

所以时间复杂度就是  $O(N)$ ，具体过程请参看如下代码中的 `sort` 方法。

```
public void sort(int[] arr) {
    if (arr == null || arr.length < 2) {
        return;
    }
    int left = -1;
    int index = 0;
    int right = arr.length;
    while (index < right) {
        if (arr[index] == 0) {
            swap(arr, ++left, index++);
        } else if (arr[index] == 2) {
            swap(arr, index, --right);
        } else {
            index++;
        }
    }
}
```

## 求最短通路值

### 【题目】

用一个整型矩阵 `matrix` 表示一个网络，1 代表有路，0 代表无路，每一个位置只要不越界，都有上下左右 4 个方向，求从最左上角到最右下角的最短通路值。

例如，`matrix` 为：

```
1  0  1  1  1
1  0  1  0  1
1  1  1  0  1
0  0  0  0  1
```

通路只有一条，由 12 个 1 构成，所以返回 12。

### 【难度】

尉 ★★☆☆

### 【解答】

使用宽度优先遍历即可，如果矩阵大小为  $N \times M$ ，本文提供的方法的时间复杂度为  $O(N \times M)$ ，具体过程如下：

1. 开始时生成 `map` 矩阵, `map[i][j]` 的含义是从  $(0,0)$  位置走到  $(i,j)$  位置最短的路径值。然后将左上角位置  $(0,0)$  的行坐标与列坐标放入行队列 `rQ`, 和列队列 `cQ`。
2. 不断从队列弹出一个位置  $(r,c)$ , 然后看这个位置的上下左右四个位置哪些在 `matrix` 上的值是 1, 这些都是能走的位置。
3. 将那些能走的位置设置好各自在 `map` 中的值, 即 `map[r][c]+1`。同时将这些位置加入到 `rQ` 和 `cQ` 中, 用队列完成宽度优先遍历。
4. 在步骤 3 中, 如果一个位置之前走过, 就不要重复走, 这个逻辑可以根据一个位置在 `map` 中的值来确定, 比如 `map[i][j]!=0`, 就可以知道这个位置之前已经走过。
5. 一直重复步骤 2~步骤 4。直到遇到右下角位置, 说明已经找到终点, 返回终点在 `map` 中的值即可, 如果 `rQ` 和 `cQ` 已经为空都没有遇到终点位置, 说明不存在这样一条路径, 返回 0。

每个位置最多走一遍, 所以时间复杂度为  $O(N \times M)$ 、额外空间复杂度也是  $O(N \times M)$ 。具体过程请参看如下代码中的 `minPathValue` 方法。

```
public int minPathValue(int[][] m) {
    if (m == null || m.length == 0 || m[0].length == 0 || m[0][0] != 1
        || m[m.length - 1][m[0].length - 1] != 1) {
        return 0;
    }
    int res = 0;
    int[][] map = new int[m.length][m[0].length];
    map[0][0] = 1;
    Queue<Integer> rQ = new LinkedList<Integer>();
    Queue<Integer> cQ = new LinkedList<Integer>();
    rQ.add(0);
    cQ.add(0);
    int r = 0;
    int c = 0;
    while (!rQ.isEmpty()) {
        r = rQ.poll();
        c = cQ.poll();
        if (r == m.length - 1 && c == m[0].length - 1) {
            return map[r][c];
        }
        walkTo(map[r][c], r - 1, c, m, map, rQ, cQ); // up
        walkTo(map[r][c], r + 1, c, m, map, rQ, cQ); // down
        walkTo(map[r][c], r, c - 1, m, map, rQ, cQ); // left
        walkTo(map[r][c], r, c + 1, m, map, rQ, cQ); // right
    }
    return res;
}

public void walkTo(int pre, int toR, int toC, int[][] m,
```

```

        int[][] map, Queue<Integer> rQ, Queue<Integer> cQ) {
    if (toR < 0 || toR == m.length || toC < 0 || toC == m[0].length
        || m[toR][toC] != 1 || map[toR][toC] != 0) {
        return;
    }
    map[toR][toC] = pre + 1;
    rQ.add(toR);
    cQ.add(toC);
}

```

## 数组中未出现的最小正整数

### 【题目】

给定一个无序整型数组 `arr`，找到数组中未出现的最小正整数。

### 【举例】

`arr=[-1,2,3,4]`。返回 1。

`arr=[1,2,3,4]`。返回 5。

### 【难度】

尉 ★★☆☆

### 【解答】

原问题。如果 `arr` 长度为  $N$ ，本题的最优解可以做到时间复杂度为  $O(N)$ ，额外空间复杂度为  $O(1)$ 。具体过程如下：

1. 在遍历 `arr` 之前先生成两个变量。变量  $l$  表示遍历到目前为止，数组 `arr` 已经包含的正整数范围是  $[1, l]$ ，所以没有开始遍历之前令  $l=0$ ，表示 `arr` 目前没有包含任何正整数。变量  $r$  表示遍历到目前为止，在后续出现最优状况的情况下，`arr` 可能包含的正整数范围是  $[1, r]$ ，所以没有开始遍历之前，令  $r=N$ ，因为还没有开始遍历，所以后续出现的最优状况是 `arr` 包含  $1 \sim N$  所有的整数。 $r$  同时表示 `arr` 当前的结束位置。

2. 从左到右遍历 `arr`，遍历到位置  $l$ ，位置  $l$  的数为 `arr[l]`。

3. 如果 `arr[l]==l+1`。没有遍历 `arr[l]` 之前，`arr` 已经包含的正整数范围是  $[1, l]$ ，此时出现了 `arr[l]==l+1` 的情况，所以 `arr` 包含的正整数范围可以扩大到  $[1, l+1]$ ，即令  $l++$ 。然后重复步骤 2。