NOIP2004 解题报告(普及组)

宝坻区牛道口二中：**唐虎**

第一题：unhappy；

【算法分析】

此题主要考察选手的编程基本能力。做题时，可先求出七天内每天学校和妈妈安排的学习时间总和存于数组中，然后设置一个循环对数组中的每个值进行处理，若学习时间超过 8 小时，则说明有不高兴的天，做上标记；若这一天的学习时间超过上一次不高兴的天学习的时间，说明这一天更不高兴，则做上标记。最后若没有不高兴的天，输出“0”，否则输出做上标记的那一天的代码。

【数据结构】

```
var a:array[1..7]of integer;
```

【程序清单】

```
program unhappy;
var a:array[1..7]of integer;
    i,x,y,sum,point:integer;
    yes:boolean;
    f1,f2:text;
begin
 assign(f1,'unhappy.in');reset(f1);                    {输入文件名"unhappy.in"}
 assign(f2,'unhappy.out');rewrite(f2);                  {输出文件名"unhappy.out"}
 for i:=1 to 7 do
  begin
   readln(f1,x,y);
   a[i]:=x+y
 end;                                  {读入数据，将每天要学习的时间存于数组 a 中}
 yes:=true;              {若有不高兴的天，即 a 中没有超过 8 的元素，布尔型变量 yes 置真值}
 sum:=0;
 for i:=1 to 7 do                              {求出最不高兴的天数，存于 point 中}
  if (a[i]>8)then
   begin
    yes:=false;
    if a[i]>sum then
     begin sum:=a[i];point:=i;end;
   end;
 if yes then writeln(f2,0)
        else writeln(f2,point);
 close(f1);
 close(f2)
end.
```

第二题：peanuts；

【算法分析】

乍一看此题，就会发现，穷举、搜索的方法对此题都不适用，因为题目中已明确

给出了摘豆子的规则。其实，我们只需要根据题目给出的规则，编程序模拟摘豆子的过程就可以了。首先，定义一个整型的二维数组 a 存储每个格子中的豆子数目，然后进行摘豆子。

设置一个二层循环求出含有最多豆子的格子的坐标，由数学知识可知，从(x1,y1)到(x2,y2)需要的单位时间为|x1-x2|+|y1-y2|，因此由当前位置（x，y）跳到含有最多豆子的格子（x1，y1）内摘下豆子并回到路边所需的时间为 step=|x-x1|+|y-y1|+1+x1，接下来判断 step 与剩余的时间 p 的大小关系。若 step<=p 则说明有足够的时间去摘豆子，则去摘：具体步骤（剩余时间 p：=p-|x-x1|+|y-y1|-1；当前点的坐标跳入（x1，y1）；摘下的总豆子数 sum：=sum+a[x，y]，a[x，y]：=0；表示此格子内的豆子已被摘没）；若 step>p，说明没有足够多的时间跳过去摘下豆子再回到路边，因此只能回到路边而不能去摘豆子了。

重复此摘法即可得出最后结果。

【数据结构】

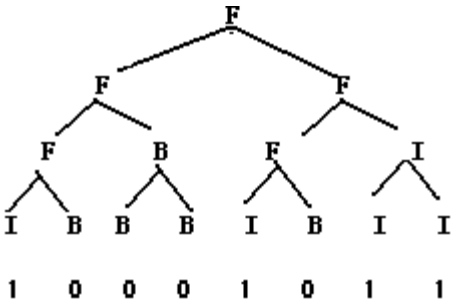
```
var a:array[0..20，0..20]of integer;                    {存每个格子内的豆子数}
```

【程序清单】

```
program peanuts;
var a:array[0..20,0..20]of integer;                     {存每个格子内的豆子数}
    m,n,k,step,i,j,x,y,x1,y1,sum:integer;
    f1,f2:text;
    yes:boolean;                                        {存是否有豆子可摘}
begin
 assign(f1,'peanuts.in');reset(f1);                     {输入文件名"peanuts.in"}
 assign(f2,'peanuts.out');rewrite(f2);                  {输出文件名"peanuts.out"}
 readln(f1,m,n,k);
 for i:=1 to m do
  begin
   for j:=1 to n do read(f1,a[i,j]);readln(f1)
  end;                                                  {读入数据}
 x:=0;y:=0;yes:=true;sum:=0;                            {初始化,x,y 存当前结点坐标}
 repeat
  x1:=0;y1:=0;
  for i:=1 to m do
    for j:=1 to n do
     if a[i,j]>a[x1,y1] then begin x1:=i;y1:=j end;     {求出含有最多豆子数的格子坐标}
  if y=0 then y:=y1;
  step:=abs(x-x1)+abs(y-y1)+x1+1;  {求出从原点到含豆子数最多点内摘豆子并回到路边所需单位时间}
  if (k<step)or(a[x1,y1]=0)then yes:=false;             {判断豆子是否可摘}
  if yes then
   begin
    sum:=sum+a[x1,y1];
    k:=k-abs(x-x1)-abs(y-y1)-1;
    x:=x1;y:=y1;a[x,y]:=0;
   end;                                                 {摘豆子}
```

```
 until yes=false;
 writeln(f2,sum);
 close(f1);close(f2)
end.
```

