排成一条线的纸牌博弈问题

【题目】

给定一个整型数组 arr，代表数值不同的纸牌排成一条线。玩家 A 和玩家 B 依次拿走每张纸牌，规定玩家 A 先拿，玩家 B 后拿，但是每个玩家每次只能拿走最左或最右的纸牌，玩家 A 和玩家 B 都绝顶聪明。请返回最后获胜者的分数。

【举例】

arr=[1,2,100,4]。

开始时玩家 A 只能拿走 1 或 4。如果玩家 A 拿走 1，则排列变为[2,100,4]，接下来玩家 B 可以拿走 2 或 4，然后继续轮到玩家 A。如果开始时玩家 A 拿走 4，则排列变为[1,2,100]，接下来玩家 B 可以拿走 1 或 100，然后继续轮到玩家 A。玩家 A 作为绝顶聪明的人不会先拿 4，因为拿 4 之后，玩家 B 将拿走 100。所以玩家 A 会先拿 1，让排列变为[2,100,4]，接下来玩家 B 不管怎么选，100 都会被玩家 A 拿走。玩家 A 会获胜，分数为 101。所以返回 101。

arr=[1,100,2]。

开始时玩家 A 不管拿 1 还是 2，玩家 B 作为绝顶聪明的人，都会把 100 拿走。玩家 B 会获胜，分数为 100。所以返回 100。

【难度】

事实上，存在先手必胜的方法，只需要计算奇数位置的和与偶数位置的和进行比较，若奇数位置上的和更大，则每次去奇数位置上的数值即可。反之亦然。

尉 ★★☆☆

【解答】

暴力递归的方法。定义递归函数 $f(i,j)$，表示如果 arr[i..j]这个排列上的纸牌被绝顶聪明的人先拿，最终能获得什么分数。定义递归函数 $s(i,j)$，表示如果 arr[i..j]这个排列上的纸牌被绝顶聪明的人后拿，最终能获得什么分数。

首先来分析 $f(i,j)$，具体过程如下：

1．如果 i==j（即 arr[i..j]）上只剩一张纸牌。当然会被先拿纸牌的人拿走，所以返回 arr[i]。

2．如果 i!=j。当前拿纸牌的人有两种选择，要么拿走 arr[i]，要么拿走 arr[j]。如果拿走 arr[i]，那么排列将剩下 arr[i+1..j]。对当前的玩家来说，面对 arr[i+1..j]排列的纸牌，他成

了后拿的人，所以后续他能获得的分数为 $s(i+1,j)$。如果拿走 arr[j]，那么排列将剩下 arr[i..j-1]。对当前的玩家来说，面对 arr[i..j-1]排列的纸牌，他成了后拿的人，所以后续他能获得的分数为 $s(i,j-1)$。作为绝顶聪明的人，必然会在两种决策中选最优的。所以返回 max{arr[i]+s(i+1,j) , arr[j]+s(i,j-1)}。

然后来分析 $s(i,j)$，具体过程如下：

1. 如果 i==j（即 arr[i..j]）上只剩一张纸牌。作为后拿纸牌的人必然什么也得不到，返回 0。

2. 如果 i!=j。根据函数 s 的定义，玩家的对手会先拿纸牌。对手要么拿走 arr[i]，要么拿走 arr[j]。如果对手拿走 arr[i]，那么排列将剩下 arr[i+1..j]，然后轮到玩家先拿。如果对手拿走 arr[j]，那么排列将剩下 arr[i..j-1]，然后轮到玩家先拿。对手也是绝顶聪明的人，所以必然会把最差的情况留给玩家。所以返回 min{f(i+1,j) , f(i,j-1)}。

具体过程请参看如下代码中的 win1 方法。

```
public int win1(int[] arr) {
        if (arr == null || arr.length == 0) {
                return 0;
        }
        return Math.max(f(arr, 0, arr.length - 1), s(arr, 0, arr.length - 1));
}

public int f(int[] arr, int i, int j) {
        if (i == j) {
                return arr[i];
        }
        return Math.max(arr[i] + s(arr, i + 1, j), arr[j] + s(arr, i, j - 1));
}

public int s(int[] arr, int i, int j) {
        if (i == j) {
                return 0;
        }
        return Math.min(f(arr, i + 1, j), f(arr, i, j - 1));
}
```

暴力递归的方法中，递归函数一共会有 N 层，并且是 f 和 s 交替出现的。$f(i,j)$会有 $s(i+1,j)$ 和 $s(i,j-1)$两个递归分支，$s(i,j)$也会有 $f(i+1,j)$和 $f(i,j-1)$两个递归分支。所以整体的时间复杂度为 $O(2^N)$，额外空间复杂度为 $O(N)$。下面介绍动态规划的方法，如果 arr 长度为 N，生成两个大小为 $N\times N$ 的矩阵 f 和 s，f[i][j]表示函数 $f(i,j)$的返回值，s[i][j]表示函数 $s(i,j)$的返回值。规定一下两个矩阵的计算方向即可。具体过程请参看如下代码中的 win2 方法。

```
public int win2(int[] arr) {
```

```
        if (arr == null || arr.length == 0) {
            return 0;
        }
        int[][] f = new int[arr.length][arr.length];
        int[][] s = new int[arr.length][arr.length];
        for (int j = 0; j < arr.length; j++) {
            f[j][j] = arr[j];
            for (int i = j - 1; i >= 0; i--) {
                f[i][j] = Math.max(arr[i] + s[i + 1][j], arr[j] + s[i][j - 1]);
                s[i][j] = Math.min(f[i + 1][j], f[i][j - 1]);
            }
        }
        return Math.max(f[0][arr.length - 1], s[0][arr.length - 1]);
    }
```

如上的 win2 方法中，矩阵 f 和 s 一共有 $O(N^2)$个位置，每个位置计算的过程都是 $O(1)$ 的比较过程，所以 win2 方法的时间复杂度为 $O(N^2)$，额外空间复杂度为 $O(N^2)$。

跳跃游戏

【题目】

给定数组 arr，arr[i]==k 代表可以从位置 i 向右跳 1~k 个距离。比如，arr[2]==3，代表从位置 2 可以跳到位置 3、位置 4 或位置 5。如果从位置 0 出发，返回最少跳几次能跳到 arr 最后的位置上。

【举例】

arr=[3,2,3,1,1,4]。

arr[0]==3，选择跳到位置 2；arr[2]==3，可以跳到最后的位置。所以返回 2。

【要求】

如果 arr 长度为 N，要求实现时间复杂度为 $O(N)$、额外空间复杂度为 $O(1)$的方法。

【难度】

士 ★☆☆☆