第五讲

暴力求解法

数学科学学院 汪小平

一、简单枚举

除法

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恰如为数字0~9的一个排列, 2<=n<=79.

- **Samples**

Input	Output
62	79546/01283 = 62 94736/01528 = 62

题目分析

- 需要列举所有10位数的排列吗?
- 只需要枚举分母就行 -- 降低复杂度.
- 在程序中如果记录哪些数字已经被用?

参考程序

```
#include <stdio.h>
#include <string.h>
int main()
{
    int b, c, f, g, h, i, j, k, t, n, a[10];
    while(scanf("%d", &n) == 1)
    {
        memset(a, 0, sizeof(a));
        for(f = 0; f < 10; f++)
        {
            a[f] = 1;
            for(g = 0; g < 10; g++)
            {
                if(a[g]) continue;
                a[g] = 1;
                for(h = 0; h < 10; h++)
                {
                    if(a[h]) continue;
                    a[h] = 1;
                    for(i = 0; i < 10; i++)
                    {
                        if(a[i]) continue;
                        a[i] = 1;
                        for(j = 0; j < 10; j++)
                        {
                            if(a[j]) continue;
                            a[j] = 1;
                            k = f*10000+g*1000+h*100+i*10+j;
                            c = t = k * n;
```

```
                        if(c<100000)
                        {
                            while(c)//被除数最高位不可能为0
                            {
                                b = c%10;
                                if(a[b]) break;
                                a[b] = 1;
                                c /= 10;
                            }
                            for(b = 0; b < 10; b++)
                                if(a[b] == 0)//检查是否是排列
                                    c = 1;
                            if(c == 0)
                                printf("%05d/%05d = %d\n", t, k, n);
                        }
                        memset(a, 0, sizeof(a));
                        a[f] = a[g] = a[h] = a[i] = 1;
                    }
                    a[i] = 0;
                }
                a[h] = 0;
            }
            a[g] = 0;
        }
        a[f] = 0;
    }
}
    return 0;
}
```

最大乘积

输入n个整数元素组成的序列S, 你需要找出一个乘积最大的连续子序列. 如果这个最大的乘积不是正数, 应输出-1. 1<=n<=18, -10 <= S_i <= 10.

- **Samples**

Input	Output
3 2 4 -3 5 2 5 -1 2 -1	8 20

题目分析

- 只须枚举起点和终点即可.
- 注意数的范围, 乘积不超过10^{18}, 可用long long存储.

参考程序

```
#include <stdio.h>
#define N 18
int n, data[N];
long long mul[N];
int main()
{
    int i, j;
    long long ma;
    while(scanf("%d", &n) == 1)
    {
        ma = -1;
        for(i = 0; i < n; i++)
        {
            scanf("%d", &data[i]);
            if(data[i] > ma)   ma = data[i];
        }
        for(i = 0; i < n; i++) //data[i]为起点
        {
            mul[i] = data[i];
            for(j = i+1; j < n; j++) //data[j]为终点
            {
                mul[j] = mul[j-1]*data[j];
                if(mul[j] > ma) ma = mul[j];
            }
        }
        printf("%lld\n", ma);
    }
    return 0;
}
```

分数拆分

输入正整数k, 找到所有的正整数x>=y, 使得1/k = 1/x + 1/y, 其中2 <= k <= 10000.

- **Samples**

Input	Output
2 12	2 1/2 = 1/6 + 1/3 1/2 = 1/4 + 1/4 8 1/12 = 1/156 + 1/13 1/12 = 1/84 + 1/14 1/12 = 1/60 + 1/15 1/12 = 1/48 + 1/16 1/12 = 1/36 + 1/18 1/12 = 1/30 + 1/20 1/12 = 1/28 + 1/21 1/12 = 1/24 + 1/24

题目分析

$$由\frac{1}{k}-\frac{1}{y}=\frac{1}{x},x\geq y\Rightarrow\frac{1}{k}-\frac{1}{y}\leq\frac{1}{y}\Rightarrow\frac{2}{y}\geq\frac{1}{k}$$

$$\Rightarrow 0\leq y\leq 2k$$

- 只须按 0 <= y <= 2k枚举y, 然后进行判断即可.
- 由于要求先输出满足要求的个数, 所以需要先记录, 后输出.

