大厂Java后端算法面试题串讲

Facebook面试官 带你刷遍大厂高频题

讲师：令狐冲

版权声明

九章的所有课程均受法律保护，不允许录像与传播录像 一经发现，将被追究法律责任和赔偿经济损失。

本节大纲

1. 字节跳动算法类面试真题串讲

    T1：LintCode 1086. 重复字符串匹配

    T2：LintCode 376. 二叉树的路径和

2. 阿里巴巴算法类面试真题串讲

    T3：LintCode 433. 岛屿的个数

    T4：LintCode 35. 翻转链表

3. 腾讯算法类面试真题串讲

    T5：LintCode 5. 第k大元素

    T6：LintCode 102. 带环链表

    T7：LintCode 100.删除排序数组中的重复数字

字节跳动真题1

T1：LintCode 1086. 重复字符串匹配

https://www.lintcode.com/problem/repeated-string-match/description

LintCode 1086. 重复字符串匹配

题目大意：

给定两个字符串A和B，找到A必须重复的最小次数，以使得B是它的子字符串。如果没有这样的解决方案，返回-1。

样例输入：

A = "abcd"

B = "cdabcdab“

样例输出：

3

样例解释：

将字符串A连续复制3次得到 “abcdabcdabcd”，此时B是A的子串

解题思路

- 如果B是A的子串，那么A的长度一定>=B的长度。
- 首先让A不断复制，直到A的长度>=B的长度，此时判断B是否为其子串。
- 若不是，再重复一次A，并判断B是否为其子串，
- 若仍不是，则输出 -1 即可。

LintCode 1086. 重复字符串匹配


```java
public int repeatedStringMatch(String A, String B) {
    //拷贝一个A
    String tempS = A;
    int count = 1;
    // 当A的长度小于B的时候，扩大A的长度
    while (A.length() < B.length()) {
        A += tempS;
        count++;
    }
    // 判断此时B在不在A中
    if(A.indexOf(B) >= 0) {
        return count;
    }
    // 再加一个A看看B是否在A中
    A = A + tempS;
    if(A.indexOf(B) >= 0) {
        return count + 1;
    }
    // 如果这还不行的话，那么A再怎么扩大也不能是让B成为子串
    return -1;
}
```

字节跳动真题2

T2：LintCode 376. 二叉树的路径和

https://www.lintcode.com/problem/binary-tree-path-sum/description

LintCode 376. 二叉树的路径和

题目大意：

给定一个二叉树，找出所有路径中各节点相加总和等于给定目标值的路径。一个有效的路径，指的是从根节点到叶节点的路径。

样例输入：

{1,2,4,2,3}

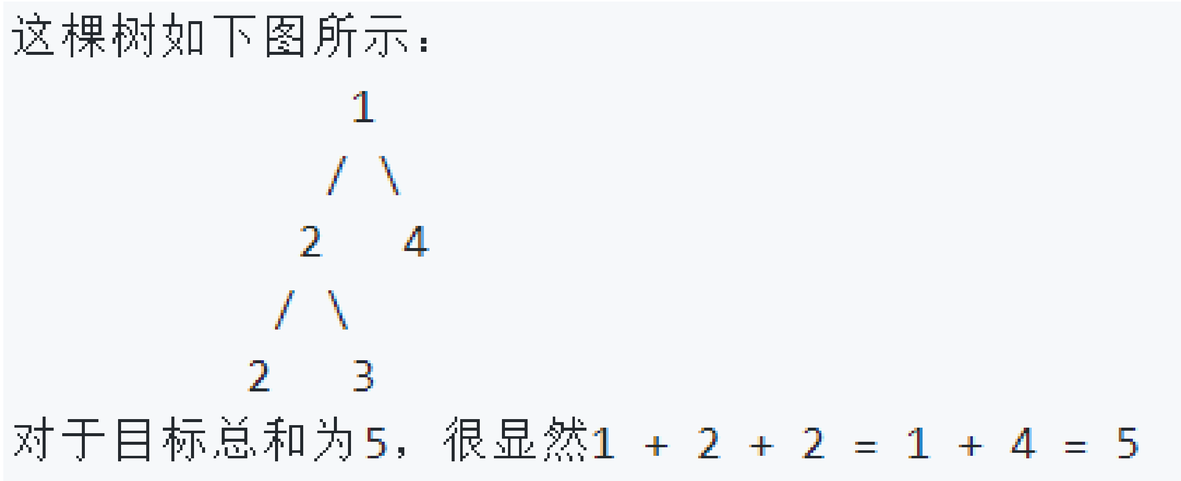
5

样例输出：

[[1, 2, 2],[1, 4]]

