# Table of Contents

50. Smallest Rectangle Enclosing Black Pixels 1.50
51. Word Pattern II 1.51
52. Unique Word Abbreviation 1.52
53. Zigzag Iterator 1.53
54. Wiggle Sort 1.54
55. Strobogrammatic Number III 1.55
56. Number of Digit One 1.56
57. Valid Number 1.57
58. Longest Substring Without Repeating Characters 1.58
59. Maximum Product of Word Lengths 1.59
60. Palindrome Linked List 1.60
61. Subsets 1.61
62. Binary Tree Maximum Path Sum 1.62
63. Find Minimum in Rotated Sorted Array 1.63
64. Decode Ways 1.64
65. Unique Binary Search Trees II 1.65
66. Largest BST Subtree 1.66
67. Word Break 1.67
68. Word Break II 1.68
69. Word Search II 1.69
70. Rotate List 1.70
71. Coins in a Line II 1.71
72. Best Time to Buy and Sell Stock 1.72
73. Best Time to Buy and Sell Stock II 1.73
74. Best Time to Buy and Sell Stock III\IV 1.74
75. Merge k sorted lists 1.75
76. Binary Tree Paths 1.76
77. Reverse Nodes in k-Group 1.77
78. Sort Colors 1.78
79. Edit String to Palindrome 1.79
80. Patching Array 1.80
81. Number of Connected Components in an Undirected Graph 1.81
82. The Skyline Problem 1.82
83. Regular Expression Matching 1.83
84. In-order Traversal 1.84
85. Max XOR in an Array 1.85
86. The Maze III 1.86

# Introduction

# LeetCode Note

2333

# Validate Binary Search Tree

# Validate Binary Search Tree

**Threaded Binary Tree**

```
    public boolean isValidBST(TreeNode root) {

        TreeNode prev = null, curr = root, preNode = null, curNode =

        while(curr != null){
            if(curr.left == null){

                preNode = curNode;
                curNode = curr;
                if(preNode != null && preNode.val >= curNode.val) re

                curr = curr.right;
            }else{
                // find predecessor
                prev = curr.left;
                while(prev.right != null && prev.right != curr)
                    prev = prev.right;

                if(prev.right == null){
                    prev.right = curr;
                    curr = curr.left;
                }else{
                    preNode = curNode;
                    curNode = curr;
                    if(preNode != null && preNode.val >= curNode.val

                    prev.right = null;
                    curr = curr.right;
                }
            }
        }

        return true;
    }
```

**Recursive**

```
    public boolean isValidBST(TreeNode root) {
        return helper(root, null, null);
    }

    private boolean helper(TreeNode root, Integer min, Integer max){
        return root == null || ((min == null || root.val > min) && (r
    }
```

# Isomorphic Strings

# Isomorphic Strings

**hash table** and **map** to record whether the char in s has a isomorphic char and char in t has been mapped.

```
public boolean isIsomorphic(String s, String t) {
        Map<Character, Character> map = new HashMap();
        boolean[] mapped = new boolean[256];
        char[] sl = s.toCharArray(), tl = t.toCharArray();

        if(sl.length != tl.length) return false;

        for(int i = 0 ; i < sl.length ; i++){
            if(map.get(sl[i]) == null){
                if(mapped[tl[i]]) return false;
                map.put(sl[i], tl[i]);
                mapped[tl[i]] = true;
            }else{
                if(map.get(sl[i]) != tl[i]) return false;
            }
        }

        return true;
    }
```

# Read N Characters Given Read4 II - Call multiple times

# Read N Characters Given Read4 II - Call multiple times

The API: int read4(char *buf) reads 4 characters at a time from a fi

The return value is the actual number of characters read. For exampl

By using the read4 API, implement the function int read(char *buf, i

---

Keep two global various **hasReadLen** and **buffStr**, which record the read data length and buffer.

Read buf as Problem 1, but keep the char into buffStr, and each time read next char from buffStr when **index < n** and **index + hasReadLen < buffStr.length**

```
/* The read4 API is defined in the parent class Reader4.
      int read4(char[] buf); */

public class Solution extends Reader4 {
    /**
     * @param buf Destination buffer
     * @param n   Maximum number of characters to read
     * @return    The number of characters read
     */

    private int hasReadLen = 0;

    private String buffStr = "";

    public int read(char[] buf, int n) {

        int idx = 0;
        boolean eof = false;

        while(idx < n){

            char [] tmp = new char[4];

            if(eof) break;

            if(read4(tmp) == 0) eof = true;

            for(char c : tmp){
                if(c != 0x00)
                    buffStr += String.valueOf(c);
```

```
            }

            while(idx < n && idx + hasReadLen < buffStr.length()){
                buf[idx] = buffStr.charAt(idx + hasReadLen);
                idx ++;
            }

        }

        hasReadLen += idx;

        return idx;
    }
}
```

# Read N Characters Given Read4

# Read 4

```
The API: int read4(char *buf) reads 4 characters at a time from a fi

The return value is the actual number of characters read. For exampl

By using the read4 API, implement the function int read(char *buf, i
```

---

Create a 4-length char array buffer to keep read data.

Read data until get n size or buf is empty (size == 0), put the data which in tmp into buf.

```
public int read(char[] buf, int n) {

        int idx = 0;
        char [] tmp = new char[4];

        while(idx < n){

            int size = read4(tmp), i = 0;

            if(size == 0) break;

            while(idx < n && i < size){
                buf[idx ++] = tmp[i ++];
            }

        }

        return idx;
    }
```

# Number of Islands II

# Number of Islands II

A 2d grid map of m rows and n columns is initially filled with water. We may perform an addLand operation which turns the water at position (row, col) into a land. Given a list of positions to operate, count the number of islands after each addLand operation. An island is surrounded by water and is formed by connecting adjacent lands horizontally or vertically. You may assume all four edges of the grid are all surrounded by water.

Example:

Given m = 3, n = 3, positions = [[0,0], [0,1], [1,2], [2,1]]. Initially, the 2d grid grid is filled with water. (Assume 0 represents water and 1 represents land).

```
0 0 0
0 0 0
0 0 0
```

Operation #1: addLand(0, 0) turns the water at grid[0][0] into a land.

```
1 0 0
0 0 0   Number of islands = 1
0 0 0
```

Operation #2: addLand(0, 1) turns the water at grid[0][1] into a land.

```
1 1 0
0 0 0   Number of islands = 1
0 0 0
```

Operation #3: addLand(1, 2) turns the water at grid[1][2] into a land.

```
1 1 0
0 0 1   Number of islands = 2
0 0 0
```

Operation #4: addLand(2, 1) turns the water at grid[2][1] into a land.

```
1 1 0
0 0 1   Number of islands = 3
0 1 0
```

We return the result as an array: [1, 1, 2, 3]

---

**Union found**

```
    public List<Integer> numIslands2(int m, int n, int[][] positions
        List<Integer> res = new ArrayList();
        if(m == 0 || n == 0 || positions.length == 0) return res;
```

```
        int union[] = new int[m * n];
        int size = 0;

        Arrays.fill(union, -1);

        for(int []pos : positions){
            int i = pos[0], j = pos[1];
            union[i * n + j] = i * n + j;
            size += findUnion(union, i, j, m, n);
            res.add(size);
        }

        return res;
    }

    private int[] x = {-1,1,0,0};
    private int[] y = {0,0,-1,1};

    private int findUnion(int [] union, int i, int j, int m, int n){

        int diff = 1;

        for(int dir = 0; dir < 4; dir ++){

            int target = i * n + j, curr = (i + x[dir]) * n + j + y[

            if(i >= - x[dir] && i < m - x[dir] && j >= - y[dir] && j

                while(union[target] != target){
                    target = union[target];
                }

                while(union[curr] != curr){
                    curr = union[curr];
                }

                if(target != curr){
                    union[curr] = target;
                    diff --;
                }
            }
        }

        return diff;
    }
```

# Missing Ranges

# Missing Ranges

Given a sorted integer array where the range of elements are [lower,

For example, given [0, 1, 3, 50, 75], lower = 0 and upper = 99, retu

---

```
    public List<String> findMissingRanges(int[] nums, int lower, int

        List<String> res = new ArrayList();
        if(nums.length == 0){
            if(lower == upper)
                res.add(lower + "");
            else res.add(lower + "->" + upper);
            return res;
        }

        if(lower < nums[0])
            if(nums[0] - 1 == lower)
                res.add(lower +"");
            else res.add(lower + "->" + (nums[0] - 1));

        for(int i = 1; i < nums.length; i++){
            int diff = nums[i] - nums[i - 1];
            if(diff > 1){
                if(diff == 2)
                    res.add((nums[i - 1] + 1) + "");
                else res.add((nums[i - 1] + 1) + "->" + (nums[i] - 1
            }
        }

        if(upper > nums[nums.length - 1])
            if(nums[nums.length - 1] + 1 == upper)
                res.add(upper + "");
            else res.add((nums[nums.length - 1] + 1) + "->" + upper)

        return res;
    }
```

# Sparse Matrix Multiplication

# Sparse Matrix Multiplication

```
Given two sparse matrices A and B, return the result of AB.

You may assume that A's column number is equal to B's row number.

Example:

A = [
  [ 1, 0, 0],
  [-1, 0, 3]
]

B = [
  [ 7, 0, 0 ],
  [ 0, 0, 0 ],
  [ 0, 0, 1 ]
]


     |  1 0 0 |   | 7 0 0 |   |  7 0 0 |
AB = | -1 0 3 | x | 0 0 0 | = | -7 0 3 |
                  | 0 0 1 |
```

---

Record all 0 row/col to avoid useless multiplication

```java
    public int[][] multiply(int[][] A, int[][] B) {

        int [][]res = new int[A.length][B[0].length];
        boolean []rowA = new boolean[A.length];
        boolean []colB = new boolean[B[0].length];

        for(int i = 0 ; i < A.length; i++){
            for(int j = 0 ; j < A[0].length; j++)
                if(A[i][j] != 0){
                    rowA[i] = true;
                    break;
                }
        }
        for(int j = 0 ; j < B[0].length; j++){
            for(int i = 0 ; i < B.length; i++)
                if(B[i][j] != 0){
                    colB[j] = true;
                    break;
                }
        }
```

```
        for(int i = 0 ; i < A.length; i ++){
            for(int k = 0 ; k < B[0].length ; k ++){
                if(!rowA[i] || !colB[k]){
                    res[i][k] = 0;
                    continue;
                }

                int sum = 0;
                for(int j = 0 ; j < A[0].length; j ++){
                    sum += A[i][j] * B[j][k];
                }
                res[i][k] = sum;
            }
        }
        return res;
    }
```

# Range Sum Query 2D - Mutable

# Range Sum Query 2D - Mutable

```
Given a 2D matrix matrix, find the sum of the elements inside the rec

Range Sum Query 2D
The above rectangle (with the red border) is defined by (row1, col1)

Example:
Given matrix = [
  [3, 0, 1, 4, 2],
  [5, 6, 3, 2, 1],
  [1, 2, 0, 1, 5],
  [4, 1, 0, 1, 7],
  [1, 0, 3, 0, 5]
]

sumRegion(2, 1, 4, 3) -> 8
update(3, 2, 2)
sumRegion(2, 1, 4, 3) -> 10
Note:
The matrix is only modifiable by the update function.
You may assume the number of calls to update and sumRegion function i
You may assume that row1 ≤ row2 and col1 ≤ col2.
```

---

**Binary Indexed Tree, O(logm) + O(logn)**

```java
public class NumMatrix {

    int[][] tree;
    int[][] nums;
    int m;
    int n;

    public NumMatrix(int[][] matrix) {
        if (matrix.length == 0 || matrix[0].length == 0) return;
        m = matrix.length;
        n = matrix[0].length;
        tree = new int[m+1][n+1];
        nums = new int[m][n];
        for (int i = 0; i < m; i++) {
            for (int j = 0; j < n; j++) {
                update(i, j, matrix[i][j]);
            }
        }
    }

    public void update(int row, int col, int val) {
```

```
        if (m == 0 || n == 0) return;
        int delta = val - nums[row][col];
        nums[row][col] = val;
        for (int i = row + 1; i <= m; i += i & (-i)) {
            for (int j = col + 1; j <= n; j += j & (-j)) {
                tree[i][j] += delta;
            }
        }
    }

    public int sumRegion(int row1, int col1, int row2, int col2) {
        if (m == 0 || n == 0) return 0;
        return sum(row2+1, col2+1) + sum(row1, col1) - sum(row1, col1
    }

    public int sum(int row, int col) {
        int sum = 0;
        for (int i = row; i > 0; i -= i & (-i)) {
            for (int j = col; j > 0; j -= j & (-j)) {
                sum += tree[i][j];
            }
        }
        return sum;
    }
}
```

