路径数组变为统计数组

【题目】

给定一个路径数组 paths，表示一张图。paths[i]==j 代表城市 i 连向城市 j，如果 paths[i]==i，则表示 i 城市是首都，一张图里只会有一个首都且图中除首都指向自己之外不会有环。例如，paths=[9,1,4,9,0,4,8,9,0,1]，代表的图如图 9-6 所示。

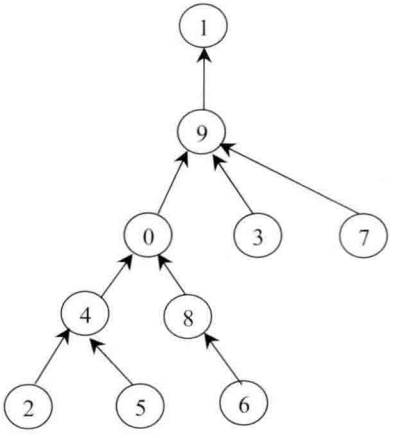

图 9-6

由数组表示的图可以知道，城市 1 是首都，所以距离为 0，离首都距离为 1 的城市只有城市 9，离首都距离为 2 的城市有城市 0、3 和 7，离首都距离为 3 的城市有城市 4 和 8，离首都距离为 4 的城市有城市 2、5 和 6。所以距离为 0 的城市有 1 座，距离为 1 的城市有 1 座，距离为 2 的城市有 3 座，距离为 3 的城市有 2 座，距离为 4 的城市有 3 座。那么统计数组为 nums=[1,1,3,2,3,0,0,0,0,0]，nums[i]==j 代表距离为 i 的城市有 j 座。要求实现一个 void 类型的函数，输入一个路径数组 paths，直接在原数组上调整，使之变为 nums 数组，即 paths=[9,1,4,9,0,4,8,9,0,1]经过这个函数处理后变成[1,1,3,2,3,0,0,0,0,0]。

【要求】

如果 paths 长度为 N，请达到时间复杂度为 $O(N)$，额外空间复杂度为 $O(1)$。

【难度】

校 ★★★☆

【解答】

本题完全考查代码实现技巧，怎么在一个数组上不停地折腾且不出错是非常锻炼边界处理能力的。本书提供的解法分为两步，第一步是将 paths 数组转换为距离数组。以题目中的例子来说，paths=[9,1,4,9,0,4,8,9,0,1]转换为[-2,0,-4,-2,-3,-4,-4,-2,-3,-1]。转换后的 paths[i]==j 代表城市 i 距离首都的距离为 j 的绝对值。至于为什么距离数组中的值要设置为负数，在以下过程中会说明。转换成距离数组的过程如下：

1．从左到右遍历 paths，先遍历位置 0。

paths[0]==9，首先令 paths[0]=-1，因为城市 0 指向城市 9，所以跳到城市 9。

跳到城市 9 之后，paths[9]==1，说明城市 9 下一步应该跳到城市 1，因为城市 9 是由城市 0 跳过来的，所以先令 paths[9]=0，然后跳到城市 1。

跳到城市 1 之后，此时 paths[1]==1，说明城市 1 是首都，停止向首都跳的过程。城市 1 是由城市 9 跳过来的，所以跳回城市 9。

根据之前的设置(paths[9]==0)，我们可以知道城市 9 下一步应该跳回城市 0，在跳回之前先设置 paths[9]==-1，表示城市 9 距离为 1，然后跳回城市 0。

根据之前的设置(paths[0]==-1)，我们知道城市 0 是整个过程的发起城市，所以不需要再回跳，设置 paths[0]=-2，表示城市 0 距离为 2。

以上在跳向首都的过程中，paths 数组有一个路径反指的过程，这是为了保证找到首都之后，能够完全跳回来。在跳回来的过程中，设置好这一路所跳城市的距离即可，此时 paths=[-2,1,4,9,0,4,8,9,0,-1]。

2．遍历到位置 1，此时 paths[1]==1，说明城市 1 是首都，令一个单独的变量 cap=1，然后不再做任何操作。

3．遍历到位置 2，paths[2]==4，先令 paths[2]=-1，因为城市 2 指向城市 4，跳到城市 4。

跳到城市 4 之后，paths[4]==0，说明城市 4 下一步应该跳到城市 0，因为城市 4 是由城市 2 跳过来的，所以先令 paths[4]=2，然后跳到城市 0。

跳到城市 0 之后，发现 paths[0]==-2，此时将距离设置为负数的作用就显现出来了，是负数标记着这是一个之前已经计算过与首都的距离的值，而不是下一跳的城市，所以向前跳的过程停止，开始跳回城市 4。

