

## 第一题 均分纸牌

### 【问题分析】

我们要使移动次数最少，就是要把浪费降至零。通过对具体情况的分析，可以看出在某相邻的两堆之间移动两次或两次以上，是一种浪费，因为我们可以把它们合并为一次或零次。因此，我们将相邻两堆间的移动次数限定在一次或零次。由于本题规模很小，完全可以进行模拟（一步一步地数）。所以现在的问题是如何确定移动顺序。

容易算出目标状态每堆的牌数  $a$ ，而达到目标状态的过程就是把牌数多于  $a$  的堆中的牌移到牌数少于  $a$  的堆。若把  $N$  堆牌分成左右两段各相邻的若干堆，则会出现三种情况：

(1) 左段平均数小于  $a$ ，则必然右段平均数大于  $a$ ，我们必须将右段多于纸牌通过两段交界处用一次移动到左段。

(2) 左段平均数大于  $a$ ，我们必须将左段多于纸牌通过两段交界处用一次移动到右段。

(3) 两段平均数均为  $a$ ，在两段交界处不需要任何移动。

若出现情况 (3)，则可以把原问题在两段交界处划分为两个子问题进行递归处理。否则，我们希望通过在两段交界处进行一次移动而达到情况 (3)。但有可能由于两段交界处纸牌过少而不能完成这一点。所以我们需要在分段上下功夫。

我们可以这样做：对于  $N$  堆纸牌，从左起找到第一堆  $P$  使他以及他左边所有堆的平均数大于等于  $a$ （即最左边  $(P-1)$  堆的平均数小于  $a$ ）。若等于，则递归处理两个子问题（1 到  $P$ ， $P+1$  到  $N$ ），但要注意一点，若  $P$  为最右边的一堆时，应将其多于  $a$  的部分向左移，然后递归（1 到  $N-1$ ）。若大于，则将多余的纸牌从  $P$  堆移动到  $(P+1)$  堆（可以证明，这种移法一定可行），再递归（1 到  $P$ ， $P+1$  到  $N$ ）。最后输出总移动次数即可。

### 【程序清单】

```
{ $A+,B-,D+,E+,F-,G-,I-,L+,N-,O-,P-,Q-,R-,S-,T-,V+,X+ }
```

```
{ $M 16384,0,655360 }
```

```
Program NOIPG1;
```

```
Var
```

```
  Infile:Text; {输入文件}
```

```
  InName:String; {输入文件名}
```

N:LongInt;

Data:Array[1..100] Of LongInt;{每堆纸牌数}

Sum,Av,i,Count:LongInt;{Sum 为纸牌总数，Av 为平均每堆纸牌数，Count 为移动次数}

Procedure Work(A,B:LongInt);{递归处理第 A 堆到第 B 堆的子问题}

Var

Small,i,P:LongInt;{Small 为当前左边若干堆纸牌总数}

Begin

    If  $A \geq B$  Then Exit;

    Small:=0;

    For i:=A To B Do

        Begin{寻找分段点 P}

            Inc(Small,Data[i]);

            If  $Small \geq (i-A+1) * Av$  Then

                Begin

                    P:=i;

                    Break;

                End;

        End;

        If  $(Small = (P-A+1) * Av)$  And  $(P < B)$  Then

            Begin{情况 (3) }

                Work(A,P);

                Work(P+1,B);

End

Else If  $P=B$  Then

Begin{将第  $P$  堆的纸牌向左移}

    Inc(Data[ $P-1$ ],Data[ $B$ ]-Av);

    Data[ $B$ ]:=Av;

    Inc(Count);

    Work(A,B-1);

End

Else Begin{情况 (1) 或 (2) }

    Inc(Data[ $P+1$ ],Small-( $P-A+1$ )\*Av);

    Dec(Data[ $P$ ],Small-( $P-A+1$ )\*Av);

    Inc(Count);

    Work(A,P);

    Work(P+1,B);

End;

End;

Begin

    ReadLn(InName);

    Assign(InFile,InName);