**Keep row sum (dp), O(m) + O(n)**

```
public class NumMatrix {
    private int[][] rowSums;
    private int[][] matrix;

    public NumMatrix(int[][] matrix) {
        if(   matrix           == null
           || matrix.length    == 0
           || matrix[0].length == 0   ){
            return;
         }

         this.matrix = matrix;

         int m   = matrix.length;
         int n   = matrix[0].length;
         rowSums = new int[m][n];
         for(int i = 0; i < m; i++){
            for(int j = 0; j < n; j++){
                if(j == 0)
                    rowSums[i][j] = matrix[i][j];
                else rowSums[i][j] = rowSums[i][j - 1] + matrix[i][j
            }
         }
    }
    //time complexity for the worst case scenario: O(m)
```

```
    public void update(int row, int col, int val) {
        for(int i = col; i < matrix[0].length; i++){
            rowSums[row][i] = rowSums[row][i] - matrix[row][col] + va
        }

        matrix[row][col] = val;
    }
    //time complexity for the worst case scenario: O(n)
    public int sumRegion(int row1, int col1, int row2, int col2) {
        int ret = 0;

        for(int j = row1; j <= row2; j++){
            if(col1 == 0) ret += rowSums[j][col2];
            else ret += rowSums[j][col2] - rowSums[j][col1 - 1];
        }

        return ret;
    }
}
```

# 3 Sum Smaller

# 3 Sum Smaller

```
Given an array of n integers nums and a target, find the number of i

For example, given nums = [-2, 0, 1, 3], and target = 2.

Return 2. Because there are two triplets which sums are less than 2:

[-2, 0, 1]
[-2, 0, 3]
Follow up:
Could you solve it in O(n2) runtime?
```

---

Keep each two sum in the array, and use one loop to find sum which is smaller than target.

```
for i from 0 -> n - 1
    for j from i + 1 -> n
        map -> a[i] + a[j]

for map from 0 -> n
    for k from 0 -> n
        if map + k < target && k is not in map
            counter ++
```

---

**Sort, Two pointer** This is O(n^3), not efficiency.

```
    public int threeSumSmaller(int[] nums, int target) {
        if(nums.length < 3) return 0;
        Arrays.sort(nums);
        int counter = 0;
        for(int i = 0 ; i < nums.length - 2; i++){
            int b = i + 1, e = i + 2;
            while(b < nums.length - 1){
                if(e < nums.length && nums[i] + nums[b] + nums[e] < 
                    e ++;
                    counter ++;
                    continue;
                }
                e = ++ b + 1;
            }
        }
        return counter;
    }
```

# Alien Dictionary

# Alien Dictionary

```
There is a new alien language which uses the latin alphabet. However,

For example,
Given the following words in dictionary,

[
  "wrt",
  "wrf",
  "er",
  "ett",
  "rftt"
]
The correct order is: "wertf".

Note:
You may assume all letters are in lowercase.
If the order is invalid, return an empty string.
There may be multiple valid order of letters, return any one of them
```

---

**Topology Sort, BFS**

```
    public String alienOrder(String[] words) {

        if(words == null || words.length == 0)
            return "";
        Map<Character, Set<Character>> map = new HashMap<Character, S
        Map<Character, Integer> degree = new HashMap<Character, Integ
        StringBuilder sb = new StringBuilder();

        // put all word in-degree 0
        for(String s: words){
            for(char c: s.toCharArray()){
                degree.put(c,0);
            }
        }

        // compare each word and its pre-word char by char,
        // if different, since c1 is in front of c2, put c2 into c1's
        for(int i=0; i < words.length-1; i++){
            String cur = words[i];
            String next = words[i+1];
            // using longer one
            int length = Math.min(cur.length(), next.length());
            for(int j=0; j<length; j++){
                char c1=cur.charAt(j);
```

```
                char c2=next.charAt(j);
                if(c1!=c2){
                    Set<Character> set = map.getOrDefault(c1, new Has
                    if(!set.contains(c2)){
                        set.add(c2);
                        map.put(c1, set);
                        degree.put(c2, degree.get(c2) + 1);
                    }
                    break;
                }
            }
        }

        // topological sort via BFS
        Queue<Character> q = new LinkedList();
        // put all 0 in-degree into queue
        for(char c: degree.keySet()){
            if(degree.get(c) == 0) q.add(c);
        }
        while(!q.isEmpty()){
            char c = q.poll();
            sb.append(c);
            if(map.containsKey(c)){
                for(char c2: map.get(c)){
                    // all next chars' in-degree abstract 1
                    degree.put(c2,degree.get(c2) - 1);
                    if(degree.get(c2) == 0) q.add(c2);
                }
            }
        }
        if(sb.length() != degree.size()) return "";

        return sb.toString();
    }
```

# Best Meeting Point

# Best Meeting Point

A group of two or more people wants to meet and minimize the total t

For example, given three people living at (0,0), (0,4), and (2,2):

```
1 - 0 - 0 - 0 - 1
|   |   |   |   |
0 - 0 - 0 - 0 - 0
|   |   |   |   |
0 - 0 - 1 - 0 - 0
```

The point (0,2) is an ideal meeting point, as the total travel dista

---

```java
    public int minTotalDistance(int[][] grid) {
        int m = grid.length;
        int n = grid[0].length;

        List<Integer> I = new ArrayList<>(m);
        List<Integer> J = new ArrayList<>(n);

        for(int i = 0; i < m; i++){
            for(int j = 0; j < n; j++){
                if(grid[i][j] == 1){
                    I.add(i);
                    J.add(j);
                }
            }
        }

        return getMin(I) + getMin(J);
    }

    private int getMin(List<Integer> list){
        int ret = 0;

        Collections.sort(list);

        int i = 0;
        int j = list.size() - 1;
        while(i < j){
            ret += list.get(j--) - list.get(i++);
        }

        return ret;
    }
```

# Binary Tree Longest Consecutive Sequence

# Binary Tree Longest Consecutive Sequence

```
Given a binary tree, find the length of the longest consecutive sequ

The path refers to any sequence of nodes from some starting node to a

For example,
   1
    \
     3
    / \
   2   4
        \
         5
Longest consecutive sequence path is 3-4-5, so return 3.
   2
    \
     3
    /
   2
  /
 1
Longest consecutive sequence path is 2-3,not3-2-1, so return 2.
```

---

private int maxCount = 0;

```
```

# Meeting Rooms

# Meeting Rooms

Given an array of meeting time intervals consisting of start and end

For example,
Given [[0, 30],[5, 10],[15, 20]],
return false.

---

**Sort**

```
public boolean canAttendMeetings(Interval[] intervals) {
        if(intervals.length<2) return true;
        Arrays.sort(intervals, (Interval i1, Interval i2) -> i1.star
        for(int i=1;i<intervals.length;i++){
            if(intervals[i].start<intervals[i-1].end)
                return false;
        }
        return true;
    }
```

# Meeting Rooms II

# Meeting Rooms II

```
Given an array of meeting time intervals consisting of start and end

For example,
Given [[0, 30],[5, 10],[15, 20]],
return 2.
```

---

**Greedy, Sort**

```
    public int minMeetingRooms(Interval[] intervals) {
        if(intervals.length == 0) return 0;

        int min = 0;
        Set<Interval> set = new LinkedHashSet();
        Arrays.sort(intervals, new Comparator<Interval>(){
            public int compare(Interval i1, Interval i2){
                if(i1.start == i2.start) return 0;
                else return i1.start > i2.start ? 1 : -1;
            }
        });

        for(Interval i : intervals)
            set.add(i);

        while(!set.isEmpty()){
            min ++;
            Iterator<Interval> iter = set.iterator();
            Interval head = iter.next();
            Interval prev = head;
            while(iter.hasNext()){
                Interval curr = iter.next();
                if(prev.end <= curr.start){
                    iter.remove();
                    prev = curr;
                }
            }
            set.remove(head);
        }

        return min;
    }
```

# Invert Binary Tree

# Invert Binary Tree

^_^

---

```
    public TreeNode invertBT(TreeNode root){
        if(root != null) {
            TreeNode tmp = invertBT(root.left);
            root.left = invertBT(root.right);
            root.right = tmp;
        }
        return root;
    }
```

# Remove Duplicate Letters

# Remove Duplicate Letters

Given a string which contains only lowercase letters, remove duplicate letters so that every letter appear once and only once. You must make sure your result is the smallest in lexicographical order among all possible results.

Example: Given "bcabc" Return "abc"

Given "cbacdcbc" Return "acdb"

---

**Stack**

```
    public String removeDuplicateLetters(String str) {
        int[] res = new int[26]; //will contain number of occurences
        boolean[] visited = new boolean[26]; //will contain if chara
        char[] ch = str.toCharArray();
        for(char c: ch){  //count number of occurences of character
            res[c-'a']++;
        }
        Stack<Character> st = new Stack<>(); // answer stack
        int index;
        for(char s:ch){
            index= s-'a';
            res[index]--;   //decrement number of characters remaini
            if(visited[index]) //if character is already present in s
                continue;
            //if current character is smaller than last character in
            //it can be removed and  added later e.g stack = bc rema
            while(!st.isEmpty() && s<st.peek() && res[st.peek()-'a']
                visited[st.pop()-'a']=false;
            }
            st.push(s); //add current character and mark it as visit
            visited[index]=true;
        }

        StringBuilder sb = new StringBuilder();
        //pop character from stack and build answer string from back
        while(!st.isEmpty()){
            sb.insert(0,st.pop());
        }
        return sb.toString();
    }
```

# Is Symmetric Tree

# Is Symmetric Tree

^_^

---

```
    public boolean isSymmetric(TreeNode root) {
        if(root == null) return true;
        else return isSymmetric(root.left, root.right);
    }

    private boolean isSymmetric(TreeNode left, TreeNode right){
        if(left == null || right == null) return left == right;
        else return left.val == right.val && isSymmetric(left.left,
    }
```

# Balanced Binary Tree

# Balanced Binary Tree

^_^

---

**Recursive**

```
    public boolean isBalanced(TreeNode root) {
        return helper(root) != -1;
    }

    private int helper(TreeNode root){
        if(root == null) return 0;
        else {
            int left = helper(root.left), right = helper(root.right)
            if(left == -1 || right == -1) return -1;
            else return Math.abs(left - right) <= 1 ? Math.max(left,
        }
    }
```

**Stack**

# Max Points on a Line

# Max Points on a Line

Given n points on a 2D plane, find the maximum number of points that lie on the same straight line.

---

Select a point and find all other points which has the same slope. Use a **hash map** to keep the same slope number.

Be care for the overlap point and same x value points.

```
    public int maxPoints(Point[] points) {

        if (points.length < 3) return points.length;

        int max = 0;
        Map<Double, Integer> map = new HashMap();
        // find same slope of with a target point
        for (int i = 0; i < points.length; i++) {
            int overlap = 1;
            for(int j = 0 ; j < points.length; j++){
                if (i == j) continue;
                double slope = 0.0;
                // same x points, y / 0 will error
                if (points[i].x == points[j].x) {
                    if(points[i].y == points[j].y){
                        overlap ++; continue;
                    }
                    else slope = Integer.MAX_VALUE;
                } else {
                    slope = 1.0 * (points[i].y - points[j].y) / (poi
                }
                map.put(slope, map.containsKey(slope) ? map.get(slop
            }
            if(map.keySet().size() == 0) max = Math.max(overlap, max
            for (double key : map.keySet()) {
                max = Math.max(overlap + map.get(key), max);
            }

            map.clear();
        }

        return max;
    }
```

```
def maxPoints(self,points):
    res = 0
    for p in points:
        dict = collections.defaultdict(int)
```

```
        itself = 0
        for q in points:
            x,y = q.x - p.x, q.y - p.y
            if x:
                dict[float(y)/x] += 1
            else:
                if y:
                    dict[float('inf')] += 1
                else:
                    itself += 1
        res = max(res,max(dict.values())+itself if dict.values() else
    return res
```

# Binary Tree Upside Down

# Binary Tree Upside Down

```
Given a binary tree where all the right nodes are either leaf nodes \

For example:
Given a binary tree {1,2,3,4,5},
    1
   / \
  2   3
 / \
4   5
return the root of the binary tree [4,5,2,#,#,3,1].
   4
  / \
 5   2
    / \
   3   1
```

---

Left child as root, root as right-most child's right child, right child as right-most child's left child.

```
    public TreeNode upsideDownBinaryTree(TreeNode root) {
        if(root == null) return null;
        else if(root.left == null && root.right == null) return root

        TreeNode leftNode = upsideDownBinaryTree(root.left), rightNo
        root.left = null;
        root.right = null;
        if(currNode != null){
            while(currNode.right != null)
                currNode = currNode.right;
            currNode.right = root;
            currNode.left = rightNode;
            return leftNode;
        }
        else return root;
    }

    public TreeNode upsideDownBinaryTree(TreeNode root) {

        if(root == null || (root.left == null && root.right == null)

        TreeNode left = upsideDownBinaryTree(root.left), t = left;

        while(t.right != null)
            t = t.right;

        t.left = root.right;
```

```
        root.right = null;

        t.right = root;
        root.left = null;

        return left;
    }
```

iterative way:

```
def upsideDownBinaryTree(self,root):
    if not root: return root
    prev,prev_right = None,None
    while root:
        root.left,root.right,prev,prev_right,root = prev_right,prev,
    return prev
```

# Walls and Gates

# Walls and Gates

```
You are given a m x n 2D grid initialized with these three possible v

-1 - A wall or an obstacle.
0 - A gate.
INF - Infinity means an empty room. We use the value 231 - 1 = 214748
Fill each empty room with the distance to its nearest gate. If it is

For example, given the 2D grid:
INF  -1  0  INF
INF INF INF  -1
INF  -1 INF  -1
  0  -1 INF INF

After running your function, the 2D grid should be:
  3  -1   0   1
  2   2   1  -1
  1  -1   2  -1
  0  -1   3   4
```

---

**DFS**

```
    public void wallsAndGates(int[][] rooms) {
        if(rooms.length == 0) return ;

        boolean [][] visited = new boolean[rooms.length][rooms[0].le

        for(int i = 0 ; i < rooms.length; i++)
            for(int j = 0 ; j < rooms[0].length; j ++)
                if(rooms[i][j] == 0)
                    dfs(rooms, visited, i, j, 0);
    }

    private void dfs(int [][] rooms, boolean visited[][], int i, int

        if(i < 0 || j < 0 || i >= rooms.length || j >= rooms[0].leng

        //when meet a root value is smaller than current dist, jump
        //Because no matter how to move, the current dist will alway
        if(rooms[i][j] < dist) return ;

        visited[i][j] = true;

        rooms[i][j] = dist;

        dfs(rooms,visited, i - 1, j, dist + 1);
```

```
        dfs(rooms,visited, i + 1, j, dist + 1);
        dfs(rooms,visited, i, j - 1, dist + 1);
        dfs(rooms,visited, i, j + 1, dist + 1);

        visited[i][j] = false;

        return;
    }
```

# Closest Binary Search Tree Value

# Closest Binary Search Tree Value

Given a non-empty binary search tree and a target value, find the value in the BST that is closest to the target.

