大概两三周前电话面试：

012
345
678
.9.

给定以上2d array作为棋盘，起始位置为0，移动方式为走L型（0可以到5和7，1可以到6，8， 3可以到2和8）。问给定步数n，有几种走法。

类似题：576 688

```
public int ways(int n) {
        List<List<Integer>> adj = new LinkedList<>();
        construct(adj);
        int count = 0;
        int[] dp = new int[10];
        dp[0] = 1;
        for(int step = 0; step < n; step++) {
            int[] temp = new int[10];
            for(int i = 0; i < 10; i++) {
                List<Integer> cur = adj.get(i);
                for(int c: cur) temp[c] += dp[i];
            }
            dp = temp;
        }
        for(int item: dp) count += item;
        return count;
    }

    public void construct(List<List<Integer>> adj) {
        adj.add(new LinkedList<>(Arrays.asList(5, 7)));
        adj.add(new LinkedList<>(Arrays.asList(6, 8)));
        adj.add(new LinkedList<>(Arrays.asList(3, 7)));
        adj.add(new LinkedList<>(Arrays.asList(2, 8, 9)));
        adj.add(new LinkedList<>());
        adj.add(new LinkedList<>(Arrays.asList(0, 6, 9)));
        adj.add(new LinkedList<>(Arrays.asList(1, 5)));
        adj.add(new LinkedList<>(Arrays.asList(0, 2)));
        adj.add(new LinkedList<>(Arrays.asList(1, 3)));
        adj.add(new LinkedList<>(Arrays.asList(3, 5)));
    }
```

第一个

|X|X|O|O|O| *
|O|X|O|O|O| *
|O|O|O|O|O| *
|O|O|O|O|O| * 5 steps

从最左边走到最右边最少几步（上图是5） 用bfs 没做过唉没有一遍bugfree 小姐姐也没有给我自己检查的机会 就暗示了一下 我就改了 最后的version肯定是bugfree的

followup: 要返回路径
链接：http://www.1point3acres.com/bbs/forum.php?mod=viewthread&tid=304922&ctid=456
注意！一定要注意放入queue的那一刻就要标注visited，否则在同一个level之间的点可能会作为同伴的路径而没有机会进行搜索
代码：

```
public int countSteps(char[][] board) {
        int step = 0;
        Queue<int[]> queue = new LinkedList<>();
        boolean[][] visited = new boolean[board.length][board[0].length];
        for(int i = 0; i < board.length; i++) {
            if(board[i][0] == 'O') {
                queue.offer(new int[]{i, 0});
                visited[i][0] = true;
            }
        }
        int[] b = new int[]{0, -1, 0, 1, 0};
        while(!queue.isEmpty()) {
            step++;
            int n = queue.size();
            while(n-- > 0) {
                int[] cur = queue.poll();
                for(int i = 0; i < 4; i++) {
                    int row = cur[0] + b[i], col = cur[1] + b[i + 1];
                    if(col == board[0].length) return step;
                    if(col < 0 || row < 0 || row >= board.length || col >= board[0].length || board[row][col] ==
'X' || visited[row][col]) continue;
                    queue.offer(new int[]{row, col});
                }
            }
        }
        return 0;
    }
```