参考程序

```
#include <stdio.h>
#define N 10000
int n, ans[2*N];
int main()
{
    int i, k, x, y;
    freopen("1.in", "r", stdin);
    freopen("1.out", "w", stdout);
    while(scanf("%d", &k) == 1)
    {
        n = 0;
        for(y = k+1; y <=2*k; y++)
            if((k*y) % (y-k) == 0)
                ans[n++] = y;
        printf("%d\n", n);
        for(i = 0; i < n; i++)
        {
            y = ans[i];
            printf("1/%d = 1/%d + 1/%d\n", k, (k*y)/(y-k), y);
        }
    }
    return 0;
}
```

$$\frac{1}{k}-\frac{1}{y}=\frac{1}{x}\Rightarrow x=\frac{ky}{y-k}$$

确定进制

6 * 9 = 42对于十进制来说是错误的, 但是对于13进制来说是正确的. 即6(13) * 9(13) = 42(13), 而42(13) = 4 * 13 + 2*1 = 54(10).

你的任务是写一段程序读入三个整数p、q和r, 然后确定一个进制B (2 <= B <= 16), 使得p * q = r.

如果B有很多选择, 则输出最小的一个. 例如: p=11,q=11,r=121, 则有11(3) * 11(3) = 121(3), 因为11(3)=4(10)和121(3)=16(10). 对于十进制10, 有11(10) * 11(10) = 121(10), 这种情况下, 应该输出3. 如果没有合适的进制, 则输出0.

- **Standard Input**

1行, 包含三个整数p、q、r, 相邻两个整数之间用单个空格隔开.

- **Standard Output**

一个整数, 即使得p * q = r成立的最小的B. 如果没有合适的B, 则输出0.

- **Samples**

Input	Output
6 9 42 11 11 121	13 3

题目分析

- 枚举进制，判断在该进制下等式是否成立即可.
- 为便于计算，可以考虑把三个数转化为十进制再进行运算判断.
- 三个数应该先判断可能的进制，有助于枚举的准确性.

参考程序

```
#include <iostream>
#include <string>
using namespace std;
int getBase(string a)
{
    int k = a[0], n = a.length();
    for(int i = 1; i < n; i++)
        if(k < a[i])
            k = a[i];
    return k - '0' + 1;
}
int getNumber(string a, int base)
{
    int k = 0, n = a.length();
    for(int i = 0; i < n; i++)
        k = k * base + (a[i] - '0');
    return k;
}
```

```
int main()
{
    string p, q, r;
    cin >> p >> q >> r;
    int i, base = getBase(p);
    int t = getBase(q);
    if(base < t) base = t;
    t = getBase(r);
    if(base < t) base = t;
    for(i = base; i <= 16; i++)
    {
        int a = getNumber(p, i);
        int b = getNumber(q, i);
        int c = getNumber(r, i);
        if(a * b == c)
            break;
    }
    if(i <= 16) cout << i << endl;
    else cout << 0 << endl;
    return 0;
}
```

称硬币

赛利有12枚银币, 其中有11枚真币和1枚假币, 假币看起来和真币没有区别, 但是重量不同. 但赛利不知道假币比真币轻还是重. 于是他向朋友借了一架天平. 朋友希望赛利称三次就能找出假币并且确定假币是轻是重.

例如: 如果赛利用天平称两枚硬币, 发现天平平衡, 说明两枚都是真的. 如果赛利用一枚真币与另一枚银币比较, 发现它比真币轻或重, 说明它是假币. 经过精心安排每次的称量, 赛利保证在称三次后确定假币.

- **Standard Input**

第一行有一个数字n, 表示有n组测试用例.

对于每组测试用例: 输入有三行, 每行表示一次称量的结果. 赛利事先将银币标号为A-L. 每次称量的结果用三个以空格隔开的字符串表示: 天平左边放置的硬币 天平右边放置的硬币 平衡状态. 其中平衡状态用“up”, “down”, 或“even”表示, 分别为右端高、右端低和平衡. 天平左右的硬币数总是相等的.

- **Standard Output**

输出哪一个标号的银币是假币, 并说明它比真币轻还是重(heavy or light).

- **Samples**

Input	Output
1 ABCD EFGH even ABCI EFJK up ABIJ EFGH even	K is the counterfeit coin and it is light.

题目分析

- 题中赛利已经设计了正确的称量方案, 保证从三组称量数据中能得到唯一的答案.
- 答案可以用两个变量表示: x-假币的标号, w-假币是比真币轻还是比真币重.
- x有12种猜测, w有2种猜测. 根据赛利设计的称量方案, (x,w)的24种猜测中, 只有唯一的猜测与三组称量数据都不矛盾. 因此, 如果猜测(x,w)满足下列条件, 这个猜测就是要找的答案:
  - 在称量结果为"even" 的天平两边, 没有出现x;
  - 如果w表示假币比真币轻, 则在称量结果为"up" 的天平右边一定出现x、在称量结果为"down" 的天平左边一定出现x;
  - 如果w表示假币比真币重，则在称量结果为"up" 的天平左边一定出现x、在称量结果为"down" 的天平右边一定出现x。

具体实现时, 要注意两点:

- 选择合适的算法

  对于每一枚硬币x 逐个试探:

  - ➔ x 比真币轻的猜测是否成立? 猜测成立则进行输出.
  - ➔ x 比真币重的猜测是否成立? 猜测成立则进行输出.

