链滴

2021.1.24-2021.1.28 题解

作者：ChenforCode
原文链接：https://ld246.com/article/1611810634214
来源网站：链滴

2021.1.24-2021.1.28题解

● 674.最长递增子序列 2021.1.24

https://leetcode-cn.com/problems/longest-continuous-increasing-subsequence/

方法具体可以类比寻找最大值，只不过最大值变成了每个序列的递增长度，设置一个curMax和一个mx，如果序列一直在递增，就curMax++，直到递减重置curMax，然后根据curMax和max的大小对mx进行更新。

```
package _2021._01._0124._1;

/**
 * @author <a href="http://blog.chenforcode.cn">PKUCoder</a>
 * @date 2021/1/24 10:02 上午
 * @description 最长递增子序列，
 * 方法具体可以类比寻找最大值，只不过最大值变成了每个序列的递增长度，设置一个curMax
 * 和一个max，如果序列一直在递增，就curMax++，直到递减重置curMax，然后根据curMax
 * 和max的大小对max进行更新。
 */
public class Solution {
    public static int findLengthOfLCIS(int[] nums) {
        if (nums.length == 0) {
            return 0;
        }
        int max = 1;
        int curMax = 1;
        for (int i = 1; i < nums.length; i++) {
            if (nums[i] > nums[i - 1]) {
                curMax++;
            } else {
                curMax = 1;
```

```
            }
            max = Math.max(curMax, max);
        }
        return max;
    }

    public static void main(String[] args) {
        int[] nums = {2,2,2};
        System.out.println(findLengthOfLCIS(nums));
    }
}
```

- 2.两数相加 2021.1.24

https://leetcode-cn.com/problems/add-two-numbers/

两数相加，数字是倒序存储的，所以可以直接遍历两个链表，让两个节点的值相加，如果存在大于等10的情况就进位，下两个节点计算加上进位，如果最后两个节点还存在进位，那就新建立一个节点存该进位。

```
package _2021._01._0124._2;

/**
 * @author <a href="http://blog.chenforcode.cn">PKUCoder</a>
 * @date 2021/1/24 11:09 上午
 * @description 2.两数相加，数字是倒序存储的，所以可以直接遍历两个链表，让两个节点的值相
,
 * 如果存在大于等于10的情况就进位，下两个节点计算加上进位，如果最后两个节点还存在进位，那
新
 * 建立一个节点存储该进位。
 */
public class Solution {
    public static ListNode addTwoNumbers(ListNode l1, ListNode l2) {
        ListNode head = new ListNode(-1);
        ListNode h = head;
        boolean jinwei = false;
        while (l1 != null || l2 != null) {
            int sum = 0;
            if (l1 != null) {
                sum += l1.val;
                l1 = l1.next;
            }
            if (l2 != null) {
                sum += l2.val;
                l2 = l2.next;
            }
            if (jinwei) {
                sum++;
            }
            h.next = new ListNode(sum % 10);
            h = h.next;
            jinwei = (sum >= 10);
        }
        if (jinwei) {
            h.next = new ListNode(1);
        }
```

```
        return head.next;
    }

    public static void main(String[] args) {
    }
}

class ListNode {
    int val;
    ListNode next;

    ListNode() {
    }

    ListNode(int val) {
        this.val = val;
    }

    ListNode(int val, ListNode next) {
        this.val = val;
        this.next = next;
    }
}
```

