### 动规优化2★

　　可以发现，当公主与王子在相邻格子上时，是永远也不可能相遇的，如图17.5所示：

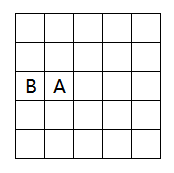


图17.5

　　如果第t天王子与公主在相邻的格子，那么如果王子与公主各往水平方向退一步，也就是第t－1天的位置，如图17.6所示：

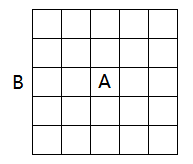


图17.6

　　王子和公主也是不可能相遇的。即如果公主往左边走去找王子，那么就会出现相邻的情况；如果公主上下移动来等王子追上来，显然也是不可能的（又会出现相邻的情况），如图17.7所示：

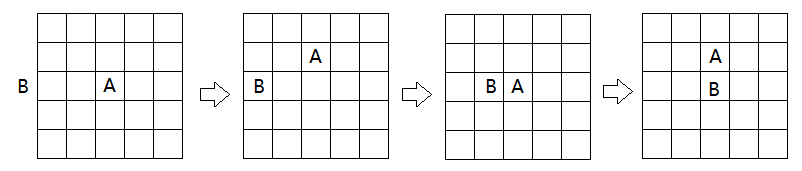


图17.7

　　由以上的分析可以继续设想：如果n更大，公主和王子继续往后退呢？显然可以得出一个结论：当王子与公主相遇的格数为偶数时，王子与公主不可能相遇。当输入n的时候，我们可以先判断（n－1）/2是否为偶数，如果不是的话，直接输出0。（写程序的时候可以判断int(n/2)是否为偶数，如果是的话，直接输出0。

参考代码如下所示。

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52 | //奇妙的相遇 — 动规优化2（部分数据有误）  #include <bits/stdc++.h>  using namespace std;  int n;  int Dir(int x,int y)  {  if((x==1&&y==1)||(x==n&&y==n)||(x==1&&y==n)||(x==n&&y==1))  return 2;  else if(x==1||y==1||x==n||y==n)  return 3;  else  return 4;  }  int main()  {  scanf("%d",&n);  if(!((n/2)&1)) //使用位运算判断是否相距偶数格  {  puts("0");  return 0;  }  int i,j,k;  double p[2][n][n],x=0;  for(i=0;i<n;i++)  for(j=0;j<n;j++)  p[0][i][j]=0,p[1][i][j]=0;  p[0][n/2][n/2]=1;  int w=0;  for(k=0;k<n;k++,w=1-w)  {  for(i=0;i<n;i++)  for(j=0;j<n;j++)  p[1-w][i][j]=0;  for(i=0;i<n;i++)  for(j=0;j<n;j++)  {  if(i-1>=0)  p[1-w][i-1][j]+=p[w][i][j]/Dir(i,j);  if(j-1>=0)  p[1-w][i][j-1]+=p[w][i][j]/Dir(i,j);  if(i+1<n)  p[1-w][i+1][j]+=p[w][i][j]/Dir(i,j);  if(j+1<n)  p[1-w][i][j+1]+=p[w][i][j]/Dir(i,j);  }  x+=p[1-w][n/2][k],p[1-w][n/2][k]=0;  }  printf("%.4f\n",x);  return 0;  } |

这里只判断了输入时的n，其实还可以对程序运行过程中的每一步进行判断。不过，在此之前，还可以进行更深入的优化。

### 动规优化3★

　　上述的优化是建立在公主与王子相邻的情况下各自往水平方向后退来实现的，那如果公主不是在水平方向上后退而是在垂直方向上后退呢？如图17.8所示：

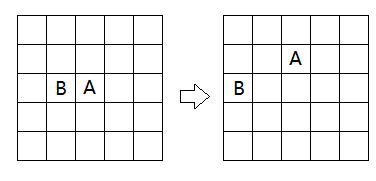


图17.8

　　也就是说，在这种情况下，他们是不会相遇的。如果再多退x步，则我们会发现，当王子的位置一定时，公主在粗斜线所经过的位置是不会相遇的，如图17.9所示。

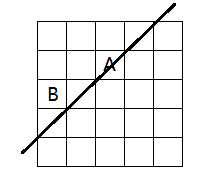


图17.9

　　如果公主先往水平方向后退t1步，再向垂直方向退t2步，显然他们也是不能相遇的。于是，我们可以得到图17.10（深色方格代表公主在这个方格里是不可能与正处于Ｂ方格的王子相遇的）。

