FB tags

10 regular expression matching

1. recursion is very short
   base case = both string are empty
   the first char is matching
   or the other is matching

```
class Solution {
    public boolean isMatch(String s, String p) {
        if (p.length() == 0) {
            return s.length() == 0;
        }
        boolean first = s.length() > 0 && (p.charAt(0) == s.ch
arAt(0) || p.charAt(0) == '.');

        if (p.length() >= 2 && p.charAt(1) == '*') {
            return isMatch(s, p.substring(2)) || (first && isM
atch(s.substring(1), p));
        } else {
            return first && isMatch(s.substring(1), p.substrin
g(1));
        }
    }
}
```

1. 2d dp problem, understand when to remove 2 and when to remove only 1.

```
class Solution {
    public boolean isMatch(String s, String p) {
        if (s == null || p == null) {
            return false;
        }
```

```
        boolean[][] dp = new boolean[s.length() + 1][p.length
() + 1];
        char[] sc = s.toCharArray();
        char[] pc = p.toCharArray();
        dp[0][0] = true;
        for (int i = 1; i <= p.length(); i++) {
            if (pc[i - 1] == '*') {
                dp[0][i] = dp[0][i - 2];
            }
        }
        for (int i = 1; i <= s.length(); i++) {
            for (int j = 1; j <= p.length(); j++) {
                if (pc[j - 1] == sc[i - 1] || pc[j - 1] ==
'.') {
                    dp[i][j] = dp[i - 1][j - 1];
                } else if (pc[j - 1] == '*') {
                    if (pc[j - 2] == sc[i - 1] || pc[j - 2] ==
'.') {
                        dp[i][j] = dp[i - 1][j] || dp[i][j -
2];
                    } else {
                        dp[i][j] = dp[i][j - 2];
                    }
                }
            }
        }
        return dp[s.length()][p.length()];
    }
}
```

follow up, optimize space complexity .... don't understand

****
****

11 Whildcard matching

1. 2d dp, 2 cases when the p is having the *, either up or right, cause you can either remove the * or use the loop

```
class Solution {
    public boolean isMatch(String s, String p) {
        if (s == null || p == null) {
            return false;
        }
        boolean[][] dp = new boolean[s.length() + 1][p.length
() + 1];
        char[] sc = s.toCharArray();
        char[] pc = p.toCharArray();
        dp[0][0] = true;
        for (int i = 1; i <= p.length(); i++) {
            if (pc[i - 1] == '*') {
                dp[0][i] = dp[0][i - 1];
            }
        }
        for (int i = 1; i <= s.length(); i++) {
            for (int j = 1; j <= p.length(); j++) {
                if (sc[i - 1] == pc[j - 1] || pc[j - 1] ==
'?') {
                    dp[i][j] = dp[i - 1][j - 1];
                } else {
                    if (pc[j - 1] == '*') {
                        dp[i][j] = dp[i - 1][j] || dp[i][j -
1];
                    }
```

```
                }
            }
        }
        return dp[s.length()][p.length()];
    }
}
```

102. Binary Tree Level Order Traversal

1. bfs
2. dfs, cool and simple using the relationship of level and size of the size

```
class Solution {
    public List<List<Integer>> levelOrder(TreeNode root) {
        List<List<Integer>> res = new ArrayList<>();
        dfs(root, res, 0);
        return res;
    }
    void dfs(TreeNode root, List<List<Integer>> res, int l) {
        if (root == null) {
            return;
        }
        if (l >= res.size()) {
            res.add(new ArrayList<Integer>());
        }
        res.get(l).add(root.val);
        dfs(root.left, res, l + 1);
        dfs(root.right, res, l + 1);
    }
}
```

121. Best Time to Buy and Sell Stock

dp solution

```
class Solution {
    public int maxProfit(int[] prices) {
        int buy = Integer.MIN_VALUE;
        int sell = 0;
        for (int p : prices) {
            buy = Math.max(-p, buy);
            sell = Math.max(p + buy, sell);
        }
        return sell;
    }
}
```

122. Best Time to Buy and Sell Stock II

```
class Solution {
    public int maxProfit(int[] prices) {
        if (prices == null || prices.length == 0) {
            return 0;
        }
        int sum = 0;
        int prev = prices[0];
        for (int p : prices) {
            sum += Math.max(0, p - prev);
            prev = p;
        }
        return sum;
    }
}
```