- 选择合适的数据结构

  以字符串数组存储称量的结果. 每次称量时, 天平左右最多有6枚硬币. 因此, 字符串的长度需要为7, 最后一位存储字符串的结束符'\0', 便于程序代码中使用字符串操作函数.

```
char left[3][7], right[3][7], result[3][7];
```

参考程序

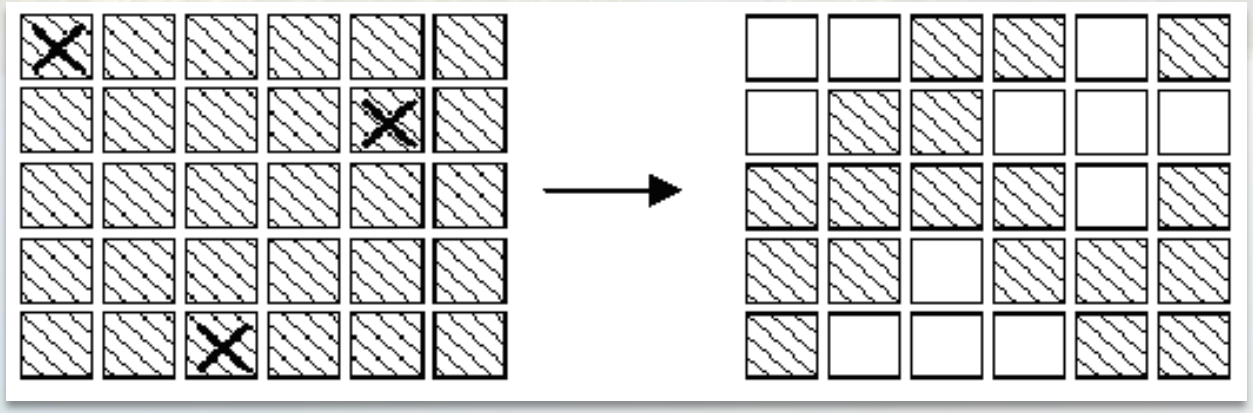
```
#include <stdio.h>
#include <string.h>
char left[3][7], right[3][7], result[3][5];
bool isHeavy(char x);
bool isLight(char x);
int main(void) {
    int i,n;
    char c;
    scanf("%d", &n);
    while ( n-- ) {
        for ( i = 0; i < 3; i++ )
            scanf("%s %s %s", left[i], right[i], result[i]);
        for ( c = 'A'; c <= 'L'; c++ ) {
            if ( isLight(c) )  {
                printf("%c is the counterfeit coin and it is light.\n", c);
                break;
            }
```