followup 解答：
思路：由于每个点只有一个前继点，所以用map来存一下是由哪个点过来的
代码：

```
public List<List<Integer>> countSteps(char[][] board) {
        LinkedList<List<Integer>> res = new LinkedList<>();
        Queue<int[]> queue = new LinkedList<>();
        boolean[][] visited = new boolean[board.length][board[0].length];
        Map<List<Integer>, List<Integer>> route = new HashMap<>();
        for(int i = 0; i < board.length; i++) {
            if(board[i][0] == 'O') {
                queue.offer(new int[]{i, 0});
                visited[i][0] = true;
            }
        }
        int[] b = new int[]{0, -1, 0, 1, 0};
        while(!queue.isEmpty()) {
            int n = queue.size();
            while(n-- > 0) {
                int[] cur = queue.poll();
                for(int i = 0; i < 4; i++) {
                    int row = cur[0] + b[i], col = cur[1] + b[i + 1];
```

```
                if(col == board[0].length) {
                    findRoute(route, res, cur[0], cur[1]);
                    return res;
                }
                if(col < 0 || row < 0 || row >= board.length || col >= board[0].length || board[row][col] ==
'X' || visited[row][col]) continue;
                visited[row][col] = true;
                queue.offer(new int[]{row, col});
                route.put(Arrays.asList(row, col), Arrays.asList(cur[0], cur[1]));
            }
        }
    }
    return res;
  }

public void findRoute(Map<List<Integer>, List<Integer>> route, LinkedList<List<Integer>> res, int
row, int col) {
    res.add(Arrays.asList(row, col));
    while(route.containsKey(Arrays.asList(row, col))) {
        List<Integer> target = route.get(Arrays.asList(row, col));
        res.addFirst(target);
        row = target.get(0);
        col = target.get(1);
    }
  }
```

11.19

3. 题目: 没有给input，说随意定义，一个set，问能不能找到四个点组成平行于坐标轴的矩形，本身很菜，看到连input都没有一脸懵逼...gg

代码：

```
public boolean findRectangle(int[][] points) {
    Set<List<Integer>> set = new HashSet<>();
    for(int[] point: points) set.add(Arrays.asList(point[0], point[1]));
    for(int i = 0; i < points.length; i++) {
        for(int j = i + 1; j < points.length; j++) {
            if(points[i][0] == points[j][0] || points[i][1] == points[j][1]) continue;
            if(set.contains(Arrays.asList(points[i][0], points[j][1])) &&
set.contains(Arrays.asList(points[j][0], points[i][1]))) return true;
        }
    }
    return false;
  }
```

follow up: 若四边不一定平行于坐标轴?

思路：算三个点，满足勾股定理的话找第四个点，第四个点横纵坐标分别为两个对角点相加减去另外一个点

代码：

```
public boolean findRectangle(int[][] points) {
    Set<List<Integer>> set = new HashSet<>();
    for(int[] point: points) set.add(Arrays.asList(point[0], point[1]));
    for(int i = 0; i < points.length; i++) {
        for(int j = i + 1; j < points.length; j++) {
```

```
                for(int k = j + 1; k < points.length; k++) {
                    int[] forth = triangle(points, i, j, k);
                    if(forth[0] == -1) continue;
                    if(set.contains(Arrays.asList(forth[0], forth[1]))) return true;
                }
            }
        }
        return false;
    }

    public int[] triangle(int[][] points, int i, int j, int k) {
        int[] l = new int[3];
        l[0] = (points[i][0] - points[j][0]) * (points[i][0] - points[j][0]) + (points[i][1] - points[j][1]) * (points[i]
[1] - points[j][1]);
        l[1] = (points[j][0] - points[k][0]) * (points[j][0] - points[k][0]) + (points[j][1] - points[k][1]) *
(points[j][1] - points[k][1]);
        l[2] = (points[k][0] - points[i][0]) * (points[k][0] - points[i][0]) + (points[k][1] - points[i][1]) *
(points[k][1] - points[i][1]);
        int max = Math.max(l[0], Math.max(l[1], l[2]));
        if(max == l[0] && l[0] == l[1] + l[2]) return new int[]{points[i][0] + points[j][0] - points[k][0],
points[i][1] + points[j][1] - points[k][1]};
        if(max == l[1] && l[1] == l[0] + l[2]) return new int[]{points[j][0] + points[k][0] - points[i][0],
points[j][1] + points[k][1] - points[i][1]};
        if(max == l[2] && l[2] == l[1] + l[0]) return new int[]{points[i][0] + points[k][0] - points[j][0],
points[i][1] + points[k][1] - points[j][1]};
        return new int[]{-1, -1};
    }
```

4. 题目

trie的搜索, 和李口儿妖妖有些不同。搜索返回所有符合wildcard的词
比如
add("car")
add("caw")
add("cauw")
search("c*w") 返回 "caw" 和 "cauw".