● 959.由斜杠划分区域 2021.1.25

https://leetcode-cn.com/problems/regions-cut-by-slashes/

将1x1的格子扩大成为3x3的迷宫,扩大成为6x6，然后把两种斜线写入迷宫计1，然后计算最终有多少区域即可

```
package _2021._01._0125;

/**
 * @author Ariel
 * @date 2021/1/25 10:37
 * @description 959. 由斜杠划分区域。
 * 将1*1的格子扩大成为3*3的迷宫，2*2扩大成为6*6，然后把两种斜线写入迷宫计1，然后计算最终多少个
 * 区域即可
 */
public class Solution {
    private static int[][] next = {{0,1}, {1,0}, {0,-1}, {-1,0}};
    public static int regionsBySlashes(String[] grid) {
        int count = 100;//记录有多少个区域，深搜一次说明就有一个
        int [][] g = new int[grid.length * 3][grid[0].length() * 3];
        //初始化
        for (int i = 0; i < g.length; i++) {
            for (int j = 0; j < g[0].length; j++){
                g[i][j] = 0;
            }
        }
        //填迷宫
        for (int i = 0; i < grid.length; i++) {
            char[] chars = grid[i].toCharArray();
            for (int j = 0; j < chars.length; j++) {
```

```
                if (chars[j] == '/') {
                    g[i * 3 + 2][j * 3] = 1;
                    g[i * 3 + 1][j * 3 + 1] = 1;
                    g[i * 3][j * 3 + 2] = 1;
                }
                if(chars[j] == '\\'){
                    g[i * 3][j * 3] = 1;
                    g[i * 3 + 1][j * 3 + 1] = 1;
                    g[i * 3 +2][j * 3 + 2] = 1;
                }
            }
        }
        //开始深搜
        for (int i = 0; i < g.length; i++) {
            for (int j = 0; j < g[0].length; j++) {
                if(g[i][j] == 0){
                    g[i][j]= count;
                    dfs(g, i, j, count);
                    count++;
                }
            }
        }
        return count - 100;
    }
    static void dfs(int[][]g, int i, int j, int count){
        for (int m = 0; m < 4; m++) {
            int nextX = next[m][0] + i;
            int nextY = next[m][1] + j;
            if (nextX >= 0 && nextX < g.length && nextY >= 0 && nextY < g[0].length && g[nex
X][nextY] == 0) {
                g[nextX][nextY] = count;
                dfs(g, nextX, nextY, count);
            }
        }
    }

    public static void main(String[] args) {
        String[] grid = new String[2];
        grid[0] = "//";
        grid[1] = "/ ";
        System.out.println(regionsBySlashes(grid));
    }
}
```