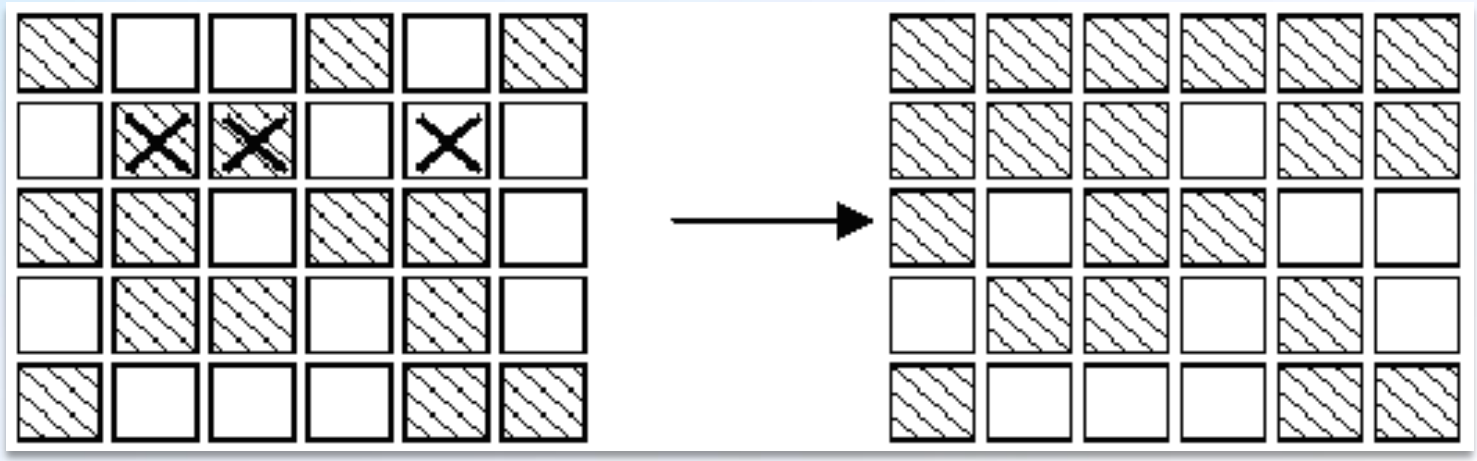
```
            else if ( isHeavy(c) ) {
                printf("%c is the counterfeit coin and it is heavy.\n", c);
                break;
            }
        }
    }
    return 0;
}
bool isLight( char x ) { // 判断硬币x 是否为轻的代码
    int i;
    for ( i = 0; i < 3; i++ ) // 判断是否与三次称量结果矛盾
        switch( result[i][0] ) {
            case 'u': if( strchr(right[i], x) == NULL )   return false;
                      break;
            case 'e': if(strchr(right[i], x) != NULL || strchr(left[i], x) != NULL)  return false;
                      break;
            case 'd': if(strchr(left[i], x) == NULL)  return false;
                      break;
        }
    return true;
}
```

```
bool isHeavy( char x ) //判断硬币x 是否为重的代码
{
   int i;
   for ( i = 0; i < 3; i++ ) // 判断是否与三次称量结果矛盾
      switch( result[i][0] )
      {
         case 'u': if( strchr(left[i], x) == NULL)
                      return false;
                   break;
         case 'e': if(strchr(right[i], x) != NULL || strchr(left[i], x) != NULL)
                      return false;
                   break;
         case 'd': if(strchr(right[i], x) == NULL)
                      return false;
                   break;
      }
   return true;
}
```

熄灯问题

一个由按钮组成的矩阵, 其中每行有6个按钮, 共5行. 每个按钮的位置上有一盏灯. 当按下一个按钮后, 该按钮以及周围位置(上边、下边、左边、右边)的灯都会改变一次. 即, 如果灯原来是点亮的, 就会被熄灭; 如果灯原来是熄灭的, 则会被点亮. 在矩阵角上的按钮改变3盏灯的状态; 在矩阵边上的按钮改变4盏灯的状态; 其他的按钮改变5盏灯的状态.





请你写一个程序, 确定需要按下哪些按钮, 恰好使得所有的灯都熄灭.

根据上面的规则, 我们知道:

(1)第2次按下同一个按钮时, 将抵消第1次按下时所产生的结果. 因此, 每个按钮最多只需要按下一次;

(2)各个按钮被按下的顺序对最终的结果没有影响;

(3)对第1行中每盏点亮的灯, 按下第2行对应的按钮, 就可以熄灭第1行的全部灯. 如此重复下去, 可以熄灭第1、2、3、4行的全部灯. 同样, 按下第1、2、3、4、5列的按钮, 可以熄灭前5列的灯.

- **Standard Input**

第一行是一个正整数N, 表示需要解决的案例数. 每个案例由5行组成, 每一行包括6个数字. 这些数字以空格隔开, 可以是0或1. 0表示灯的初始状态是熄灭的, 1表示灯的初始状态是点亮的.

- **Standard Output**

对每个案例, 首先输出一行, 输出字符串“PUZZLE #m”, 其中m是该案例的序号. 接着按照该案例的输入格式输出5行, 其中的1表示需要把对应的按钮按下, 0则表示不需要按对应的按钮. 每个数字以一个空格隔开.

- **Samples**

Input	Output
2 0 1 1 0 1 0 1 0 0 1 1 1 0 0 1 0 0 1 1 0 0 1 0 1 0 1 1 1 0 0 0 0 1 0 1 0 1 0 1 0 1 1 0 0 1 0 1 1 1 0 1 1 0 0 0 1 0 1 0 0	PUZZLE #1 1 0 1 0 0 1 1 1 0 1 0 1 0 0 1 0 1 1 1 0 0 1 0 0 0 1 0 0 0 0 PUZZLE #2 1 0 0 1 1 1 1 1 0 0 0 0 0 0 0 1 0 0 1 1 0 1 0 1 1 0 1 1 0 1

题目分析

- 为了叙述方便, 按下图所示, 为按钮矩阵中的每个位置分别指定一个坐标. 用数组元素puzzle[i][j]表示位置(i, j)上灯的初始状态: 1表示灯是被点亮的; 0表示灯是熄灭的. 用数组元素press[i][j]表示为了让全部的灯都熄灭, 是否要按下位置(i, j)上的按钮: 1表示要按下; 0表示不用按下.