---

**Binary Search**

```
    public int closestValue(TreeNode root, double target) {
        if(root == null) throw new RuntimeException("Error");

        int res = root.val;

        double diff = (double) root.val - target;

        while(root != null){

            double currDiff = (double) root.val - target;

            if(Math.abs(diff) > Math.abs(currDiff))
                res = root.val;

            diff = currDiff;

            if(diff == 0.0) return target;
            else if(diff > 0.0)
                root = root.left;
            else root = root.right;
        }

        return res;
    }
```

**Stack**

```
    public int closestValue(TreeNode root, double target) {
        Stack <TreeNode> pred = new Stack(), succ = new Stack();
        while(root != null){
            if(root.val <= target){
                pred.push(root);
                root = root.right;
            }
            else{
                succ.push(root);
                root = root.left;
            }
        }
```

```
        return pred.isEmpty() ? succ.peek().val : succ.isEmpty() ? pr
    }
```

# Closest Binary Search Tree Value II

# Closest Binary Search Tree Value II

Given a non-empty binary search tree and a target value, find k values in the BST that are closest to the target.

---

**Two Stack**

```
    public List<Integer> closestKValues(TreeNode root, double target,
        List<Integer> list = new ArrayList<>();
        Stack<TreeNode> pred = new Stack<>(), succ = new Stack<>();
        initStack(pred, succ, root, target);
        while(k-- > 0){
            if(succ.isEmpty() || !pred.isEmpty() && target - pred.pe
                list.add(pred.peek().val);
                getPredecessor(pred);
            }
            else{//Since N > k, always have something to add
                list.add(succ.peek().val);
                getSuccessor(succ);
            }
        }
        return list;
    }

    // find closet elements until Null.
    // the elements in the top of two stacks must be the predecessor
    private void initStack(Stack<TreeNode> pred, Stack<TreeNode> succ
        while(root != null){
            if(root.val <= target){
                pred.push(root);
                root = root.right;
            }
            else{
                succ.push(root);
                root = root.left;
            }
        }
    }

    // put all node on path into predecessor stack
    private void getPredecessor(Stack<TreeNode> st){
        TreeNode node = st.pop();
        if(node.left != null){
            st.push(node.left);
            while(st.peek().right != null)  st.push(st.peek().right)
        }
    }
```

```
// put all node on path into successor stack
private void getSuccessor(Stack<TreeNode> st){
    TreeNode node = st.pop();
    if(node.right != null){
        st.push(node.right);
        while(st.peek().left != null)   st.push(st.peek().left);
    }
}
```

# Encode and Decode Strings

# Encode and Decode Strings

```
Design an algorithm to encode a list of strings to a string. The enc

Machine 1 (sender) has the function:

string encode(vector<string> strs) {
  // ... your code
  return encoded_string;
}
Machine 2 (receiver) has the function:
vector<string> decode(string s) {
  //... your code
  return strs;
}
So Machine 1 does:

string encoded_string = encode(strs);
and Machine 2 does:

vector<string> strs2 = decode(encoded_string);
strs2 in Machine 2 should be the same as strs in Machine 1.

Implement the encode and decode methods.
```

---

Format: **length%{data}**

```java
public class Codec {

    public String encode(List<String> strs) {
        StringBuffer result = new StringBuffer();

        if(strs == null || strs.size() == 0)
            return result.toString();

        for(String str: strs){
            result.append(str.length());
            result.append("%");
            result.append(str);
        }

        return result.toString();
    }

    // Decodes a single string to a list of strings.
    public List<String> decode(String s) {
        List<String> result = new ArrayList();
```

```
        if(s == null || s.length() == 0)
            return result;

        int current = 0;
        while(true){
            if(current == s.length())
                break;
            StringBuffer sb = new StringBuffer();
            while(s.charAt(current) != '%'){
                sb.append(s.charAt(current));
                current++;
            }
            int len = Integer.parseInt(sb.toString());
            int end = current + 1 + len;
            result.add(s.substring(current+1, end));
            current = end;
        }
        return result;
    }
}
```

# Find the Celebrity

# Find the Celebrity

Suppose you are at a party with n people (labeled from 0 to n - 1) and among them, there may exist one celebrity. The definition of a celebrity is that all the other n - 1 people know him/her but he/she does not know any of them.

Now you want to find out who the celebrity is or verify that there is not one. The only thing you are allowed to do is to ask questions like: "Hi, A. Do you know B?" to get information of whether A knows B. You need to find out the celebrity (or verify there is not one) by asking as few questions as possible (in the asymptotic sense).

You are given a helper function bool knows(a, b) which tells you whether A knows B. Implement a function int findCelebrity(n), your function should minimize the number of calls to knows.

Note: There will be exactly one celebrity if he/she is in the party. Return the celebrity's label if there is a celebrity in the party. If there is no celebrity, return -1.

---

```
    def findCelebrity(self, n):
        """
        :type n: int
        :rtype: int
        """
        x = 0
        for i in xrange(n): # find the possible candidate x
            if knows(x,i):
                x = i
        for k in xrange(n): # check if everybody knows x
            if not knows(k,x):
                return -1
        for j in xrange(x): # check if he knows anybody.
            if knows(x,j):
                return -1
        return x
```

# Graph Valid Tree

# Graph Valid Tree

Given n nodes labeled from 0 to n - 1 and a list of undirected edges (each edge is a pair of nodes), write a function to check whether these edges make up a valid tree.

For example:

Given n = 5 and edges = [[0, 1], [0, 2], [0, 3], [1, 4]], return true.

Given n = 5 and edges = [[0, 1], [1, 2], [2, 3], [1, 3], [1, 4]], return false.

No cycle, Connected undirected graph.

---

**Union found**

```
    public boolean validTree(int n, int[][] edges) {
        if(n <= 1) return true;

        int [] union = new int[n];
        int unionNumber = n;

        Arrays.fill(union, -1);

        for(int []edge : edges){

            if(edge[0] > edge[1]) swap(edge, 0, 1);
            int e1 = edge[0], e2 = edge[1];

            while(union[e1] != -1)
                e1 = union[e1];
            while(union[e2] != -1)
                e2 = union[e2];

            if(e1 == e2) return false;

            union[e2] = e1;
            unionNumber --;
        }

        return unionNumber == 1;
    }

    private void swap(int []nums, int i, int j){
        int tmp = nums[i];
        nums[i] = nums[j];
        nums[j] = tmp;
    }
```

# Group Shifted Strings

# Group Shifted Strings

```
Given a string, we can "shift" each of its letter to its successive l

"abc" -> "bcd" -> ... -> "xyz"
Given a list of strings which contains only lowercase alphabets, grou

For example, given: ["abc", "bcd", "acef", "xyz", "az", "ba", "a", "z
Return:

[
  ["abc","bcd","xyz"],
  ["az","ba"],
  ["acef"],
  ["a","z"]
]
Note: For the return value, each inner list's elements must follow th
```

---

Use **hash table** to keep the strings that shift all elements with length of the first element to 'a' (e.g. 'c' -> 'a' is 2)

```
    public List<List<String>> groupStrings(String[] strings) {

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

        if(strings.length == 0) return res;

        HashMap<String, List<String>> map = new HashMap();

        for(String word : strings){

            String key = "";

            int diff = word.charAt(0) - 'a';

            for(int i = 1; i < word.length(); i++){
                key += (word.charAt(i) - diff + 26) % 26;
            }

            List<String> arr = map.getOrDefault(key ,new ArrayList<St

            arr.add(word);

            map.put(key, arr);
        }

        for(List<String> list : map.values()){
```

```
            Collections.sort(list);
            res.add(list);
        }

        return res;
    }
```

# Inorder Successor in BST

# Inorder Successor in BST

Given a binary search tree and a node in it, find the in-order successor of that node in the BST.

---

**Binary Search**

```
    public TreeNode inorderSuccessor(TreeNode root, TreeNode p) {
        if(root == null || p == null) return null;
        else if(p.right != null){
            // find successor directly
            p = p.right;
            while(p.left != null)
                p = p.left;
            return p;
        }else{

            //binary search to find p and prev node
            TreeNode curr = root, prev = null;

            while(curr != p){
                if(curr.val > p.val){
                    prev = curr;
                    curr = curr.left;
                }else{
                    curr = curr.right;
                }

            }
            return prev;
        }
    }
```

O(n), flat BST to list.

```
private List<TreeNode> flatList = new ArrayList<>();

    public TreeNode inorderSuccessor(TreeNode root, TreeNode p) {
        if(root == null || p == null)
            return null;
        inorderTraversal(root);
        for(int i = 0; i < flatList.size(); i++){
            if(flatList.get(i) == p && i + 1 < flatList.size())
                return flatList.get(i + 1);
        }
        return null;
    }
```

```
    private void inorderTraversal(TreeNode root){
        if(root == null)
            return;
        inorderTraversal(root.left);
        flatList.add(root);
        inorderTraversal(root.right);
    }
```

# Paint House

# Paint House

There are a row of n houses, each house can be painted with one of the three colors: red, blue or green. The cost of painting each house with a certain color is different. You have to paint all the houses such that no two adjacent houses have the same color.

The cost of painting each house with a certain color is represented by a n x 3 cost matrix. For example, costs[0][0] is the cost of painting house 0 with color red; costs[1][2] is the cost of painting house 1 with color green, and so on... Find the minimum cost to paint all houses.