样例解释：

这棵树如下图所示：



对于目标总和为5，很显然1 + 2 + 2 = 1 + 4 = 5

LintCode 376. 二叉树的路径和

使用算法：DFS

实现方式：递归

解题思路：

- 递归的定义：返回路径和等于给定目标值的路径。
- 递归的拆解：左右子树不为空的时候往左右子树走。在当前递归层，路径的总价值要加上当前节点的值，存放路径的 List 增加当前的节点。
- 递归的出口：路径和为给定值时，保存当前路径并返回。

LintCode 376. 二叉树的路径和


```java
public List<List<Integer>> binaryTreePathSum(TreeNode root, int target) {
    List<List<Integer>> res = new ArrayList<>();
    if (root == null){
        return res;
    }
    ArrayList<Integer> path = new ArrayList<Integer>();
    dfs(root, target, path, res);
    return res;
}
private void dfs(TreeNode root, int target, ArrayList<Integer> path, List<List<Integer>> res){
    if (root == null) { // 空节点
        return;
    }
    path.add(root.val);
    if (root.left == null && root.right == null) { // 叶节点
        if (target == root.val){
            res.add(new ArrayList<>(path));
        }
        path.remove(path.size() - 1);
        return;
    }
    // 非叶节点
    dfs(root.left, target - root.val, path, res);
    dfs(root.right, target - root.val, path, res);
    path.remove(path.size() - 1);
}
```

相关问题

- 二叉树中的最大路径和

https://www.lintcode.com/problem/binary-tree-maximum-path-sum/description

- 二叉树最近公共祖先

https://www.lintcode.com/problem/lowest-common-ancestor-of-a-binary-tree/description

阿里巴巴真题1

T3：LintCode 433. 岛屿的个数

https://www.lintcode.com/problem/number-of-islands/description

LintCode 433. 岛屿的个数

题目大意：

给一个 01 矩阵，求不同的岛屿的个数。0 代表海，1 代表岛，如果两个 1 相邻，那么这两个 1 属于同一个岛。我们只考虑上下左右为相邻。

样例输入：

[[1,1,0,0,0],
[0,1,0,0,1],
[0,0,0,1,1],
[0,0,0,0,0],
[0,0,0,0,1]]

样例输出：3

样例解释：

[[1,1,0,0,0],
[0,1,0,0,1],
[0,0,0,1,1],
[0,0,0,0,0],
[0,0,0,0,1]]

上图中用三种颜色标记了三个岛屿

LintCode 433. 岛屿的个数

使用算法：BFS

实现方式：队列

解题思路：

1. 使用 n*m 的循环依次对每个点判断，如果该点没有标记，则以该点为起点开始宽度优先搜索
2. 使用方向数组进行四个方向搜索，并将入队的坐标标记起来
3. 队列空时搜索结束，继续步骤1的循环，岛屿数量加一
4. 步骤1的循环结束后，得到岛屿数量。

LintCode 433. 岛屿的个数

九章算法
www.jiuzhang.com

```java
void bfs(boolean[][] grid, boolean[][] visited, int x, int y) {
    Queue<Point> queue = new LinkedList<>();
    queue.offer(new Point(x, y));
    visited[x][y] = true;
    while(!queue.isEmpty()) {
        extendQueue(grid, visited, queue);
    }
}

void extendQueue(boolean[][] grid, boolean[][] visited, Queue<Point> queue) {
    int queueLength = queue.size();
    for (int i = 0; i < queueLength; i++) {
        Point thisPoint = queue.poll();
        for (int j = 0; j < 4; j++) {
            int newX = thisPoint.x + DIRECTIONS[j][0];
            int newY = thisPoint.y + DIRECTIONS[j][1];
            if (isValid(newX, newY, visited, grid)) {
                queue.offer(new Point(newX, newY));

                visited[newX][newY] = true;
            }
        }
    }
}
```