    Reset(InFile);

    ReadLn(InFile,N);

    For  $i:=1$  To  $N$  Do

```
    Read(InFile,Data[i]);

Close(InFile);

Sum:=0;

For i:=1 To N Do Inc(Sum,Data[i]);

Av:=Sum Div N;

Count:=0;

Work(1,N);

WriteLn(Count);

End.
```

## 第二题 字符变换

### 【问题分析】

对于这道题，我没有找到比宽度搜索更好的算法。虽然我的程序可以顺利的通过所有的测试数据，但依照题目叙述的规模较大的数据仍无法通过。例如：

aaaaaaaaaaaaaaaaaaaaa bbbbbbbbbbbbbbbbbbbbbb

a b

b a

aa bb

bb aa

aaa bbb

bbb aaa

我的算法是标准的宽度搜索，所以在这里只说一些应该注意的地方。

(1) 题目中没有明确指出变换过程中字符串长度的上限。(但我为了节约空间,仍认为是 20)

(2) 为了提高程序的性能,应进行双向宽度搜索(其实只需在单向搜索的基础上略加改动即可)。

(3) 为存储更多的状态,应使用指针。

(4) 在竞赛中,超时即为零分。所以应进行卡时,并根据经验输出某一可能正确的解。(尤其是在结果的情况数很少的情况下)

### 【程序清单】

```
{ $A+,B-,D+,E+,F-,G-,I-,L+,N-,O-,P-,Q-,R-,S-,T-,V+,X+ }
```

```
{ $M 16384,0,655360 }
```

```
Program NOIPG2;
```

```
Type
```

```
StrType=String[20];
```

```
SType=Array[1..2800] Of Record Name:StrType; Step:Integer; End;
```

```
Var
```

```
ct:longint absolute $40:$6C;
```

```
ot:Longint; { ct 和 ot 为卡时所用变量,相关语句均为卡时所用 }
```

```
Infile:Text;
```

```
InName:String;
```

```
A,B:StrType; { A 为初始状态, B 为目标状态 }
```

```
Data:Array[1..6,1..2] Of StrType; { 变换方案 }
```

```
DataC:Integer; { 变换方案数 }
```

```
S,S2:^SType; { 分别为正向和逆向搜索时记录状态的变量 }
```

```
Str1:String; { 临时变量 }
```

i,j,P,Head,Rear,Head2,Rear2:Integer;

IsSame:Boolean;

Begin

ot:=ct;

New(S); New(S2);

{读入数据}

ReadLn(InName);

Assign(InFile,InName);

Reset(InFile);

ReadLn(InFile,Str1);

A:=Copy(Str1,1,Pos(' ',Str1)-1);

B:=Copy(Str1,Pos(' ',Str1)+1,255);

DataC:=0;

While Not Eof(InFile) Do

Begin

Inc(DataC);

ReadLn(InFile,Str1);

Data[DataC,1]:=Copy(Str1,1,Pos(' ',Str1)-1);

Data[DataC,2]:=Copy(Str1,Pos(' ',Str1)+1,255);

End;

Close(InFile);

{初始化}

Head:=1; Rear:=2;

S^[1].Name:=A;

S^[1].Step:=0;

Head2:=1; Rear2:=2;

S2^[1].Name:=B;

S2^[1].Step:=0;

{开始搜索}

Repeat

    if ct-ot>182 then{卡时为 10 秒}

    begin

        writeln(10);

        halt;

    end;

{正向搜索}

If S^[Head].Step>=10 Then

    Break;{“NO ANSWER”的情况}

For i:=1 To DataC Do

    Begin{按每种方案进行变换}

        P:=0;

        While Pos(Data[i,1],Copy(S^[Head].Name,P+1,255))<>0 Do

            Begin

P:=Pos(Data[i,1],Copy(S^[Head].Name,P+1,255))+P;

S^[Rear].Name:=Copy(S^[Head].Name,1,P-1)+Data[i,2]+Copy(S^[Head].Name,P+Length(Data[i,1]),255);