---

**DP**

```
    public int minCost(int[][] costs) {
        if(costs.length == 0) return 0;

        int len = costs.length;

        for(int i = 1 ; i < len ; i++){
            costs[i][0] += Math.min(costs[i - 1][1],costs[i - 1][2])
            costs[i][1] += Math.min(costs[i - 1][0],costs[i - 1][2])
            costs[i][2] += Math.min(costs[i - 1][0],costs[i - 1][1])
        }

        return Math.min(Math.min(costs[len - 1][0], costs[len - 1][1
    }
```

# Binary Tree Longest Consecutive Seq

# Binary Tree Longest Consecutive Seq

```
Given a binary tree, find the length of the longest consecutive sequ

The path refers to any sequence of nodes from some starting node to a

For example,
   1
    \
     3
    / \
   2   4
        \
         5
Longest consecutive sequence path is 3-4-5, so return 3.
   2
    \
     3
    /
   2
  /
 1
Longest consecutive sequence path is 2-3,not3-2-1, so return 2.
```

---

```java
    private int max = 1;
    public int longestConsecutive(TreeNode root){
        Helper(root);
        return max;
    }
    public int Helper(TreeNode root) {
        if(root == null) return 0;
        else if(root.left == null && root.right == null){
            return 1;
        }
        else{
            int left = Helper(root.left), right = Helper(root.right)
            if(root.left != null && root.val != root.left.val - 1){
                left = 0;
            }
            if(root.right != null && root.val != root.right.val - 1)
                right = 0;
            }
            int curMax = Math.max(left + 1, right + 1);
            max = Math.max(max, curMax);
            return curMax;
        }
    }
```

# Count Univalue Subtrees

# Count Univalue Subtrees

```
Given a binary tree, count the number of uni-value subtrees.

A Uni-value subtree means all nodes of the subtree have the same valu

For example:
Given binary tree,
              5
             / \
            1   5
           / \   \
          5   5   5
return 4.
```

---

```
    int sum = 0;

    public int countUnivalSubtrees(TreeNode root) {
        if(Helper(root)) sum ++;
        return sum;
    }

    private boolean Helper(TreeNode root){
        if(root == null) return false;
        else if(root.left == null && root.right == null){
            return true;
        }else{
            boolean isSame = true;
            if(root.left != null) {
                boolean left = Helper(root.left);
                if(left) sum ++;
                if(!left || root.val != root.left.val) isSame = false
            }
            if(root.right != null) {
                boolean right = Helper(root.right);
                if(right) sum ++;
                if(!right || right && root.val != root.right.val) isS
            }
            return isSame;
        }
    }
```

# Factor Combinations

# Factor Combinations

```
Numbers can be regarded as product of its factors. For example,

8 = 2 x 2 x 2;
  = 2 x 4.
Write a function that takes an integer n and return all possible coml

Note:
Each combination's factors must be sorted ascending, for example: Th
You may assume that n is always positive.
Factors should be greater than 1 and less than n.
Examples:
input: 1
output:
[]
input: 37
output:
[]
input: 12
output:
[
  [2, 6],
  [2, 2, 3],
  [3, 4]
]
input: 32
output:
[
  [2, 16],
  [2, 2, 8],
  [2, 2, 2, 4],
  [2, 2, 2, 2, 2],
  [2, 4, 4],
  [4, 8]
]
```

---

```java
    public List<List<Integer>> getFactors(int n) {
        List<List<Integer>> res = new ArrayList();
        if(n <= 1) return res;
        Set<List<Integer>> set = new HashSet();
        Helper(set, new ArrayList(), n);
        res.addAll(set);
        return res;
    }

    private void Helper(Set<List<Integer>> res, List<Integer> comb, 
```

```
        if(n == 1){
            res.add(comb);
        }else{
            int sqrt = (int)Math.sqrt(n);
            for(int i = 2; i <= sqrt ; i++){
                if(n % i == 0){
                    List <Integer> tmp = new ArrayList(comb);
                    tmp.add(i);
                    Helper(res, tmp, n / i);
                    tmp.add(n / i);
                    Collections.sort(tmp);
                    res.add(tmp);
                }
            }
        }
    }
```

# Flatten 2D Vector

# Flatten 2D Vector

```
Implement an iterator to flatten a 2d vector.

For example,
Given 2d vector =

[
  [1,2],
  [3],
  [4,5,6]
]
By calling next repeatedly until hasNext returns false, the order of
```

---

```
public class Vector2D {

    Iterator<List<Integer>> listIter;
    Iterator<Integer> numberIter;

    public Vector2D(List<List<Integer>> vec2d) {
        listIter = vec2d.iterator();
    }

    public int next() {

        return numberIter.next();

    }

    public boolean hasNext() {

        while(numberIter == null || !numberIter.hasNext()){
            if(listIter != null && listIter.hasNext())
                numberIter = listIter.next().iterator();
            else return false;
        }

        return true;
    }
}
```

# Flip Game

# Flip Game

You are playing the following Flip Game with your friend: Given a st

Write a function to compute all possible states of the string after

For example, given s = "++++", after one move, it may become one of

```
[
  "--++",
  "+--+",
  "++--"
]
```

If there is no valid move, return an empty list [].

---

```
    def generatePossibleNextMoves(self, s):
        """
        :type s: str
        :rtype: List[str]
        """
        res = []
        for i in xrange(len(s)-1):
            if s[i] == s[i+1] == "+":
                res.append(s[:i] + "--" + s[i+2:])
        return res
```

# Flip Game ll

# Flip Game ll

You are playing the following Flip Game with your friend: Given a st

Write a function to determine if the starting player can guarantee a

For example, given s = "++++", return true. The starting player can

Follow up:
Derive your algorithm's runtime complexity.

---

```
    public boolean canWin(String s) {
        Set<Integer> set = new HashSet<>();
        for (int i = 0; i + 1< s.length(); i ++) {
            if (s.charAt(i) == s.charAt(i + 1) && s.charAt(i) == '+'
                set.add(i);
            }
        }

        return helper(set);
    }

    public boolean helper(Set<Integer> set) {
        for (int move : set) {

            Set<Integer> newSet = new HashSet<>(set);
            newSet.remove(move);
            boolean hasLeft = newSet.remove(move - 1);
            boolean hasRight = newSet.remove(move + 1);
            if (!helper(newSet)) {
                return true;
            }

            newSet.add(move);
            if (hasLeft) newSet.add(move - 1);
            if (hasRight) newSet.add(move + 1);
        }
        return false;
    }
```

# Longest Substring with At Most Two Distinct Characters

# Longest Substring with At Most Two Distinct Characters

Given a string, find the length of the longest substring T that contains at most 2 distinct characters.

For example, Given s = “eceba”,

T is "ece" which its length is 3.

---

**Two pointers, Hash table**

```
public int lengthOfLongestSubstringTwoDistinct(String s) {
    if(s == null || s.length() == 0)
        return 0;
    Map<Character, Integer> map = new HashMap<>();
    int b = 0, e = 0, max = 1;
    while(e < s.length()){
        char ce = s.charAt(e);
        map.put(ce, map.getOrDefault(ce, 0) + 1);
        while(map.size() > 2){
            char cb = s.charAt(b);
            int amount = map.get(cb);
            if(-- amount > 0){
                map.put(cb, amount);
            }else map.remove(cb);
            b ++;
        }
        max = Math.max(max, e - b + 1);
        e ++;
    }
    return max;
}
```

K Distinct Characters

```
public int lengthOfLongestSubstringKDistinct(String s, int k) {
    if(s == null || s.length() == 0 || k == 0)
        return 0;
    Map<Character, Integer> map = new HashMap<>();
    int b = 0, e = 0, max = 1;
    while(e < s.length()){
        char ce = s.charAt(e);
        map.put(ce, map.getOrDefault(ce, 0) + 1);
        while(map.size() > k){
```

```
            char cb = s.charAt(b);
            int amount = map.get(cb);
            if(-- amount > 0){
                map.put(cb, amount);
            }else map.remove(cb);
            b ++;
        }
        max = Math.max(max, e - b + 1);
        e ++;
    }
    return max;
}
```

# One Edit Distance

# One Edit Distance

Given two strings S and T, determine if they are both one edit distance apart.

---

```
    public boolean isOneEditDistance(String s, String t) {
        if(Math.abs(s.length() - t.length()) > 1 || s.equals(t)) retu

        if(s.length() >= t.length())
            return helper(s, t);
        else return helper(t, s);
    }

    private boolean helper(String s, String t){
        int diff = s.length() - t.length(), si = 0, ti = 0, editTimes
        while(si < s.length() && ti < t.length()){
            if(s.charAt(si) == t.charAt(ti)){
                si ++;
                ti ++;
            }else {
                if(diff > 0)
                    si ++;
                else {
                    si ++;
                    ti ++;
                }
                if(-- editTimes < 0) return false;
            }
        }
        return true;
    }
```

# Palindrome Permutation

# Palindrome Permutation

Given a string, determine if a permutation of the string could form a palindrome.

For example, "code" -> False, "aab" -> True, "carerac" -> True.

---

```
    def canPermutePalindrome(self, s):
        """
        :type s: str
        :rtype: bool
        """
        dict = {}
        for i in range(len(s)):
            if s[i] in dict:
                if dict[s[i]] == 0:
                    dict[s[i]] = 1
                else:
                    dict[s[i]] -= 1
            else:
                dict[s[i]] = 1
        return sum(dict.values()) == 1 or sum(dict.values()) == 0
```

# Palindrome Permutation II

# Palindrome Permutation II

Given a string s, return all the palindromic permutations (without duplicates) of it. Return an empty list if no palindromic permutation could be form.

For example:

Given s = "aabb", return ["abba", "baab"].

Given s = "abc", return [].

---

```
    public List<String> generatePalindromes(String s) {

        List<String> res = new ArrayList();

        if(s.length() == 0) return res;

        int odd = 0;

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

        for(char c : s.toCharArray()){
            int num = map.getOrDefault(c, 0);
            map.put(c, ++ num);
            odd ++;
            if((num & 1) == 0) odd -= 2;
        }

        if(odd > 1) return res;

        Set <String> set = new HashSet();

        Helper(set, map, "",s.length());

        res.addAll(set);
        return res;
    }

    private void Helper(Set <String> res, Map<Character, Integer> ma

        if(tmp.length() > 0 && (len & 1) == 0 && tmp.length() == len
            res.add(tmp + new StringBuilder(tmp).reverse().toString(
            return ;
        }
        Iterator it = map.entrySet().iterator();

        while (it.hasNext()) {
```

```
            Map.Entry pair = (Map.Entry)it.next();
            char c = (char) pair.getKey();
            int num = (int) pair.getValue();

            if(num == 1 && tmp.length() * 2 + 1 == len){
                res.add(tmp + String.valueOf(c) + new StringBuilder(
                return;
            }

            int curr_num = num;
            while(curr_num >= 2){
                curr_num -= 2;
                map.put(c, curr_num);
                Helper(res, map, tmp + String.valueOf(c), len);
            }
            map.put(c, num);
        }

    }
```

# Reverse Words in a String II

# Reverse Words in a String II

Given an input string, reverse the string word by word. A word is defined as a sequence of non-space characters.

The input string does not contain leading or trailing spaces and the words are always separated by a single space.

For example, Given s = "the sky is blue", return "blue is sky the".

Could you do it in-place without allocating extra space?

---

```
    public void reverseWords(char[] s) {
        reverse(s, 0, s.length - 1);
        int left = 0, right = 0;
        while(right < s.length){
            if(s[right] == ' '){
                reverse(s, left, right - 1);
                left = ++ right;
            }else right ++;
        }
        reverse(s, left, right - 1);
    }

    private void reverse(char[] s, int start, int end){
        int i = start, j = end;
        while(i < j){
            char tmp = s[i];
            s[i] = s[j];
            s[j] = tmp;
            i ++;
            j --;
        }
    }
```

# Shortest Distance from All Buildings

# Shortest Distance from All Buildings

```
You want to build a house on an empty land which reaches all buildin

Each 0 marks an empty land which you can pass by freely.
Each 1 marks a building which you cannot pass through.
Each 2 marks an obstacle which you cannot pass through.
The distance is calculated using Manhattan Distance, where distance(

For example, given three buildings at (0,0), (0,4), (2,2), and an ob

1 - 0 - 2 - 0 - 1
|   |   |   |   |
0 - 0 - 0 - 0 - 0
|   |   |   |   |
0 - 0 - 1 - 0 - 0
The point (1,2) is an ideal empty land to build a house, as the tota

Note:
There will be at least one building. If it is not possible to build
```

---

**BFS**

```java
public int shortestDistance(int[][] grid) {
    int row = grid.length;
    if (row == 0) {
        return -1;
    }
    int col = grid[0].length;
    int[][] record1 = new int[row][col]; // visited num
    int[][] record2 = new int[row][col]; // distance
    int num1 = 0;
    for (int r = 0; r < row; r++) {
        for (int c = 0; c < col; c++) {
            if (grid[r][c] == 1) {
                num1 ++;
                boolean[][] visited = new boolean[row][col];
                Queue<int[]> queue = new LinkedList<int[]>();
                queue.offer(new int[]{r, c});
                int dist = 0;
                while (!queue.isEmpty()) {
                    int size = queue.size();
                    for (int i = 0; i < size; i++) {
                        int[] node = queue.poll();
                        int x = node[0];
                        int y = node[1];
                        record2[x][y] += dist;
```

```
                        record1[x][y] ++;
                        if (x > 0 && grid[x - 1][y] == 0 && !visited
                            queue.offer(new int[]{x - 1, y});
                            visited[x - 1][y] = true;
                        }
                        if (x + 1 < row && grid[x + 1][y] == 0 && !v
                            queue.offer(new int[]{x + 1, y});
                            visited[x + 1][y] = true;
                        }
                        if (y > 0 && grid[x][y - 1] == 0 && !visited
                            queue.offer(new int[]{x, y - 1});
                            visited[x][y - 1] = true;
                        }
                        if (y + 1 < col && grid[x][y + 1] == 0 && !v
                            queue.offer(new int[]{x, y + 1});
                            visited[x][y + 1] = true;
                        }
                    }
                    dist ++;
                }
            }
        }
    }
    int result = Integer.MAX_VALUE;
    for (int r = 0; r < row; r++) {
        for (int c = 0; c < col; c++) {
            if (grid[r][c] == 0 && record1[r][c] == num1 && record2[
                result = record2[r][c];
            }
        }
    }
    return result == Integer.MAX_VALUE ? -1 : result;
}
```

# Two Sum III - Data structure design

# Two Sum III - Data structure design

```
Design and implement a TwoSum class. It should support the following

add - Add the number to an internal data structure.
find - Find if there exists any pair of numbers which sum is equal t

For example,
add(1); add(3); add(5);
find(4) -> true
find(7) -> false
```