代码:

```
public class wordDic {
    class TrieNode {
        TrieNode[] next;
        public TrieNode() {
            next = new TrieNode[27];
        }
    }

    private TrieNode root;
    public wordDic() {
        root = new TrieNode();
    }
```

```
int maxLength = 0;
public void addWord(String word) {
    maxLength = Math.max(word.length(), maxLength);
    TrieNode cur = root;
    for(char c: word.toCharArray()) {
        if(cur.next[c - 'a'] == null) cur.next[c - 'a'] = new TrieNode();
        cur = cur.next[c - 'a'];
    }
    cur.next[26] = new TrieNode();
}

public List<String> searchWord(String word) {
    List<String> res = new LinkedList<>();
    TrieNode cur = root;
    searchHelper(word, 0, cur, res, new StringBuilder());
    return res;
}


public void searchHelper(String word, int start, TrieNode cur, List<String> res, StringBuilder sb) {
    if(sb.length() > maxLength) return;
    if(start == word.length()) {
        if(cur.next[26] != null) {
            res.add(sb.toString());
        }
        return;
    }
    char c = word.charAt(start);
    if(c != '*') {
        if(cur.next[c - 'a'] == null) return;
        sb.append(c);
        searchHelper(word, start + 1, cur.next[c - 'a'], res, sb);
        sb.deleteCharAt(sb.length() - 1);
        return;
    }

    for(char cc = 'a'; cc <= 'z'; cc++) {
        if(cur.next[cc - 'a'] != null) {
            sb.append(cc);
            searchHelper(word, start + 1, cur.next[cc - 'a'], res, sb);
            searchHelper(word, start, cur.next[cc - 'a'], res, sb);
            sb.deleteCharAt(sb.length() - 1);
        }
    }
}
```

5. 题目

还有一道给定 一个字符串和一个切分长度k 比如:
"This is a good day" k=10
切分为:
"This (1/4)"
"is a (2/4)"
"good (3/4)"
"day (4/4)"
也就是说每个切分后面还要带上一个后缀 而且这个后缀还是算长度的 最后返回的切分最小需要多少下

思路： binary search
代码
```
public int cut(String str, int k) {
        int start = str.length()/k;
        String[] word = str.split(" ");
        int end = word.length;
        while(start < end) {
            int mid = start + (end - start)/2;
            System.out.println(start + " " + end + " " + mid);
            if(splitMore(word, mid, k)) start = mid + 1;
            else end = mid;
        }

        return start;
    }
    public boolean splitMore(String[] word, int length, int k) {

        int kk = k - 3 - String.valueOf(length).length();
        int count = 1, left = kk - 1;
        for(int i = 0; i < word.length; i++) {
            if(left < word[i].length() + 1) {
                count++;
                left = kk - String.valueOf(count).length() - word[i].length();
            } else {
                left = left - word[i].length() - 1;
            }
        }
        System.out.println(length + " " + count);
        return count > length;
    }
```

6 题目： 给定两个压缩表示的vector 比如[(3,2),(4,1),(2,3)] 就是[2, 2, 2, 1, 1, 1, 1, 3, 3]的缩写 现在给你两个这样压缩好的vetcor 让你求内积 当两个长度不同的时候 按最小一个的长度算
代码：
```
public int function(int[][] s, int[][] t) {
        int countS = 0, countT = 0, offSetS = 0, offSetT = 0;
        int sum = 0;
        while(true) {
            if(countS == s.length || countT == t.length) break;
            sum += s[countS][1] * t[countT][1];
            offSetS++;
            offSetT++;
            if(offSetS == s[countS][0]) {
                offSetS = 0;
                countS++;
            }
            if(offSetT == t[countT][0]) {
                offSetT = 0;
```

```
                countT++;
            }
        }
        return sum;
    }
```