{判断是否结束}

IsSame:=False;

For j:=1 To Rear2-1 Do

If S^[Rear].Name=S2^[j].Name Then

Begin

If S^[Head].Step+1+S2^[j].Step<=10 Then

WriteLn(S^[Head].Step+1+S2^[j].Step)

Else WriteLn('NO ANSWER!');

Halt;

End;

{判重}

For j:=1 To Rear-1 Do

If S^[Rear].Name=S^[j].Name Then

Begin

IsSame:=True;

Break;

End;

If Not IsSame Then

Begin



S^[Rear].Step:=S^[Head].Step+1;

Inc(Rear);

End;

End;

End;

Inc(Head);

{逆向搜索，基本与上一段程序相同}

If S2^[Head2].Step>=10 Then

Break;

For i:=1 To DataC Do

Begin

P:=0;

While Pos(Data[i,2],Copy(S2^[Head2].Name,P+1,255))<>0 Do

Begin

P:=Pos(Data[i,2],Copy(S2^[Head2].Name,P+1,255))+P;

S2^[Rear2].Name:=Copy(S2^[Head2].Name,1,P-1)+Data[i,1]+Copy(S2^[Head2].Name,P+Length(Data[i,2]),255);

IsSame:=False;

For j:=1 To Rear-1 Do

If S2^[Rear2].Name=S^[j].Name Then

Begin

If S2^[Head2].Step+1+S^[j].Step<=10 Then

```

        WriteLn(S2^[Head2].Step+1+S^[j].Step)

    Else WriteLn('NO ANSWER!');

    Halt;

End;

For j:=1 To Rear2-1 Do

    If S2^[Rear2].Name=S2^[j].Name Then

        Begin

            IsSame:=True;

            Break;

        End;

    If Not IsSame Then

        Begin

            S2^[Rear2].Step:=S2^[Head2].Step+1;

            Inc(Rear2);

        End;

    End;

    End;

    Inc(Head2);

Until (Head>=Rear) Or (Head2>=Rear2);

WriteLn('NO ANSWER!');

End.

```

### 第三题 自由落体

#### 【问题分析】

这是本次比赛中最简单的一道题。显然，小车接到的球一定是相邻的若干个。因此，只需要算出被接到的最左与最右边的两个球即可。要注意几点：

- (1) 小车只用左面和上面接球。
- (2) 注意误差处理。
- (3) 小车有可能接到零个球。

#### 【程序清单】

```
{ $A+,B-,D+,E+,F-,G-,I-,L+,N-,O-,P-,Q-,R-,S-,T-,V+,X+ }
```

```
{ $M 16384,0,655360 }
```

```
Program NOIPG3;
```

```
Var
```

```
  H,S1,V,L,K:Real;
```

```
  N,Left,Right:LongInt;
```

```
  t1,t2,x1,x2:Real;
```

```
Begin
```

```
  ReadLn(H,S1,V,L,K,N);
```

```
  If H<K Then
```

```
    t1:=0
```

```
  Else t1:=Sqrt(2*(H-K)/10);
```

```
  t2:=Sqrt(2*H/10);
```

```
  x1:=S1-V*t2;{小球落地时小车左端的坐标}
```

$x2:=S1-V*t1+L$ ; {小球落至小车上表面高度时小车右端的坐标}

If  $x1<0$  Then  $x1:=0$ ;

If  $x2>N-1$  Then  $x2:=N-1$ ;

If  $x1-\text{Trunc}(x1)\leq 0.00001$  Then Left:=Trunc( $x1$ ) Else Left:=Trunc( $x1$ )+1;

If  $\text{Trunc}(x2)+1-x2\leq 0.00001$  Then Right:=Trunc( $x2$ )+1 Else Right:=Trunc( $x2$ );

If Right<Left Then WriteLn(0) Else WriteLn(Right-Left+1);

End.