---

```
public class TwoSum {

    // Add the number to an internal data structure.

    private Map <Integer, Integer> map;

    public TwoSum(){
        map = new HashMap();
    }

    public void add(int number) {
       map.put(number, map.get(number) == null ? 1 : 2);
    }

    // Find if there exists any pair of numbers which sum is equal t
    public boolean find(int value) {

        for(Map.Entry<Integer, Integer> entry : map.entrySet()){
            int key = entry.getKey(), val = entry.getValue();
            if(map.get(value - key) != null && (value != key * 2 || 
                return true;
            }
        }
        return false;
    }
}
```

# Shortest Word Distance

# Shortest Word Distance

```
Given a list of words and two words word1 and word2, return the shor

For example,
Assume that words = ["practice", "makes", "perfect", "coding", "make

Given word1 = “coding”, word2 = “practice”, return 3.
Given word1 = "makes", word2 = "coding", return 1.

Note:
You may assume that word1 does not equal to word2, and word1 and wor
```

---

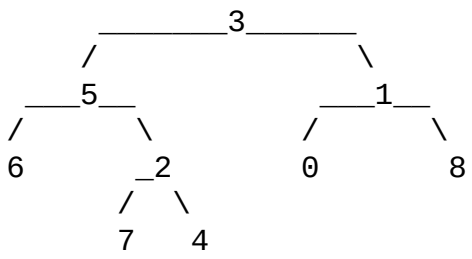
```
    public int shortestDistance(String[] words, String word1, String
        int dis = 1234567, w1 = -32767, w2 = 32767;
        for(int i = 0 ; i < words.length; i ++){
            if(words[i].equals(word1)) w1 = i;
            else if(words[i].equals(word2)) w2 = i;
            dis = Math.min(Math.abs(w1 - w2), dis);
        }

        return dis;
    }
```

```
def shortestDistance(self, words, word1, word2):
        """
        :type words: List[str]
        :type word1: str
        :type word2: str
        :rtype: int
        """
        p1 = p2 = float('inf')
        result = float('inf')
        if word1 != word2:
            for i, w in enumerate(words):
                if w == word1:
                    p1 = i
                elif w == word2:
                    p2 = i
                result = min(abs(p2 - p1), result)
        else:
            for i, w in enumerate(words):
                if w == word1:
                    p2 = p1
                    p1 = i
                result = min(abs(p2 - p1), result)
        return result
```

# Lowest Common Ancestor of a Binary Tree

# Lowest Common Ancestor of a Binary Tree

```
Given a binary tree, find the lowest common ancestor (LCA) of two giv

According to the definition of LCA on Wikipedia: “The lowest common a
```



```
For example, the lowest common ancestor (LCA) of nodes 5 and 1 is 3.
```

---

The node which has both left and right will be the LCA.

```
    public TreeNode lowestCommonAncestor(TreeNode root, TreeNode p, T
        if(root == null || root == p || root == q) return root;

        TreeNode left = lowestCommonAncestor(root.left, p, q), right

        if(left != null && right != null) return root;
        else return left == null ? right : left;
    }
```

# Verify Preorder Sequence in Binary Search Tree

# Verify Preorder Sequence in Binary Search Tree

Given an array of numbers, verify whether it is the correct preorder traversal sequence of a binary search tree.

You may assume each number in the sequence is unique.

Follow up: Could you do it using only constant space complexity?

---

Recursive is not O(1) space

```
public boolean verifyPreorder(int[] preorder) {
    return Helper(preorder, 0, preorder.length - 1);
}

private boolean Helper(int [] preorder, int b, int e){
    if(b >= e) return true;
    else{
        int root = preorder[b], i = b;
        for(; i <= e; i ++){
            if(preorder[i] > root)
                break;
        }

        for(int j = i + 1; j <= e; j++)
            if(preorder[j] < root){
                return false;
            }

        return Helper(preorder, b + 1, i - 1) && Helper(preorder,

    }

}
```

# Strobogrammatic Number

# Strobogrammatic Number

A strobogrammatic number is a number that looks the same when rotated 180 degrees (looked at upside down).

Write a function to determine if a number is strobogrammatic. The number is represented as a string.

For example, the numbers "69", "88", and "818" are all strobogrammatic.

---

```
    public boolean isStrobogrammatic(String num) {
        if(num.length() == 0) return false;
        int [] map = new int[10];
        for(int n : map)
            n = -1;
        map[0] = 0;
        map[1] = 1;
        map[6] = 9;
        map[8] = 8;
        map[9] = 6;

        for(int i = 0 ; i < num.length(); i++){
            char c = num.charAt(num.length() - i - 1);
            if(map[c - '0'] == -1 || map[c - '0'] != num.charAt(i) -
                return false;
        }
        return true;
    }
```

# Strobogrammatic Number II

# Strobogrammatic Number II

A strobogrammatic number is a number that looks the same when rotate

Find all strobogrammatic numbers that are of length = n.

For example,
Given n = 2, return ["11","69","88","96"].

---

```
    public List<String> findStrobogrammatic(int n) {
        List<String> res = new ArrayList();
        if(n <= 0) return res;
        Helper(res, new StringBuilder(), n);
        return res;
    }

    private void Helper(List<String> res, StringBuilder comb, int n)
        int len = comb.length();
        if(len > 0 && len >= n / 2.0){
            for(int i = len + 1; i <= n ; i++)
                comb.append(getStroNum(comb.charAt(n - i)));
            res.add(comb.toString());
        }else {
            char []stroNum = {'0', '1', '6', '8', '9'};
            for(int i = 0 ; i < stroNum.length; i++){
                if(i == 0 && len == 0 && n > 1)
                    continue;
                if((i == 2 || i == 4) && ((n & 1) == 1 && len == (1
                    continue;
                StringBuilder sb = new StringBuilder(comb);
                sb.append(stroNum[i]);
                Helper(res, sb, n);
            }

        }
    }

    private char getStroNum(char n){
        switch (n){
            case '0': return '0';
            case '1': return '1';
            case '6': return '9';
            case '8': return '8';
            case '9': return '6';
        }
        return ' ';
    }
```

# Smallest Rectangle Enclosing Black Pixels

# Smallest Rectangle Enclosing Black Pixels

```
An image is represented by a binary matrix with 0 as a white pixel ar

For example, given the following image:

[
  "0010",
  "0110",
  "0100"
]
and x = 0, y = 2,
Return 6.
```

---

```java
    public int minArea(char[][] image, int x, int y) {
        if(image.length == 0) return 0;

        int left = y, right = y, top = x, bottom = x;
        //get topmost
        for(int i = 0 ; i < x; i++){
            if(findOneByRow(image, i, 0, image[0].length - 1)){
                top = i;
                break;
            }
        }
        //get leftmost
        for(int i = 0 ; i < y ; i++){
            if(findOneByCol(image, i, 0, image.length - 1)){
                left = i;
                break;
            }
        }
        //get rightmost
        for(int i = image[0].length - 1; i > y; i--){
            if(findOneByCol(image, i, 0, image.length - 1)){
                right = i;
                break;
            }
        }
        //get bottommost
        for(int i = image.length - 1; i > x; i--){
            if(findOneByRow(image, i, 0, image[0].length - 1)){
                bottom = i;
                break;
            }
        }
```

```
        return (right - left + 1) * (bottom - top + 1);
    }

    private boolean findOneByRow(char [][]image, int row, int i, int

        if(i <= j){
            int mid = (i + j) / 2;
            if(image[row][mid] == '1')
                return true;
            else{
                return findOneByRow(image, row, i, mid - 1) ||
                    findOneByRow(image, row, mid + 1, j);
            }
        }

        return false;
    }

    private boolean findOneByCol(char [][]image, int col, int i, int
        if(i <= j){
            int mid = (i + j) / 2;
            if(image[mid][col] == '1')
                return true;
            else{
                return findOneByCol(image, col, i, mid - 1) ||
                    findOneByCol(image, col, mid + 1, j);
            }
        }

        return false;
    }
```

# Word Pattern II

# Word Pattern II

Given a pattern and a string str, find if str follows the same pattern.

Here follow means a full match, such that there is a bijection between a letter in pattern and a non-empty substring in str.

Examples:

pattern = "abab", str = "redblueredblue" should return true.

pattern = "aaaa", str = "asdasdasdasd" should return true.

pattern = "aabb", str = "xyzabcxzyabc" should return false.

Notes: You may assume both pattern and str contains only lowercase letters.

---

```
    public boolean wordPatternMatch(String pattern, String str) {
        if(pattern.length() == 0 || str.length() == 0)
            return pattern.length() == str.length();

        Map<Character, String> pToS = new HashMap();
        Map<String, Character> sToP = new HashMap();

        return Helper(str, pattern, pToS, sToP);
    }

    private boolean Helper(String str, String pattern, Map<Character,

        if(str.length() == 0 || pattern.length() == 0)
            return str.length() == pattern.length();

        boolean flag = false;
        char p = pattern.charAt(0);

        if(pToS.get(p) != null){
            String ms = pToS.get(p);
            Character mp = sToP.get(ms);
            if(str.startsWith(ms) && mp != null && mp == p){
                flag |= Helper(str.substring(ms.length()), pattern.su
            }else return false;
        }
        else{
            for(int i = 0 ; i < str.length(); i++){
                String s = str.substring(0, i + 1);
                if(sToP.get(s) == null){
                    pToS.put(p, s);
                    sToP.put(s, p);
```

```
                    flag |= Helper(str.substring(i + 1), pattern.subs

                    pToS.remove(p);
                    sToP.remove(s);
                }
            }
        }

        return flag;
    }
```

# Unique Word Abbreviation

# Unique Word Abbreviation

```
An abbreviation of a word follows the form <first letter><number><las

a) it                      --> it    (no abbreviation)

     1
b) d|o|g                   --> d1g

               1    1  1
     1---5----0----5--8
c) i|nternationalizatio|n  --> i18n

               1
     1---5----0
d) l|ocalizatio|n          --> l10n
Assume you have a dictionary and given a word, find whether its abbr

Example:
Given dictionary = [ "deer", "door", "cake", "card" ]

isUnique("dear") -> false
isUnique("cart") -> true
isUnique("cane") -> false
isUnique("make") -> true
```

---

```python
class ValidWordAbbr(object):
    def __init__(self, dictionary):
        """
        initialize your data structure here.
        :type dictionary: List[str]
        """
        self.dict = collections.defaultdict(list)
        for word in dictionary:
            if len(word)>2:
                temp = word[0] + str(len(word) - 2 ) + word[-1]
            else:
                temp = word
            if word not in self.dict[temp]:
                self.dict[temp].append(word)

    def isUnique(self, word):
        """
        check if a word is unique.
        :type word: str
        :rtype: bool
        """
```

```
        if len(word) > 2:
            temp = word[0] + str(len(word) - 2) + word[-1]
        else:
            temp = word
        if self.dict[temp] == [] or (self.dict[temp]==[word]):
            return True
        else:
            return False

public class ValidWordAbbr {
    Map<String, String> dict;
    public ValidWordAbbr(String[] dictionary) {
        dict = new HashMap<>();
        String abbr;
        for (String word : dictionary) {
            abbr = getAbbr(word);
            if (!dict.containsKey(abbr)) //first time to encounter tl
                dict.put(abbr, word);
            else if (!dict.get(abbr).equals(word)) // next time, ori
                dict.put(abbr, "-1"); // -1: not unique
        }
    }

    public boolean isUnique(String word) {
        String abbr = getAbbr(word);
        if (!dict.containsKey(abbr))
            return true;
        else
            return dict.get(abbr).equals(word) ? true : false;
    }

    private String getAbbr(String word) {
        int n = word.length();
        if (n < 3)
            return word;
        return "" + word.charAt(0) + (n - 2) + word.charAt(n - 1);
    }
}
```

# Zigzag Iterator

# Zigzag Iterator

```
Given two 1d vectors, implement an iterator to return their elements

For example, given two 1d vectors:

v1 = [1, 2]
v2 = [3, 4, 5, 6]
By calling next repeatedly until hasNext returns false, the order of

Follow up: What if you are given k 1d vectors? How well can your cod

The "Zigzag" order is not clearly defined and is ambiguous for k > 2

[1,2,3]
[4,5,6,7]
[8,9]
It should return [1,4,8,2,5,9,3,6,7].
```

---

```
class ZigzagIterator(object):

    def __init__(self, v1, v2):
        """
        Initialize your data structure here.
        :type v1: List[int]
        :type v2: List[int]
        """
        self.v = [v1,v2]
        self.start = 0
        self.cur = 0
        self.len = len(v1) + len(v2)

    def next(self):
        """
        :rtype: int
        """
        while True:
            if self.v[self.cur%2]:
                temp = self.v[self.cur % 2].pop(0)
                self.start += 1
                self.cur += 1
                return temp
            else:
                self.cur += 1

    def hasNext(self):
        """
```

```
        :rtype: bool
        """
        if self.start < self.len:
            return True
        else:
            return False
```

```
public class ZigzagIterator {

    LinkedList<Iterator> list;
    public ZigzagIterator(List<Integer> v1, List<Integer> v2) {
        list = new LinkedList<Iterator>();
        if(!v1.isEmpty()) list.add(v1.iterator());
        if(!v2.isEmpty()) list.add(v2.iterator());
    }

    public int next() {
        Iterator poll = list.remove();
        int result = (Integer)poll.next();
        if(poll.hasNext()) list.add(poll);
        return result;
    }

    public boolean hasNext() {
        return !list.isEmpty();
    }
}
```

# Wiggle Sort

# Wiggle Sort

Given an unsorted array nums, reorder it in-place such that nums[0] <= nums[1] >= nums[2] <= nums[3]....