11.20

7. 题目：第二轮面试是onsite。应该是中国小姐姐我先自我介绍，不太记得问了啥了，问了一下我的prediction用了什么features。。然后做题。。首先是longest increasing subarray，秒了，跑了个case。然后longest increasing subsequence，假装想了一分钟，然后说思路然后秒写，跑了个case。最后还是subsequence 但是输出的不是maxLen，输出的是最长那一个所有元素

相似题：300. Longest Increasing subsequence

代码

```
public List<Integer> function(int[] nums) {
        int[] tails = new int[nums.length + 1];
        tails[0] = Integer.MIN_VALUE;
        int size = 0;
        List<List<Integer>> route = new LinkedList<>();
        //the ith item means route of length i, the 0th is null
        route.add(new LinkedList<>());
        for(int num: nums) {
            if(num > tails[size]) {
                tails[++size] = num;
                route.add(new LinkedList<>(route.get(size - 1)));
                route.get(size).add(num);
            } else {
                int start = 0, end = size;
                while(start < end) {
                    int mid = (end + start) /2;
                    if(num > tails[mid]) start = mid + 1;
                    else end = mid;
                }
                tails[start] = num;
                List<Integer> cur = route.get(start);
                cur.remove(cur.size() - 1);
                cur.add(num);
            }
        }
        return route.get(size);
    }
```

8. 题目： 给一个array 比如 [9,9,5,3,3,7,1] 里面的数字都是0到9之间 inclusive 给你个k 给出top k largest number sum

比如 top 3 = 9+9+7

思路： bucket sort

代码：

```
public int function(int[] nums, int k) {
        int[] bucket = new int[10];
        for(int num: nums) bucket[num]++;
        int sum = 0, count = 0;
        int index = 9;
        while(count < k) {
```

```
            for(int i = 0; i < Math.min(k - count, bucket[index]); i++) {
                count++;
                sum += index;
            }
            index--;
        }
        return sum;
    }
```

11.21
9. 链接：http://www.1point3acres.com/bbs/forum.php?mod=viewthread&tid=303348&ctid=456
题目: 字符串含+，*和数字，return int结果
思路： 用+split， 然后对于每一个string再用*split
注意：这里要注意+和*都是特殊符号不能直接用split，所以要进行替换之后再操作
代码：

```
public int cal(String str) {
        int res = 0;
        if(str.length() == 0) return 0;
        str = str.replaceAll("[+]", "+");
        str = str.replaceAll("[*]", "*");
        String p = "[+]";
        String[] plus = str.split(p);
        for(String s: plus) {
            String[] cur = s.split("[*]");
            int temp = 1;
            for(String c: cur) {
                temp = temp * Integer.parseInt(c);
            }
            res += temp;
        }
        return res;
    }
```

11.22
10. 题目：给一个array 是Btree的inorder traverse顺序 build一个minheap tree
reference: http://www.geeksforgeeks.org/convert-bst-min-heap/
链接： http://www.1point3acres.com/bbs/thread-303099-1-1.html
LeetCode 654

11. Convert BST into a MIN-Heap
思路：inorder traverse the BST and get the array, then preorder traverse and put the items in array one by on into the tree or we can traverse the tree level to level
follow up: do it in place?
reference: http://www.geeksforgeeks.org/in-place-convert-bst-into-a-min-heap/
solution: convert the BST to sorted linked list first and then use a queue to construct the tree

not in place, O(N) space代码:

```
public TreeNode bstToMinHeap(TreeNode root) {
        if(root == null) return null;
        LinkedList<Integer> list = new LinkedList<>();
        constructList(root, list);
        Queue<TreeNode> queue = new LinkedList<>();
        TreeNode newRoot = new TreeNode(list.removeFirst());
        queue.offer(newRoot);
```

```
        while(!queue.isEmpty()) {
            int n = queue.size();
            while(n-- > 0) {
                TreeNode cur = queue.poll();
                if(list.size() == 0) return newRoot;
                cur.left = new TreeNode(list.removeFirst());
                if(list.size() == 0) return newRoot;
                cur.right = new TreeNode(list.removeFirst());
                queue.offer(cur.left);
                queue.offer(cur.right);
            }
        }
        return newRoot;
    }

    public void constructList(TreeNode root, List<Integer> list) {
        if(root == null) return;
        constructList(root.left, list);
        list.add(root.val);
        constructList(root.right, list);
    }
```