#### 第四题 矩形覆盖

##### 【问题分析】

由于  $k$  的取值只有四种情况，所以我采用对  $k$  分类讨论的办法。

$k=1$  时，很简单，不说了。

$k=2$  时，将所有点按横坐标排序，用一条平行与  $y$  轴的直线将所有点分为两部分，分别用一个矩形覆盖，可得到一个解。枚举所有这样的划分方案进行求解（即对点进行枚举，用每个点及其后一个点之间的直线划分）。再按纵坐标排序，用平行与  $x$  轴的直线划分，同样方法求解。最后得到最优解。

$k=3$  时，类似于  $k=2$  时的情况。只是划分后，一部分用一个矩形覆盖，另一部分用两个矩形覆盖，并求解。然后颠倒过来再求解。

$k=4$  时，简单地用直线划分不一定能求得最优解（如右图，其最优解是零）。我的算法是枚举每个点，将它以及它左下方（包括正左方和正下方）的点用一个矩形覆盖，而其余的点用三个矩形覆盖。这个算法有一个缺陷，即有可能违背“各个矩形必须完全分开”的限制条件，而得到比实际最优解更小的结果。但对于绝大部分数据是没有问题的。另外，对于绝大部分随机生成的数据，甚至直接用直线划分也可求得最优解。

##### 【程序清单】

{ $A+,B-,D+,E-,F-,G-,I-,L+,N-,O-,P-,Q-,R-,S-,T-,V+,X+$ }

{ $M\ 16384,0,655360$ }

Program NOIPG4;

Type

SetType=Array[1..50,1..2] Of LongInt;

Var

Infile:Text;

InName:String;

N,K:LongInt;

Data:SetType;{存储所有的点}

i:LongInt;

Function Solve1(Var CurSet:SetType;A,B:Integer):LongInt;

{用一个矩形覆盖 CurSet 中的第 A 个点到第 B 个点}

Var

i:LongInt;

Up,Down,Left,Right:LongInt;

{分别为最大纵坐标、最小纵坐标、最小横坐标、最大横坐标}

Begin

If B-A+1<=1 Then

Begin

Solve1:=0;

Exit;

End;

Up:=0;

Down:=500;

```

Left:=500;

Right:=0;

For i:=A To B Do

Begin

    If CurSet[i,1]<Left Then Left:=CurSet[i,1];

    If CurSet[i,1]>Right Then Right:=CurSet[i,1];

    If CurSet[i,2]<Down Then Down:=CurSet[i,2];

    If CurSet[i,2]>Up Then Up:=CurSet[i,2];

End;

Solve1:=(Up-Down)*(Right-Left);

End;{Solve1}

Function Solve2(CurSet:SetType;A,B:Integer):LongInt;

{用两个矩形覆盖 CurSet 中的第 A 个点到第 B 个点}

Var

    i,j,X,Y,Best,q:LongInt;

Begin

    If B-A+1<=2 Then

        Begin{只有或不到两个点}

            Solve2:=0;

            Exit;

        End;

    Best:=MaxLongInt;

```

For i:=A To B-1 Do{以横坐标排序（冒泡排序）}

For j:=i+1 To B Do

If CurSet[i,1]>CurSet[j,1] Then

Begin

X:=CurSet[i,1]; Y:=CurSet[i,2];

CurSet[i]:=CurSet[j];

CurSet[j,1]:=X; CurSet[j,2]:=Y;

End;

For i:=A To B-1 Do

Begin

If CurSet[i,1]=CurSet[i+1,1] Then Continue;

{第 i 个点与第(i+1)个点横坐标相同，不能在它们之间划分}

q:=Solve1(CurSet,A,i)+Solve1(CurSet,i+1,B);

If Best>q Then Best:=q;

End;

For i:=A To B-1 Do{按纵坐标排序}

For j:=i+1 To B Do

If CurSet[i,2]>CurSet[j,2] Then

Begin

X:=CurSet[i,1]; Y:=CurSet[i,2];

