Betsy Tour 一类问题的 DP 解法

By Baoan

仔细琢磨了 lrj 的黑书，最近又看到了有位同学发的求助 Formula1 的帖子。
想到当年苦搞 OI 陷入茫然的窘迫，自己狂搞了一下这个题。但是目前还是 TLE test18 了。
但是对于 Betsy Tour 这个题在时间限制上已经绰绰有余了。
此文难免有错，还请大牛们多多指教。

Betsy Tour：（USACO）一个方形的小镇被分成 N*N 个小方格。（N<=9），Betsy 要从左上角的方格到达左下角的方格，并且经过每个方格仅一次。编程求出路线数目。

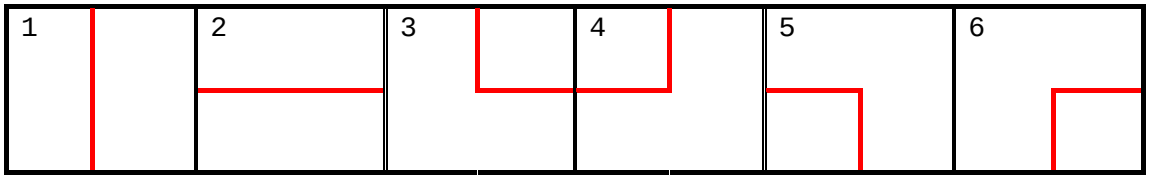
为了方便，先讨论此题的加强版。Ural1519——Formula 1： 给你一个 N×M 的方格阵，上面有些小方格是坏的。现在问你一共有多少条回路，每条回路穿过所有的好的方格并且有且只有 1 次。（这个题目前还是 tle 了，但是对于 Betsy tour 来说是很快了）。

Ps：Betsy 问题只需要把 ural1519 的数据改成

```
..........
.xxxxxxxxx.
.........x.
.........x.
.........x.
.........x.
.........x.
.........x.
.........x.
.........x.
.........x.
.xxxxxxxxx.
..........
```