in place, O(1) space代码：

12. BST to double linked list
reference: http://www.geeksforgeeks.org/convert-given-binary-tree-doubly-linked-list-set-3/
思路：用递归转成inOrder的dll
代码：

```
    TreeNode pre = null;
    TreeNode head = null;
    public void bstTodll(TreeNode root) {
        if(root == null) return;
        bstTodll(root.left);
        if(pre == null) {
            head = root;
        }
        else {
            pre.right = root;
            root.left = pre;
        }
        pre = root;
        bstTodll(root.right);
    }

    public void print(TreeNode root) {
        bstTodll(root);
        if(head == null) return;
        System.out.print("inorder: ");
        TreeNode tail = null;
        while(head != null) {
            System.out.print(head.val + "->");
            if(head.right == null) tail = head;
            head = head.right;
        }
```

```
        System.out.println();
        System.out.print("reversed: ");
        while(tail != null) {
            System.out.print(tail.val + "<-");
            tail = tail.left;
        }
    }
```

in place method: morris traverse
代码：

```
TreeNode pre = null;
    public void bstToDll(TreeNode root) {
        if(root == null) return;
        TreeNode cur = root;
        while(cur != null) {
            if(cur.left == null) {
                visit(cur);
                cur = cur.right;
            } else {
                TreeNode scanner = cur.left;
                while(scanner.right != null && scanner.right != cur) scanner = scanner.right;
                if(scanner.right == cur) {
                    visit(cur);
              //      scanner.right = null;
                    cur = cur.right;
                } else {
                    scanner.right = cur;
                    cur = cur.left;
                }
            }

        }
    }

    public void visit(TreeNode cur) {
        if(pre != null) {
            cur.left = pre;
            pre.right = cur;
        }
        pre = cur;
    }
```

13. BST to circle double linked list
reference: http://www.geeksforgeeks.org/convert-a-binary-tree-to-a-circular-doubly-link-list/

```
TreeNode pre = null;
    TreeNode head = null;
    TreeNode tail = null;
    public void bstTodll(TreeNode root) {
        if(root == null) return;
        bstTodll(root.left);
        if(pre == null) {
            head = root;
        }
```

```
        else {
            pre.right = root;
            root.left = pre;
        }
        pre = root;
        bstTodll(root.right);
        if(root.right == null) {
            tail = root;
            head.left = tail;
            tail.right = head;
        }
    }

public void printLoop(TreeNode root) {
        bstTodll(root);
        if(head == null) return;
        System.out.print("inorder: ");
        TreeNode cur = head;
        do {
            System.out.print(cur.val + "->");
            cur = cur.right;
        } while(cur != head);
        System.out.print(cur.val + "->");

        System.out.println();
        System.out.print("reversed: ");
        cur = tail;
        do {
            System.out.print(tail.val + "<-");
            tail = tail.left;
        } while(cur != tail);
        System.out.print(tail.val + "<-");
    }
```

14. 题目： Max Sum of Subarray with size k, 相当于Easy难度，我说用一个sum array存sum，然后做减法就行。中国小哥说让优化空间，于是说可以只用两个数。
注意：跳出循环之后还是要更新max
要考虑的case：k > nums.length, k <= 0, k = 1;

```
public int cal(int[] nums, int k) {
        if(nums.length < k || k <= 0) return 0;
        int sum = 0, start = 0, end = 0, max = 0;
        //start inclusive, end exclusive
        while(end < nums.length && end - start < k){
            sum += nums[end++];
        }

        while(end < nums.length) {
            max = Math.max(sum, max);
            sum += nums[end++];
            sum -= nums[start++];
        }
        max = Math.max(max, sum);
        return max;
    }
```

