

0101-0200

0101-0200

- 101. 对称二叉树
 - 题目
 - 解答思路
- 104. 二叉树的最大深度
 - 题目
 - 解答思路
- 107. 二叉树的层次遍历II
 - 题目
 - 解题思路
- 108. 将有序数组转换为二叉搜索树
 - 题目
 - 解题思路
- 110. 平衡二叉树(缺少多种思路)
 - 题目
 - 解题思路
- 111. 二叉树的最小深度
 - 题目
 - 解题思路
- 112. 路径总和
 - 题目
 - 解题思路
- 118. 杨辉三角
 - 题目
 - 解题思路
- 119. 杨辉三角II
 - 题目
 - 解题思路
- 121. 买卖股票的最佳时机
 - 题目
 - 解题思路

101. 对称二叉树

题目

- 1

给定一个二叉树，检查它是否是镜像对称的。
- 2

例如，二叉树 `[1,2,2,3,4,4,3]` 是对称的。
- 3

1
- 4

/ \
- 5

2 2

```

6      / \ / \
7      3  4 4  3
8
9      但是下面这个 [1,2,2,null,3,null,3] 则不是镜像对称的：
10     1
11    / \
12   2   2
13  \   \
14   3   3
15
16  说明：
17  如果你可以运用递归和迭代两种方法解决这个问题，会很加分。

```

解答思路

No.	思路	时间复杂度	空间复杂度
01(最优)	递归	O(n)	O(n)
02	迭代	O(n)	O(n)

```

1  // 递归
2  func isSymmetric(root *TreeNode) bool {
3      if root == nil {
4          return true
5      }
6      return recur(root.Left, root.Right)
7  }
8
9  func recur(left, right *TreeNode) bool {
10     if left == nil && right == nil {
11         return true
12     }
13     if left == nil || right == nil {
14         return false
15     }
16
17     return left.Val == right.Val &&
18         recur(left.Left, right.Right) &&
19         recur(left.Right, right.Left)
20 }
21 // 迭代
22 func isSymmetric(root *TreeNode) bool {
23     leftQ := make([]*TreeNode, 0)
24     rightQ := make([]*TreeNode, 0)
25     leftQ = append(leftQ, root)
26     rightQ = append(rightQ, root)
27

```

```

28     for len(leftQ) != 0 && len(rightQ) != 0 {
29         leftCur, rightCur := leftQ[0], rightQ[0]
30         leftQ, rightQ = leftQ[1:], rightQ[1:]
31
32         if leftCur == nil && rightCur == nil {
33             continue
34         } else if leftCur != nil && rightCur != nil && leftCur.Val ==
rightCur.Val {
35             leftQ = append(leftQ, leftCur.Left, leftCur.Right)
36             rightQ = append(rightQ, rightCur.Right, rightCur.Left)
37         } else {
38             return false
39         }
40     }
41
42     if len(leftQ) == 0 && len(rightQ) == 0 {
43         return true
44     } else {
45         return false
46     }
47 }

```

104.二叉树的最大深度

题目

No.	思路	时间复杂度	空间复杂度
01(最优)	递归	O(n)	O(log(n))
02	迭代	O(n)	O(n)

解答思路

```

1 // 递归
2 func maxDepth(root *TreeNode) int {
3     if root == nil {
4         return 0
5     }
6     left := maxDepth(root.Left)
7     right := maxDepth(root.Right)
8
9     return max(left, right) + 1
10 }
11
12 func max(a, b int) int {
13     if a > b {

```

```

14     return a
15 }
16 return b
17 }
18
19
20 // 迭代
21 func maxDepth(root *TreeNode) int {
22     if root == nil {
23         return 0
24     }
25     queue := make([]*TreeNode, 0)
26     queue = append(queue, root)
27     depth := 0
28
29     for len(queue) > 0 {
30         length := len(queue)
31
32         for i := 0; i < length; i++ {
33             node := queue[0]
34             queue = queue[1:]
35             if node.Left != nil {
36                 queue = append(queue, node.Left)
37             }
38             if node.Right != nil {
39                 queue = append(queue, node.Right)
40             }
41         }
42         depth++
43     }
44     return depth
45 }