For example, given nums = [3, 5, 2, 1, 6, 4], one possible answer is [1, 6, 2, 5, 3, 4].

---

```
    public void wiggleSort(int[] nums) {

        for (int i = 0; i < nums.length - 1; i++) {
            if ((i % 2 == 0) == (nums[i] > nums[i + 1])) {
                swap(nums, i, i + 1);
            }

        }
    }
```

II

```
Given an unsorted array nums, reorder it such that nums[0] < nums[1]

Example:
(1) Given nums = [1, 5, 1, 1, 6, 4], one possible answer is [1, 4, 1,
(2) Given nums = [1, 3, 2, 2, 3, 1], one possible answer is [2, 3, 1,
```

Not a good solution, time O(nlogn), space O(n)...

```
public void wiggleSort(int[] nums) {
    if(nums.length < 2)
        return;
    Arrays.sort(nums);
    int[] first = Arrays.copyOfRange(nums, 0, (nums.length + 1) / 2),
        second = Arrays.copyOfRange(nums, (nums.length + 1) / 2, nums
    int i = first.length - 1, j = second.length - 1;
    for(int k = 0; k < nums.length; k ++){
        if((k & 1) == 0){
            nums[k] = first[i --];
        }else {
            nums[k] = second[j --];
        }
    }
}
```

# Strobogrammatic Number III

# Strobogrammatic Number III

A strobogrammatic number is a number that looks the same when rotated 180 degrees (looked at upside down).

Write a function to count the total strobogrammatic numbers that exist in the range of low <= num <= high.

For example,

Given low = "50", high = "100", return 3. Because 69, 88, and 96 are three strobogrammatic numbers.

Note: Because the range might be a large number, the low and high numbers are represented as string.

---

```
char[][] pairs = {{'0', '0'}, {'1', '1'}, {'6', '9'}, {'8', '8'}, {'9
int count = 0;

public int strobogrammaticInRange(String low, String high) {
    for(int len = low.length(); len <= high.length(); len++) {
        dfs(low, high, new char[len], 0, len - 1);
    }
    return count;
}

public void dfs(String low, String high, char[] c, int left, int rigl
    if(left > right) {
        String s = new String(c);
        if((s.length() == low.length() && s.compareTo(low) < 0) ||
           (s.length() == high.length() && s.compareTo(high) > 0)) re
        count++;
        return;
    }

    for(char[] p : pairs) {
        c[left] = p[0];
        c[right] = p[1];
        if(c.length != 1 && c[0] == '0') continue;
        if(left < right || left == right && p[0] == p[1]) dfs(low, hi
    }
}
```

```
    public int strobogrammaticInRange(String low, String high){
        int count = 0;
        List<String> rst = new ArrayList<String>();
        for(int n = low.length(); n <= high.length(); n++){
            rst.addAll(helper(n, n));
        }
```

```
        for(String num : rst){

            if((num.length() == low.length()&&num.compareTo(low) < 0
                count++;
        }
        return count;
    }

    private List<String> helper(int cur, int max){
        if(cur == 0) return new ArrayList<String>(Arrays.asList(""))
        if(cur == 1) return new ArrayList<String>(Arrays.asList("1",

        List<String> rst = new ArrayList<String>();
        List<String> center = helper(cur - 2, max);

        for(int i = 0; i < center.size(); i++){
            String tmp = center.get(i);
            if(cur != max) rst.add("0" + tmp + "0");
            rst.add("1" + tmp + "1");
            rst.add("6" + tmp + "9");
            rst.add("8" + tmp + "8");
            rst.add("9" + tmp + "6");
        }
        return rst;
    }
```

# Number of Digit One

# Number of Digit One

Given an integer n, count the total number of digit 1 appearing in all non-negative integers less than or equal to n.

For example:

Given n = 13, Return 6, because digit 1 occurred in the following numbers: 1, 10, 11, 12, 13.

---

```
0 - 10 -> 2
0 - 100 -> 21
0 - 1000 - > 301

0 - 20 -> 11 -> 10 + 1*1
0 - 200 -> 140 -> 100 + 20 * 2
0 - 2000 -> 1600 -> 1000 + 300 * 2

---

d = 0 -> 0
d = 1 -> 1 + remain + len * 10 ^ (len - 1)
1 < d < 9 -> (10 + len * d) * 10 ^ (len - 1)

...
```

```java
    public int countDigitOne(int n) {

        if(n <= 0) return 0;

        String s = String.valueOf(n);
        int count = 0, len = s.length();

        for(int i = 0 ; i < len ; i++){
            int digit = s.charAt(i) - '0', sub = i != len - 1 ? Inte
            count += digit > 0 ? tmp + (digit > 1 ? (int) Math.pow(1
        }

        return count;
    }
```

# Valid Number

# Valid Number

Validate if a given string is numeric.

Some examples:

"0" => true

" 0.1 " => true

"abc" => false

"1 a" => false

"2e10" => true

---

Pay attention to ' ', '+'/ '-', 'e', '.'

```
    def isNumber(self, s):
        """
        :type s: str
        :rtype: bool
        """
        isValid = False
        i = 0
        n = len(s)
        while i<n and s[i] == ' ':
            i += 1
        if i<n and s[i] in ['+','-']:
            i += 1
        while i<n and s[i].isdigit():
            i += 1
            isValid = True
        if i<n and s[i] == '.':
            i += 1
        while i<n and s[i].isdigit():
            i += 1
            isValid = True
        if i<n and isValid and s[i] =='e':
            isValid = False
            i += 1
            if i < n and s[i] in ['+','-']:
                i += 1
            while i<n and s[i].isdigit():
                i += 1
                isValid = True
        while i < n and s[i] == ' ':
            i += 1
```

```
        return isValid if i == n else False
```

# Longest Substring Without Repeating Characters

# Longest Substring Without Repeating Characters

Given a string, find the length of the longest substring without repeating characters. For example, the longest substring without repeating letters for "abcabcbb" is "abc", which the length is 3. For "bbbbb" the longest substring is "b", with the length of 1.

---

```
    public int lengthOfLongestSubstring(String s) {
        Set <Character> set = new HashSet();
        char [] sl = s.toCharArray();
        int b = 0, e = 0, max = 0;
        while(e < sl.length){
            if(set.add(sl[e])){
                max = Math.max(e - b + 1, max);
            }else{
                while(sl[b] != sl[e]){
                    // remove b before b increase 1
                    set.remove(sl[b ++]);
                }
                b ++;
            }
            e ++;
        }
        return max;
    }
```

# Maximum Product of Word Lengths

# Maximum Product of Word Lengths

```
Given a string array words, find the maximum value of length(word[i]
```

```
Example 1:
Given ["abcw", "baz", "foo", "bar", "xtfn", "abcdef"]
Return 16
The two words can be "abcw", "xtfn".

Example 2:
Given ["a", "ab", "abc", "d", "cd", "bcd", "abcd"]
Return 4
The two words can be "ab", "cd".

Example 3:
Given ["a", "aa", "aaa", "aaaa"]
Return 0
No such pair of words.
```

---

**Bit Map**

```
    public int maxProduct(String[] words) {
        int res = 0;
        if(words.length == 0) return 0;
        List<boolean[]> bml = new ArrayList();
        for(String w : words){
            boolean [] bitmap = new boolean[26];
            char [] wl = w.toCharArray();
            for(char c : wl){
                bitmap [c - 'a'] = true;
            }
            bml.add(bitmap);
        }

        for(int i = 0 ; i < bml.size() - 1; i ++){
            for(int j = i + 1; j < bml.size(); j ++){
                boolean canProduct = true;
                for(int k = 0 ; k < 26 ; k++){
                    if(bml.get(i)[k] && bml.get(j)[k]) {canProduct =
                }
                if(canProduct) res = Math.max(res, words[i].length()
            }
        }

        return res;
    }
```

# Palindrome Linked List

# Palindrome Linked List

Given a singly linked list, determine if it is a palindrome.

---

**Recursive**

```
public class Solution {

    private ListNode head;

    private boolean helper(ListNode tail) {
        if(tail == null) return true;

        if(helper(tail.next) && head.val == tail.val){
            head = head.next;
            return true;
        }else return false;
    }

    public boolean isPalindrome(ListNode tail){
        head = tail;
        return helper(tail);
    }
}
```

# Subsets

# Subsets

```
Given a set of distinct integers, nums, return all possible subsets.

Note:
Elements in a subset must be in non-descending order.
The solution set must not contain duplicate subsets.
For example,
If nums = [1,2,3], a solution is:

[
  [3],
  [1],
  [2],
  [1,2,3],
  [1,3],
  [2,3],
  [1,2],
  []
]
```

Iterator

```
    public List<List<Integer>> subsets(int[] nums) {
        List<List<Integer>> res = new ArrayList();
        res.add(new ArrayList());
        if(nums.length == 0) return res;
        Arrays.sort(nums);
        for(int i = 0 ; i < nums.length; i++){
            int n = nums[i], size = res.size();
            for(int j = 0 ; j < size; j++){
                List <Integer> arr = new ArrayList();
                arr.addAll(res.get(j));
                arr.add(n);
                res.add(arr);
            }
        }
        return res;
    }
```

# Binary Tree Maximum Path Sum

# Binary Tree Maximum Path Sum

```
Given a binary tree, find the maximum path sum.

For this problem, a path is defined as any sequence of nodes from so

For example:
Given the below binary tree,

       1
      / \
     2   3
Return 6.
```

---

When meet negative node, compare and cut.

```
    private int res;

    private int helper(TreeNode root) {
        if(root == null) return 0;

        int left = helper(root.left), right = helper(root.right);

        if(left < 0) {res = Math.max(left, res); left = 0;}

        if(right < 0) {res = Math.max(right, res); right = 0;}

        res = Math.max(left + right + root.val, res);

        return Math.max(left, right) + root.val;
    }

    public int maxPathSum(TreeNode root) {
        if(root == null) return 0;

        res = root.val;

        helper(root);

        return res;
    }
```

# Find Minimum in Rotated Sorted Array

# Find Minimum in Rotated Sorted Array

Suppose a sorted array is rotated at some pivot unknown to you before

(i.e., 0 1 2 4 5 6 7 might become 4 5 6 7 0 1 2).

Find the minimum element.

You may assume no duplicate exists in the array.

---

Binary search will closet to smaller element (also the larger element will not be the target), so let b = mid + 1 to prevent dead loops.

```
    public int findMin(int[] nums) {
        int b = 0, e = nums.length - 1;

        while(b < e){
            int mid = (e - b) / 2 + b;
            if(nums[mid] >= nums[b] && nums[mid] >= nums[e]){
                b = mid + 1;
            }else e = mid;
        }

        return nums[b];
    }
```

# Decode Ways

# Decode Ways

A message containing letters from A-Z is being encoded to numbers us

'A' -> 1
'B' -> 2
...
'Z' -> 26
Given an encoded message containing digits, determine the total numb

For example,
Given encoded message "12", it could be decoded as "AB" (1 2) or "L"

The number of ways decoding "12" is 2.