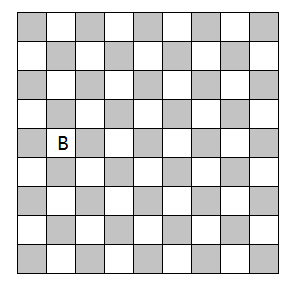


图17.10

　　如何表示这些不可能相遇的点呢？通过观察可以发现，任意一个深色方格与Ｂ方格的纵向与横向的方格数之和为偶数。于是得出：设在某个状态下王子的坐标为（x1，y1），公主的坐标为(x2，y2)，如果｜x1－x2－1｜+|y1－y2－1|为偶数的话，公主就不会与王子相遇。

　　还可以继续“剪枝”。可以发现，当公主在如图17.11所示的两条射线的左方时是不可能追上王子的。

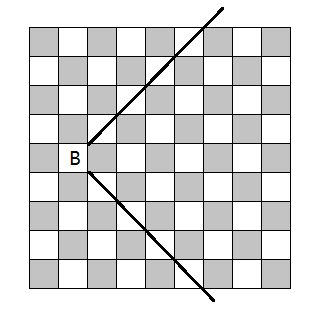


图17.11

　　于是，公主能追上王子的范围变成了如图17.12所示的白色方格。

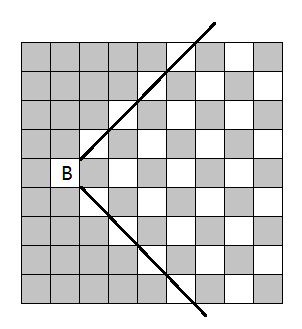


图17.12

　　这样可以大大减少程序计算的次数，提高效率。

参考代码如下所示。

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53 | //奇妙的相遇 — 动规优化3（部分数据有误）  #include <bits/stdc++.h>  using namespace std;  int n;  int Dir(int x,int y)  {  if((x==1&&y==1)||(x==n&&y==n)||(x==1&&y==n)||(x==n&&y==1))  return 2;  else if(x==1||y==1||x==n||y==n)  return 3;  else  return 4;  }  int main()  {  scanf("%d",&n);  if(!((n/2)&1))  {  puts("0");  return 0;  }  int i,j,k;  double p[2][n][n],x=0;  for(i=0; i<n; i++)  for(j=0; j<n; j++)  p[0][i][j]=0,p[1][i][j]=0;  p[0][n/2][n/2]=1;  int w=0;  for(k=0; k<n; k++,w=1-w)  {  for(i=0; i<n; i++)  for(j=0; j<n; j++)  p[1-w][i][j]=0;  for(i=0; i<n; i++)  for(j=k; j<n; j++) //k为当前天数和王子的横坐标，公主在王子左边则不会相遇  if((abs(j-k-1)+abs(i-n/2-1))%2!=0) //剪枝  {  if(i-1>=0)  p[1-w][i-1][j]+=p[w][i][j]/Dir(i,j);  if(j-1>=0)  p[1-w][i][j-1]+=p[w][i][j]/Dir(i,j);  if(i+1<n)  p[1-w][i+1][j]+=p[w][i][j]/Dir(i,j);  if(j+1<n)  p[1-w][i][j+1]+=p[w][i][j]/Dir(i,j);  }  x+=p[1-w][n/2][k],p[1-w][n/2][k]=0;  }  printf("%.4f\n",x);  return 0;  } |