● 1128.等价多米诺骨牌对的数量 2021.1.26

https://leetcode-cn.com/problems/number-of-equivalent-domino-pairs/

对于12，21这种我们统一看成12，并将12看成key，value看成有多少个组是12，然后对value进行合计算，求和得到最终的结果。

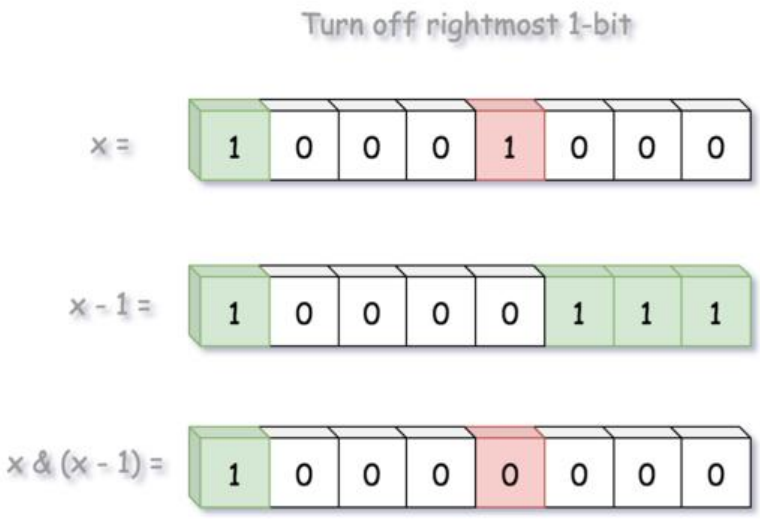
```
package _2021._01._0126;

import java.util.HashMap;
import java.util.Set;
```

```java
/**
 * @author <a href="http://blog.chenforcode.cn">PKUCoder</a>
 * @date 2021/1/26 9:52 上午
 * @description 1128.等价多米诺骨牌对的数量
 * 给出n个二元组，两个组，相同或者相反代表是相同的一组，查一共有多少个相同组。
 * 例如 (1,2)(2,1)(3,4)(4,5)只有前两个是相同的一组，输出为1
 * 解法：对于12，21这种我们统一看成12，并将12看成key，value看成有多少个组是12，然后对val
e进行
 * 组合计算，求和得到最终的结果。
 */
public class Solution {
    public static int numEquivDominoPairs(int[][] dominoes) {
        HashMap<Integer, Integer> hashMap = new HashMap<>();
        for (int i = 0; i < dominoes.length; i++) {
            int[] row = dominoes[i];
            int key = 0;
            if (row[0] > row[1]) {
                key = row[0] * 10 + row[1];
            } else {
                key = row[1] * 10 + row[0];
            }

            if (!hashMap.containsKey(key)) {
                hashMap.put(key, 1);
            } else {
                hashMap.put(key, hashMap.get(key) + 1);
            }
        }
        Set<Integer> integers = hashMap.keySet();
        int ans = 0;
        for (Integer integer : integers) {
            int val = hashMap.get(integer);
            if (val >= 2) {
                ans += (val * (val - 1) / 2);
            }
        }
        return ans;
    }

    public static void main(String[] args) {
        int [][]res = {{1,2},{2,1},{2,1},{5,6}};
        System.out.println(numEquivDominoPairs(res));
    }
}
```

● 461.汉明距离 2021.1.26

https://leetcode-cn.com/problems/hamming-distance/

汉明距离是求两个数字中，不同位的数字的个数。因此可以先进行异或，然后结果里边的所有的1就最终结果。因此提供两种解法：第一种就是一位一位的移动，计算1的个数。第二种是叫做布赖恩·克根算法，他是可以直接计算1，可以跳过0。如图，让x & (x-1)就可以去除掉x中的最后一个1，以此推，该操作的次数就是结果里边1的个数。




```
package _2021._01._0126._2;

/**
 * @author <a href="http://blog.chenforcode.cn">PKUCoder</a>
 * @date 2021/1/26 11:16 上午
 * @description 461.汉明距离
 */
public class Solution {
    public static int hammingDistance(int x, int y) {
        int result = x ^ y;
        int dis = 0;
        //相与计算
//        while (result != 0) {
//            //每做一次该操作，都消除了一个1
//            result = (result & (result - 1));
//            dis++;
//        }
        //移位计算
        while (result != 0) {
            if (result % 2 == 1) {
                dis++;
            }
            result = result >> 1;
        }
        return dis;
    }

    public static void main(String[] args) {
        System.out.println(hammingDistance(1, 4));
    }
}
```

- 1579.保证图可完全遍历 2021.1.27

https://leetcode-cn.com/problems/remove-max-number-of-edges-to-keep-graph-fully-traversable/

这个题的主要思路有点类似于克鲁斯卡尔求最小生成树，但是不同的是这个图没有权重，因此可以按边的顺序进行依次加边。而且判断连通也不是整个图连通，而是判断两个点是否连通，如果这两个点