BFS算法与合法性判断

```java
boolean isValid(int x,
                int y,
                boolean[][] visited,
                boolean[][] grid) {
    if (x < 0 || x >= visited.length) {
        return false;
    }
    if (y < 0 || y >= visited[0].length) {
        return false;
    }
    if (visited[x][y] == true) {
        return false;
    }
    return grid[x][y];
}
```

LintCode 433. 岛屿的个数


```java
int[][] DIRECTIONS = {
    {1, 0},
    {-1, 0},
    {0, 1},
    {0, -1},
};

public int numIslands(boolean[][] grid) {
    if (grid == null || grid.length == 0 || grid[0].length == 0) {
        return 0;
    }

    int ans = 0, n = grid.length, m = grid[0].length;
    boolean[][] visited = new boolean[n][m];
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < m; j++) {
            if (grid[i][j] && !visited[i][j]) {
                bfs(grid, visited, i, j);
                ans++;
            }
        }
    }

    return ans;
}
```

方向数组，程序入口

以及坐标类定义

```java
class Point {
    int x, y;
    public Point(int x, int y) {
        this.x = x;
        this.y = y;
    }
}
```

LintCode 433. 岛屿的个数

使用算法：并查集

解题思路：

并查集是一个可以快速查询和快速合并两个集合的树形结构。

一开始我们把每个看做陆地的点都视作一个集合。

使用一个 n*m 的循环遍历所有格子，如果它的四方向存在另外一块陆地，则查询两个陆地所在集合，若两块陆地不在一个集合，则总集合数量减一，并将两个集合（当前陆地和周边陆地）合并。

