整数 N。

第 2 行为 N 个字符的字符串，每个字符是一个 C、O 或 W。

[输出格式]

输出 COW 作为输入字符串的字串出现的次数（不一定是连续的）。

提示：答案会很大，建议用 64 位整数（long long）。

6 [6861] 奶牛碑文

[输入格式]

第 1 行为 1 个整数 N 。

第 2 行为 N 个字符的字符串，每个字符是一个 C、O 或 W。

[输出格式]

输出 COW 作为输入字符串的字串出现的次数（不一定是连续的）。

提示：答案会很大，建议用 64 位整数（long long）。

[输入样例]

6

COOWWW

[输出样例]

6

[数据规模]

对于 50% 的数据满足： $N \leq 60$ 。

对于 100% 的数据满足： $N \leq 105$ 。

分析

因为只有 3 个字母，所以可以穷举字符串中的每一个 "0"，假设位置 i ，然后分别计算其左边 "C" 的个数 $l[i]$ 和右边 "W" 的个数 $r[i]$ ，再利用乘法原理进行计数 $l[i]*r[i]$ ，每次把答案累加到 ans 中。

```
3  const int maxn=100001;
4  long long c[maxn],w[maxn];
5  char str[maxn];
6  long long int n,i,k1,k2,ans;
7  int main()
8  {
9      scanf("%lld%s",&n,str);
10     k1=k2=ans=0;
11     for(i=0;i<n;i++)
12     {
13         if(str[i]=='C') k1++;
14         c[i]=k1; //记录到每个位置c的个数
15     }
16     for(i=n-1;i>=0;i--)
17     {
18         if(str[i]=='W') k2++;
19         w[i]=k2; //从后往前记录到每个位置w的个数
20     }
21     for(i=0;i<n;i++)
22     {
23         if(str[i]=='0')
24             ans=ans+c[i]*w[i]; //求解
25     }
26     printf("%lld\n",ans);
27     return 0;
28 }
```

第二种写法

```
1  #include<bits/stdc++.h>
2  using namespace std;
3  const int maxn=100000;
4  char t[maxn];
5  long long dp[4];
6  int main()
7  {
8      int n;
9      scanf("%d%s",&n,t+1);
10     for(int i=1;i<=n;i++)
11     {
12         if(t[i]=='C') dp[0]++;
13         if(t[i]=='O') dp[1]+=dp[0];
14         if(t[i]=='W') dp[2]+=dp[1];
15     }
16     cout<<dp[2]<<endl;
17 }
```

7 [6862]三角形个数

输入一根木棒的长度 n , $1 \leq n \leq 10000$, 将该木棒分成三段, 每段的长度为正整数, 输出由该三段小木棒组成的不一样的三角形个数。

[输入样例]10

[输出样例]2

[样例说明]

两个能组成的三角形边长分别为 2、4、4 和 3、3、4。

[问题分析]

穷举三角形三条边长 (假设为 a 、 b 、 c) 的可能值, 判断能否构成一个三角形, 若能则计数, 最后输出计数器的值。为了保证组成的三角形不重复, 只要在穷举时设定 $1 \leq a \leq b \leq c \leq n-2$ 。优化思想很简单但很重要, "能算不举", 穷举两条边, 根据木棒长度直接计算出第三条边长。

分析

穷举三角形三条边长（假设为 a 、 b 、 c ）的可能值，判断能否构成一个三角形，若能则计数，最后输出计数器的值。为了保证组成的三角形不重复，只要在穷举时设定 $1 \leq a \leq b \leq c \leq n/2$ 。优化思想很简单但很重要，**"能算不举"**，穷举两条边，根据木棒长度直接计算出第三条边长。

```
1  #include<bits/stdc++.h>
2  using namespace std;
3  int main()
4  {
5      int n;
6      long long total=0;
7      cin>>n;
8      for(int a=1;a<=n/2;a++)
9          //因为每条边不能超过n/2
10         for(int b=a;b<=n/2;b++)
11             {
12                 int c=n-a-b;
13                 if((c>=b)&&(a+b>c))total++;
14             }
15     }
16     cout<<total<<endl;
17 }
```

8 [1278] 波浪数

波浪数是在一对数字之间交替转换的数，如1212121，双重波浪数则是指在两种进制下都是波浪数的数，如十进制数191919是一个十进制下的波浪数，它对应的十一进制数121212也是一个波浪数，所以十进制数191919是一个双重波浪数。类似的可以定义三重波浪数，三重波浪数在三种不同的进制中都是波浪数，甚至还有四重波浪数，如十进制 $300=606$ （七进制） $=363$ （九进制） $=454$ （八进制） $=1A1$ （十三进制）...，你的任务就是在指定范围内找出双重、三重、四重波浪数。

输入

单独一行包含五个用空格隔开的十进制整数，前两个数表示进制的范围（2..32），第三与第四个数表示指定的范围（1..10000000），第五个数为2,3,4中的一个，表示要找的波浪数的重数。

例4.5 绑定正多边形(Bounding Box)

题目来源：University of Waterloo Local Contest 2001.09.22

题号：ZOJ1892, POJ2504

题目描述：

考古学家发现文物位于**正多边形**的顶点。沙漠的移动沙丘使得挖掘工作十分艰难，所以一旦发现了**正多边形的三个顶点**，就必须用保护性的建筑物来**覆盖整个正多边形**。

输入描述：

输入文件包含多个测试数据，每个测试数据描述了一个正多边形。描述信息起始于一个整数 n ， $n \leq 50$ ，即顶点的个数，接下来是三对实数，**给出了这个多边形的三个顶点的x坐标和y坐标**。每对实数用空格隔开，每对实数占一行。当 n 等于0时，表示输入结束，这个测试数据不需处理。



第二种输入方式!!!!

输出描述：

要求输出能够覆盖多边形所有顶点的、面积最小的矩形的面积，这个矩形是平行x轴和y轴的。

样例输入：

```
4
10.00000 0.00000
0.00000 -10.00000
-10.00000 0.00000
6
22.23086 0.42320
-4.87328 11.92822
1.76914 27.57680
23
156.71567 -13.63236
139.03195 -22.04236
137.96925 -11.70517
0
```

样例输出：

```
Polygon 1: 400.000
Polygon 2:
1056.172
Polygon 3: 397.673
```



今天的课程结束啦.....



下课了...
同学们再见!