经连通了，那么如果再对这两个点加边就是多余的，此时最终的答案+1。然后具体判断两个点是否连可以用并查集。

```
package _2021._01._0127;

import java.util.Arrays;

/**
 * @author <a href="http://blog.chenforcode.cn">PKUCoder</a>
 * @date 2021/1/27 11:26 上午
 * @description 1579.保证图可完全遍历
 * 因为没有权重，所以按照顺序遍历，依次加边，然后判断联通，如果两点已经联通，那么下次连接
 * 这两点的边就是多余的，不再加边，记录可删除数字+1。先加公共边，然后加alice，加bob。
 * 二者分别有一个并查集来判断任意两点是否联通。最后各自的并查集数目为1代表合并完成，可连
。
 * 并查集可以看成是几个联通点的集合，他们几个之间类似树的关系，有一个祖先节点。
 */
public class Solution {
    public static int maxNumEdgesToRemove(int n, int[][] edges) {
        init(edges);
        Union alice = new Union(n);
        Union bob = new Union(n);
        int ans = 0;
        for (int[] edge : edges) {
            if (edge[0] == 3) {
                if (!alice.isConnected(edge[1], edge[2])) {
                    alice.merge(edge[1], edge[2]);
                    bob.merge(edge[1], edge[2]);
                } else {
                    ans++;
                }
            }
        }
        for (int[] edge : edges) {
            if (edge[0] == 1) {
                if (!alice.isConnected(edge[1], edge[2])) {
                    alice.merge(edge[1], edge[2]);
                } else {
                    ans++;
                }
            }
            if (edge[0] == 2) {
                if (!bob.isConnected(edge[1], edge[2])) {
                    bob.merge(edge[1], edge[2]);
                } else {
                    ans++;
                }
            }
        }
        return (alice.count == 1 && bob.count == 1) ? ans : -1;
    }

    public static void init(int[][] edges) {
        for (int[] edge : edges) {
```

```
                edge[1]--;
                edge[2]--;
            }
        }

        public static void main(String[] args) {
//            int[][] edges = {{3, 1, 2}, {3, 2, 3}, {1, 1, 3}, {1, 2, 4}, {1, 1, 2}, {2, 3, 4}};
//            System.out.println(maxNumEdgesToRemove(4, edges));
//            int n = 2;
//            int[][] edges = {{1, 1, 2}, {2, 1, 2}, {3, 1, 2}};
            int n = 13;
            int[][] edges = {{1,1,2},{2,1,3},{3,2,4},{3,2,5},{1,2,6},{3,6,7},{3,7,8},{3,6,9},{3,4,10},{2,3,11},{1,5,
2},{3,3,13},{2,1,10},{2,6,11},{3,5,13},{1,9,12},{1,6,8},{3,6,13},{2,1,4},{1,1,13},{2,9,10},{2,1,6},{2,10,13},
2,2,9},{3,4,12},{2,4,7},{1,1,10},{1,3,7},{1,7,11},{3,3,12},{2,4,8},{3,8,9},{1,9,13},{2,4,10},{1,6,9},{3,10,13}
{1,7,10},{1,1,11},{2,4,9},{3,5,11},{3,2,6},{2,1,5},{2,5,11},{2,1,7},{2,3,8},{2,8,9},{3,4,13},{3,3,8},{3,3,11},{
,9,11},{3,1,8},{2,1,8},{3,8,13},{2,10,11},{3,1,5},{1,10,11},{1,7,12},{2,3,5},{3,1,13},{2,4,11},{2,3,9},{2,6,9}
{2,1,13},{3,1,12},{2,7,8},{2,5,6},{3,1,9},{1,5,10},{3,2,13},{2,3,6},{2,2,10},{3,4,11},{1,4,13},{3,5,10},{1,4,
0},{1,1,8},{3,3,4},{2,4,6},{2,7,11},{2,7,10},{2,3,12},{3,7,11},{3,9,10},{2,11,13},{1,1,12},{2,10,12},{1,7,13}
{1,4,11},{2,4,5},{1,3,10},{2,12,13},{3,3,10},{1,6,12},{3,6,10},{1,3,4},{2,7,9},{1,3,11},{2,2,8},{1,2,8},{1,11
13},{1,2,13},{2,2,6},{1,4,6},{1,6,11},{3,1,2},{1,1,3},{2,11,12},{3,2,11},{1,9,10},{2,6,12},{3,1,7},{1,4,9},{1,
0,12},{2,6,13},{2,2,12},{2,1,11},{2,5,9},{1,3,8},{1,7,8},{1,2,12},{1,5,11},{2,7,12},{3,1,11},{3,9,12},{3,2,9},
3,10,11}};
            System.out.println(maxNumEdgesToRemove(n, edges));
        }
    }

    class Union {
        int[] parent;
        int[] size;
        int count;
        int n;

        public Union(int n) {
            this.n = n;
            this.count = n;
            parent = new int[n];
            size = new int[n];
            Arrays.fill(size, 1);
            for (int i = 0; i < parent.length; i++) {
                parent[i] = i;
            }
        }

        public int findRoot(int x) {
            //判断是否找到了最后的祖先。即判断x本身是否为祖先
            if (x == parent[x]) {
                return x;
            }
            //如果x不是祖先，那就接着判断x的父亲是不是祖先，并且更新x的父亲为找到的祖先结果
            parent[x] = findRoot(parent[x]);
            return parent[x];
        }

        public boolean merge(int x, int y) {
```

```
        //注意这里并查集的合并不能简单的将两个点合并，而是将两个点所在的集合合并
        //因此操作的是root(x)和root(y)
        if (findRoot(x) == findRoot(y)) {
            return false;
        }
        parent[findRoot(x)] = findRoot(y);
        parent[x] = y;
        this.count--;
        return true;
    }

    public boolean isConnected(int x, int y) {
        return findRoot(x) == findRoot(y);
    }
}
```

