开灯游戏

开灯问题就是：有 n 乘 n 个房间，一开始灯都是关的，如果将某个房间的灯打开或关闭，那么相邻房间的灯也会状态翻转。那么，要想将灯都打开，要如何操作？也叫 lights-out 或关灯问题等等等等。

请参见http://mathworld.wolfram.com/LightsOutPuzzle.html。

Q1：开灯问题总有解么？

答：总有解的，参见下面的网页。可以看到对于任意图（不光是格子图），都是有解的。也就是说：Ax==b 在二阶域 F2 中总是有线性解的。其中，A 是任意 0-1 方阵，且 A 是以主对角线对称的（A 除了主对角线外是某个图的关联矩阵），并且 A 的主对角线上都是 1，x 和 b 是一维矢量，b 中全都是 1，x 是要求的线性解。

http://www.brand.site.co.il/riddles/201103q.html

http://www.matrix67.com/blog/archives/4263#more-4263

下面全文引用 matrix67 的网页。

“某公司有 n 间办公室。每间办公室都有一盏灯，拉动它的开关即可改变电灯的状态。某些办公室之间存在“业务相关”的关系（这是一个对称的关系）。一个办公室可以和 0 到任意多个办公室相关。愚人节那天，有人在大家上班之前偷偷对办公室的电灯开关做了手脚：拉动任何一个办公室的电灯开关，都会同时改变该办公室以及所有相关办公室的电灯状态。初始时，所有灯都是关着的。证明：等到大家来上班后，总能用有限次的开关，最终把所有办公室的灯都打开。

证明：对 n 施归纳。只有一间办公室时，结论显然成立。下面假设我们已经有办法让任意 n-1 个办公室的灯全部打开。如果把其中某 n-1 个办公室的灯全打开后，发现剩下的那个办公室的灯正好也亮了，问题就解决了。否则，我们就相当于有办法同时改变任意 n-1 个办公室的电灯状态（并且不对剩下的那个办公室造成影响）。

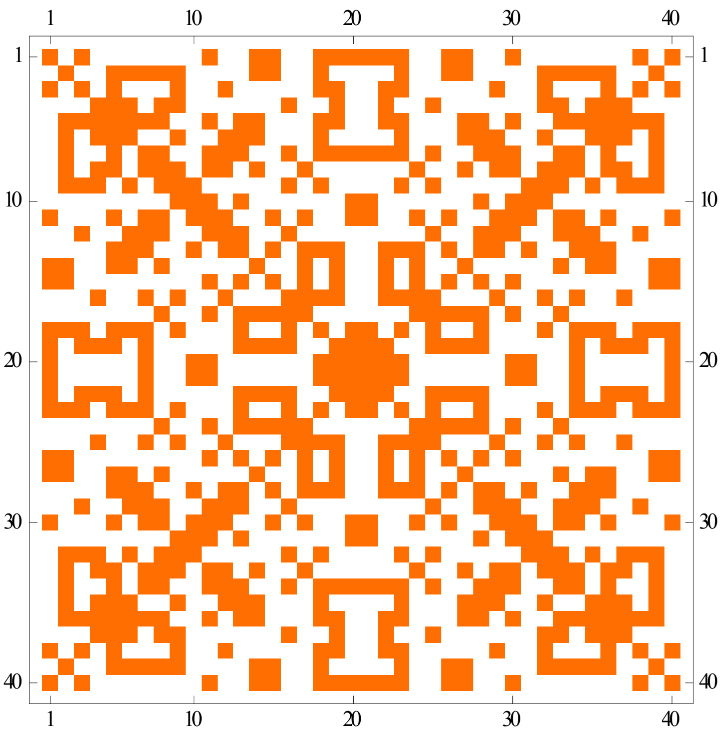
考虑这样的操作：先改变除了办公室 A 以外的所有办公室的电灯状态，再改变除办公室 B 以外的所有办公室的电灯状态。这样下来的结果就是，只有办公室 A 、 B 的电灯状态真的被改变了，其它办公室的电灯状态又都变了回去。也就是说，我们可以同时改变任意两个办公室的电灯状态了（并且不影响其它办公室）。

