1.ReverseWordString:

+ revese full string, find words and reverse each of them

2.Maximum Size Subarray Sum Equals k

+ consider k as subtraction totalSum - <sum from 0 to i> where i - array index and totalSum is the sum of first i items.

+Save previous ith sum to map

3.Longest Palindromic Substring

+ use DP and consider even(i-1,i) and odd(i-1,i+1) substrings as polyndrom.

4.Product of Array Except Self

+ Use resulting array to calculation. First get product(i) = product(i-1)\*num(i-1) product(0) = 1. Then product(n-1) = right \* product(n-1) where right = 1 for n-1,then right = right \* num(i)

 5.Find the Duplicate Number

+ Sort the item and find same adjacent items.

6.Intersection of Two Linked Lists

- make while until list1 != list2 list1 = list1.next and list2 = list2.next,

If list1 == null list1 = head1 and if list2 == null list2 = head2 Floyd algorithm????? – fix it!!!

7.Symmetric Tree

Create method for x and y to compare x.left and y.right and x.right and y.left. To pass (root, root) as first parameters to this method.

8.Binary Tree Vertical Order Traversal

public List<List<Integer>> verticalOrder(TreeNode root) {

List<List<Integer>> res = new ArrayList<>();

if (root == null) {

return res;

}

Map<Integer, ArrayList<Integer>> map = new HashMap<>();

Queue<TreeNode> q = new LinkedList<>();

Queue<Integer> cols = new LinkedList<>();

q.add(root);

cols.add(0);

int min = 0;

int max = 0;

while (!q.isEmpty()) {

TreeNode node = q.poll();

int col = cols.poll();

if (!map.containsKey(col)) {

map.put(col, new ArrayList<Integer>());

}

map.get(col).add(node.val);

if (node.left != null) {

q.add(node.left);

cols.add(col - 1);

min = Math.min(min, col - 1);

}

if (node.right != null) {

q.add(node.right);

cols.add(col + 1);

max = Math.max(max, col + 1);

}

}

for (int i = min; i <= max; i++) {

res.add(map.get(i));

}

return res;

Graph Valid Tree:

Use union-find algorithm with rank and path-compression

9.Alien Language

**import** collections  
**class** Solution:  
 **def** alienOrder(self, words):  
 pre = collections.defaultdict(set)  
 suc = collections.defaultdict(set)  
  
 **for** pair **in** zip(words, words[1:]):  
 **for** a, b **in** zip(\*pair):  
 **if** a != b:  
 suc[a].add(b)  
 pre[b].add(a)  
 **break** chars = set(**''**.join(words))  
 *#get first char without predecesesor* charToProcess = chars - set(pre)  
 order = **''  
 while** charToProcess:  
 ch = charToProcess.pop()  
 order += ch  
 **for** b **in** suc[ch]:  
 pre[b].discard(ch)  
 **if not** pre[b]: *# if processed all predecesesors let's add curret char to charToProcess* charToProcess.add(b)  
 **return** order \* (set(order) == chars)