我们正在玩一个猜数游戏，游戏规则如下：
我从 **1** 到 **n** 之间选择一个数字，你来猜我选了哪个数字。
每次你猜错了，我都会告诉你，我选的数字比你的大了或者小了。
然而，当你猜了数字 x 并且猜错了的时候，你需要支付金额为 x 的现金。直到你猜到我选的数字，你才算赢得了这个游戏。
示例:

```
n = 10, 我选择了8.

第一轮：你猜我选择的数字是5，我会告诉你，我的数字更大一些，然后你需要支付5块。
第二轮：你猜是7，我告诉你，我的数字更大一些，你支付7块。
第三轮：你猜是9，我告诉你，我的数字更小一些，你支付9块。

游戏结束。8 就是我选的数字。

你最终要支付 5 + 7 + 9 = 21 块钱。
```

给定 n ≥ 1，计算你至少需要拥有多少现金才能确保你能赢得这个游戏。

那么我们需要建立一个二维的dp数组，其中dp[i][j]表示从数字i到j之间猜中任意一个数字最少需要花费的钱数，那么我们需要遍历每一段区间[j, i]，维护一个全局最小值global_min变量，然后遍历该区间中的每一个数字，计算局部最大值local_max = k + max(dp[j][k - 1], dp[k + 1][i])，这个正好是将该区间在每一个位置都分为两段，然后取当前位置的花费加上左右两段中较大的花费之和为局部最大值，为啥要取两者之间的较大值呢，因为我们要cover所有的情况，就得取最坏的情况。然后更新全局最小值，最后在更新dp[j][i]的时候看j和i是否是相邻的，相邻的话赋为i，否则赋为global_min。这里为啥又要取较小值呢，因为dp数组是求的[j, i]范围中的最低cost，比如只有两个数字1和2，那么肯定是猜1的cost低，是不有点晕，没关系，博主继续来绕你。我们想，如果只有一个数字，那么我们不用猜，cost为0。如果有两个数字，比如1和2，我们猜1，即使不对，我们cost也比猜2要低。如果有三个数字1，2，3，那么我们就先猜2，根据对方的反馈，就可以确定正确的数字，所以我们的cost最低为2。如果有四个数字1，2，3，4，那么情况就有点复杂了，那么我们的策略是用k来遍历所有的数字，然后再根据k分成的左右两个区间，取其中的较大cost加上k。
当k为1时，左区间为空，所以cost为0，而右区间2，3，4，根据之前的分析应该取3，所以整个cost就是1+3=4。
当k为2时，左区间为1，cost为0，右区间为3，4，cost为3，整个cost就是2+3=5。

当k为3时，左区间为1，2，cost为1，右区间为4，cost为0，整个cost就是3+1=4。
当k为4时，左区间1，2，3，cost为2，右区间为空，cost为0，整个cost就是4+2=6。
综上k的所有情况，此时我们应该取整体cost最小的，即4，为最后的答案，这就是极小化极大算法，参见代码如下

```
class Solution {
public:
    int getMoneyAmount(int n) {
        if(n<=2)return n-1;
        vector<vector<int>> dp(n + 1, vector<int>(n+1,0));
       for(int i=2;i<=n;i++)
           for(int j=i-1;j>0;j--)
           {
               int global=999;
               int local=0;
               for(int k=j+1;k<i;k++)
               {
                   local=k+max(dp[k+1][i], dp[j][k-1]);
                   global=min(global,local);
               }
                 dp[j][i] = j + 1 == i ? j : global;
           }
        return dp[1][n];

    }
};
```