(0 0)	(0 1)	(0 2)	(0 3)	(0 4)	(0 5)	(0 6)	(0 7)
(1 0)	(1 1)	(1 2)	1 3)	(1 4)	1 5)	(1 6)	(1 7)
(2 0)	(2 1)	(2 2)	(2 3)	(2 4)	(2 5)	(2 6)	(2 7)
(3 0)	(3 1)	(3 2)	(3 3)	(3 4)	(3 5)	(3 6)	(3 7)
(4 0)	(4 1)	(4 2)	(4 3)	(4 4)	(4 5)	(4 6)	(4 7)
(5 0)	(5 1)	(5 2)	(5 3)	(5 4)	(5 5)	(5 6)	(5 7)

- 由于第0 行、第0 列和第7 列不属于按钮矩阵的范围, 没有按钮, 可以假设这些位置上的灯总是熄灭的、按钮也不用按下. 其它30 个位置上的按钮是否需要按下是未知的.

- 因此数组press 共有2^30种取值. 如果直接搜索答案, 显然代价太大.
- 根据熄灯规则, 如果矩阵press 是寻找的答案, 那么按照press 的第一行对矩阵中的按钮操作之后, 此时在矩阵的第一行上:

➔ 如果位置(1, j)上的灯是点亮的, 则要按下位置(2, j)上按钮, 即press[2][j]一定取1;

➔ 如果位置(1, j)上的灯是熄灭的, 则不能按位置(2, j)上按钮, 即press[2][j]一定取0.

- 这样依据press 的第一、二行操作矩阵中的按钮, 才能保证第一行的灯全部熄灭. 而对矩阵中第三、四、五行的按钮无论进行什么样的操作, 都不影响第一行各灯的状态. 依此类推, 可以确定press第三、四、五行的值.

- 因此, 一旦确定了press 第一行的值之后, 为熄灭矩阵中第一至四行的灯, 其他行的值也就随之确定了. press 的第一行共有2^6 种取值, 分别对应唯一的一种press 取值, 使得矩阵中前四行的灯都能熄灭. 只要对这2^6 种情况进行判断就可以了: 如果按照其中的某个press对矩阵中的按钮进行操作后, 第五行的所有灯也恰好熄灭, 则找到了答案.
- 解决方案: 对press 第一行的元素press[1][1]~ press [1][6]的各种取值情况进行枚举, 根据熄灯规则计算出press 的其他行的值. 判断这个press 能否使得矩阵第五行的所有灯也恰好熄灭. 如果能够, 也就找到了熄灯的按开关方案.

参考程序

```
#include <stdio.h>
int puzzle[6][8],press[6][8];
bool guess(void)
{
    int c,r;
    for (r=1;r<5;r++ )
        for (c=1;c<7;c++)
            press[r+1][c]=(puzzle[r][c]+press[r][c]
                           +press[r-1][c]+press[r][c-1]
                           +press[r][c+1])%2;
    for(c=1;c<7;c++) //判断最后一行是否熄灭
        if ((press[5][c-1]+press[5][c]
             +press[5][c+1]+press[4][c])%2!=puzzle[5][c])
            return(false);
    return true;
}
```

```
void enumate(void)
{
    int c;
    for (c=1;c<7;c++)
        press[1][c]=0;
    while(guess()==false)
    {
        press[1][1]++;
        c=1;
        while(press[1][c]>1)
        {
            press[1][c]=0;
            c++;
            press[1][c]++;
        }
    }
}
```

```
0 0 0 0 0 0
1 0 0 0 0 0
0 1 0 0 0 0
1 1 0 0 0 0
0 0 1 0 0 0
……
1 1 1 1 1 1
```

```
int main()
{
    int cases,i,r,c;
    scanf("%d", &cases);
    for (r=0;r<6;r++) press[r][0]=press[r][7]=0;
    for (c=1;c<7;c++) press[0][c]=0;
    for (i=0;i<cases;i++)
    {
        for (r=1;r<6;r++)
            for (c=1;c<7;c++)
                scanf("%d", &puzzle[r][c]);
        enumate();
        printf("PUZZLE #%d\n",i+1);
        for (r=1;r<6;r++)
        {
            for (c=1;c<7;c++)
                printf("%d ",press[r][c]);
            printf("\n");
        }
    }
    return 0;
}
```