### 动规优化4★★

上述的优化当n取得非常大时还是会超时（例如当n≥339时）。所以还需要进一步优化。可以发现，王子与公主的初始位置的连线刚好平分整个“棋盘”，连线的上方和下方完全对称，且王子仅能在该线上行走。所以可以把该连线作为一个镜面，镜面的上下各个对称点的概率显然是完全相等的，因此只需要计算上面那个镜像的概率之和，就可以得到整个“棋盘”的概率。为了方便，可以把王子所走的那条线定义为x轴，如图17.13所示：

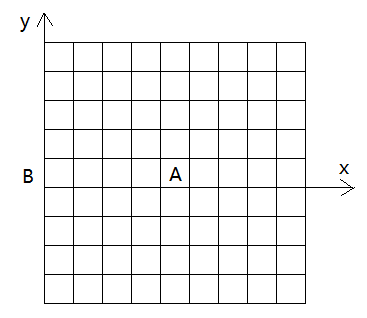


图17.13

由于上下是对称的，所以计算到x轴上时几率要乘2。

参考代码如下所示。

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56 | //奇妙的相遇 — 动规优化4（部分数据有误）  #include <bits/stdc++.h>  using namespace std;  int n;  double p[2][501][1000],x=0; //只需计算一半，所以数组只用开辟一半  int Dir(int x,int y)  {  if((x==1&&y==1)||(x==n&&y==n)||(x==1&&y==n)||(x==n&&y==1))  return 2;  else if(x==1||y==1||x==n||y==n)  return 3;  else  return 4;  }  int main()  {  scanf("%d",&n);  if(!((n/2)&1))  {  puts("0");  return 0;  }  int i,j,k;  for(i=0; i<n/2+1; i++)  for(j=0; j<n; j++)  p[0][i][j]=0,p[1][i][j]=0;  p[0][n/2][n/2]=1;  int w=0;  for(k=0; k<n; k++,w=1-w)  {  for(i=0; i<n/2+1; i++)  for(j=0; j<n; j++)  p[1-w][i][j]=0;  for(i=0; i<n/2+1; i++)  for(j=k; j<n; j++)  if((abs(j-k-1)+abs(i-n/2-1))%2!=0)  {  if(i-1>=0)  p[1-w][i-1][j]+=p[w][i][j]/Dir(i,j);  if(j-1>=0)  p[1-w][i][j-1]+=p[w][i][j]/Dir(i,j);  if(i+1<n/2)  p[1-w][i+1][j]+=p[w][i][j]/Dir(i,j);  if(i+1==n/2) //计算到中间时概率要乘以2  p[1-w][i+1][j]+=(p[w][i][j]/Dir(i,j))\*2;  if(j+1<n)  p[1-w][i][j+1]+=p[w][i][j]/Dir(i,j);  }  x+=p[1-w][n/2][k],p[1-w][n/2][k]=0;  }  printf("%.4f\n",x);  return 0;  } |

　　此程序在n≤463时可以一秒出解。

### 动规优化5★★

　　虽然经过了上述大量的优化，程序的提升却并不是非常大，这是为什么呢？可以发现，虽然经过了大量优化，但是由于程序判断较多，且每当天数变化时，要用两重循环来刷新数组。大大降低了程序的运行效率，所以我们要对程序的运行效率进行优化。

　　在对运行效率进行优化之前，先使用逆推的思想进行优化。之前的算法都是建立在从前往后退的基础上。如果从后往前退呢？

　　假设第t天公主与王子在（i，j）相遇了，那么当前位置的概率为1，如果往前推，一直推到初始的位置，此时公主所在格子的概率就为其在（i，j）与王子相遇的概率，但需要注意的是，在往回退的过程中，公主与王子不能再相遇（一次相遇之后就会停止，不会出现多次相遇）。