```

107. 二叉树的层次遍历II

题目

```

1  给定一个二叉树，返回其节点值自底向上的层次遍历。（即按从叶子节点所在层到根节点所在的层，
   逐层从左向右遍历）
2
3  例如：
4  给定二叉树 [3,9,20,null,null,15,7],
5      3
6     / \
7    9  20
8   /  \
9  15   7
10

```

```

11  返回其自底向上的层次遍历为：
12  [
13    [15,7],
14    [9,20],
15    [3]
16  ]

```

解题思路

No.	思路	时间复杂度	空间复杂度
01(最优)	递归	O(n)	O(n)
02	迭代	O(n)	O(n)

```

1  // 迭代
2  func levelOrderBottom(root *TreeNode) [][]int {
3      if root == nil {
4          return nil
5      }
6      queue := make([]*TreeNode, 0)
7      out := make([][]int, 0)
8      queue = append(queue, root)
9
10     for len(queue) != 0 {
11         l := len(queue)
12         arr := make([]int, 0)
13         for i := 0; i < l; i++ {
14             pop := queue[i]
15             arr = append(arr, pop.Val)
16             if pop.Left != nil {
17                 queue = append(queue, pop.Left)
18             }
19             if pop.Right != nil {
20                 queue = append(queue, pop.Right)
21             }
22         }
23         out = append(out, arr)
24         queue = queue[l:]
25     }
26
27     out2 := make([][]int, len(out))
28     for i := 0; i < len(out); i++ {
29         out2[len(out)-1-i] = out[i]
30     }
31
32     return out2
33 }

```

```

34
35 // 递归
36 func levelOrderBottom(root *TreeNode) [][]int {
37     result := make([][]int, 0)
38     level := 0
39     if root == nil {
40         return result
41     }
42
43     orderBottom(root, &result, level)
44
45     left, right := 0, len(result)-1
46     for left < right {
47         result[left], result[right] = result[right], result[left]
48         left++
49         right--
50     }
51     return result
52 }
53
54 func orderBottom(root *TreeNode, result *[][]int, level int) {
55     if root == nil {
56         return
57     }
58     if len(*result) > level {
59         fmt.Println(level, result, root.Val)
60         (*result)[level] = append((*result)[level], root.Val)
61     } else {
62         *result = append(*result, []int{root.Val})
63     }
64     orderBottom(root.Left, result, level+1)
65     orderBottom(root.Right, result, level+1)
66 }

```

108.将有序数组转换为二叉搜索树

题目

- 1 将一个按照升序排列的有序数组，转换为一棵高度平衡二叉搜索树。
- 2
- 3 本题中，一个高度平衡二叉树是指一个二叉树每个节点 的左右两个子树的高度差的绝对值不超过 1。
- 4
- 5 示例：
- 6
- 7 给定有序数组：[-10,-3,0,5,9]，
- 8
- 9 一个可能的答案是：[0,-3,9,-10,null,5]，它可以表示下面这个高度平衡二叉搜索树：

```

10
11      0
12     / \
13    -3  9
14   /  /
15  -10 5

```

解题思路

No.	思路	时间复杂度	空间复杂度
01(最优)	递归	O(n)	O(log(n))
02	迭代	O(n)	O(n)

```

1  // 递归
2  func sortedArrayToBST(nums []int) *TreeNode {
3      if len(nums) == 0 {
4          return nil
5      }
6
7      mid := len(nums) / 2
8
9      return &TreeNode{
10         Val:  nums[mid],
11         Left: sortedArrayToBST(nums[:mid]),
12         Right: sortedArrayToBST(nums[mid+1:]),
13     }
14 }
15
16 // 迭代
17 type MyTreeNode struct {
18     root *TreeNode
19     start int
20     end int
21 }
22
23 func sortedArrayToBST(nums []int) *TreeNode {
24     if len(nums) == 0 {
25         return nil
26     }
27
28     queue := make([]MyTreeNode, 0)
29     root := &TreeNode{Val: 0}
30     queue = append(queue, MyTreeNode{root, 0, len(nums)})
31     for len(queue) > 0 {
32         myRoot := queue[0]
33         queue = queue[1:]

