> 若干层楼，若干个鸡蛋，让你算出最少的尝试次数，找到鸡蛋恰好摔不碎的那层楼。

分析：

1. 其状态分别为鸡蛋数k,和楼层数N;
2. 选择就是从哪个楼层扔鸡蛋。

分析递归条件： 当在第i层向下扔鸡蛋有如下两种情况：

- 如果鸡蛋没碎，则继续上面楼层搜索dp(k, N-i), 此时鸡蛋数未变；
- 如果鸡蛋碎了，则向下层开始搜索dp(k-1, i-1), 此时鸡蛋已经碎了。

递归解法：

```
int dp(int k, int N) {
    if(k == 1) {
        return N;
    }
    if(N == 0) {
        return 0;
    }

    int res = INT_MAX;
    for(int i = 1;i<N;++i){
        res = std::min(res, std::max(dp(k-1,i-1), dp(k, n-i))+1)
    }

    return res;
}
```

备忘录优化；

```
#include <climits>
#include <vector>

int minegg(int k, int n, std::vector<std::vector<int>> memo) {
  if (k == 1) {
    return n;
  }
  if (n == 0) {
    return 0;
  }

  if (memo[k][n] != 0) {
    return memo[k][n];
  }
```

```
    int res = INT_MAX;
    for (int i = 1; i <= n; i++) {
      res = std::min(
          res,
          1 + std::max(minegg(k - 1, i - 1, memo), minegg(k, n - i, memo)));
    }

    memo[k][n] = res;

    return memo[k][n];
  }

  int superEggDrop(int k, int n) {
    std::vector<std::vector<int>> memo =
        std::vector<std::vector<int>>(k + 1, std::vector<int>(n + 1, 0));
    return minegg(k, n, memo);
  }
```

修改动态规划方程：

```
// k个鸡蛋扔m层，最多可以测试t层楼
dp[k][m] = t
```

基于下面两个事实： 1、**无论你在哪层楼扔鸡蛋，鸡蛋只可能摔碎或者没摔碎，碎了的话就测楼下，没碎的话就测楼上。** 2、**无论你上楼还是下楼，总的楼层数 = 楼上的楼层数 + 楼下的楼层数 + 1（当前这层楼）。**

```
dp[k][m] = 1 + dp[k-1][m-1] + dp[k][m-1]
```

算法有：

```
int superEggDrop(int K, int N) {
    std::vector<std::vector<int>> dp= std::vector<std::vector<int>>(K+1,
std::vector<int>(N+1,0));

    int m = 0;
    for(;dp[K][m] < N; m++){
      for(int i=1;i<=k;i++){
        dp[i][m] = 1 + dp[i-1][m-1]+dp[i][m-1];
      }
    }

    return m;
}
```