122. Best Time to Buy and Sell Stock III

1. dp

```
class Solution {
    public int maxProfit(int[] prices) {
        int buy1 = Integer.MIN_VALUE;
        int buy2 = Integer.MIN_VALUE;
        int sell1 = 0;
        int sell2 = 0;
        for (int p : prices) {
            buy1 = Math.max(-p, buy1);
            sell1 = Math.max(p + buy1, sell1);
            buy2 = Math.max(sell1 - p, buy2);
            sell2 = Math.max(p + buy2, sell2);
        }
        return sell2;
    }
}
```

2. left and right side max, final total max

188. Best Time to Buy and Sell Stock IV

1. dp similar to previous ones, only build new int array to fill buy and sell

```
class Solution {
    public int maxProfit(int k, int[] prices) {
        if (prices == null || prices.length == 0) {
            return 0;
        }
        int n = prices.length;
        int[] buy = new int[k + 1];
        int[] sell = new int[k + 1];
        if (k > n / 2) {
            return maxall(prices);
        }
```

```
        Arrays.fill(buy, Integer.MIN_VALUE);

        for (int i = 0; i < n; i++) {
            for (int j = 1; j <= k; j++) {
                buy[j] = Math.max(sell[j - 1] - prices[i], buy
[j]);
                sell[j] = Math.max(buy[j] + prices[i], sell
[j]);
            }
        }
        return sell[k];
    }
    int maxall(int[] prices) {
        int sum = 0;
        for (int i = 1; i < prices.length; i++) {
            if (prices[i] - prices[i - 1] > 0) {
                sum += prices[i] - prices[i - 1];
            }
        }
        return sum;
    }
}
```

309. Best Time to Buy and Sell Stock with Cooldown

dp solution only with slight changes

```
class Solution {
    public int maxProfit(int[] prices) {
        if (prices == null || prices.length <= 1) {
            return 0;
```

```
        }
        int n = prices.length;
        int[] buy = new int[n];
        int[] sell = new int[n];
        Arrays.fill(buy, Integer.MIN_VALUE);
        buy[0] = -prices[0];
        sell[0] = 0;
        buy[1] = Math.max(-prices[0], -prices[1]);
        sell[1] = Math.max(0, prices[1] - prices[0]);
        for (int i = 2; i < n; i++) {
            buy[i] = Math.max(sell[i - 2] - prices[i], buy[i -
1]);
            sell[i] = Math.max(buy[i - 1] + prices[i], sell[i
- 1]);
        }
        return sell[n - 1];
    }
}
```

133. Clone Graph

1. dfs

```
public class Solution {
    public UndirectedGraphNode cloneGraph(UndirectedGraphNode
node) {
        Map<UndirectedGraphNode, UndirectedGraphNode> map = ne
w HashMap<>();
        return dfs(node, map);
    }
    UndirectedGraphNode dfs(UndirectedGraphNode node, Map<Undi
rectedGraphNode, UndirectedGraphNode> map) {
        if (node == null) {
```

```
            return null;
        }
        if (map.containsKey(node)) {
            return map.get(node);
        }
        UndirectedGraphNode temp = new UndirectedGraphNode(nod
e.label);
        map.put(node, temp);
        for (UndirectedGraphNode n : node.neighbors) {
            temp.neighbors.add(dfs(n, map));
        }
        return temp;
    }
}
```

1. bfs

138. Copy List with Random Pointer

1. Use a hashmap. First round, make a copy of list aside, set up the next pointer. Then do the random copy.

```
public class Solution {
    public RandomListNode copyRandomList(RandomListNode head)
{
        RandomListNode dummy = new RandomListNode(0);
        RandomListNode trav = dummy;
        Map<RandomListNode, RandomListNode> map = new HashMap<
>();
        while (head != null) {
            RandomListNode temp = new RandomListNode(head.labe
l);
            map.put(head, temp);
            temp.random = head.random; // copy the random here
            trav.next = temp;
```

```
            trav = trav.next;
            head = head.next;
        }
        trav = dummy.next;
        while (trav != null) {
            if (trav.random != null) {
                trav.random = map.get(trav.random);
            }
            trav = trav.next;
        }
        return dummy.next;
    }
}
```

139. Word Break

1. return true or false, use dp

```
class Solution {
    public boolean wordBreak(String s, List<String> wordDict)
{
        Set<String> st = new HashSet<>(wordDict);
        boolean[] dp = new boolean[s.length() + 1];
        dp[0] = true;
        for (int i = 1; i < s.length() + 1; i++) {
            for (int j = 0; j < i; j++) {
                if (dp[j]) {
                    String temp = s.substring(j, i);
                    if (st.contains(temp)) {
                        dp[i] = true;
                        break;
                    }
                }
```

```
            }
        }
        return dp[s.length()];
    }
}
```

