[**动态规划解抛鸡蛋(玻璃球)问题**](http://www.cppblog.com/xiaoyisnail/archive/2009/09/18/96638.html)

    昨天早上看了题经典老题，抛玻璃球，也有的版本是抛鸡蛋，可惜昨天早上愣是没做出来，下午忙别的事去了，到了晚上看了ChinaUnix上的一篇[讨论帖](http://bbs.chinaunix.net/viewthread.php?tid=868766&extra=&page=1)才知道如何解，事实上我一开始对题目的理解就错了，于是根本没有想到用DP。今天总算有时间整理一下思路，并把代码实现出来了。  
    题目是这样的：一个100层的大厦，你手中有两个相同的玻璃球。从这个大厦的某一层扔下围棋  
子就会碎，用你手中的这两个玻璃围棋子，找出一个最优的策略，来得知那个临界层面。  
    这里的最优策略指的是在这种策略下无论哪个临界层面在第几层，测试的次数最少。我一开始就是把题意理解错了，给了一个非最优解，后来看了CU那的讨论后才明白了是用动态规划来做，并可以把题目扩展为n层大厦用k个玻璃球来测试。  
    设F(n,k)为用k个玻璃球来测试n层大厦的临界层的最少次数，状态转移方程如下：  
    F(n,k)=min{max{F(r,k-1), F(n-r,k)}+1, 1<=r<=n}  
    边界条件:F(n,1)=n-1, F(1,k)=F(0,k)=0  
    状态转移方程可以这样来考虑，假设在n层楼中的第r层抛一次(对应方程中的"+1")，会有两种情况发生：  
    (1)玻璃球碎，说明在第1到第r层楼中必有一层为临界层，问题转化为一个子问题：求F(r,k-1)  
    (2)玻璃球不碎，说明临界层在第r+1层到第n层这n-r层楼中，问题转化为子问题:求F(n-r,k)  
    因为考虑的是最坏情况下抛球策略的所需测试次数的最小值，所以取这两种情况中的较大值，并遍历每一个可能的r，取其最小值即得到F(n,k)。  
    实现代码如下：

 1 #include <iostream>  
 2 #include <fstream>  
 3 #include <sstream>  
 4 #include <string>  
 5 #include <cmath>  
 6 #include <iomanip>  
 7 #include <vector>  
 8 #include <deque>  
 9 #include <list>  
10 #include <queue>  
11 #include <stack>  
12 #include <map>  
13 #include <algorithm>  
14 #include <limits>  
15 #include <utility>  
16 #include <ctime>  
17 #include <bitset>  
18 using namespace std;  
19   
20 #define MAX\_FLOOR 512  
21 #define MAX\_BALL  100  
22   
23 int dp(int n, int k)  
24 {  
25     if(k<1 || n<1) return -1;    //错误输入  
26   
27     if(k==1) return n-1;        //去掉一些trivial case  
28     if(n==1) return 0;  
29   
30     int M[MAX\_BALL][MAX\_FLOOR];  
31     int i,j,r;  
32     int temp, min;  
33   
34     for(i=0;i<=k;i++) M[i][0]=M[i][1]=0;    //F(1,k)=F(0,k)=0  
35     for(j=2;j<=n;j++) M[1][j]=j-1;            //F(n,1)=n-1  
36   
37     /\*  
38     状态转移方程：  
39     F(n,k)=min{max{F(r,k-1)+1, F(n-r,k)+1}, 1<=r<=n}  
40     \*/  
41     for(i=2;i<=k;i++)  
42         for(j=2;j<=n;j++)  
43         {  
44             min = numeric\_limits<int>::max();  
45             for(r=1;r<=j;r++)  
46             {  
47                 temp = max(M[i-1][r], M[i][j-r])+1;  
48                 if(temp<min)  
49                     min = temp;  
50             }  
51             M[i][j] = min;  
52         }  
53   
54     return M[k][n];//F(n,k)  
55 }  
56   
57 int main()  
58 {  
59     int n,k;  
60       
61     cin>>n>>k;  
62     cout<<dp(n, k)<<endl;  
63       
64     return 0;  
65 }

input: 100 2  output: 14  
input: 300 3  output: 13