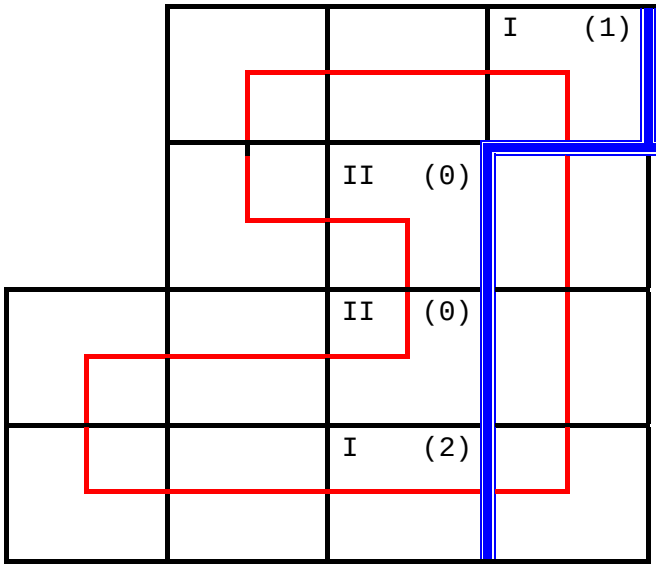
这样用 ural1519 的程序算出的答案就是 Besty 这题的答案。

正文：
在这里不说干搜索的方法，只详细谈谈 dp 的方法是怎么做的。
我们知道，对于每个好的小方格，最多只有 6 种走线方法。分别是：



如果按照一般的状态压缩 dp 写，把状态当作以上这 6 种走线方法。那么时间复杂度必然要 >O(n^2*6^n) n=m=9 的话，这个方法就太慢了。

怎么样才能快一些呢？先看个图（只显示了好的小方格）。



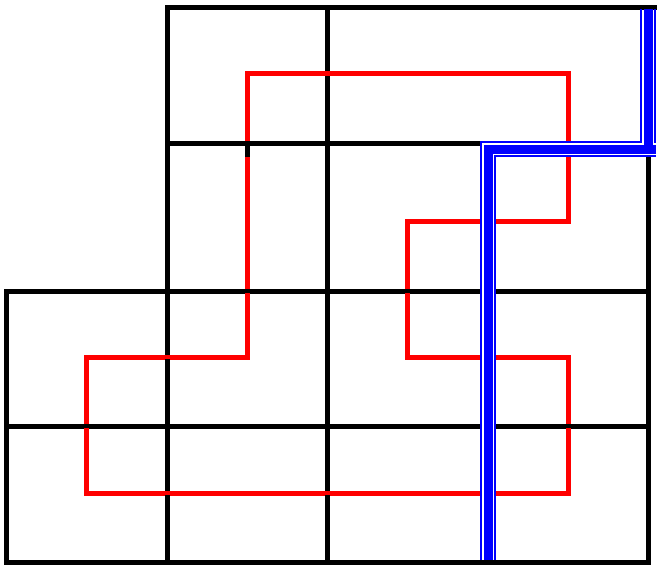
红色的线给出的是一个可行的回路。

现在看紧挨粗蓝线左边的小方格。换个角度，你可以发现无非就只有 2 种方格。

1. 图中 I（1）号和 I（2）号方格。他们的特点是方格中的红线是与蓝线相交的。我们把这种方格叫做**端点**。其中 I（1）号叫做**头**，I（2）号叫做**尾**。(下面具体解释头和尾的区别)
2. 图中 II 号方格。他们的特点是方格中的红线与蓝线不相交。我们把这种方格叫做**茎**。

聪明的你也许想到了：如果用状态压缩 DP，把**端点**和**茎**作为状态。用 **CEOI2002 Bugs** 那个题的**官方解法**。这个题能做到 O(n*n*2^n*C) C 是转移常数。转移原则是：只要能连就连。这个题也许就 ac 了！

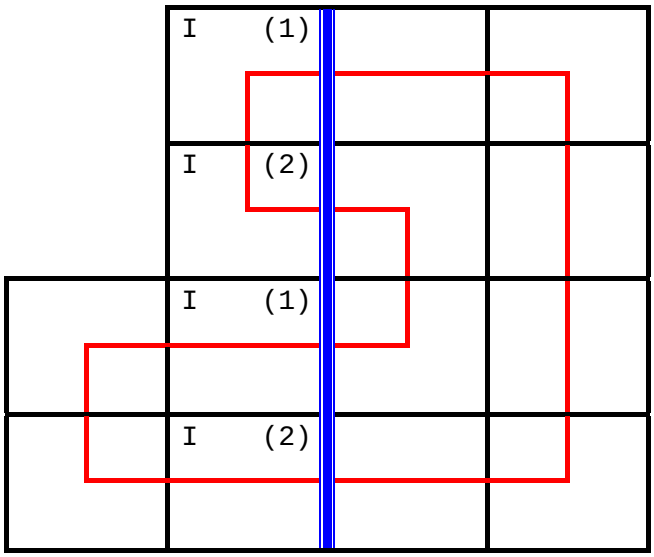
的确，差不多了。但是有种情况对于只用**端点**和**茎**作为状态的状态压缩 DP 是会得到错误答案的。比如就是刚才这个图：



这个图和第一个图中，紧靠蓝线左边的方格是一样的，但是这两个图却代表了两种不同的答案。由于采用能连就连的转移规则，这种非法情况还是无法避免的。（我没有想到怎么样用O(n*n*2^n)的方法解决）

怎么办呢？实际上已经很接近了……

现在再看这个图：



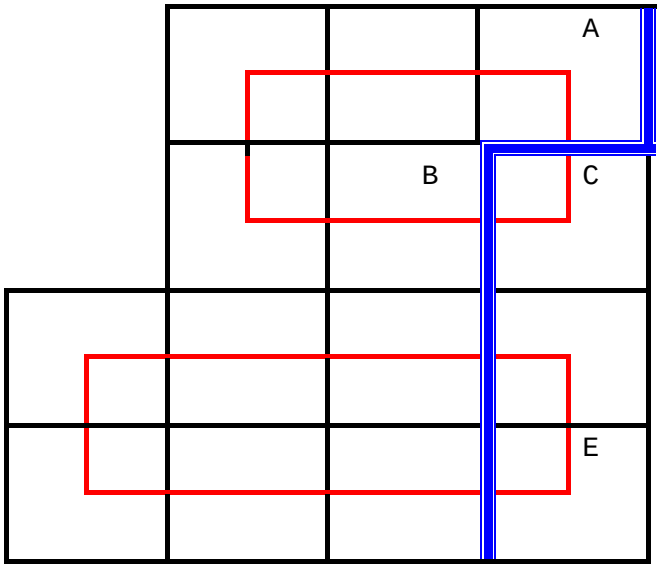
还记得刚才我说得**头**和**尾**的名词吗？
现在我给出**头**和**尾**的概念。
此题如果采用 **CEOI2002 bugs** 的**官方解法**的状态压缩 dp，在此把蓝线称为**状态线,**紧挨状态线左边的方格的**排列**经过压缩就是当前的状态。从上向下看，**头**和**尾**满足以下条件：
1. 头和尾都是 I 类方格。
2. 每一个头都对应着一个尾，并且这个**头**和这个**尾**的红线是**连通**的。
3. 每一个头都对应着一个尾，并且这个**头**的位置总在这个**尾**的上方。