- 617.合并二叉树 2021.1.27

https://leetcode-cn.com/problems/merge-two-binary-trees/

这个题就正常做，遍历两棵树，计算相应的值作为新树的节点，然后新树的左子树为递归遍历两棵树左子树。右子树同理

```
package _2021._01._0127._2;

/**
 * @author <a href="http://blog.chenforcode.cn">PKUCoder</a>
 * @date 2021/1/27 10:56 下午
 * @description 617.合并二叉树
 */
public class Solution {
    public TreeNode mergeTrees(TreeNode t1, TreeNode t2) {
        if (t1 == null) {
            return t2;
        }
        if (t2 == null) {
            return t1;
        }
        TreeNode root = new TreeNode(t1.val + t2.val);
        root.left = mergeTrees(t1.left, t2.left);
        root.right = mergeTrees(t1.right, t2.right);
        return root;
    }
}

class TreeNode {
    int val;
    TreeNode left;
    TreeNode right;

    TreeNode() {
    }

    TreeNode(int val) {
        this.val = val;
    }
```

```
    TreeNode(int val, TreeNode left, TreeNode right) {
        this.val = val;
        this.left = left;
        this.right = right;
    }
}
```

- 724.寻找数组的中心索引 2021.1.28

https://leetcode-cn.com/problems/find-pivot-index/

这题的思路也很明确，就是遍历每一个点，都尝试当做中心索引，然后是否左右相等。但是该题如果每个中心节点的左右都计算一次太慢。我们只需要判断最后的条件，即sum左 + sum右 + num[i] == otal。然后sum左和sum右是相等的，因此直接2 * sum左 + num[i] == total即可，然后sum左可以遍历num[i]的时候就计算出来。

```
package _2021._01._0128;

import java.util.Arrays;

/**
 * @author <a href="http://blog.chenforcode.cn">PKUCoder</a>
 * @date 2021/1/28 9:22 上午
 * @description 724.寻找数组的中心索引
 */
public class Solution {
    public static int pivotIndex(int[] nums) {
        int total = Arrays.stream(nums).sum();
        int sum = 0;
        for (int i = 0; i < nums.length; i++) {
            if (sum * 2 + nums[i] == total) {
                return i;
            }
            sum += nums[i];
        }

        return -1;
    }

    public static void main(String[] args) {
//        int[] nums = {-1,-1,-1,-1,-1,-1};
        int []nums = {1, 7, 3, 6, 5, 6};
        System.out.println(pivotIndex(nums));
    }
}
```