```
            res += num1To9999(yi) + "亿";
            if (rest == 0) {
                    return res;
            } else {
                    if (rest < 10000000) {
                            return res + "零" + num1To99999999(rest);
                    } else {
                            return res + num1To99999999(rest);
                    }
            }
    }
```

该类型的代码面试题目实际上是相当棘手的。通常是由小的、简单的场景出发，把复杂的事情拆解成简单的场景，最终得到想要的结果。

分糖果问题

【题目】

一群孩子做游戏，现在请你根据游戏得分来发糖果，要求如下：

1．每个孩子不管得分多少，起码分到 1 个糖果。

2．任意两个相邻的孩子之间，得分较多的孩子必须拿多一些的糖果。

给定一个数组 arr 代表得分数组，请返回最少需要多少糖果。

例如：arr=[1,2,2]，糖果分配为[1,2,1]，即可满足要求且数量最少，所以返回 4。

【进阶题目】

原题目中的两个规则不变，再加一条规则：

3．任意两个相邻的孩子之间如果得分一样，糖果数必须相同。

给定一个数组 arr 代表得分数组，返回最少需要多少糖果。

例如：arr=[1,2,2]，糖果分配为[1,2,2]，即可满足要求且数量最少，所以返回 5。

【要求】

arr 长度为 N，原题与进阶题都要求时间复杂度为 $O(N)$，额外空间复杂度为 $O(1)$。

【难度】

校 ★★★☆

【解答】

原问题。先引入爬坡和下坡的概念，从左到右依次考虑每个孩子，如果一个孩子的右邻居比他大，那么爬坡过程开始。如果一直单调递增，就一直爬坡，否则爬坡结束，下坡开始。如果一直单调递减，就一直下坡，直到遇到一个孩子的右邻居大于或等于他，则下坡结束。爬坡中的叫左坡，下坡中的叫右坡。

比如[1,2,3,2,1]，左坡为[1,2,3]，右坡为[3,2,1]。比如[1,2,2,1]，第一个左坡为[1,2]，第一个右坡为[2](只含有第一个 2)，第二个左坡为[2](只含有第二个 2)，第二个右坡为[2,1]。比如[1,2,3,1,2]，第一个左坡[1,2,3]，第一个右坡为[3,1]，第二个左坡为[1,2]，第二个右坡为[2]。

定义了爬坡过程和下坡过程之后，大家可以看到，arr 数组可以被分解成很多对左坡和右坡，利用左坡和右坡来看糖果如何分。假设有一对左坡和右坡，分别为[1,4,5,9]和[9,3,2]。对左坡来说，从左到右分的糖果应该为[1,2,3,4]，对右坡来说，从左到右分的糖果应该为[3,2,1]。但这两种分配方式对 9 这个坡顶的分配是不同的，怎么决定呢？看左坡和右坡的坡度哪个更大，坡度是指坡中除去相同的数字之后(也就是纯升序或纯降序）的序列长度。而根据我们定义的爬坡和下坡过程，左坡和右坡中都不可能有重复数字，所以坡度就是各自的序列长度。[1,2,3,4]坡度为 4，[3,2,1]坡度为 3。如果左坡的坡度更大，坡顶就按左坡的分配，如果右坡的坡度更大，就按右坡的分配，所以最终分配为[1,2,3,4,2,1]。

成对的左坡和右坡都按照这种处理方式，从左到右处理得分数组 arr，统计总体的糖果数即可。具体过程请参看如下代码中的 candy1 方法。

```
public int candy1(int[] arr) {
        if (arr == null || arr.length == 0) {
                return 0;
        }
        int index = nextMinIndex1(arr, 0);
        int res = rightCands(arr, 0, index++);
        int lbase = 1;
        int next = 0;
        int rcands = 0;
        int rbase = 0;
        while (index != arr.length) {
                if (arr[index] > arr[index - 1]) {
                        res += ++lbase;
                        index++;
                } else if (arr[index] < arr[index - 1]) {
                        next = nextMinIndex1(arr, index - 1);
                        rcands = rightCands(arr, index - 1, next++);
                        rbase = next - index + 1;
                        res += rcands + (rbase > lbase ? -lbase : -rbase);
```

```
                        lbase = 1;
                        index = next;
                } else {
                        res += 1;
                        lbase = 1;
                        index++;
                }
        }
        return res;
}

public int nextMinIndex1(int[] arr, int start) {
        for (int i = start; i != arr.length - 1; i++) {
                if (arr[i] <= arr[i + 1]) {
                        return i;
                }
        }
        return arr.length - 1;
}

public int rightCands(int[] arr, int left, int right) {
        int n = right - left + 1;
        return n + n * (n - 1) / 2;
}
```