如果 n 是偶数，两个两个地把它们的灯打开，问题直接就解决了。麻烦的就是，如果 n 是奇数的话，该怎么办呢？要是有一个办公室正好有偶数个相关的办公室就好了，这样的话就可以先拉下它的开关，剩下灯没亮的办公室正好偶数个，问题也就解决 了。下面我们就证明，如果 n 是奇数，那么一定存在一个办公室，它正好有偶数个相关办公室。

注意到，把所有办公室的相关办公室数加起来，结果一定是一个偶数（因为每个相关关系都被算了两次）。但是，我们一共有奇数个办公室，如果它 们各自的相关办公室数目都是奇数，加起来也还是个奇数。因此，至少有一间办公室，它有偶数个相关办公室。这就完成了整个证明过程的最后一环。”

Q2：对于 20 乘 20 一共 400 个房间，要如何操作？

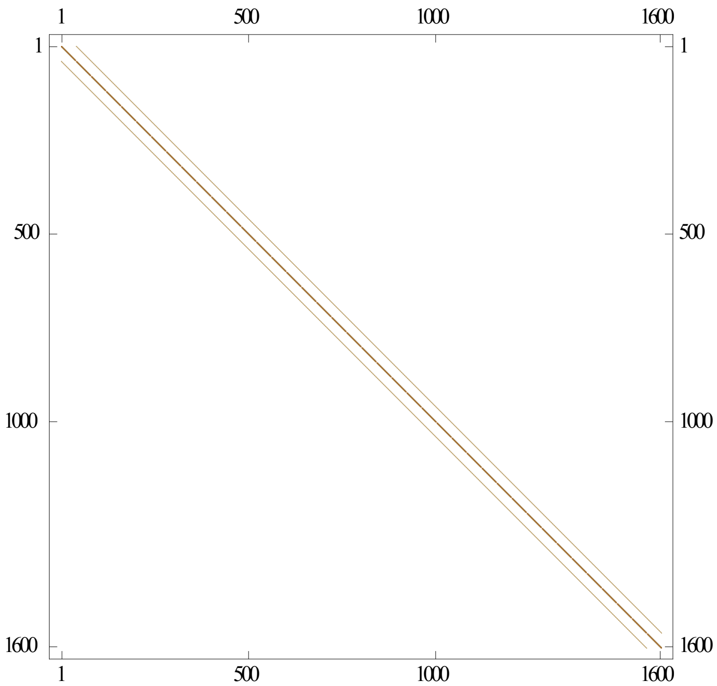
答：操作的方法是这样的，如下图，将有颜色的格子对应的房间的灯拉一下就可以了。



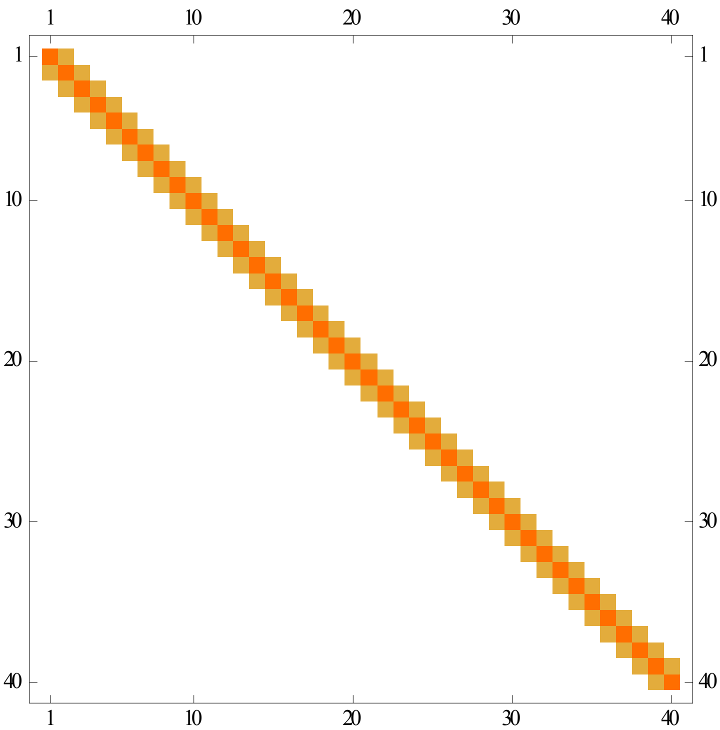
那么，上面的答案是如何求出的呢？有一种比较直接的方法，如下：

```
c[x_,y_,n_]:=Module[{x1,x2,y1,y2},If[x==y,Return[1];];
x1=Mod[x-1,n]+1;x2=(x-x1)/n;y1=Mod[y-1,n]+1;y2=(y-y1)/n;
If[(x1==y1&&Abs[x2-y2]==1)||(x2==y2&&Abs[x1-y1]==1),1,0]];
n=40;s=Table[c[i,j,n],{i,n n},{j,n n}];b=Table[1,{n n}];
(*n n-MatrixRank[s,Modulus->2]*)
a=LinearSolve[s,b,Modulus->2];