---

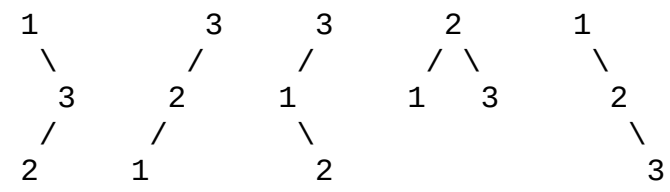
**DP, dp[i] = dp[i - 1] + dp[i - 2]**

```
public int numDecodings(String s) {
    if(s.length() == 0) return 0;

    char []sArr = s.toCharArray();
    int []dp = new int[sArr.length + 1];

    dp[0] = 1;
    dp[1] = canDecode(String.valueOf(sArr[0])) ? 1 : 0;

    for(int i = 1; i < sArr.length; i++){
        if(canDecode(String.valueOf(sArr[i]))) dp[i + 1] += dp[i];
        if(canDecode(s.substring(i - 1, i + 1))) dp[i + 1] += dp[i -
    }

    return dp[sArr.length];
}

private boolean canDecode(String c){
    if(c.length() < 1 || c.length() > 2) return false;
    else return c.charAt(0) != '0' && Integer.parseInt(c) <= 26;
}
```

# Unique Binary Search Trees II

# Unique Binary Search Trees II

```
Given n, generate all structurally unique BST's (binary search trees

For example,
Given n = 3, your program should return all 5 unique BST's shown bel
```



---

**Backtracing**

```
public List<TreeNode> generateTrees(int n) {
    return generateTrees(1, n);
}
private List<TreeNode> generateTrees(int b, int e){
    List<TreeNode> res = new ArrayList<>();
     if (b > e) {
         res.add(null);
     } else if (b == e) {
         res.add(new TreeNode(b));
     } else {
         for (int i = b; i <= e; i++) {
             List<TreeNode> left = generateTrees(b, i - 1);
             List<TreeNode> right = generateTrees(i + 1, e);
             int lsize = left.size();
             int rsize = right.size();
             for (int j = 0; j < lsize; j++) {
                 for (int k = 0; k < rsize; k++) {
                     TreeNode root = new TreeNode(i);
                     root.left = left.get(j);
                     root.right = right.get(k);
                     res.add(root);
                 }
             }
         }
     }

     return res;
}
```

# Largest BST Subtree

# Largest BST Subtree

Given a binary tree, find the largest subtree which is a Binary Searc

---

```
public int largestBSTSubtree(TreeNode root) {

    helper(root);

    return maxSubTree;
}

private int maxSubTree = 0;

private SubTreeInfo helper(TreeNode root){

    if(root == null) return null;

    SubTreeInfo leftTreeInfo = helper(root.left),
                rightTreeInfo = helper(root.right);

    SubTreeInfo currNodeInfo = new SubTreeInfo(root.val, root.val);
    boolean isBST = true;

    if(leftTreeInfo != null){
        if(leftTreeInfo.max < root.val && leftTreeInfo.count != -1){
            currNodeInfo.min = leftTreeInfo.min;
            currNodeInfo.count += leftTreeInfo.count;
        }else isBST = false;
    }

    if(rightTreeInfo != null){
        if(rightTreeInfo.min > root.val && rightTreeInfo.count != -1
            currNodeInfo.max = rightTreeInfo.max;
            currNodeInfo.count += rightTreeInfo.count;
        }else isBST = false;
    }

    if(!isBST){
        currNodeInfo.count = -1;
    }else maxSubTree = Math.max(maxSubTree, currNodeInfo.count);

    return currNodeInfo;
}


class SubTreeInfo{
```

```
    int count = 1;
    int max;
    int min;

    SubTreeInfo(int max, int min){
        this.max = max;
        this.min = min;
    }
}
```

# Word Break

# Word Break

```
Given a string s and a dictionary of words dict, determine if s can b

For example, given
s = "leetcode",
dict = ["leet", "code"].

Return true because "leetcode" can be segmented as "leet code".
```

---

```
public boolean wordBreak(String s, Set<String> wordDict) {
    if(wordDict.contains(s)){ return true;}
    boolean[] dp = new boolean[s.length()];
    dp[0] = true;
    for(int i = 1; i < s.length(); i++){
        if(wordDict.contains(s.substring(0, i))){
            dp[i] = true;
        }
        else{
            for(int j = i - 1; j >= 0; j--){
                if(wordDict.contains(s.substring(j, i)) && dp[j]){
                    dp[i] = true;
                    break;
                }
            }
        }
        if(dp[i] && wordDict.contains(s.substring(i, s.length()))){
            return true;
        }
    }
    return false;
}
```

# Word Break II

# Word Break II

```
Given a string s and a dictionary of words dict, add spaces in s to c

Return all such possible sentences.

For example, given
s = "catsanddog",
dict = ["cat", "cats", "and", "sand", "dog"].

A solution is ["cats and dog", "cat sand dog"].
```

---

**DP + Backtracing**, quite like **Unique Binary Search Trees II**

res = word in Dict + func (subArray)

```
public List<String> wordBreak(String s, Set<String> wordDict) {

    return helper(s, wordDict, new HashMap());
}

private List<String> helper(String s, Set<String> wordDict, Map<Stri

    if(map.get(s) != null) return map.get(s);

    if(s.length() == 0) return new ArrayList(Arrays.asList(""));

    List<String> res = new ArrayList();

    for(String word : wordDict){

        if(s.startsWith(word)){

            List<String> subArrays = helper(s.substring(word.length(

            for(String sub : subArrays){
                res.add(word + (sub.length() == 0 ? "" :" " + sub));
            }
        }
    }

    map.put(s, res);

    return res;
}
```

# Word Search II

# Word Search II

```
Given a 2D board and a list of words from the dictionary, find all wo

Each word must be constructed from letters of sequentially adjacent c

For example,
Given words = ["oath","pea","eat","rain"] and board =

[
  ['o','a','a','n'],
  ['e','t','a','e'],
  ['i','h','k','r'],
  ['i','f','l','v']
]
Return ["eat","oath"].
```

---

```
public class Solution {
    public List<String> findWords(char[][] board, String[] words) {
        List <String> res = new ArrayList();
        if(words.length == 0 || board.length == 0) return res;
        // define a Trie
        TrieNode root = new TrieNode();
        // result set
        Set <String> set = new HashSet();
        // keep visited point
        boolean [][]visited = new boolean[board.length][board[0].leng

        // build Trie
        for(String w : words)
            insert(root, w);

        //dfs
        for(int i = 0; i < board.length ; i++)
            for(int j = 0 ; j < board[0].length; j++){
                dfs(set, root, board, visited, i, j, new StringBuilde
            }

        res.addAll(set);

        return res;
    }

    private void dfs(Set<String> res, TrieNode root, char [][] board,

        if(root == null) return ;
```

```
        if(root.isLeaf) res.add(sb.toString());

        if(i < 0 || j < 0 || i >= board.length || j >= board[0].leng
            return ;
        }

        TrieNode child = root.children.get(board[i][j]);

        visited[i][j] = true;
        sb.append(board[i][j]);

        dfs(res, child, board, visited, i - 1, j, sb);
        dfs(res, child, board, visited, i, j - 1, sb);
        dfs(res, child, board, visited, i + 1, j, sb);
        dfs(res, child, board, visited, i, j + 1, sb);

        visited[i][j] = false;
        sb.deleteCharAt(sb.length() - 1);
    }

    private void insert(TrieNode root, String word){
        TrieNode t = root;
        for(int i = 0; i < word.length(); i++){
            char c = word.charAt(i);
            if(t.children.get(c) == null)
                t.children.put(c, new TrieNode());
            t = t.children.get(c);
        }
        t.isLeaf = true;
    }
}

class TrieNode{
    public Map<Character, TrieNode> children;
    public boolean isLeaf = false;
    public TrieNode(){
        children = new HashMap();
    }
}
```

# Rotate List

# Rotate List

Given a list, rotate the list to the right by k places, where k is n

For example:
Given 1->2->3->4->5->NULL and k = 2,
return 4->5->1->2->3->NULL.

---

```
public ListNode rotateRight(ListNode head, int k) {

    if (head == null || head.next == null) return head;

    ListNode fast = head, slow = head, dummyHead = head;
    int len = 0;

    while (fast.next != null){
        fast = fast.next;
        if (++len > k)
            slow = slow.next;
    }

    len ++;

    if(k >= len){
        k %= len;
        int offset = len - k;
        while (-- offset > 0)
            slow = slow.next;
    }

    //if(k == 0) return head;
    if(slow.next != null){
        dummyHead = slow.next;
        slow.next = null;
        fast.next = head;
    }

    return dummyHead;
}
```

# Coins in a Line II

# Coins in a Line II

There are n coins with different value in a line. Two players take t

Could you please decide the first player will win or lose?

Example
Given values array A = [1,2,2], return true.

Given A = [1,2,4], return false.

---

**DP**

```
public boolean firstWillWin(int[] values) {
    // write your code here
    int size = values.length;
    if(2 >= size) return true;
    int []f = new int[size];
    f[size-1] = values[size-1];
    f[size-2] = values[size-2]+values[size-1];

    for(int i = size-3; i >=0; --i)
    {
        f[i] = Math.max(values[i]-f[i+1], values[i]+values[i+1]-f[i+
    }
    return f[0] > 0;
}
```

# Best Time to Buy and Sell Stock

# Best Time to Buy and Sell Stock

Say you have an array for which the ith element is the price of a given stock on day i.

If you were only permitted to complete at most one transaction (ie, buy one and sell one share of the stock), design an algorithm to find the maximum profit.

---

**Find max diff values in an array, O(n)**

```
public int maxProfit(int[] prices) {
    if(prices.length < 2) {
        return 0;
    }
    int maxProfit = 0, min = prices[0];
    for(int i = 1; i < prices.length; i++){
        if(prices[i] >= min)
            maxProfit = Math.max(maxProfit, prices[i] - min);
        else min = prices[i];
    }
    return maxProfit;
}
```

# Best Time to Buy and Sell Stock II

# Best Time to Buy and Sell Stock II

Say you have an array for which the ith element is the price of a given stock on day i.

Design an algorithm to find the maximum profit. You may complete as many transactions as you like (ie, buy one and sell one share of the stock multiple times). However, you may not engage in multiple transactions at the same time (ie, you must sell the stock before you buy again).

---

**Find/count all ascending sequences**

```
public int maxProfit(int[] prices) {
    int profit = 0;
    for(int i = 1; i < prices.length; i ++) {
        int diff = prices[i] - prices[i - 1];
        if(diff > 0){
            profit += diff;
        }
    }
    return profit;

}
```

# Best Time to Buy and Sell Stock III\IV

DP

```
public int maxProfit(int k, int[] prices) {
 int len = prices.length;
 if (k >= len / 2) {
 return quickSolve(prices);

 }

 int[][] dp = new int[k + 1][len];

 for (int i = 1; i <= k; i++) {

 int tmpMax = -prices[0];

 for (int j = 1; j < len; j++) {

 // Sell the stock at j: not sell/sell.

 dp[i][j] = Math.max(dp[i][j - 1], tmpMax + prices[j]);

 // Buy the stock: not buy/buy.

 tmpMax = Math.max(tmpMax, dp[i - 1][j - 1] - prices[j]);

 }

 }

 return dp[k][len - 1];

 }


 private int quickSolve(int[] prices) {

 int len = prices.length, profit = 0;

 for (int i = 1; i < len; i++)

 // as long as there is a price gap, we gain a profit.

 if (prices[i] > prices[i - 1]) {

 profit += prices[i] - prices[i - 1];

 }
```

```
 return profit;
 }
```

# Merge k sorted lists

# Merge k sorted lists

rt

---

```
public ListNode mergeKLists(List<ListNode> lists) {
    // write your code here
    if (lists.size() == 0) return null;
    Queue<ListNode> pq = new PriorityQueue(10, new Comparator<ListNo
        public int compare(ListNode n1, ListNode n2) {
            return n1.val - n2.val;// minHeap
        }
    });
    ListNode dummy = new ListNode(-1), p = dummy;
    for(ListNode n : lists)
        if(n != null)
            pq.add(n);
    while(!pq.isEmpty()){
        ListNode n = pq.poll();
        p.next = n;
        p = p.next;
        if(n.next != null)
            pq.add(n.next);
    }
    return dummy.next;

}
```

# Binary Tree Paths

# Binary Tree Paths

```
Given a binary tree, return all root-to-leaf paths.

For example, given the following binary tree:

   1
 /   \
2     3
 \
  5
All root-to-leaf paths are:

["1->2->5", "1->3"]
```

---

**DFS**

```
public List<String> binaryTreePaths(TreeNode root) {
    List<String> res = new ArrayList();

    if(root == null) return res;

    Helper(res, String.valueOf(root.val), root);

    return res;
}

public void Helper(List<String> res, String comb, TreeNode root) {
    if(root.left == null && root.right == null){
        res.add(comb);
    }else{
        if(root.left != null){
            Helper(res, comb + "->" + root.left.val, root.left);
        }
        if(root.right != null){
            Helper(res, comb + "->" + root.right.val, root.right);
        }
    }
}
```

**BFS**

```
public List<String> binaryTreePaths(TreeNode root) {
    List<String> res = new ArrayList();

    if(root == null) return res;
```

```
    Queue<TreeNode> que = new LinkedList();
    Queue<String> path = new LinkedList();

    que.add(root);
    path.add(String.valueOf(root.val));

    while(!que.isEmpty()){
        TreeNode n = que.poll();
        String currPath = path.poll();

        if(n.left == null && n.right == null){
            res.add(currPath);
        }

        if(n.left != null){
            que.add(n.left);
            path.add(currPath + "->" + n.left.val);
        }

        if(n.right != null){
            que.add(n.right);
            path.add(currPath + "->" + n.right.val);
        }
    }

    return res;
}
```

# Reverse Nodes in k-Group

# Reverse Nodes in k-Group

Given a linked list, reverse the nodes of a linked list k at a time and return its modified list.

If the number of nodes is not a multiple of k then left-out nodes in the end should remain as it is.

You may not alter the values in the nodes, only nodes itself may be changed. Only constant memory is allowed.