进阶问题。针对进阶问题所加的新规则，需要对爬坡和下坡的过程进行修改。从左到右依次考虑每个孩子，如果一个孩子的右邻居大于或等于他，那么爬坡过程开始，如果一直不降序，就一直爬坡，否则爬坡结束，下坡开始。如果一直不升序，就一直下坡，直到遇到一个孩子的右邻居大于他，则下坡结束。爬坡中的叫左坡，下坡中的叫右坡。比如，[1,2,3,2,1]，左坡为[1,2,3]，右坡为[3,2,1]。再如，[1,2,2,1]，左坡为[1,2,2]，右坡为[2,1]。

依然是利用左坡和右坡来决定糖果如何分配，还是举例说明整个分配过程。比如，[0,1,2,3,3,3,2,2,2,2,2,1,1]，左坡为[0,1,2,3,3,3]，右坡为[3,2,2,2,2,2,1,1]。对左坡来说，从左到右分的糖果应该为[1,2,3,4,4,4]，对右坡来说，从左到右分的糖果应该为[3,2,2,2,2,2,1,1]。所以左坡和右坡的分配方案对整个坡顶的分配其实是矛盾的。注意，在这种情况下，其实坡顶为 3 个元素，即[3,3,3]。根据新的规则，相邻的且得分相等的孩子拿的糖果数要一样。所以坡顶究竟按谁的来呢？同样是根据左坡和右坡的坡度决定，左坡[0,1,2,3,3,3]的坡度为 4，右坡[3,2,2,2,2,2,1,1]的坡度为 3，坡顶分的糖果数同样按照坡度大的来决定。所以总的分配方案为[1,2,3,4,4,4,2,2,2,2,2,1,1]，也就是说，坡顶的所有小朋友都根据坡度大的一方决定。具体过程请参看如下代码中的 candy2 方法。

```
public int candy2(int[] arr) {
    if (arr == null || arr.length == 0) {
        return 0;
```

```
    }
    int index = nextMinIndex2(arr, 0);
    int[] data = rightCandsAndBase(arr, 0, index++);
    int res = data[0];
    int lbase = 1;
    int same = 1;
    int next = 0;
    while (index != arr.length) {
        if (arr[index] > arr[index - 1]) {
            res += ++lbase;
            same = 1;
            index++;
        } else if (arr[index] < arr[index - 1]) {
            next = nextMinIndex2(arr, index - 1);
            data = rightCandsAndBase(arr, index - 1, next++);
            if (data[1] <= lbase) {
                res += data[0] - data[1];
            } else {
                res += -lbase * same + data[0] - data[1] + data[1] * same;
            }
            index = next;
            lbase = 1;
            same = 1;
        } else {
            res += lbase;
            same++;
            index++;
        }
    }
    return res;
}

public int nextMinIndex2(int[] arr, int start) {
        for (int i = start; i != arr.length - 1; i++) {
                if (arr[i] < arr[i + 1]) {
                        return i;
                }
        }
        return arr.length - 1;
}

public int[] rightCandsAndBase(int[] arr, int left, int right) {
        int base = 1;
        int cands = 1;
        for (int i = right - 1; i >= left; i--) {
                if (arr[i] == arr[i + 1]) {
                        cands += base;
                } else {
                        cands += ++base;
                }
        }
        return new int[] { cands, base };
}
```