跳回到城市 4 之后，根据之前的设置(paths[4]==2)，可以知道城市 4 下一步应该跳回城市 2。但先设置 paths[4]=-3，因为城市 4 跳到城市 0 之后发现 paths[0]已经等于-2，所以自己距离首都的距离应该再远一步，然后跳回城市 2。

跳回到城市 2 之后，根据之前的设置(paths[2]==-1)，我们知道城市 2 是整个过程的发起城市，所以不需要再回跳，设置 paths[2]=-4，表示城市 2 距离为 4，此时 paths=[-2,1,-4,9,-3,4,8,9,0,-1]

4．遍历到位置 3，paths[3]==9，先令 paths[3]=-1，因为城市 3 指向城市 9，跳到城市 9。

跳到城市 9 之后，发现 paths[9]==-1，说明城市 9 之前已经计算过与首都的距离，所以向前跳的过程停止，开始跳回城市 3。

跳回到城市 3 之后，根据之前的设置(paths[3]==-1)，知道城市 3 是整个过程的发起城市，所以不需要再回跳，设置 paths[3]=-2（因为之前 paths[9]==-1）。所以此时 paths=[-2,1,-4,-2,-3,4,8,9,0,-1]

5．遍历到位置 4，发现 paths[4]==-3，说明之前计算过城市 4 的值，直接继续下一步。

6．遍历到位置 5，paths[5]==4，首先令 paths[5]=-1，因为城市 5 指向城市 4，跳到城市 4。

跳到城市 4 之后，发现 paths[4]==-3，说明城市 4 之前已经计算过与首都的距离，所以向前跳的过程停止，跳回城市 5。

跳回到城市 5 之后，根据之前的设置(paths[5]==-1)，我们知道城市 5 是整个过程的发起城市，所以不需要再回跳，设置 paths[5]=-4，此时 paths=[-2,1,-4,-2,-3,-4,8,9,0,-1]

7．遍历到位置 6，paths[6]==8，先令 paths[6]=-1，因为城市 6 指向城市 8，跳到城市 8。

跳到城市 8 之后，发现 paths[8]==0，说明城市 8 下一步应该跳到城市 0，因为城市 8 是由城市 6 跳过来的，所以先令 paths[8]=6，然后跳到城市 0。

跳到城市 0 之后，发现 paths[0]==-2，说明城市 0 计算过了，向前跳停止，跳回城市 8。

跳回城市 8 之后，根据之前的设置(paths[8]==6)，知道城市 8 下一步应该跳回城市 6，依然与步骤 1 的情况一样，通过之前 paths 数组的反指找到回去的路径。先设置 paths[8]=-3，然后跳回城市 6。

跳回城市 6 之后，根据之前的设置(paths[6]==-1)，我们知道城市 6 是整个过程的发起城市，所以不需要再回跳，设置 paths[6]=-4，此时 paths=[-2,1,-4,-2,-3,-4,-4,9,-3,-1]

8．遍历到位置 7，paths[7]==9，先令 paths[7]=-1，因为城市 7 指向城市 9，跳到城市 9。

跳到城市 9 之后，发现 paths[9]==-1，说明城市 9 之前已经计算过与首都的距离，所以向前跳的过程停止，跳回城市 7。

跳回到城市 7 之后，根据之前的设置(paths[7]==-1)，我们知道城市 7 是整个过程的发起城市，所以不需要再回跳，设置 paths[7]=-2（因为之前 paths[9]==-1），此时 paths=[-2,1,-4,-2,-3,-4,-4,-2,-3,-1]

9．位置 8 和位置 9 都已经是负数，所以可知之前已经计算过，所以不用调整，遍历结束。

10．根据步骤 2 的 cap 变量，可知首都是城市 1，所以单独设置 paths[1]=0，此时 paths=[-2,0,-4,-2,-3,-4,-4,-2,-3,-1]。

paths 数组转换为距离数组的详细过程请参看如下代码中的 pathsToDistans 方法。

```
public void pathsToDistans(int[] paths) {
    int cap = 0;
    for (int i = 0; i != paths.length; i++) {
        if (paths[i] == i) {
            cap = i;
        } else if (paths[i] > -1) {
            int curI = paths[i];
            paths[i] = -1;
            int preI = i;
            while (paths[curI] != curI) {
                if (paths[curI] > -1) {
                    int nextI = paths[curI];
                    paths[curI] = preI;
                    preI = curI;
                    curI = nextI;
                } else {
                    break;
                }
            }
            int value = paths[curI] == curI ? 0 : paths[curI];
            while (paths[preI] != -1) {
                int lastPreI = paths[preI];
                paths[preI] = --value;
                curI = preI;
                preI = lastPreI;
            }
            paths[preI] = --value;
        }
    }
    paths[cap] = 0;
}
```

paths 变成了距离数组，数组中的距离值都用负数表示，接下来进行第二步，将 paths 转换为我们最终想要的统计数组的过程，即 paths=[-2,0,-4,-2,-3,-4,-4,-2,-3,-1]需要变为[1,1,3,2,3,0,0,0,0,0]。转换过程如下：