你可能要问了，光标出紧挨状态线左边的**头**和**尾**，怎么样区别是哪个**头**对应哪个**尾**呢？
对于任何一条从上到下的状态线，我只要标出紧靠状态线左边的**头,**就可以确定对于每个**头**所对应的**尾**是谁。（有点像堆栈，是头的话进栈，是尾出栈）这是因为对于所有**对应的头与尾**的连线**不可能会相交的。(**体会这句话，我语文不好，说不清)

用堆栈的办法确定对应的头和尾，复杂度 O（n）。

这个题说到这里，应该差不多了。剩下的就是最重要的状态转移了。具体太麻烦了，我不想写了。提示：看完 CEOI2002 bugs 的**官方**解题报告后，分情况讨论吧。

在这里我只简单提示一下如何避免出现以下这种情况：



如果当前的图中没有两个环，即当前的图是合法的，那么用茎，头，尾的排列作为状态则可以说明当前图的连通性。
为了防止有两个环出现，关键在于如果状态转移。
比如，看上图中的 A 方块和 B 方块：
1．如果 A 和 B 都是头，那么可以让 C 方块把 AB 连起来，同时维护底下 A、B 对应的尾，把 B 对应的尾改成头。
2．如果 A 和 B 都是尾，那么可以让 C 方块把 AB 连起来，同时维护上方 A、B 对应的头，把 A 对应的头改成尾。
3．如果 A 是头，B 是尾。除了 C 在终结处（方针的右下角，这里是 E 方块），其它地方都**不能**把 AB 连起来。
4．如果 A 是尾，B 是头。那么则可以让 C 方块把 AB 连起来。

如果紧靠蓝线有一个头和一个尾，头在上，尾在下，并且他们之间没有头和尾了。那么这对头尾是**连通的**，也就是说不能把他们连到一起。所以用这四点就可以避免出现两个环的可能。

每个单独状态是 O(3^n)，转移是 O（C）（是平均情况，因为有时候还要确定头和尾，理论来讲应该是 O（n）。但是通过优化，实际情况远小于 O（n））参考 bugs 的解法，最后时间复杂度是 O（n*n*3^n*C）约等于 O(n*n*3^n)

Betsy’s Tour 只要按照开头我说得那样，人为把 readdata 改一下就可以 ac 了。我测了一下，的确如同 lrj 所说：“实际程序是很快的。”（黑书 P186）