MatrixPlot[Partition[a,n]]
```

其中，S 是一个 1600 乘 1600 的矩阵，它是 40 乘 40 的格子图（姑且叫格子图吧，它有 1600 个节点）的关联矩阵，我们最核心的操作是求它的一组线性解 a（a 表示了对 1600 个房间依次进行的操作。这里和下面所有的操作都是在有限域 F2 中进行的）。看，还是要花几分钟的吧（如果 n=20，就比较快了。注释的地方是算 Rank，如果为 0，就说明是唯一解）。要想算的更快，就要了解 S 的摸样，如下（s//MatrixPlot）：



看不清楚吧，可以看下图：



上图中有 40 乘 40 个方块，如果将主对角线上的每一个深色方块换成某个 40 乘 40 的方阵 M，将浅色方块换成

40 乘 40 的单位方阵 E，那么就和看不清的那图的 S 一样了。那个 40 乘 40 的方阵 M 就是上图中带颜色的部分为 1 的方阵。可以看到，S 是由两层 M 嵌套的，于是可能会有更好的方法求线性解的吧。（如果采用处理稀疏矩阵找线性解的方法，自然效率会提升，而象 GNFS 过程中求大个（大约 10^8）的稀疏矩阵的线性解的方法是现成的，有兴趣的可以去看看，但是这样就和 S 的结构无关了，呵呵。）

假定我们用高斯消去法来弄 S，但是不必老实的一行一行的来，可以一块一块的来，一下就弄 40 行，就是把 S 的 41-80 行中的每一块都乘上 M，再和 1-40 行的块相减，就消去了前 40 列。这就相当于求 40 乘 40 的方阵 Mx（其在主对角线上深色的方块是 x，浅色的方块是 1）的判别式 d，然后把 M 假想成数 x，硬带入进去，得到的方阵再次求线性解。（下面的记号中 d(i)表示以 x 为变量的 i 次多项式，d(i,M)表示把 M 带入到 d(i)之后得到的方阵。）