15. read 4k，要求print line。这题没做过，写的磕磕巴巴，写完小哥说如果一直没有换行符会怎样。
给个无限大的文件， 调用read4k API，读4k个character 返回值是char[]， 遇到 '\n' 一行结束
然后写个方法 每次返回该文件的一行String，
tricky的地方是，每次调用read4k之前 要考虑上次调用返回的char[]
abc, bcd, cde, '/n', def, efg, fgh， ijk, jkl, klm, lmn, opq, '/n', pqr
假如调用一次返回6个string长度， 第二次调用的时候要考虑第一次调用 '\n'之后剩下的char[]
思路？：在换行符没有出现之前都先将读出来的放进一个缓存里面
http://www.1point3acres.com/bbs/forum.php?mod=viewthread&tid=294662&e

16. Delete Double Linked List Node. Input is head & target(node), delete target, return head. 因为我用了dummy head，所以问了我如果有环怎么办
if(target.pre == null) return head.next;
if(target.next == null) target.pre.next = null; return head;
if(target.next == target) return null;
target.pre.next = target.next;
target.next.pre = target.pre;

17. 题目：
给一堆node，判断这一堆node是不是属于一个完整的binary tree
比如给两个node， n1和n2
n1.left = n2.
n1.right = null
n2.left = n2.right = null
这个就返回true

如果n1.left = n2
n1.right = n2.
n2.left=n2.right = null
这个就返回false.
链接： http://www.1point3acres.com/bbs/thread-302637-1-1.html
思路：建一个set,遍历node,如果node有孩子就往set里面添加，如果set里本来就有这个孩子node就return false 最后再return set.size==# of nodes -1 。 我的想法就是每个node最多只有一个父母，所以最多只能被加进set一次，n个节点的树应该有n-1的父母关系才是连通的。
弊端：如果1，2，3构成环，4单独一个，好像这个方法检查不出来。。。
解决： 再遍历一次检验是否有环
1. 封闭性 需要保证所有的left, right 都指向list内部的点
2. 不能存在环 包括自环
3. 必须全部联通
http://www.1point3acres.com/bbs/forum.php?mod=viewthread&tid=299725&ctid=456

18. sorted DDL to BST

19.同时判断是否递增或者递减
[1,2,3,4,5] true
[5,4,3,2,1] true.
[1,2,3,2,1] false
代码：

```
public boolean func(int[] nums) {
        if(nums.length < 2) return true;
        boolean flag = nums[1] - nums[0] > 0;
        for(int i = 1; i < nums.length; i++) {
            if(nums[i] - nums[i - 1] > 0 != flag) return false;
            if(nums[i] == nums[i - 1]) return false;
        }
        return true;
    }
```

20. 找离原点最近的k个点，要nlogk的算法，用优先队列
input是(List<Points>, k), output(List<Points>)，有个Point class，求离原点最近的K个点。好像是fb经常考的一道非lc题
pq解

```
public List<Point> kNearest(List<Point> points, int k) {
        if(k >= points.size()) {
            printList(points);
            return points;
        }
        Comparator<Point> comp = new Comparator<Point>() {
            public int compare(Point o1, Point o2) {
                return distance(o2) - distance(o1);
            }
        };
        PriorityQueue<Point> queue = new PriorityQueue<Point>(k, comp);
        for(int i = 0; i < k; i++) queue.offer(points.get(i));
        int p = k;
        while(p < points.size()) {
            Point top = queue.peek();
            Point cur = points.get(p);
            if(distance(cur) < distance(top)) {
                queue.poll();
                queue.offer(cur);
            }
            p++;
        }
        List<Point> res = new LinkedList<>();
        for(int i = 0; i < k; i++) res.add(queue.poll());
        printList(res);
        return res;
    }
    public int distance(Point a) {
        return a.x * a.x + a.y * a.y;
    }

    public void printList(List<Point> list) {
        for(Point p: list) {
            System.out.print("(" + p.x + "," + p.y + ")");
        }
        System.out.println();
    }
```

21. 给一个字符串比如 12345=67 可以在等号两边随便加加减号使得等式成立，求所有的方法.(只有加减号，但是可以在第一位加减号，要返回等式的样子)

