

【解答】

具体过程如下：

1. 整型变量 `jump`，代表目前跳了多少步。整型变量 `cur`，代表如果只能跳 `jump` 步，最远能够达到的位置。整型变量 `next`，代表如果再多跳一步，最远能够达到的位置。初始时，`jump=0`，`cur=0`，`next=0`。

2. 从左到右遍历 `arr`，假设遍历到位置 `i`。

1) 如果 `cur>=i`，说明跳 `jump` 步可以到达位置 `i`，此时什么也不做。

2) 如果 `cur<i`，说明只跳 `jump` 步不能到达位置 `i`，需要多跳一步才行。此时令 `jump++`，`cur=next`。表示多跳了一步，`cur` 更新成跳 `jump+1` 步能够达到的位置，即 `next`。

3) 将 `next` 更新成 `math.max(next, i+arr[i])`，表示下一次多跳一步到达的最远位置。

3. 最终返回 `jump` 即可。

具体过程请参看如下代码中的 `jump` 方法。

```
public int jump(int[] arr) {
    if (arr == null || arr.length == 0) {
        return 0;
    }
    int jump = 0;
    int cur = 0;
    int next = 0;
    for (int i = 0; i < arr.length; i++) {
        if (cur < i) {
            jump++;
            cur = next;
        }
        next = Math.max(next, i + arr[i]);
    }
    return jump;
}
```

数组中的最长连续序列

【题目】

给定无序数组 `arr`，返回其中最长的连续序列的长度。

【举例】

`arr=[100,4,200,1,3,2]`，最长的连续序列为`[1,2,3,4]`，所以返回 4。

【难度】

尉 ★★☆☆

【解答】

本题利用哈希表可以实现时间复杂度为 $O(N)$ 、额外空间复杂度为 $O(N)$ 的方法。具体过程如下：

1. 生成哈希表 `HashMap<Integer, Integer> map`，key 代表遍历过的某个数，value 代表 key 这个数所在的最长连续序列的长度。同时 map 还可以表示 arr 中的一个数之前是否出现过。

2. 从左到右遍历 arr，假设遍历到 `arr[i]`。如果 `arr[i]` 之前出现过，直接遍历下一个数，只处理之前没出现过的 `arr[i]`。首先在 map 中加入记录 `(arr[i], 1)`，代表目前 `arr[i]` 单独作为一个连续序列。然后看 map 中是否含有 `arr[i]-1`，如果有，则说明 `arr[i]-1` 所在的连续序列可以和 `arr[i]` 合并，合并后记为 A 序列。利用 map 可以得到 A 序列的长度，记为 `lenA`，最小值记为 `leftA`，最大值记为 `rightA`，只在 map 中更新与 `leftA` 和 `rightA` 有关的记录，更新成 `(leftA, lenA)` 和 `(rightA, lenA)`。接下来看 map 中是否含有 `arr[i]+1`，如果有，则说明 `arr[i]+1` 所在的连续序列可以和 A 合并，合并后记为 B 序列。利用 map 可以得到 B 序列的长度为 `lenB`，最小值记为 `leftB`，最大值记为 `rightB`，只在 map 中更新与 `leftB` 和 `rightB` 有关的记录，更新成 `(leftB, lenB)` 和 `(rightB, lenB)`。

3. 遍历的过程中用全局变量 `max` 记录每次合并出的序列的长度最大值，最后返回 `max`。

整个过程中，只是每个连续序列最小值和最大值在 map 中的记录有意义，中间数的记录不再更新，因为再也不会使用到。这是因为我们只处理之前没出现的数，如果一个没出现的数能够把某个连续区间扩大，或把某两个连续区间连在一起，毫无疑问，只需要 map 中有关这个连续区间最小值和最大值的记录。

具体过程请参看如下代码中的 `longestConsecutive` 方法。

```
public int longestConsecutive(int[] arr) {
    if (arr == null || arr.length == 0) {
        return 0;
    }
    int max = 1;
    HashMap<Integer, Integer> map = new HashMap<Integer, Integer>();
    for (int i = 0; i < arr.length; i++) {
        if (!map.containsKey(arr[i])) {
            map.put(arr[i], 1);
            if (map.containsKey(arr[i] - 1)) {
                max = Math.max(max, merge(map, arr[i] - 1, arr[i]));
            }
        }
    }
}
```

```

        if (map.containsKey(arr[i] + 1)) {
            max = Math.max(max, merge(map, arr[i], arr[i] + 1));
        }
    }
    return max;
}

public int merge(HashMap<Integer, Integer> map, int less, int more) {
    int left = less - map.get(less) + 1;
    int right = more + map.get(more) - 1;
    int len = right - left + 1;
    map.put(left, len);
    map.put(right, len);
    return len;
}

```

N 皇后问题

【题目】

N 皇后问题是指在 $N \times N$ 的棋盘上要摆 N 个皇后，要求任何两个皇后不同行、不同列，也不在同一条斜线上。给定一个整数 n ，返回 n 皇后的摆法有多少种。

【举例】

$n=1$ ，返回 1。

$n=2$ 或 3，2 皇后和 3 皇后问题无论怎么摆都不行，返回 0。

$n=8$ ，返回 92。

【难度】

校 ★★★★★

【解答】

本题是非常著名的问题，甚至可以用人工智能相关算法和遗传算法进行求解，同时可以用多线程技术达到缩短运行时间的效果。本书不涉及专项算法，仅提供在面试过程中 10 至 20 分钟内可以用代码实现的解法。本书提供的最优解做到在单线程的情况下，计算 16 皇后问题的运行时间约为 13 秒左右。在介绍最优解之前，先来介绍一个容易理解的解法。

如果在 (i, j) 位置(第 i 行第 j 列)放置了一个皇后，接下来在哪些位置不能放置皇后呢？