## 面试题 01.08. 零矩阵

```
func setZeroes(matrix [][]int)  {
    n := len(matrix)
    if n == 0 {return}
    m := len(matrix[0])
    //哪一行，哪一列要清空为 0
    zeroRows := make([]bool, n)
    zeroColumns := make([]bool, m)
    for i := 0; i < n; i++ {
        for j := 0; j < m; j++ {
            if matrix[i][j] == 0 {
                zeroRows[i] = true
                zeroColumns[j] = true
            }
        }
    }
    //设置为 0
    for i := 0; i < n; i++ {
        for j := 0; j < m; j++ {
            if zeroRows[i] || zeroColumns[j] {
                matrix[i][j] = 0
            }
        }
    }
}
```

## 剑指 Offer 61. 扑克牌中的顺子

```
func isStraight(nums []int) bool {
    dup := make([]bool, 14) //去重用的
    min := 100
    max := -1
    for i := 0; i < 5; i++ {
        if nums[i] != 0 {
```

```
            if dup[nums[i]] {
                return false
            } else {
                dup[nums[i]] = true
            }

            if nums[i] < min {min = nums[i]}
            if nums[i] > max {max = nums[i]}
        }
    }
    return (max-min) < 5
}
```

## 面试题 16.11. 跳水板

```
func divingBoard(shorter int, longer int, k int) []int {
    //特殊情况处理
    if k == 0 {return []int{}}
    if shorter == longer {return []int{k*shorter}}

    result := make([]int, k+1)
    //长板子个数：0,1,2...k
    for i := 0; i <= k; i++ {
        result[i] = i * longer + (k-i) * shorter
    }
    return result
}
```

## 面试题 01.05. 一次编辑

```
func oneEditAway(first string, second string) bool {
    n := len(first)
    m := len(second)
    if math.Abs(float64(n-m))>1 {return false} //长度相差大于 1，无法通过一次编辑匹配

    i := 0
    j := 0
```

```
    //碰到第一个不能匹配的字符
    for i < n && j < m && first[i] == second[j] {
        i++
        j++
    }
    if n == m { //替换 abdf  abcf
        i++
        j++
    } else if n > m { //删除或插入 abcf  abf
        i++
    } else { //m>n 删除或插入
        j++
    }
    //继续考察后面的元素，必须完全匹配
    for i < n && j < m {
        if first[i] != second[j] {
            return false
        }
        i++
        j++
    }
    return true
}
```

## 面试题 16.15. 珠玑妙算

```
func masterMind(solution string, guess string) []int {
    n := len(solution)
    hited := make([]bool, n) //guess 中哪些字符已经猜中了
    used := make([]bool, n) //solution 中哪些字符已经被匹配用掉了
    //先计算猜中的
    hitCount := 0
    for i := 0; i < n; i++ {
        if solution[i] == guess[i] {
            hited[i] = true
            used[i] = true
            hitCount++
```

```
        }
    }
    //再计算伪猜中的
    fakeHitCount := 0
    for i := 0; i < n; i++ {
        if hited[i] {continue}
        //拿每个 guess 中的字符到 solution 中查找
        for j := 0; j < n; j++ {
            if solution[j] == guess[i] && !used[j] {
                used[j] = true
                fakeHitCount++
                break
            }
        }
    }
    return []int{hitCount, fakeHitCount}
}
```

## 面试题 16.04. 井字游戏(中等)

```
func tictactoe(board []string) string {
    n := len(board)
    boards := make([][]byte, n)

    for i := 0; i < len(boards); i++ {
        boards[i] = make([]byte, n)
        boards[i] = []byte(board[i])
    }

    determined := false //表示是否已经发现有人赢了
    //检查行
    for i := 0; i < n; i++ {
        if boards[i][0] == ' ' {continue}
        determined = true
        for j := 1; j < n; j++ {
            if boards[i][j] != boards[i][0] {
                determined = false
```

```
            break
        }
    }
    if determined {return string(boards[i][0])}
}
//检查列
for j := 0; j < n; j++ {
    if boards[0][j] == ' ' {continue}
    determined = true
    for i := 1; i < n; i++ {
        if boards[i][j] != boards[0][j] {
            determined = false
        }
    }
    if determined {return string(boards[0][j])}
}
//检查对角线：左上->右下
if boards[0][0] != ' ' {
    i := 1
    j := 1
    determined = true
    for i < n && j < n {
        if boards[i][j] != boards[0][0] {
            determined = false
            break
        }
        i++
        j++
    }
    if determined {return string(boards[0][0])}
}
//检查对角线：左下->右上
if boards[n-1][0] != ' ' {
    i := n-2
    j := 1
    determined = true
    for i >= 0 && j < n {
```

```
            if board[i][j] != boards[n-1][0] {
                determined = false
                break
            }
            i--
            j++
        }
        if determined {return string(boards[n-1][0])}
    }
    //上面没有找到哪方赢， 判断游戏是否还能继续玩
    for i := 0; i < n; i++ {
        for j := 0; j < n ; j++ {
            if boards[i][j] == ' ' {return "Pending"}
        }
    }
    //游戏结束了，平局
    return "Draw"
}
```