140. Word Break II

1. dfs + memo

```
class Solution {
    Map<String, List<String>> map = new HashMap<>();
    public List<String> wordBreak(String s, List<String> wordD
ict) {
        List<String> res = new ArrayList<>();
        if (s == null || s.length() == 0) {
            return res;
        }
        if (map.containsKey(s)) {
            return map.get(s);
        }
        if (wordDict.contains(s))    res.add(s);
        for (int i = 1; i < s.length(); i++) {
            String temp = s.substring(i);
            if (wordDict.contains(temp)) {
                List<String> next = wordBreak(s.substring(0,
i), wordDict);
                if (next.size() > 0) {
                    for (String n : next) {
                        res.add(n + " " + temp);
                    }
                }
            }
```

```
        }
        map.put(s, res);
        return res;
    }
}
```

151. Reverse Words in a String

此题不难，注意clarification里的细节，要去除头尾空格，单词中间多个空格reverse后只能有一个。

186. Reverse Words in a String II

152. Maximum Product Subarray

max positive, min negative, 分情况讨论

```
class Solution {
    public int maxProduct(int[] nums) {
        if (nums == null || nums.length == 0) {
            return 0;
        }
        int maxPos = nums[0];
        int maxNeg = nums[0];
        int res = nums[0];
        for (int i = 1; i < nums.length; i++) {
            if (nums[i] >= 0) {
                maxPos = Math.max(maxPos * nums[i], nums[i]);
                maxNeg = Math.min(maxNeg * nums[i], nums[i]);
            } else {
                int temp = maxNeg;
                maxNeg = Math.min(maxPos * nums[i], nums[i]);
                maxPos = Math.max(temp * nums[i], nums[i]);
```

```
            }
            res = Math.max(res, maxPos);
        }
        return res;
    }
}
```

161. One Edit Distance

think about the 3 cases. Either use equals substring or compare one by one.

```
public boolean isOneEditDistance(String s, String t) {
    int len = Math.min(s.length(), t.length());
    for (int i = 0; i < len; i++) {
        if (s.charAt(i) != t.charAt(i)) {
            if (s.length() == t.length())   return s.substring
(i + 1).equals(t.substring(i + 1)); // replace
            else if (s.length() < t.length())   return s.subst
ring(i).equals(t.substring(i + 1)); // delete t
            else    return s.substring(i + 1).equals(t.substri
ng(i)); // delete s
        }
    }
    return Math.abs(s.length() - t.length()) == 1; // corner c
ase: ""
}
```

Not using equals or substring, use recursion to pass one second boolean

```
class Solution {
    public boolean isOneEditDistance(String s, String t) {
        if(s == null && t == null){
            return false;
        }
        if(s == null && t.length() != 1){
```

```
            return false;
        }
        if(t == null && s.length() != 1){
            return false;
        }
        if(Math.abs(s.length() - t.length()) > 1){
            return false;
        }
        return helper(s, 0, t, 0, false);
    }
    boolean helper(String s, int sstart, String t, int tstart,
boolean second){
        while(sstart < s.length() && tstart < t.length()){
            if(s.charAt(sstart) == t.charAt(tstart)){
                sstart++;
                tstart++;
            }else{
                if(second){
                    return false;
                }
                if(s.length() > t.length()){
                    return helper(s, sstart + 1, t, tstart, tr
ue);
                }else if(t.length() > s.length()){
                    return helper(s, sstart, t, tstart + 1, tr
ue);
                }else{
                    return helper(s, sstart + 1, t, tstart +
1, true);
                }
            }
```

```
        }
        if(!second && Math.abs(s.length() - t.length()) == 1){
            return true;
        }
        if(second && sstart == s.length() && tstart == t.lengt
h()){
            return true;
        }
        return false;
    }
}
```

17. Letter Combinations of a Phone Number

classic dfs

173. Binary Search Tree Iterator

push all function, use next to have all the logic

Tree preorder, inorder, postorder traversal.