第三题：fbi；



【算法分析】

此题的后序遍历适合用递归算法完成。由题目可知，输入数据给出的字符为所要遍历的fbi树的叶结点代码，又因为字符串的长度为2的整数次幂，故此fbi树为完全二叉树。由于题中明确规定：子符串中的字符都是‘0’，为B串；都是‘1’，为I串；既有‘0’又有‘1’，为F串。即：二叉树的子结点都是B，父结点为B；子结点都是I，父结点为I；既有I又有B，父结点为F。因此，根据树的子结点可以求出父结点。我们要做的是根据子结点后序遍历二叉树。基本算法为：

```
procedure bianli（x，y：integer）； //遍历过程；x，y：为子结点在数组a中的位置。
 begin
  if x=y then 输出叶结点          //结束递归条件，结点为一个字符。
        else begin
             求出父结点；
             遍历左子树；
             遍历右子树；
             输出父结点；
            end；
 end；                          //递归算法，详情请参阅程序清单。
```

【数据结构】

```
var a:array[1..10000]of '0'..'1';
```

【程序清单】

```
program fbi;
var a:array[1..10000]of '0'..'1';
    n,i,l:integer;
    f1,f2:text;
    treeb,treei:boolean;
    tree:char;
procedure bianli(x,y:integer);              {递归过程，x,y为所要遍历的树的叶结点在数组a中位置}
 begin
  if x=y then case a[x] of
              '0':write(f2,'B');
```

```
            '1':write(f2,'I');
           end                                          {输出叶结点}
        else begin
            bianli(x,x+(y-x+1) div 2-1);                {遍历左子树}
            bianli(x+(y-x+1)div 2,y);                   {遍历右子树}
            treei:=false;treeb:=false;
            for i:=x to y do if a[i]='0' then treeb:=true else treei:=true; {求出树的父结点}
            if treei and treeb then tree:='F'
             else begin
                  if treei then tree:='I';
                  if treeb then tree:='B'               {输出父结点}
                 end;
            write(f2,tree);
           end
 end;
begin
 assign(f1,'fbi.in'); reset(f1);                        {输入文件名"fbi,in"}
 assign(f2,'fbi.out');rewrite(f2);                      {输出文件名"fbi.out"}
 readln(f1,n);l:=1;
 for i:=1 to n do l:=l*2;
 for i:=1 to l do read(f1,a[i]);                        {读入数据,l 为字符串长度}
 bianli(1,l);writeln(f2);
 close(f1); close(f2)
end.
```

第四题：martian;

【算法分析】

此题难度较大。

此题的常规算法为根据题中给出的组合算出其表示的值，加上要加的数 m，最后“翻译”成组合。但这种方法会受到时间限制，当数据规模大一些时，会超时或无法得出结果。使用好一点的算法是利用数学中的组合知识计算出所给的组合所表示的数存于高精度数中，再进行处理，但当数据规模达到 100 甚至 10000 时，从存储空间上或从时间上来说都是不可能实现的。

这就启发我们寻求更为简洁有效的算法。

我们以 n 为例，从小到大的组合为：

第 n 位	第 n-1 位	第 n-2 位	…	第 x 位	…	第 1 位	序列在组合中的位置数
1	2	3	…	x	…	n	1
1	2	3	…	…	…	n-1	2
1	2	3	…	…	…	n-2	3
		.					
		.					
		.					
n	n-1	n- 2	…	…	…	1	n!

对于给定的整数 n，共有 n!序列,对应于 1 打头的序列有（n-1）！个。

对序列中任一位 x(x<n)，可以看出，第 x+1 位不变动，组合的方式有 x！种，用 a[1、、 n]表示每一位上的数字，由此我们得到递推公式：

组合中前 i 位（a[I]、、 a[1]）所表示的数=（a[I]在 a[1、、 I]中的位置-1）×(I-1)!+（a[I-1]在 a[1、、 I-1]中的位置-1）×(I-2)!+、、 + 1；（后面源程序代码中“前 p 位”指最低位开始的 p 位；“x 在 s 中的位置”指的是数 s 在集合中或某一部分数中排序后的位置，如 3 在[2，3，4，1，0]中的位置为 4。），如：

1 5 3 4 2 前 3 位表示的数：

（2-1）×2！+（2-1）×1！+（1-1）×0！+1=4

首先，我们求出 m 使组合数需要变动的最小长度 p 求出，然后求出组合前 p 位表示的数 s，m：=m+s；若数 m 仍能用长度为 p 的组合数表示出，即原组合数+m 后所得的新组合数不需要进位，就将 a[1]、、 a[p]之间的数（包括 a[1]，a[p]）构成一个集合，将集合中的 p 个数重新填入 a[1]、、 a[p]，使前 p 位所表示的数等于 m；若 m>p！数 m 不能用长度为 p 的组合数表示出即原组合数+m 后所得的新组合数需要进位，则求出需要进位的数在数组 a 中的位置 p1，将 a[1]、、 a[p1]之间的数（包括 a[1]，a[p1]）重新构成一个集合。a[p1]：=集合中比它在集合中的位置大 1 的数，a[p1]出集合，在 p1-1、、 p+1 之间依次填入集合中的最小数，填过的数出集合 m：=m-p！将集合中剩下的 p 个数重新填入 a[1]、、 a[p]，使第 p 位到第 1 位所表示的数等于 m。新组合数已构成，详情参见程序清单。