## 55. 跳跃游戏

```
func canJump(nums []int) bool {
    reachedMax := 0
    for i := 0; i < len(nums); i++ {
        if i > reachedMax {return false} //表示前边任意一个走法都到达不了 i
        if i + nums[i] > reachedMax {
            reachedMax = i + nums[i]
        }
        if reachedMax >= len(nums)-1 {return true}
    }
    return false
}
```

## 48. 旋转图像

```go
//解法 1 借助辅助数组
func rotate(matrix [][]int)  {
    n := len(matrix)
    tmp := make([][]int, n)
    for i := 0; i < n; i++ {
        tmp[i] = make([]int, n)
    }
    for i := 0; i < n; i++ {
        for j := 0; j < n; j++ {
            tmp[j][n-i-1] = matrix[i][j]
        }
    }
    for i := 0; i < n; i++ {
        for j := 0; j < n; j++ {
            matrix[i][j] = tmp[i][j]
        }
    }
}

//解法 2 用翻转代替旋转
func rotate(matrix [][]int)  {
    n := len(matrix)
    //先上下翻转
    for i := 0; i < n/2; i++ {
        for j := 0; j < n; j++ {
            swap(matrix, i, j, n-i-1, j)
        }
    }
    //再对角翻转（左上-右下）
    for i := 0; i < n; i++ {
        for j := 0; j < i; j++ {
            swap(matrix, i, j, j, i)
        }
    }
}

func swap (matrix [][]int, i, j, p, q int) {
```

```
    matrix[i][j], matrix[p][q] = matrix[p][q], matrix[i][j]
}

//解法 3 标准原地旋转
func rotate(matrix [][]int)  {
    n := len(matrix)
    s1_i := 0
    s1_j := 0
    for n > 1 {
        s2_i := s1_i
        s2_j := s1_j + n-1
        s3_i := s1_i + n-1
        s3_j := s1_j + n-1
        s4_i := s1_i + n-1
        s4_j := s1_j

        for move := 0; move<=n-2; move++ {
            p1_i := s1_i + 0
            p1_j := s1_j + move
            p2_i := s2_i + move
            p2_j := s2_j + 0
            p3_i := s3_i + 0
            p3_j := s3_j - move
            p4_i := s4_i - move
            p4_j := s4_j + 0
            swap4(matrix, p1_i, p1_j, p2_i, p2_j, p3_i, p3_j, p4_i, p4_j)
        }
        s1_i++
        s1_j++
        n-=2
    }
}

func swap4(a [][]int, i1, j1, i2, j2, i3, j3, i4, j4 int) {
    tmp := a[i1][j1]
    a[i1][j1] = a[i4][j4]
    a[i4][j4] = a[i3][j3]
```

```
    a[i3][j3] = a[i2][j2]
    a[i2][j2] = tmp
}
```

## 54. 螺旋矩阵

```
func spiralOrder(matrix [][]int) []int {
    m := len(matrix)
    n := len(matrix[0])
    result := make([]int, 0)
    left := 0
    right := n-1
    top := 0
    bottom := m-1
    for left <= right && top <= bottom {
        for j := left; j <= right; j++ {
            result = append(result, matrix[top][j])
        }
        for i := top+1; i <= bottom; i++ {
            result = append(result, matrix[i][right])
        }
        if top != bottom {
            for j := right-1; j >= left; j-- {
                result = append(result, matrix[bottom][j])
            }
        }
        if left != right {
            for i := bottom-1; i > top; i-- {
                result = append(result, matrix[i][left])
            }
        }
        left++
        right--
        top++
        bottom--
    }
```

```
    return result
}
```

## 240. 搜索二维矩阵 II

```
func searchMatrix(matrix [][]int, target int) bool {
    h := len(matrix)
    w := len(matrix[0])
    i := 0
    j := w-1
    //根据 matrix[i][j]跟 target 的大小关系，一层一层的剥离
    for i <= h-1 && j >= 0 {
        if matrix[i][j] == target {
            return true
        }
        if matrix[i][j] > target {
            j--
            continue
        }
        if matrix[i][j] < target {
            i++
            continue
        }
    }
    return false
}
```