求 d 的过程如下：

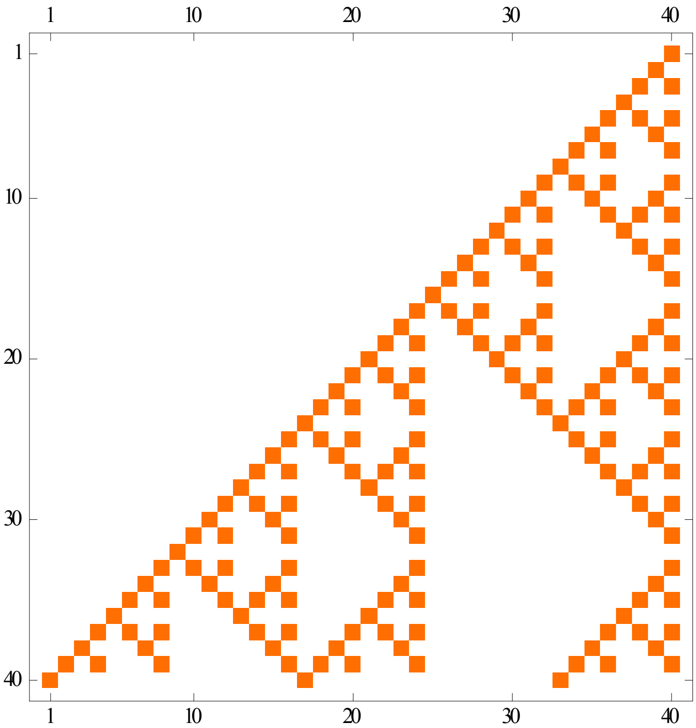
```
d[m_]:=PolynomialMod[Det[Table[If[i==j,x,If[Abs[i-j]==1,1,0]],{i,m},{j,m}]],2];
d[40]
```

我们看到，还是比较快的，但是求 d(1000)就比较慢了（多项式 d 和 M 的特征多项式相关，是有名字的，呵呵）。而我们求 d(大)，可以用迭代的方法，如下：

```
ds={1,x};Do[AppendTo[ds,PolynomialMod[x ds[[-1]]+ds[[-2]],2]],{38}];
```

其实观察一下 d(i)的结构，就可以看到 d(i)的系数是一个躺着的杨辉三角形，如下：

```
dn=ds/.x->2;Map[IntegerDigits[#,2,40]&,dn]//MatrixPlot
```

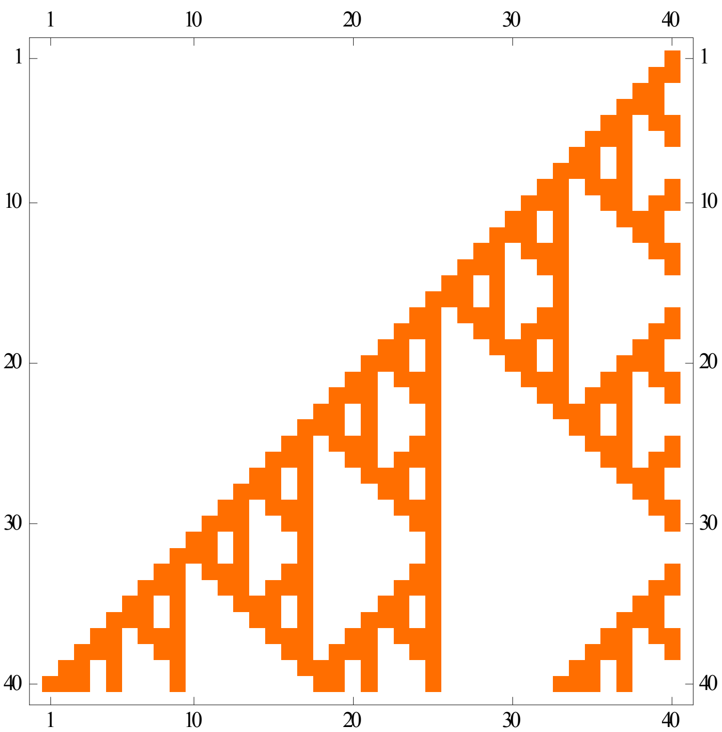


于是，下面的代码可以求出唯一解的边长 n：

```
s={};mm[n_]:=Table[If[Abs[i-j]<=1,1,0],{i,n},{j,n}];
Do[n=i;ms={IdentityMatrix[n]};m0=mm[n] ;
Do[AppendTo[ms,Mod[Last[ms].m0,2]],{n}];
a=Mod[Apply[Plus,Reverse[IntegerDigits[dn[[n+1]],2]]ms],2];
If[Mod[Det[a],2]==1,AppendTo[s,i]];,{i,39}];
```

