```
                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×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？

1．整个第 i 行的位置都不能放置。

2．整个第 j 列的位置都不能放置。

3．如果位置(a,b)满足|a-i|==|b-j|，说明(a,b)与(i,j)处在同一条斜线上，也不能放置。

把递归过程直接设计成逐行放置皇后的方式，可以避开条件 1 的那些不能放置的位置。接下来用一个数组保存已经放置的皇后位置，假设数组为 record，record[i]的值表示第 i 行皇后所在的列数。在递归计算到第 i 行第 j 列时，查看 record[0..k]($k<i$)的值，看是否有 j 相等的值，若有，则说明(i,j)不能放置皇后，再看是否有|k-i|==|record[k]-j|，若有，也说明(i,j)不能放置皇后。具体过程请参看如下代码中的 num1 方法。

```
public int num1(int n) {
        if (n < 1) {
                return 0;
        }
        int[] record = new int[n];
        return process1(0, record, n);
}

public int process1(int i, int[] record, int n) {
        if (i == n) {
                return 1;
        }
        int res = 0;
        for (int j = 0; j < n; j++) {
                if (isValid(record, i, j)) {
                        record[i] = j;
                        res += process1(i + 1, record, n);
                }
        }
        return res;
}

public boolean isValid(int[] record, int i, int j) {
    for (int k = 0; k < i; k++) {
        if (j == record[k] || Math.abs(record[k] - j) == Math.abs(i - k)) {
                return false;
        }
    }
    return true;
}
```

下面介绍最优解，基本过程与上面的方法一样，但使用了位运算来加速。具体加速的递归过程中，找到每一行还有哪些位置可以放置皇后的判断过程。因为整个过程比较超自然，所以先列出代码，然后对代码进行解释，请参看如下代码中的 num2 方法。

```
public int num2(int n) {
```

```
        // 因为本方法中位运算的载体是 int 型变量，所以该方法只能算 1~32 皇后问题
        // 如果想计算更多的皇后问题，需使用包含更多位的变量
        if (n < 1 || n > 32) {
                return 0;
        }
        int upperLim = n == 32 ? -1 : (1 << n) - 1;
        return process2(upperLim, 0, 0, 0);
}

public int process2(int upperLim, int colLim, int leftDiaLim,
                int rightDiaLim) {
        if (colLim == upperLim) {
                return 1;
        }
        int pos = 0;
        int mostRightOne = 0;
        pos = upperLim & (~(colLim | leftDiaLim | rightDiaLim));
        int res = 0;
        while (pos != 0) {
                mostRightOne = pos & (~pos + 1);
                pos = pos - mostRightOne;
                res += process2(upperLim, colLim | mostRightOne,
                                (leftDiaLim | mostRightOne) << 1,
                                (rightDiaLim | mostRightOne) >>> 1);
        }
        return res;
}
```

num2 方法中，变量 upperLim 表示当前行哪些位置是可以放置皇后的，1 代表可以放置，0 代表不能放置。8 皇后问题中，初始时 upperLim 为 00000000000000000000000011111111，即 32 位整数的 255。32 皇后问题中，初始时 upperLim 为 11111111111111111111111111111111，即 32 位整数的-1。

接下来解释一下 process2 方法，先介绍每个参数。

- upperLim：已经解释过了，而且这个变量的值在递归过程中是始终不变的。
- colLim：表示递归计算到上一行为止，在哪些列上已经放置了皇后，1 代表已经放置，0 代表没有放置。
- leftDiaLim：表示递归计算到上一行为止，因为受已经放置的所有皇后的左下方斜线的影响，导致当前行不能放置皇后，1 代表不能放置，0 代表可以放置。举个例子，如果在第 0 行第 4 列放置了皇后。计算到第 1 行时，第 0 行皇后的左下方斜线影响的是第 1 行第 3 列。当计算到第 2 行时，第 0 行皇后的左下方斜线影响的是第 2 行第 2 列。当计算到第 3 行时，影响的是第 3 行第 1 列。当计算到第 4 行时，影响的是第 4 行第 0 列。当计算到第 5 行时，第 0 行的那个皇后的左下方斜

线对第 5 行无影响，并且之后的行都不再受第 0 行皇后左下方斜线的影响。也就是说，leftDiaLim 每次左移一位，就可以得到之前所有皇后的左下方斜线对当前行的影响。

- rightDiaLim：表示递归计算到上一行为止，因为已经放置的所有皇后的右下方斜线的影响，导致当前行不能放置皇后的位置，1 代表不能放置，0 代表可以放置。与 leftDiaLim 变量类似，rightDiaLim 每次右移一位就可以得到之前所有皇后的右下方斜线对当前行的影响。

process2 方法的返回值代表剩余的皇后在之前皇后的影响下，有多少种合法的摆法。其中，变量 pos 代表当前行在 colLim、leftDiaLim 和 rightDiaLim 这三个状态的影响下，还有哪些位置是可供选择的，1 代表可以选择，0 代表不能选择。变量 mostRightOne 代表在 pos 中，最右边的 1 是在什么位置。然后从右到左依次筛选出 pos 中可选择的位置进行递归尝试。