CurSet[i]:=CurSet[j];

```

    CurSet[j,1]:=X; CurSet[j,2]:=Y;

End;

For i:=A To B-1 Do

Begin

    If CurSet[i,2]=CurSet[i+1,2] Then Continue;

    q:=Solve1(CurSet,A,i)+Solve1(CurSet,i+1,B);

    If Best>q Then Best:=q;

End;

Solve2:=Best;

End;{Solve2}

Function Solve3(CurSet:SetType;A,B:Integer):LongInt;

{用三个矩形覆盖 CurSet 中的第 A 个点到第 B 个点，类似于 Solve2}

Var

    i,j,X,Y,Best,q:LongInt;

Begin

    If B-A+1<=3 Then

    Begin

        Solve3:=0;

        Exit;

    End;

    Best:=MaxLongInt;

    For i:=A To B-1 Do

```



For j:=i+1 To B Do

If CurSet[i,1]>CurSet[j,1] Then

Begin

X:=CurSet[i,1]; Y:=CurSet[i,2];

CurSet[i]:=CurSet[j];

CurSet[j,1]:=X; CurSet[j,2]:=Y;

End;

For i:=A To B-1 Do

Begin

If CurSet[i,1]=CurSet[i+1,1] Then Continue;

{直线一边用一个矩形覆盖，另一边用两个矩形覆盖}

q:=Solve1(CurSet,A,i)+Solve2(CurSet,i+1,B);

If Best>q Then Best:=q;

q:=Solve2(CurSet,A,i)+Solve1(CurSet,i+1,B);

If Best>q Then Best:=q;

End;

For i:=A To B-1 Do

For j:=i+1 To B Do

If CurSet[i,2]>CurSet[j,2] Then

Begin

X:=CurSet[i,1]; Y:=CurSet[i,2];

```

CurSet[i]:=CurSet[j];

CurSet[j,1]:=X; CurSet[j,2]:=Y;

End;

For i:=A To B-1 Do

Begin

    If CurSet[i,2]=CurSet[i+1,2] Then Continue;

    q:=Solve1(CurSet,A,i)+Solve2(CurSet,i+1,B);

    If Best>q Then Best:=q;

    q:=Solve2(CurSet,A,i)+Solve1(CurSet,i+1,B);

    If Best>q Then Best:=q;

End;

Solve3:=Best;

End;{Solve3}

Function Solve4(CurSet:SetType;A,B:Integer):LongInt;

{用四个矩形覆盖 CurSet 中的第 A 个点到第 B 个点}

Var

    i,j,X,Y,Best,q,P1,P2:LongInt;

    Set1,Set2:SetType;

Begin

    If B-A+1<=4 Then

        Begin

            Solve4:=0;

```

Exit;

End;

Best:=MaxLongInt;

For i:=A To B Do{枚举每个点}

Begin

{将第 i 个点及其左下方（包括正左方和正下方）的点置于 Set1 中，其余的点置于 Set2 中}

X:=Data[i,1]; Y:=Data[i,2];

P1:=0; P2:=0;

For j:=A To B Do

If (Data[j,1]<=X) And (Data[j,2]<=Y) Then

Begin

Inc(P1);

Set1[P1]:=Data[j];

End

Else Begin

Inc(P2);

Set2[P2]:=Data[j];

End;

q:=Solve1(Set1,1,P1)+Solve3(Set2,1,P2);

{用一个矩形覆盖 Set1 中的点，用三个矩形覆盖 Set2 中的点}

If Best>q Then Best:=q;

End;

Solve4:=Best;

End;{Solve4}

Begin

ReadLn(InName);

Assign(InFile,InName);

Reset(InFile);

ReadLn(InFile,N,K);

For i:=1 To N Do

ReadLn(InFile,Data[i,1],Data[i,2]);

Close(InFile);

Case k Of {分类讨论}

1:

Begin

WriteLn(Solve1(Data,1,N));

End;

2:

Begin

WriteLn(Solve2(Data,1,N));

End;

3:

Begin

    WriteLn(Solve3(Data,1,N));

End;

4:

Begin

    WriteLn(Solve4(Data,1,N));

End;

End;

End.