求组合第 p 位到第 1 位表示的数详细算法：

```
function jisuan(p:integer):longint;
 begin
   for i:=p downto 1 do
   begin
     x:=只变动组合前（p-1）位能表示的最大数；
     s:=s+((a[i]在集合中位置)-1)*x;
     a[I]出集合；
   end;
  jisuan:=s+1；
 end；
```

将集合中的数填入 a[p]、、 a[1]中，使前 p 位表示的数为 m；详细算法：

```
procedure fill(p:integer;m:longint);
 begin
  for i:=p downto 2 do
   begin
    s：=只变动前（p-1）位能表示的最大数；
    if m mod s=0 then begin a[i]:=集合中位置为(m div s)的数;m:=s; end
                 else  begin a[i]:=集合中位置为(m div s+1)的数;m:=m mod s end;
    a[I]出集合；
   end;
  a[1]:=集合中最后剩下的数;
 end;
```

【数据结构】

```
var a:array[1..10000]of 0..10000;//存组合中的数；
    b:array[1..10000]of 0..1;  //表示集合；
```

【程序清单】

```
program martian;
var m,n,p1,p:longint;
```

```
    i,x,s,max:longint;
    yes:boolean;
    a:array[1..10000]of 0..10000;
    b:array[1..10000]of 0..1;
    f1,f2:text;
function jiecheng(p:integer):longint;
 var i,s:longint;
 begin
  s:=1;for i:=1 to p do s:=s*i;jiecheng:=s
 end;                                              {计算阶乘函数}
function find1(p:integer):integer;
 var i,s:integer;
 begin
  s:=0;
  for i:=1 to p do
   if b[i]=1 then inc(s);
  find1:=s;
 end;                                              {求出数 p 在集合中位置}
function find2(p:integer):integer;
 var i,s:integer;
 begin
  s:=0;i:=0;
  while i<p do
   begin
    inc(s);if b[s]=1 then inc(i);
   end;
  find2:=s;                                        {求出集合中位置为 p 的数}
 end;
function jisuan(p:integer):longint;
 var i,s,x:longint;
 begin
  x:=max;s:=0;
  for i:=p downto 1 do
   begin
    x:=x div i;s:=s+(find1(a[i])-1)*x;b[a[i]]:=0;
   end;
  jisuan:=s+1
 end;                                             {计算组合数中由前 p 位所表示的数}
procedure fill(p:integer;m:longint);
 var i,s:longint;
 begin
  s:=max;
  for i:=p downto 2 do
   begin
```

```
    s:=s div i;
    if m mod s=0 then begin a[i]:=find2(m div s);m:=s; end
                 else  begin a[i]:=find2(m div s+1);m:=m mod s end;
    b[a[i]]:=0;
   end;
  a[1]:=find2(1);                          {把集合中的数按 m 大小填入 a[1--p]中}
 end;
begin
 assign(f1,'martian.in');reset(f1);                    {输入文件"martian.in"}
 assign(f2,'martian.out');rewrite(f2);                 {输出文件"martian.out"}
 readln(f1,n);readln(f1,m);
 for i:=n downto 1 do read(f1,a[i]);                       {读入数据}
 p:=0;for i:=1 to n do b[i]:=0;                           {初始化}
 repeat inc(p) until jiecheng(p)>m;                  {计算 m 应处理的位数 p}
 max:=jiecheng(p);                                  {max 为常量，存放 p!}
 for i:=1 to p do b[a[i]]:=1;                             {置集合}
 m:=m+jisuan(p);                          {计算前 p 位所表示的数与 m 相加后所得值位置}
 for i:=1 to p do b[a[i]]:=1;                             {重置集合}
 if m<=max then fill(p,m)
          else begin
               m:=m-max;                                  {进位后 m 余的数}
               p1:=p;yes:=false;
               repeat
                inc(p1);b[a[p1]]:=1;
                if find1(a[p1])<>p1 then yes:=true;
               until yes;                          {求出要进位的数所在位置 p1}
               s:=find2(find1(a[p1])+1);b[s]:=0;a[p1]:=s;                {进位}
               for i:=p1-1 downto p+1 do
                begin
                 s:=find2(1);a[i]:=s;b[s]:=0;
                end;                        {组合数中第 p1-1 位到第 p+1 位按从小到大排列}
               fill(p,m);
              end;
 for i:=n downto 2 do write(f2,a[i],' ');
 writeln(f2,a[1]);
 close(f1); close(f2);
end.
```