结束后只需要查看最后的集合数量就是岛屿的数量。

阿里巴巴真题2

T4：LintCode 35. 翻转链表

https://www.lintcode.com/problem/reverse-linked-list/description

LintCode 35. 翻转链表

题目大意：

翻转给出的链表，并返回新链表的头指针。

样例输入：

1->2->3->null

样例输出：

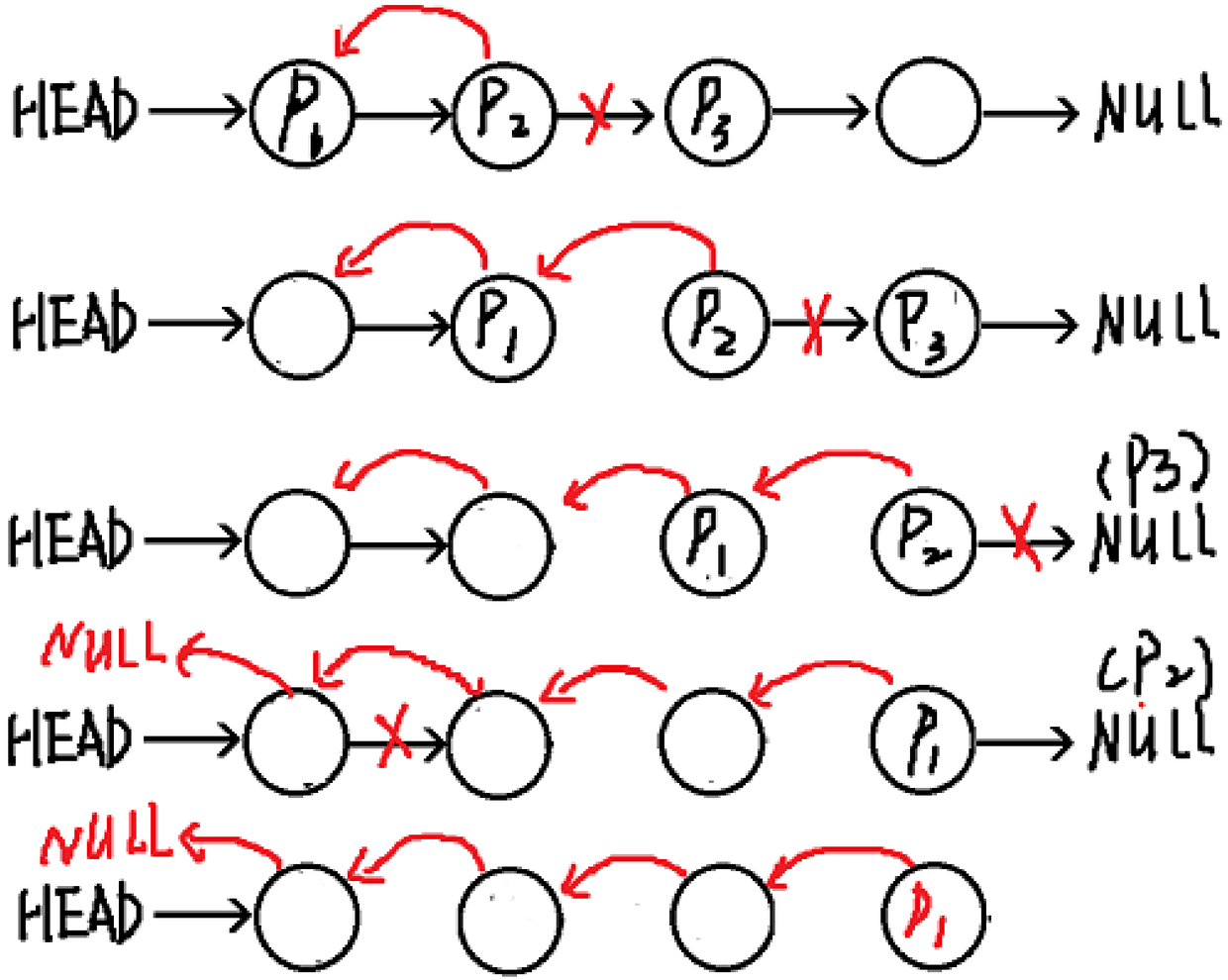
3->2->1->null

LintCode 35. 翻转链表

使用算法：迭代法

解题思路：

迭代法从前往后遍历链表，定义三个指针分别指向相邻的三个结点，反转前两个结点，即让第二个结点指向第一个结点。然后依次往后移动指针，直到第二个结点为空结束，再处理链表头尾即可。



LintCode 35. 翻转链表

```java
public ListNode reverse(ListNode head) {
    // 空链或只有一个结点，直接返回头指针
    if (head == null || head.next == null) {
        return head;
    }
    // 第一个结点
    ListNode  p1 = head;
    // 第二个结点
    ListNode  p2 = p1.next;
    // 第三个结点
    ListNode  p3 = p2.next;
    // 第二个结点为空，到链尾，结束
    while (p2 != null) {
        p3 = p2.next;
        // 第二个结点指向第一个结点，进行反转
        p2.next = p1;
        // 第一个结点往后移
        p1 = p2;
        // 第二个结点往后移
        p2 = p3;
    }
    // 第一个结点也就是反转后的最后一个节点指向mull
    head.next = null;
    // 头结点指向反转后的第一个节点
    head = p1;
    return head;
}
```

实现方法：递归

解题思路：

- 递归的定义：返回链表翻转后的头指针
- 递归的出口：空链或只有一个结点时，直接返回头指针
- 递归的拆解：调用函数本身，返回子链表翻转后的头指针

LintCode 35. 翻转链表


```java
public ListNode reverse(ListNode head) {
    // 空链或只有一个结点，直接返回头指针
    if (head == null || head.next == null) {
        return head;
    }
    // 反转以第二个结点为头的子链表
    ListNode newHead = reverse(head.next);
    // head.next 此时指向子链表的最后一个结点
    // 将之前的头结点放入子链尾
    head.next.next = head;
    head.next = null;
    return newHead;
}
```

递归方法的问题是什么？

链表问题都不可以使用递归的方法来做

腾讯真题1

T5：Lintcode 5. 第k大元素

https://www.lintcode.com/problem/kth-largest-element/description

LintCode 5. 第k大元素

题目大意：找到数组中的第k大元素

样例输入：

k = 1

nums = [1,3,4,2]

样例输出：4

使用算法：快速选择

解题思路：

通过快速排序算法的partition步骤，可以将小于pivot的值划分到pivot左边，大于pivot的值划分到pivot右边，所以可以直接得到pivot的rank。从而缩小范围继续找第k大的值。