```

```

34     start := myRoot.start
35     end := myRoot.end
36     mid := (start + end) / 2
37     curRoot := myRoot.root
38     curRoot.Val = nums[mid]
39     if start < mid {
40         curRoot.Left = &TreeNode{Val: 0}
41         queue = append(queue, MyTreeNode{curRoot.Left, start, mid})
42     }
43     if mid+1 < end {
44         curRoot.Right = &TreeNode{Val: 0}
45         queue = append(queue, MyTreeNode{curRoot.Right, mid + 1, end})
46     }
47 }
48 return root
49 }

```

110.平衡二叉树(缺少多种思路)

题目

```

1  给定一个二叉树，判断它是否是高度平衡的二叉树。
2
3  本题中，一棵高度平衡二叉树定义为：
4
5      一个二叉树每个节点 的左右两个子树的高度差的绝对值不超过1。
6
7  示例 1：
8
9  给定二叉树 [3,9,20,null,null,15,7]
10
11      3
12     / \
13    9  20
14   /  \
15  15   7
16
17  返回 true 。
18
19 示例 2：
20
21 给定二叉树 [1,2,2,3,3,null,null,4,4]
22
23      1
24     / \
25    2   2
26   / \
27  3   3

```



```
28     / \
29    4   4
30
31  返回 false 。
```

解题思路

No.	思路	时间复杂度	空间复杂度
01	递归	O(n)	O(log(n))

```
1  func isBalanced(root *TreeNode) bool {
2      _, isBalanced := recur(root)
3      return isBalanced
4
5  }
6
7  func recur(root *TreeNode) (int, bool) {
8      if root == nil {
9          return 0, true
10     }
11
12     leftDepth, leftIsBalanced := recur(root.Left)
13     if leftIsBalanced == false{
14         return 0,false
15     }
16     rightDepth, rightIsBalanced := recur(root.Right)
17     if rightIsBalanced == false{
18         return 0,false
19     }
20
21     if -1 <= leftDepth-rightDepth &&
22         leftDepth-rightDepth <= 1 {
23         return max(leftDepth, rightDepth) + 1, true
24     }
25     return 0, false
26 }
27
28 func max(a, b int) int {
29     if a > b {
30         return a
31     }
32     return b
33 }
```

111.二叉树的最小深度

题目

```
1  给定一个二叉树，找出其最小深度。
2
3  最小深度是从根节点到最近叶子节点的最短路径上的节点数量。
4  说明：叶子节点是指没有子节点的节点。
5
6  示例：
7  给定二叉树 [3,9,20,null,null,15,7],
8      3
9     /\
10    9 20
11   /\  \
12  15  7
13
14  返回它的最小深度 2.
```

解题思路

No.	思路	时间复杂度	空间复杂度
01(最优)	递归	O(n)	O(log(n))
02	广度优先	O(n)	O(n)

```
1  // 递归
2  func minDepth(root *TreeNode) int {
3      if root == nil {
4          return 0
5      } else if root.Left == nil {
6          return 1 + minDepth(root.Right)
7      } else if root.Right == nil {
8          return 1 + minDepth(root.Left)
9      } else {
10         return 1 + min(minDepth(root.Left), minDepth(root.Right))
11     }
12 }
13
14 func min(a, b int) int {
15     if a < b {
16         return a
17     }
18     return b
19 }
20
21 // 广度优先搜索
22 func minDepth(root *TreeNode) int {
23     if root == nil{
```

```

24     return 0
25 }
26
27 list := make([]*TreeNode,0)
28 list = append(list,root)
29 depth := 1
30
31 for len(list) > 0{
32     length := len(list)
33     for i := 0; i < length; i++){
34         node := list[0]
35         list = list[1:]
36         if node.Left == nil && node.Right == nil{
37             return depth
38         }
39         if node.Left != nil{
40             list = append(list,node.Left)
41         }
42         if node.Right != nil{
43             list = append(list,node.Right)
44         }
45     }
46     depth++
47 }
48 return depth
49 }

```

112.路径总和

题目

- 1 给定一个二叉树和一个目标和，判断该树中是否存在根节点到叶子节点的路径，这条路径上所有节点值相加等于目标和。
- 2 说明：叶子节点是指没有子节点的节点。
- 3 示例：
- 4 给定如下二叉树，以及目标和 $sum = 22$ ，
- 5

