小 v 最近在玩一款挖矿的游戏，该游戏介绍如下：

1、每次可以挖到多个矿石，每个矿石的重量都不一样，挖矿结束后需要通过一款平衡矿车运送下山；
2、平衡矿车有左右 2 个车厢，中间只有 1 个车轮沿着导轨滑到山下，且矿车只有在 2 个车厢重量完全相等且矿石数量相差不超过 1 个的情况下才能成功运送矿石，否则在转弯时可能出现侧翻。

假设小 v 挖到了 n（n<100）个矿石，每个矿石重量不超过 100，为了确保一次性将 n 个矿石都运送出去，一旦矿车的车厢重量不一样就需要购买配重砝码。请问小 v 每次**最少**需要购买多少重量的砝码呢？（假设车厢足够放下这些矿石和砝码，砝码重量任选）


```
private static int solution(int[] input) {
        // 矿石总重量
        int sum = 0;
        int n = input.length;
        for(int i = 0; i < n; i++) sum += input[i];
        int halfNum = (n + 1) / 2;              // 一半的数量
        int halfWeight = (sum + 1) / 2;       // 一半的重量
        int diff = 10000;                          // 两车矿石的重量差
        // 动态规划，dp[i][j]表示用 i 块矿石能否使重量为 j
        boolean[][] dp = new boolean[105][10005];
        dp[0][0] = true;        // 0 块矿石凑重量 0 是可以的
        int weight = 0;          // 一辆矿石车的矿石重量
        for(int i = 0; i < n; i++){
            for(int j = sum; j >= input[i]; j--){
                for(int k = n; k >= 0; k--){
                    // 如果可以用 k 块矿石凑 j-input[i]的重量
                    if(dp[k][j - input[i]]){
                        dp[k + 1][j] = true;      // 选上第 i 块矿石凑出重量 j
                        if(k + 1 == halfNum && Math.abs(j - halfWeight) < diff){
                            // 如果数量满足题意，则将重量差更新为小的那个，以最大限度逼近总重量的一半
                            diff = Math.abs(j - halfWeight);
```

```
                                weight = j;
                            }
                        }
                    }
                }
            }
            return Math.abs(sum - weight * 2);
        }
```

今年 7 月份 vivo 迎来了新入职的大学生，现在需要为每个新同事分配一个工号。人力资源部同事小 v 设计了一个方法为每个人进行排序并分配最终的工号，具体规则是：

将 N（N<10000）个人排成一排，从第 1 个人开始报数；如果报数是 M 的倍数就出列，报到队尾后则回到队头继续报，直到所有人都出列；
一个简单不易错的方法。
其他人的方法有些使用 erase 或者 remove 的操作删除元素，这个操作的复杂度是 O(n)，使用队列可降到O(1).

```
def solution(N,M):
    from collections import deque
    que = deque(list(range(1,N+1)))
    i = 0
    while que:
        i += 1
        if i%M == 0: #  如果是 M 的倍数，移出队列并打印
            print(que.popleft() ,end = " ")
        else: #  否则把队首放到队尾
            que.append(que.popleft())

N,M = [int(i) for i in input().split()]
solution(N,M)
```

小 v 在公司负责游戏运营，今天收到一款申请新上架的游戏“跳一跳”，为了确保提供给广大玩家朋友们的游戏都是高品质的，按照运营流程小 v 必须对新游戏进行全方位了解体验和评估。这款游戏的规则如下：

有 n 个盒子排成了一行，每个盒子上面有一个数字 a[i]，表示在该盒子上的人最多能向右移动 a[i]个盒子（比如当前所在盒子上的数字是 3，则表示可以一次向右前进 1 个盒子，2 个盒子或者 3 个盒子）。

现在小 v 从左边第一个盒子上开始体验游戏，请问**最少**需要移动几次能到最后一个盒子上？

```
private static int solution(int[] input) {
```

```
        if(input.length==1){
            //如果只有一个盒子，那么起点即为终点，无需跳跃
            return 0;
        }
        if(input.length>1&&input[0]==0){
            //如果不只一个盒子，那么肯定需要跳动。但是起点位置能跳跃的距离为 0
            //  那么肯定到不了终点
            return -1;
        }
        //下面的代码处理需要跳动才能到达终点的情况
        int[] dp=new int[input.length];

        for (int i=0;i<input.length;i++){
            //计算能够到达最右的盒子
            int range=Math.min(input.length-1,i+input[i]);
            for (int j=i+1;j<=range;j++){
                if(dp[j]==0){
                    dp[j]=dp[i]+1;
                }
            }
        }
        int ret=dp[input.length-1];
        //需要跳动的次数至少大于 0，结果出现跳动 0 次，说明不可达，返回-1
        return ret==0?-1:ret;

    }
```