1．从左到右遍历 paths，遍历到位置 0，paths[0]==-2，说明距离为 2 的城市发现了 1 座。先把 paths[0]设置为 0，表示 paths[0]的值已经不表示城市 0 与首都的距离，表示以后

可以用来统计距离为 0 的城市数量。

因为距离为 2 的城市发现了 1 座，所以应该设置 paths[2]=1，说明此时 paths[2]开始表示距离 2 的城市数量，而不再是城市 2 与首都的距离。

但在设置 paths[2]时发现 paths[2]==-4，说明 paths[2]在改变它的意义之前，还代表城市 2 与首都的距离为 4，所以先设置 paths[2]=1，然后设置 paths[4]的值，因为距离 4 的城市又发现了 1 座。

但在设置 paths[4]时发现 paths[4]==-3，依然说明 paths[4]在改变它的意义之前，还代表城市 4 与首都的距离为 3，所以先设置 paths[4]=1，然后设置 paths[3]的值，因为距离 3 的城市又发现了 1 座。

但在设置 paths[3]时发现 paths[3]==-2，依然说明 paths[3]在改变它的意义之前，还代表城市 3 与首都的距离为 2，所以先设置 paths[3]=1，然后设置 paths[2]的值，因为距离 2 的城市又发现了 1 座。

此时 paths={0,0,1,1,1,-4,-4,-2,-3,-1}，所以在设置 paths[2]时发现 paths[2]==1，值已经为正数，说明 paths[2]的意义已经不代表城市 2 与首都的距离，而完全是距离为 2 的城市数量统计，所以直接令 paths[2]++，跳的过程停止，此时 paths=[0,0,2,1,1,-4,-4,-2,-3,-1]。

2．遍历到位置 1，paths[1]==0，如果是正值，可以直接忽略，因为意义已经变成城市数量统计。这里值是 0，我们也忽略，因为一张图上距离为 0 的城市只有首都，所以等全部过程完毕后单独设置距离为 0 的城市数量。

3．位置 2~4 上值已经为正数，一律忽略。

4．遍历到位置 5，paths[5]==-4，说明距离为 4 的城市发现了 1 座。先把 paths[5]设置为 0，表示 paths[5]的值已经不表示城市 5 与首都的距离，表示以后可以用来统计距离为 5 的城市数量。此时发现 paths[4]==1，说明不需要跳，直接进行 paths[4]++操作，过程停止。此时 paths=[0,0,2,1,2,0,-4,-2,-3,-1]。

5．遍历位置 6~8，过程与步骤 4 基本相同，处理后 paths=[0,1,3,2,3,0,0,0,0,0]。

6．单独设置 paths[0]==1，因为距离为 0 的城市只有首都。

此时可以说明为什么生成距离数组的时候要把值都弄成负数，因为可以标记状态来让转换成统计数组的过程变得更加顺利。距离数组转换为统计数组的过程请参看如下代码中的 distansToNums 方法。

```
public void distansToNums(int[] disArr) {
        for (int i = 0; i != disArr.length; i++) {
                int index = disArr[i];
                if (index < 0) {
```

```
                        disArr[i] = 0; // 重要
                        while (true) {
                                index = -index;
                                if (disArr[index] > -1) {
                                        disArr[index]++;
                                        break;
                                } else {
                                        int nextIndex = disArr[index];
                                        disArr[index] = 1;
                                        index = nextIndex;
                                }
                        }
                }
        }
        disArr[0] = 1;
}
```

paths 转成距离数组的过程中，每一个城市只经历跳出去和跳回来两个过程，距离数组转成统计数组的过程也是如此，所以时间复杂度为 $O(N)$，整个过程没有使用额外的数据结构，只使用了有限几个变量，所以额外空间复杂度为 $O(1)$。全部过程请参看如下代码中的 pathsToNums 方法，这也是主方法。

```
public void pathsToNums(int[] paths) {
        if (paths == null || paths.length == 0) {
                return;
        }
        // citiesPath -> distancesArray
        pathsToDistans(paths);

        // distancesArray -> numArray
        distansToNums(paths);
}
```

正数数组的最小不可组成和

【题目】

给定一个正数数组 arr，其中所有的值都为整数，以下是最小不可组成和的概念：

- 把 arr 每个子集内的所有元素加起来会出现很多值，其中最小的记为 min，最大的记为 max。
- 在区间[min,max]上，如果有数不可以被 arr 某一个子集相加得到，那么其中最小的那个数是 arr 的最小不可组成和。
- 在区间[min,max]上，如果所有的数都可以被 arr 的某一个子集相加得到，那么