5
- 6

/ \
- 7

4 8
- 8

/ / \
- 9

11 13 4
- 10

/ \ \
- 11

7 2 1
- 12 返回 `true`，因为存在目标和为 22 的根节点到叶子节点的路径 $5 \rightarrow 4 \rightarrow 11 \rightarrow 2$ 。

解题思路

No.	思路	时间复杂度	空间复杂度
01(最优)	递归	$O(n)$	$O(\log(n))$
02	迭代	$O(n)$	$O(n)$

```

1 // 递归
2 func hasPathSum(root *TreeNode, sum int) bool {
3     if root == nil {
4         return false
5     }
6     sum = sum - root.Val
7     if root.Left == nil && root.Right == nil {
8         return sum == 0
9     }
10    return hasPathSum(root.Left, sum) || hasPathSum(root.Right, sum)
11 }
12
13 // 迭代
14 func hasPathSum(root *TreeNode, sum int) bool {
15     if root == nil {
16         return false
17     }
18     list1 := list.New()
19     list2 := list.New()
20
21     list1.PushFront(root)
22     list2.PushFront(sum - root.Val)
23     for list1.Len() > 0 {
24         length := list1.Len()
25
26         for i := 0; i < length; i++ {
27             node := list1.Remove(list1.Back()).(*TreeNode)
28             currentSum := list2.Remove(list2.Back()).(int)
29             if node.Left == nil && node.Right == nil && currentSum == 0 {
30                 return true
31             }
32             if node.Left != nil {
33                 list1.PushFront(node.Left)
34                 list2.PushFront(currentSum - node.Left.Val)
35             }
36             if node.Right != nil {
37                 list1.PushFront(node.Right)
38                 list2.PushFront(currentSum - node.Right.Val)
39             }
40         }
41     }
42     return false
43 }

```

118.杨辉三角

题目

```
1  给定一个非负整数 numRows，生成杨辉三角的前 numRows 行。
2  在杨辉三角中，每个数是它左上方和右上方的数的和。
3
4  示例：
5  输入：5
6  输出：
7  [
8      [1],
9      [1,1],
10     [1,2,1],
11     [1,3,3,1],
12     [1,4,6,4,1]
13 ]
```

解题思路

No.	思路	时间复杂度	空间复杂度
01	动态规划	$O(n^2)$	$O(n^2)$
02(最优)	递推	$O(n^2)$	$O(n^2)$

```
1  // 动态规划
2  func generate(numRows int) [][]int {
3      var result [][]int
4      for i := 0; i < numRows; i++ {
5          var row []int
6          for j := 0; j <= i; j++ {
7              tmp := 1
8              if j == 0 || j == i {
9
10             } else {
11                 tmp = result[i-1][j-1] + result[i-1][j]
12             }
13             row = append(row, tmp)
14         }
15         result = append(result, row)
16     }
17     return result
18 }
19
20 // 递推
```

```

21 func generate(numRows int) [][]int {
22     res := make([][]int, 0)
23     if numRows == 0 {
24         return res
25     }
26
27     res = append(res, []int{1})
28     if numRows == 1 {
29         return res
30     }
31
32     for i := 1; i < numRows; i++ {
33         res = append(res, genNext(res[i-1]))
34     }
35     return res
36 }
37
38 func genNext(p []int) []int {
39     res := make([]int, 1, len(p)+1)
40     res = append(res, p...)
41
42     for i := 0; i < len(res)-1; i++ {
43         res[i] = res[i] + res[i+1]
44     }
45     return res
46 }

```

119.杨辉三角II

题目

- 1 给定一个非负索引 k ，其中 $k \leq 33$ ，返回杨辉三角的第 k 行。
- 2 在杨辉三角中，每个数是它左上方和右上方的数的和。
- 3
- 4 示例：
- 5 输入：3
- 6 输出：[1,3,3,1]
- 7
- 8 进阶：
- 9 你可以优化你的算法到 $O(k)$ 空间复杂度吗？