　　在第2天公主与王子在（1，2）相遇，该处概率为1，如图17.14所示。

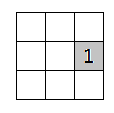


图17.14

　　接着往前退，此时公主有2个方向可以走（注意不能往左走），如图17.15所示。

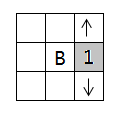


图17.15

　　和前面的算法一样，概率平分，都为0.5，如图17.16所示。

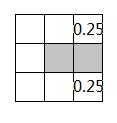


图17.16

　　以此类推，一直推到初始状态，如图17.17所示：

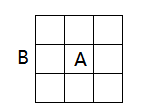


图17.17

　　此时公主所在位置的概率就是公主在第2天与王子在（1，2）位置相遇的概率，然后进行概率累加。

　　这样的算法看起来和之前的顺推是差不多的，那么它有什么好处呢？仔细思考后可以发现，不管公主与王子在哪里相遇，逆推后的最终位置是一定的，也就是公主的初始位置。所以就可以免去了刷新数组，直接输出公主的初始位置的概率。同时，还可以进行数组的优化。可以发现，第（i，j）位置上的概率事实上可以看成是该位置四个方向上各位置的概率之和除以（i，j）位置上能扩展出的方向数（事实上就是能一步到达（i，j）位置的各位置概率的平均值）。这样一来我们只需开辟一个二维数组，在过程中进行奇偶性判断。我们还可以用位运算来代替许多计算，如奇偶性的判断等。在这个程序中，我们把原点位置定义为（1，0），如图17.18所示：

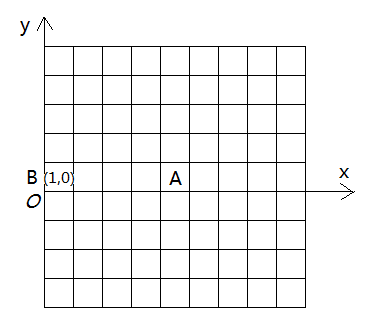


图17.18

参考代码如下所示。

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52 | //奇妙的相遇 — 动规优化5  #include <bits/stdc++.h>  #define MAX 1555  using namespace std;  int n,m;  double f[MAX][MAX>>1]; //MAX>>1相当于MAX/2  int inline d(int x,int y) //用来计算(x,y)位置能扩展出来的路径数  {  return 2+(x>1 && x<n)+(y<m); //判断是否越界，能扩展出的路径数至少为2  }  int inline minn(int ax,int bx) //返回ax,bx的较小值  {  return ax<=bx?ax:bx;  }  int inline maxn(int ax,int bx) //返回ax,bx的较大值  {  return ax>=bx?ax:bx;  }  int inline absn(int ax) //返回ax的绝对值  {  return ax>=0?ax:-ax;  }  double Dp() //计算过程  {  if(m&1^1) //初始奇偶性判断，使用了位运算  return 0;  int i,j,k,end\_k,end\_j;  for(i=n;i>=0;f[i--][0]=1) //此处开始逆推，逆推的初始位置的概率为1  for(j=maxn(i,m+1-i),end\_j=minn(n,m+1+i);j<=end\_j;++j)  { //此处通过maxn(i,m+1-i)来构造镜像，同时对公主的范围进行剪枝  if(j-i&1^1) //过程中使用位运算判断奇偶,表示当i与j同奇偶时返回1  f[j][0]=(f[j][1]+f[j][1]+f[j+1][0]+f[j-1][0])/d(j,0);  for(k=2-(j-i&1),end\_k=minn(minn(j-i,m),i-absn(m+1-j));k<=end\_k;k+=2)  //（1）j-i&1同样是过程中的奇偶性判断，当i+j为偶数时返回0，否则返回1  //（2）minn(j-i,m)为镜像选择,也是公主追赶王子的剪枝,absn(m+1-j)为公主范围剪枝  f[j][k]=(f[j+1][k]+f[j-1][k]+f[j][k+1]+f[j][k-1])/d(j,k);//滚动数组的合并  }  return f[m+1][0]; //返回公主的初始位置概率值  }  int main()  {  cin>>n;  m=n>>1;  cout<<fixed<<setprecision(4)<<Dp()<<'\n';  return 0;  } |

　　上述的代码可以通过n≤1 535的数据。