链滴

各类算法解题模板

作者：wlgzs-sjl

原文链接：https://ld246.com/article/1605795834785

来源网站：链滴

1、前序，中序，后序遍历

```
//前序遍历
public static void preOrder(TreeNode tree) {
    if (tree == null)
        return;
    System.out.printf(tree.val + "");
    preOrder(tree.left);
    preOrder(tree.right);
}

//中序遍历
public static void inOrderTraversal(TreeNode node) {
    if (node == null)
        return;
    inOrderTraversal(node.left);
    System.out.println(node.val);
    inOrderTraversal(node.right);
}

//后序遍历
public static void postOrder(TreeNode tree) {
    if (tree == null)
        return;
    postOrder(tree.left);
    postOrder(tree.right);
    System.out.println(tree.val);
}
```

2、深度优先搜索（DFS）

```
public static void treeDFS(TreeNode root) {
    if (root == null)
        return;
    System.out.println(root.val);
    treeDFS(root.left);
    treeDFS(root.right);
}
```

3、广度优先搜索（BFS）

```
public static void levelOrder(TreeNode tree) {
     if (tree == null)
         return;
     Queue<TreeNode> queue = new ArrayDeque<>();
     queue.add(root);
     while (!queue.isEmpty()) {
        TreeNode node = queue.poll();
        if (node.left != null) {
            queue.add(node.left);
        }
        if (node.right != null) {
            queue.add(node.right);
        }
     }
}
```

4、回溯

```
private void backtrack("原始参数") {
    //终止条件(递归必须要有终止条件)
    if ("终止条件") {
        //一些逻辑操作（可有可无，视情况而定）
        return;
    }

    for (int i = "for循环开始的参数"; i < "for循环结束的参数"; i++) {
        //一些逻辑操作（可有可无，视情况而定）

        //做出选择

        //递归
        backtrack("新的参数");
        //一些逻辑操作（可有可无，视情况而定）

        //撤销选择
    }
}
```

5、二分查找

```
//常规法
public int search(int[] nums, int target) {
```

```
    int len = nums.length;

    int left = 0;
    int right = len - 1;
    // 目标元素可能存在在区间 [left, right]
    while (left <= right) {
        int mid = left + (right - left) / 2;
        if (nums[mid] == target) {
            return mid;
        } else if (nums[mid] < target) {
            // 目标元素可能存在在区间 [mid + 1, right]
            left = mid + 1;
          } else {
          // 目标元素可能存在在区间 [left, mid - 1]
          right = mid - 1;
        }
    }
    return -1;
}

//排除法（1）
public int search(int[] nums, int target) {
    int len = nums.length;

    int left = 0;
    int right = len - 1;
    // 目标元素可能存在在区间 [left, right]
    while (left < right) {
        int mid = left + (right - left) / 2;
        if (nums[mid] < target) {
            // 下一轮搜索区间是 [mid + 1, right]
            left = mid + 1;
        } else {
            // 下一轮搜索区间是 [left, mid]
            right = mid;
        }
    }

    if (nums[left] == target) {
        return left;
    }
    return -1;
}

//排除法（2）
public int search(int[] nums, int target) {
    int len = nums.length;

    int left = 0;
    int right = len - 1;
    while (left < right) {
        int mid = left + (right - left + 1) / 2;
        if (nums[mid] > target) {
            // 下一轮搜索区间是 [left, mid - 1]
```

```
            right = mid - 1;
        } else {
            // 下一轮搜索区间是 [mid, right]
            left = mid;
        }
    }

    if (nums[left] == target) {
        return left;
    }
    return -1;
}
```

6、动态规划

```
//1.问题拆解，找到问题之间的具体联系
//2.状态定义
//3.递推方程推导
//4.实现

//初始化 base case
dp[0][0][...] = base
//进行状态转移
for 状态1 in 状态1的所有取值：
    for 状态2 in 状态2的所有取值：
        for ...
            dp[状态1][状态2][...] = 求最值(选择1，选择2...)


//举例：leetcode120.三角形最小路径和
public int minimumTotal(List<List<Integer>> triangle) {
    int n = triangle.size();
    // dp[i][j] 表示从点 (i, j) 到底边的最小路径和。
    int[][] dp = new int[n + 1][n + 1];
    // 从三角形的最后一行开始递推。
    for (int i = n - 1; i >= 0; i--) {
        for (int j = 0; j <= i; j++) {
            dp[i][j] = Math.min(dp[i + 1][j], dp[i + 1][j + 1]) + triangle.get(i).get(j);
        }
    }
    return dp[0][0];
}

//空间优化版本
public int minimumTotal(List<List<Integer>> triangle) {
    int n = triangle.size();
    int[] dp = new int[n + 1];
    for (int i = n - 1; i >= 0; i--) {
        for (int j = 0; j <= i; j++) {
            dp[j] = Math.min(dp[j], dp[j + 1]) + triangle.get(i).get(j);
        }
    }
    return dp[0];
}
```