和 mathe 同学（呵呵）的结果比较一下，发现少了 5,17,41,53,77,113,137,161,173,221,233,245 等项，而这些项恰好是结果图像不对称的，因此判别式可不认为对称后重合的是同一个解。

下面开始真正求解了，我们要求线性相关 Ax==b，其中 A 是 40 乘 40 的方阵，就是上面求判别式的那个 a，即 d(i,M)，x 是要求的矢量，由 40 个 0 或 1 构成，表示解的第一行，即对第一行房间的操作，b 是一个长度 40 的已知矢量，b=sum(d(i,M),i=1->n-1)I，其中 I 是 40 个 1 的矢量，I 左面是一个 40 乘 40 的方阵，我们叫它 sd 吧，在推导成块的高斯消去时，可以得到这个结果，对应 d 的图，sd 如下，也是一种分形的结构。



下面的代码可以输出对第一行房间的操作：

```
n=40;
ds={1,x};sds={1,1+x};
Do[AppendTo[ds,PolynomialMod[x ds[[-1]]+ds[[-2]],2]];
AppendTo[sds,PolynomialMod[Last[sds]+Last[ds],2]];,{n-1}];
dn=ds/.x->2;sdn=sds/.x->2;
ms={IdentityMatrix[n]};m0=mm[n];Do[AppendTo[ms,Mod[Last[ms].m0,2]],{n}];
a=Mod[Apply[Plus,Reverse[IntegerDigits[dn[[n+1]],2]]ms],2];
b=Mod[Mod[Apply[Plus,Reverse[IntegerDigits[sdn[[n]],2]]Most[ms]],2].Table[1,{n}],2];
ans=LinearSolve[a,b,Modulus->2]
```

可以看到结果 ans 对应对前面答案的一条边。求出了第一行，下面的就好求了，用下面的代码就可以了。

```
js={ans,1-Mod[m0.ans,2]};
Do[AppendTo[js,1-Mod[js[[-1]]+Append[Rest[js[[-1]]],0]+Prepend[Most[js[[-
1]]],0]+js[[-2]],2]];,{n-2}]
js//MatrixPlot
```

可以看到：如果有唯一解，那么，两条对角线上的房间必然都是要拉灯的。如果将上面代码多算一行，即将其中的 n-2 改为 n-1，那么的多出一行是空的。

将所有代码放在一起，如下：

```
n=1000;ds={1,x};sds={1,1+x};
Do[AppendTo[ds,PolynomialMod[x ds[[-1]]+ds[[-2]],2]];
AppendTo[sds,PolynomialMod[Last[sds]+Last[ds],2]];,{n-1}];
dn=ds/.x->2;sdn=sds/.x->2;
ta=Reverse[IntegerDigits[dn[[n+1]],2]];tb=Append[Reverse[IntegerDigits[sdn[[n]],2]],0];
a=Table[0,{n},{n}];b=Table[0,{n}];ib=Table[1,{n}];
ms=IdentityMatrix[n];m0=Table[If[Abs[i-j]<=1,1,0],{i,n},{j,n}];
Timing[Do[If[ta[[i]]==1,a=Mod[a+ms,2]];If[tb[[i]]==1,b=Mod[b+ms.ib,2]];ms=Mod[ms.m0,2],
{i,n+1}]];
ans=LinearSolve[a,b,Modulus->2];
js={ans,1-Mod[m0.ans,2]};
Do[AppendTo[js,1-Mod[js[[-1]]+Append[Rest[js[[-1]]],0]+Prepend[Most[js[[-1]]],0]+js[[-
2]],2]];,{n-2}];
```

```
js//MatrixPlot
```

最后放上边长 1000 的作为结束。