---

```
private ListNode reverseListNode(ListNode head, ListNode tail){
    if(head == tail || head == null) return head;
    ListNode dummy = new ListNode(-1), p = head, tNext = tail.next;
    dummy.next = head;
    while(p.next != null && p.next != tNext){
        ListNode next = p.next;
        p.next = next.next;
        next.next = dummy.next;
        dummy.next = next;

    }
    return dummy.next;
}

public ListNode reverseKGroup(ListNode head, int k) {
    if(head == null || head.next == null || k <= 1) return head;
    ListNode dummy = new ListNode(-1), b = head, e = head, pre = dum
    dummy.next = head;
    while(e != null){
        int step = k;
        while(-- step > 0 && e.next != null){
            e = e.next;
        }
        // k's length > list/remain 's length
        if(step > 0) break;
        pre.next = reverseListNode(b, e);
        while(pre.next != b.next)
            pre = pre.next;
        e = b.next;
        b = b.next;
    }
    return dummy.next;
}
```

# Sort Colors

# Sort Colors

Given an array with n objects colored red, white or blue, sort them so that objects of the same color are adjacent, with the colors in the order red, white and blue.

Here, we will use the integers 0, 1, and 2 to represent the color red, white, and blue respectively.

---

**Two pointers**

```
public void sortColors(int[] nums) {

    if(nums.length == 0) return ;

    int i = 0, start = 0, end = nums.length - 1;

    while(i <= end){
        if(nums[i] == 0){
            swap(nums, start, i);
            start ++;
            if(start > i)
                i = start;
        }else if(nums[i] == 2){
            swap(nums, end, i);
            end --;
        }else i ++;
    }
}

private void swap(int []nums, int i, int j){
    int tmp = nums[i];
    nums[i] = nums[j];
    nums[j] = tmp;
}
```

**K loops**

```
public void sortColors(int[] nums) {
    int index = 0;

    for(int k = 0 ; k < 2; k++){
        for(int j = index; j < nums.length; j++){
            if(nums[j] == k){
                nums[j] = nums[index];
                nums[index] = k;
                index ++;
            }
        }
```

```
        }
}
```

# Edit String to Palindrome

# Edit String to Palindrome

Given a string, find the minimum number of characters to edited to convert it to palindrome.

---

**Recursive**

```
public int edit(char str[], int l, int h)
{
    if (l > h) return Integer.MAX_VALUE;
    if (l == h) return 0;
    if (l == h - 1) return (str[l] == str[h])? 0 : 1;

    return (str[l] == str[h])? edit(str, l + 1, h - 1):
                               Math.min(edit(str, l + 1, h - 1) + 1,
                              (Math.min(edit(str, l, h - 1),
                                  edit(str, l + 1, h)) + 1));
}
```

# Patching Array

# Patching Array

```
Given a sorted positive integer array nums and an integer n, add/pat

Example 1:
nums = [1, 3], n = 6
Return 1.

Combinations of nums are [1], [3], [1,3], which form possible sums o
Now if we add/patch 2 to nums, the combinations are: [1], [2], [3],
Possible sums are 1, 2, 3, 4, 5, 6, which now covers the range [1, 6
So we only need 1 patch.

Example 2:
nums = [1, 5, 10], n = 20
Return 2.
The two patches can be [2, 4].

Example 3:
nums = [1, 2, 2], n = 5
Return 0.
```

---

Since [1, 2, 4, ..., 2^n] can form all the numbers from 1 to 2^(n+1) (not included), so we can simply add the numbers in given array which smaller than current probably missing number, or double the current missing number then add one to result.

```java
public int minPatches(int [] nums, int n) {
    long miss = 1;
    int added = 0, i = 0;
    while (miss <= n) {
        if (i < nums.length && nums[i] <= miss) {
            miss += nums[i++];
        } else {
            miss <<= 1;
            added++;
        }
    }
    return added;
}
```

# Number of Connected Components in an Undirected Graph

Given `n` nodes labeled from `0` to `n - 1` and a list of undirected edges (each edge is a pair of nodes), write a function to find the number of connected components in an undirected graph.

**Example 1:**

```
     0          3
     |          |
     1 --- 2    4
```

Given `n = 5` and `edges = [[0, 1], [1, 2], [3, 4]]`, return `2`.

**Example 2:**

```
     0           4
     |           |
     1 --- 2 --- 3
```

Given `n = 5` and `edges = [[0, 1], [1, 2], [2, 3], [3, 4]]`, return `1`.

**Note:**

You can assume that no duplicate edges will appear in `edges`. Since all edges are undirected, `[0, 1]` is the same as `[1, 0]` and thus will not appear together in `edges`.

---

```
public int countComponents(int n, int[][] edges) {

 if(n < 1)

 return 0;

 if(edges.length == 0)

 return n;

 int[] union = new int[n];

 Arrays.fill(union, -1);

 for(int[] edge: edges){

 if(edge[0] > edge[1])

 swap(edge);

 int b = edge[0], e = edge[1];

 while(union[b] != -1){
```

```
b = union[b];
}
while(union[e] != -1){
e = union[e];
}
if(b != e){
union[e] = b;
n --;
}
}
return n;
}

private void swap(int[] edge){
int tmp = edge[1];
edge[1] = edge[0];
edge[0] = tmp;
}
```

# The Skyline Problem

LC 218

---

TreeMap.

```
public List<int[]> getSkyline(int[][] buildings) {
    List<int[]> res = new ArrayList<>();
    if(buildings.length == 0) {
        return res;
    }
    PriorityQueue<int[]> queue = new PriorityQueue<>((a, b) -> (a[0]
    for(int[] building: buildings) {
        queue.offer(new int[]{building[0], building[2]});
        queue.offer(new int[]{building[1], -building[2]});
    }

    // TreeMap => RBT. So we can do both: get max/min value and inse
    TreeMap<Integer, Integer> heightTree = new TreeMap<>(Collections
    heightTree.put(0, 0);
    while(!queue.isEmpty()) {
        int[] point = queue.poll();
        // New building occured.
        if(point[1] > 0) {
            // The new building is the heightest one.
            if(point[1] > heightTree.firstKey()) {
                res.add(new int[] {point[0], point[1]});
            }
            int count = heightTree.getOrDefault(point[1], 0);
            heightTree.put(point[1], count + 1);
        }
        // A building reaches its end.
        else {
            int count = heightTree.get(-point[1]);
            // The new building is the heightest one, and the only o
            if(count == 1) {
                int firstKey = heightTree.firstKey();
                heightTree.remove(-point[1]);
                if(firstKey == -point[1]) {
                    res.add(new int[] {point[0], heightTree.firstKey
                }
            } else {
                heightTree.put(-point[1], count - 1);
            }
        }
    }
    return res;
}
```

# Regular Expression Matching

```
Implement regular expression matching with support for '.' and '*'.

'.' Matches any single character.
'*' Matches zero or more of the preceding element.
The matching should cover the entire input string (not partial). The
bool isMatch(const char *s, const char *p)
Some examples:
isMatch("aa","a") → false
isMatch("aa","aa") → true
isMatch("aaa","aa") → false
isMatch("aa", "a*") → true
isMatch("aa", ".*") → true
isMatch("ab", ".*") → true
isMatch("aab", "c*a*b") → true
```

---

```java
public boolean isMatch(String s, String p) {
    int sLen = s.length(), pLen = p.length();
    boolean[][] dp = new boolean[sLen + 1][pLen + 1];
    dp[0][0] = true;

    // Judges if p can match empty string.
    for(int i = 1; i <= pLen; i++) {
        if(p.charAt(i - 1) == '*') {
            dp[0][i] = dp[0][i - 2];
        }
    }

    for(int i = 1; i <= sLen; i++) {
        for(int j = 1; j <= pLen; j++) {
            if(s.charAt(i - 1) == p.charAt(j - 1) || p.charAt(j - 1)
                dp[i][j] = dp[i - 1][j - 1];
            } else if(p.charAt(j - 1) == '*') {
                if(s.charAt(i - 1) != p.charAt(j - 2) && p.charAt(j
                    dp[i][j] = dp[i][j - 2];
                }else {
                    // match empty || match many
                    dp[i][j] = dp[i][j - 2] || dp[i - 1][j];
                }
            }
        }
    }
    return dp[sLen][pLen];
}
```

# In-order Traversal

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

# Max XOR in an Array

Trie

```
class BitTrie {
 BitTrie left = null;
 BitTrie right = null;
 }
 public boolean containsDuplicate(int[] nums) {
 BitTrie root = new BitTrie();
 for(int num: nums) {
 BitTrie cur = root;
 for(int i = 31; i >= 0; i --) {
 if(((num >>> i) & 1) == 1) {
 if(cur.left == null) {
 cur.left = new BitTrie();
 }
 cur = cur.left;
 }else {
 if(cur.right == null) {
 cur.right = new BitTrie();
 }
 cur = cur.right;
 }
 }
 }


 int max = 0;
```

```
for(int num: nums) {
BitTrie cur = root;
int i = 31, curNum = num;
for(; i >= 0; i --) {
if(((curNum >>> i) & 1) == 1) {
if(cur.right != null) {
cur = cur.right;
curNum |= (1 << i);
}else {
cur = cur.left;
}
}else {
if(cur.left != null) {
cur = cur.left;
curNum |= (1 << i);
}else {
cur = cur.right;
}
}
max = Math.max(curNum, max);
}
}
System.out.println(max);
return true;
}
```

# The Maze III

There is a **ball** in a maze with empty spaces and walls. The ball can go through empty spaces by rolling **up** (u), **down** (d), **left** (l) or **right** (r), but it won't stop rolling until hitting a wall. When the ball stops, it could choose the next direction. There is also a **hole** in this maze. The ball will drop into the hole if it rolls on to the hole.

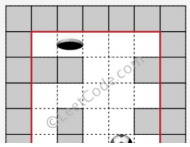
- Total Accepted: **1166**
- Total Submissions: **3637**
- Difficulty: **Hard**
- Contributors: **fallcreek**

Given the **ball position**, the **hole position** and the **maze**, find out how the ball could drop into the hole by moving the **shortest distance**. The distance is defined by the number of **empty spaces** traveled by the ball from the start position (excluded) to the hole (included). Output the moving **directions** by using 'u', 'd', 'l' and 'r'. Since there could be several different shortest ways, you should output the **lexicographically smallest** way. If the ball cannot reach the hole, output "impossible".

The maze is represented by a binary 2D array. 1 means the wall and 0 means the empty space. You may assume that the borders of the maze are all walls. The ball and the hole coordinates are represented by row and column indexes.

**Example 1**

```
Input 1: a maze represented by a 2D array

0 0 0 0 0
1 1 0 0 1
0 0 0 0 0
0 1 0 0 1
0 1 0 0 0

Input 2: ball coordinate (rowBall, colBall) = (4, 3)
Input 3: hole coordinate (rowHole, colHole) = (0, 1)

Output: "lul"
Explanation: There are two shortest ways for the ball to drop into the hole.
The first way is left -> up -> left, represented by "lul".
The second way is up -> left, represented by 'ul'.
Both ways have shortest distance 6, but the first way is lexicographically smaller because 'l' < 'u'. So the output is "lul".
```




```java
// Be careful of the lexicographical sequence: d, l, r, u.
private static final int[][] DIRS = {{1, 0}, {0, -1}, {0, 1}, {-1

private String shortestPath = "";

public String findShortestWay(int[][] maze, int[] ball, int[] hol
    String path = "";
    int[][] dist = new int[maze.length][maze[0].length];
    dist[ball[0]][ball[1]] = 1;
    dfs(ball[0], ball[1], hole, maze, dist, "");
    return shortestPath.equals("")? "impossible": shortestPath;
}

private void dfs(int row, int col, int[] hole, int[][] maze, int
    int m = maze.length, n = maze[0].length;
    for (int d = 0; d < 4; d ++) {
        int i = row, j = col, p = DIRS[d][0], q = DIRS[d][1], len
        while (i + p >= 0 && i + p < m && j + q >= 0 && j + q < n
            i += p;
            j += q;
            len ++;
            if (i == hole[0] && j == hole[1] && (dist[i][j] == 0
                if(len < dist[i][j] || isShorterPath(curPath, sho
                    shortestPath = curPath + getDir(p, q);
                    dist[i][j] = len;
                }
                return;
            }
        }
```

```
            if (dist[i][j] > 0 && len >= dist[i][j]) {
                continue;
            }
            dist[i][j] = len;
            dfs(i, j, hole, maze, dist, curPath + getDir(p, q));
        }
    }

    private boolean isShorterPath(String curPath, String shortestPatl
        if(shortestPath.equals("")) {
            return true;
        }
        for(int i = 0; i < curPath.length() && i < shortestPath.leng
            if(curPath.charAt(i) < shortestPath.charAt(i)) {
                return true;
            }else if(curPath.charAt(i) > shortestPath.charAt(i)) {
                return false;
            }
        }
        return curPath.length() < shortestPath.length();
    }

    private String getDir(int p, int q) {
        if(p == 0 && q == 1) {
            return "r";
        }else if(p == 0 && q == -1) {
            return "l";
        }else if(p == 1 && q == 0) {
            return "d";
        }else {
            return "u";
        }
    }
```