下面是我的程序:

```
program ural1519;
Const
  fin = 'e:\input\ural1519.in';
  fou = 'e:\input\ural1519.out';
  maxN =  13;
  maxOpt = 531441;

Var
  N , M    :   longint;
  hole     :   array[0..maxN,0..maxN] of boolean;
  d        :   array[0..2,0..maxOpt] of longint;
  ft       :   array[-1..maxN] of longint;

  maxUpOpt  :   longint;
  endY     :   longint;


Procedure readdata;
Var
  i , j  :   longint;
  x     :   char;
Begin
  ft[-1]:=0;
  ft[0]:=1;
  for i:=1 to maxN do ft[i]:=ft[i-1]*3;

  readln( N , M );
  fillchar( hole , sizeof( hole ) , true );
  for i:=1 to N do begin
    for j:=1 to M do begin
      read( x );
      if x='.' then hole[i][j]:=false;
    end;
    readln;
  end;

  for j:=M downto 1 do
    if not hole[N][j] and hole[N][j+1] then begin
      endY:=j;
      break;
    end;

  maxUpOpt := ft[M]-1;
```

```
End;


Procedure work;
Var
  adder , x,ii,kk,i , j ,k1,k2,k3 ,j0,j1,j2 , i1 ,ip  :   longint;
  f         :   array[0..maxN+1] of longint;
  flag1     : longint;

  Tmpx      :  longint;


Begin
  fillchar( d , sizeof(d) , 0);

  // initialzation
  d[0][0]:=1;
  k1:=0;  k2:=1;  k3:=2;
  if hole[1,1] then d[k2][0]:=1;
  if not hole[1,1] and not hole[1,2] then d[k3][ft[0]+ft[1]*2]:=1;

  //process
  for i:=1 to N do
  for j:=1 to M do begin
    k1:=(k1+1) mod 3;
    k2:=(k2+1) mod 3;
    k3:=(k3+1) mod 3;
    fillchar( f, sizeof(f) , 0);
    fillchar( d[k3] , sizeof(d[k3]) , 0);

    for x:=0 to maxUpOpt do begin
      if d[k1][x]<>0 then begin
        adder:=d[k1][x];
        j0:=j; j1:=j+1; j2:=j+2;
        i1:=i;
        if j1=M+1 then begin
          j0:=0; j1:=1; j2:=2;
          i1:=i+1;
        end;
        {option transfer}
        if f[j1]=0 then begin

          if hole[i1,j1] then
            d[k2][x]:=d[k2][x]+adder;
```

```
          if (f[j0]<>0) and not hole[i1,j1] then begin
            tmpx:=x-ft[j0-1]*F[j1-1]+ft[j1-1]*F[j1-1];
            d[k2][tmpx]:=d[k2][tmpx] + adder;
          end;

          if (j2<=M) and not hole[i1,j1] and not hole[i1,j2] then
            if f[j2]=0 then begin
              tmpx:=x+ft[j1-1]+ft[j2-1]*2;
              d[k3][tmpx]:=d[k3][tmpx] + adder;
            end
            else begin
              tmpx:=x+ft[j1-1]*F[j2]-ft[j2-1]*F[j2];
              d[k3][tmpx]:=d[k3][tmpx] + adder;
            end;

        end
        else
        if (f[j1]=1) and not hole[i1,j1] then begin
          d[k2][x]:=d[k2][x]+adder;
          if             f[j0]=1             then             begin
//**
            kk:=0;
            for ii:=j1 to M do begin
              if f[ii]=0 then continue;
              if f[ii]=1 then inc(kk)
              else begin
                dec( kk );
                if kk=0 then begin
                  flag1:=ii;
                  break;
                end;
              end;
            end;{ii}
            tmpx:=x-ft[j0-1]-ft[j1-1]-ft[flag1-1];
            d[k2][tmpx]:=d[k2][tmpx] + adder;
          end;
          if             f[j0]=2             then             begin
//**
            tmpx:=x-ft[j0-1]*2-ft[j1-1];
            d[k2][tmpx]:=d[k2][tmpx] + adder;
          end;
        end
        else
```

```
      if (f[j1]=2) and not hole[i1,j1] then begin
        d[k2][x]:=d[k2][x]+adder;
        if  (f[j0]=1)  and  (i1=N)  and  (j1=endY)  then  begin
//**
          tmpx:=x-ft[j0-1]-ft[j1-1]*2;
          d[k2][tmpx]:=d[k2][tmpx] + adder;
        end;
        if             f[j0]=2             then             begin
//**
          kk:=0;
          for ii:=j0 downto 1 do begin
            if f[ii]=0 then continue;
            if f[ii]=2 then inc(kk)
            else begin
              dec( kk );
              if kk=0 then begin
                flag1:=ii;
                break;
              end;
            end;
          end;
          tmpx:=x-ft[j0-1]*2-ft[j1-1]*2 +ft[flag1-1];
          d[k2][tmpx]:=d[k2][tmpx] + adder;
        end;
      end;
    end;{adder<>0}

    inc( f[1] );
    ip:=1;
    while f[ip]=3 do begin
      f[ip]:=0;
      inc( ip );
      inc( f[ip] );
    end;
  end;{x}
 end;{i , j }

 writeln( d[k1][0] );
End;



begin
 assign( input , fin ); reset( input );
 assign( output, fou ); rewrite( output );
```

```
  readdata;
  work;

  close( input );
  close( output );
end.
```