解题思路

No.	思路	时间复杂度	空间复杂度
01	动态规划	$O(n^2)$	$O(n^2)$
02	递推	$O(n^2)$	$O(n)$
03(最优)	二项式定理	$O(n)$	$O(n)$

```

1  // 动态规划
2  func getRow(rowIndex int) []int {
3      var result [][]int
4      for i := 0; i < rowIndex+1; i++ {
5          var row []int
6          for j := 0; j <= i; j++ {
7              tmp := 1
8              if j == 0 || j == i {
9
10             } else {
11                 tmp = result[i-1][j-1] + result[i-1][j]
12             }
13             row = append(row, tmp)
14         }
15         result = append(result, row)
16     }
17     return result[rowIndex]
18 }
19
20 // 递推
21 func getRow(rowIndex int) []int {
22     res := make([]int, 1, rowIndex+1)
23     res[0] = 1
24     if rowIndex == 0 {
25         return res
26     }
27
28     for i := 0; i < rowIndex; i++ {
29         res = append(res, 1)
30         for j := len(res) - 2; j > 0; j-- {
31             res[j] = res[j] + res[j-1]
32         }
33     }
34     return res
35 }
36
37
38 // 二项式定理
39 func getRow(rowIndex int) []int {
40     res := make([]int, rowIndex+1)
41     res[0] = 1

```

```

42     if rowIndex == 0{
43         return res
44     }
45
46     // 公式
47     //  $C(n,k) = n! / (k! * (n-k)!)$ 
48     //  $C(n,k) = (n-k+1)/k * C(n,k-1)$ 
49     for i := 1; i <= rowIndex; i++){
50         res[i] = res[i-1] * (rowIndex-i+1)/i
51     }
52     return res
53 }

```

121.买卖股票的最佳时机

题目

```

1  给定一个数组，它的第  $i$  个元素是一支给定股票第  $i$  天的价格。
2
3  如果你最多只允许完成一笔交易（即买入和卖出一支股票），设计一个算法来计算你能获取的最大
   利润。
4
5  注意你不能在买入股票前卖出股票。
6
7  示例 1：
8
9  输入：[7,1,5,3,6,4]
10 输出：5
11 解释：在第 2 天（股票价格 = 1）的时候买入，在第 5 天（股票价格 = 6）的时候卖出，最大利
   润 =  $6-1 = 5$  。
12     注意利润不能是  $7-1 = 6$ ，因为卖出价格需要大于买入价格。
13
14 示例 2：
15
16 输入：[7,6,4,3,1]
17 输出：0
18 解释：在这种情况下，没有交易完成，所以最大利润为 0。

```

解题思路

No.	思路	时间复杂度	空间复杂度
01	暴力法	$O(n^2)$	$O(1)$
02(最优)	动态规划(从前到后) 最大利润= $\max\{\text{前一天最大利润, 今天的价格} - \text{之前最低价格}\}$	$O(n)$	$O(1)$
03	动态规划(从后到前)	$O(n)$	$O(1)$

```

1 // 暴力法
2 func maxProfit(prices []int) int {
3     max := 0
4     length := len(prices)
5
6     for i := 0; i < length-1; i++{
7         for j := i+1; j <= length-1; j++{
8             if prices[j] - prices[i] > max{
9                 max = prices[j] - prices[i]
10            }
11        }
12    }
13    return max
14 }
15
16 // 动态规划(从前到后)
17 func maxProfit(prices []int) int {
18     if len(prices) < 2 {
19         return 0
20     }
21
22     min := prices[0]
23     profit := 0
24
25     for i := 1; i < len(prices); i++ {
26         if prices[i] < min {
27             min = prices[i]
28         }
29         if profit < prices[i]-min {
30             profit = prices[i] - min
31         }
32     }
33     return profit
34 }
35
36
37 // 动态规划(从后到前)
38 func maxProfit(prices []int) int {

```

```
39     if len(prices) < 2 {
40         return 0
41     }
42
43     max := 0
44     profit := 0
45
46     for i := len(prices) - 1; i >= 0; i-- {
47         if max < prices[i] {
48             max = prices[i]
49         }
50         if profit < max-prices[i] {
51             profit = max - prices[i]
52         }
53     }
54
55     return profit
56 }
```