preorder

```
public List<Integer> preorderTraversal(TreeNode root) {
    List<Integer> res = new ArrayList<>();
    if (root == null)   return res; // corner check
    Stack<TreeNode> stack = new Stack<>();
    stack.push(root);
    while (!stack.empty()) {
        res.add(stack.pop().val);
        if (root.right != null)     stack.push(root.right);
        if (root.left != null)      stack.push(root.left);
```

```
    }
    return res;
}
```

inorder

```
public List<Integer> inorderTraversal(TreeNode root) {
    List<Integer> res = new ArrayList<>();
    Stack<TreeNode> stack = new Stack<>();
    while (root != null || !stack.empty()) {
        while (root != null) {
            stack.push(root);
            root = root.left;
        }
        res.add(stack.pop().val);
        root = root.right;
    }
    return res;
}
```

postorder

```
public List<Integer> postorderTraversal(TreeNode root) {
    List<Integer> res = new ArrayList<>();
    Stack<TreeNode> stack = new Stack<>();
    TreeNode prev = null;
    while (root != null || !stack.empty()) {
        if (root != null) {
            stack.push(root);
            root = root.left;
        } else {
            TreeNode tmp = stack.peek();
            if (tmp.right != null && tmp.right != prev)
```

```
                root = tmp.right;
            else {
                stack.pop();
                res.add(tmp.val);
                prev = tmp;
            }
        }
    }
    return res;
}
```

200 Number of Islands

1. DFS
2. BFS
3. Union Find

follow up perimeter of islands. Make sure to understand what is the perimeter

206 Reverse LinkedList

2. iterative
3. recursive

211. Add and Search Word - Data structure design.java

1. trie, for the . , use dfs with the len to count the level

215. Kth Largest Element in an Array

221. Maximal Square

1. dp 滚动数组 int[] prev, int[] cur, 最后 prev = cur

230. Kth Smallest Element in a BST

1. keep a stack, count the number of prev you have, return the correct prev
2. dfs can also work

235. Lowest Common Ancestor of Binary Search Tree

236. Lowest Common Ancestor of Binary Tree

similar

253. Meeting Rooms II

1. find the relationship between start and end

```
class Solution {
    public int minMeetingRooms(Interval[] intervals) {
        int idx = 0;
        int[] start = new int[intervals.length];
        int[] end = new int[intervals.length];
        for (Interval i : intervals) {
            start[idx] = i.start;
            end[idx] = i.end;
            idx++;
        }
        Arrays.sort(start);
        Arrays.sort(end);
        int start_idx = 0;
        int end_idx = 0;
        int res = 0;
        while (start_idx < intervals.length) {
            if (start[start_idx] < end[end_idx]) {
                res++;
                start_idx++;
```

```
            } else {
                start_idx++;
                end_idx++;
            }
        }
        return res;
    }
}
```

2. priority queue to get the earliest meeting end time.

257. Binary Tree Paths

261. Graph Valid Tree

1. union find

```
class Solution {
    public boolean validTree(int n, int[][] edges) {
        int[] father = new int[n];
        for (int i = 0; i < n; i++) {
            father[i] = i;
        }
        for (int[] e : edges) {
            int f1 = find(father, e[0]);
            int f2 = find(father, e[1]);
            if (f1 == f2) {
                return false;
            }
            father[f1] = father[f2];
        }
        return edges.length == n - 1;
```

```
    }
    int find(int[] nums, int i) {
        while (i != nums[i]) {
            nums[i] = nums[nums[i]];
            i = nums[i];
        }
        return i;
    }
}
```

2. dfs or bfs

273. Integer to English Words

1. recursion with hardcoded

277. Find the Celebrity

1. O(n) time 2 passes

278. First Bad Version

28. Implement strStr()

282. Expression Add Operators

1. DFS, remember to carry a number to reduce from when doing multiplying

283 Move Zeros

285 Inorder Sucessor in BST

29 Divide Two Integers

use bits manipulation, watch out for overflow

```
class Solution {
    public int divide(int dividend, int divisor) {
        if (divisor == 0 || (dividend == Integer.MIN_VALUE &&
divisor == -1)) {
            return Integer.MAX_VALUE;
        }
        int sign = 1;
        if ((dividend < 0 && divisor > 0) || (dividend > 0 &&
divisor < 0)) {
            sign = -1;
        }
        long ddd = Math.abs((long)dividend);
        long dvs = Math.abs((long)divisor);
        int res = 0;

        while (dvs <= ddd) {
            long temp = dvs;
            long mul = 1;
            while (ddd >= temp << 1) {
                temp <<= 1;
                mul <<= 1;
            }
            ddd -= temp;
            res += mul;
        }
        return sign * res;
    }
}
```

295. Find Median from Data Stream

297. Serialize and Deserialize Binary Tree

use preorder and when deserialize, use a queue
comparing to 449. Serialize and Deserialize BST, all elements on the left can be smaller than the root

300. Longest Increasing Subsequence

A binary search trick, use binary search to update the length of the array, time complexity is O(n logn)