LintCode 5. 第k大元素


```java
public int kthLargestElement(int k, int[] nums) {
    if (nums == null || nums.length == 0 || k < 1 || k > nums.length){
        return -1;
    }
    return partition(nums, 0, nums.length - 1, nums.length - k);
}
private int partition(int[] nums, int start, int end, int k) {
    if (start >= end) {
        return nums[k];
    }
    int left = start, right = end;
    int pivot = nums[(start + end) / 2];
    while (left <= right) {
        while (left <= right && nums[left] < pivot) {
            left++;
        }
        while (left <= right && nums[right] > pivot) {
            right--;
        }
        if (left <= right) {
            swap(nums, left, right);
            left++;
            right--;
        }
    }
    if (k <= right) {
        return partition(nums, start, right, k);
    }
    if (k >= left) {
        return partition(nums, left, end, k);
    }
    return nums[k];
}
private void swap(int[] nums, int i, int j) {
    int tmp = nums[i];
    nums[i] = nums[j];
    nums[j] = tmp;
}
```

腾讯真题2

T6：LintCode 102. 带环链表

https://www.lintcode.com/problem/linked-list-cycle/description

题目大意：给出一个链表，判断其是否有环。

样例输入：

21->10->4->5，then tail connects to node index 1(value 10).

样例输出：

true

使用算法：快慢双指针

解题思路：

快指针每次走两步，慢指针每次走一步。 在慢指针进入环之后，快慢指针之间的距离每次缩小1，所以最终能相遇。

LintCode 102. 带环链表


```java
public Boolean hasCycle(ListNode head) {
    if (head == null || head.next == null) {
        return false;
    }

    ListNode fast, slow;
    fast = head.next;
    slow = head;
    while (fast != slow) {
        if(fast==null || fast.next==null)
            return false;
        fast = fast.next.next;
        slow = slow.next;
    }
    return true;
}
```

腾讯真题3

T7：LintCode 100 删除排序数组中的重复数字

https://www.lintcode.com/problem/remove-duplicates-from-sorted-array/description

LintCode 100. 删除排序数组中的重复数字

题目大意：

给定一个排序数组，在原数组中“删除”重复出现的数字，使得每个元素只出现一次，并且返回“新”数组的长度。同时不使用额外的数组空间。

样例输入：

[1,1,2]

样例输出：

2

样例解释：

1和2是两个原数组中不同的数字，所以返回2。

LintCode 100. 删除排序数组中的重复数字

使用算法：同相双指针

解题思路：

- 此题的删除并不是真正意义上的删除，只需要在返回时保证新数组中的前 k 项是原数组中的不同的值即可。要注意题目所给的数组是一个排序后的数组。
- 定义慢指针 index = 0，快指针 i = 1，每次快指针自增 1，如果当前两个指针所指数字不等，则先令慢指针向前走一步，然后交换两指针数字。
- 算法结束（快指针到达数组末尾）后，index + 1 即为答案。

LintCode 100. 删除排序数组中的重复数字


```java
public int removeDuplicates(int[] nums) {
    if (nums == null || nums.length == 0) {
        return 0;
    }
    int index = 0;
    for (int i = 1; i < nums.length; i++) {
        if (nums[index] != nums[i]) {
            nums[++index] = nums[i];
        }
    }
    return index + 1;
}
```

时间复杂度练习

面试必会

归并排序的时间复杂度分析方法

方法1：$T(n) = 2T(n/2) + O(n)$

T(n)=规模为n的问题在该算法下的时间复杂度

归并排序的时间复杂度分析方法

方法2： 画递归树

下面这个时间复杂度是多少

假设图中有 n 个点，m 条边

```
for (Node node : nodes)
    for (Node neighbor : node.get_neighbors())
        do something
```

下面这个时间复杂度是多少

```
i=0; j =0;
while (i < n) {
    while (j < n && 某个条件)  {
        j++;
    }
    i++
}
```

结论

多重循环的时间复杂度

取决于最内层循环主体的执行次数

并不一定等于多重循环的最坏循环次数相乘