注意02不是合法啊大胸弟
思路：分别找到左右字符串的所有可能结果，结果值作为key，对应的string表达式的list作为value，用map存。
代码：

```
public List<String> equation(String e) {
        List<String> res = new LinkedList<>();
        if(e.length() == 0) return res;
        String[] temp = e.split("=");
        String left = temp[0], right = temp[1];
        Map<Integer, List<String>> l = new HashMap<>();
        findResult(l, left, 0, 0, "");
        Map<Integer, List<String>> r = new HashMap<>();
        findResult(r, right, 0, 0, "");
        for(Integer num1: l.keySet()) {
            if(r.containsKey(num1)) {
                for(String ll: l.get(num1)) {
                    for(String rr: r.get(num1)) {
                        res.add(ll + "=" + rr);
                    }
                }
            }
        }
        return res;
    }

    public void findResult (Map<Integer, List<String>> map, String str, int start, int sum, String cur) {
        if(start == str.length()) {
            if(!map.containsKey(sum)) map.put(sum, new LinkedList<>());
            map.get(sum).add(cur);
            return;
        }
        for(int i = start; i < str.length(); i++) {
            String s = str.substring(start, i + 1);
            if(s.length() > 1 && s.charAt(0) == '0') break;
            int num = Integer.parseInt(s);
            findResult(map, str, i + 1, sum + num, cur + "+" + s);
            findResult(map, str, i + 1, sum - num, cur + "-" + s);
        }
    }
```

22. 一个array，有m段递增序列，输出排序好的
链接：http://www.1point3acres.com/bbs/forum.php?mod=viewthread&tid=302198&ctid=456

```
public List<Integer> sort(int[] nums) {
        List<Integer> res = new LinkedList<>();
        //same all the pivot
        List<Integer> index = new LinkedList<>();
        index.add(0);
        for(int i = 1; i < nums.length; i++) {
            if(nums[i] <= nums[i - 1]) {
                index.add(i);
            }
        }
        PriorityQueue<Integer> queue = new PriorityQueue<>((a, b)->(nums[a] - nums[b]));
        Set<Integer> set = new HashSet<>();
```

```
        for(int i: index) {
            queue.offer(i);
            set.add(i);
        }
        set.add(nums.length);
        while(!queue.isEmpty()) {
            int cur = queue.poll();
            //if cur + 1 is in the set, that means the numbers of cur sublist has been used up
            if(!set.contains(cur + 1)) queue.offer(cur + 1);
            res.add(nums[cur]);
        }
        return res;
    }
```

23. 打印root到leaf的所有path

类似257

链接: http://www.1point3acres.com/bbs/forum.php?mod=viewthread&tid=302198&ctid=456

代码：

```
public List<List<Integer>> binaryTreePaths(TreeNode root) {
        List<List<Integer>> res = new LinkedList<>();
        backtracking(root, res, new LinkedList<>(), new HashSet<>());
        return res;
    }

    public void backtracking(TreeNode root, List<List<Integer>> res, List<Integer> list) {
        if(root == null) return;
        if(root.left == null && root.right == null) {
            list.add(root.val);
            res.add(new LinkedList<>(list));
            list.remove(list.size() - 1);
            return;
        }
        list.add(root.val);
        backtracking(root.left, res, list, visited);
        backtracking(root.right, res, list, visited);
        list.remove(list.size() - 1);
    }
```

24. 压缩数字： 比如输入1111334222 返回41231432

note：很简单

```
public String compress(String number) {
        if(number.length() == 0) return number;
        int count = 0;
        StringBuilder sb = new StringBuilder();
        char pre = '#';
        for(char c: number.toCharArray()) {
            if(c != pre) {
                if(pre != '#') {
                    sb.append(count);
                    sb.append(pre);
                }
                pre = c;
                count = 1;
            } else count++;
```

```
        }
        sb.append(count);
        sb.append(pre);
        return sb.toString();
    }
```

25. 判断两个图是否同构
链接：http://www.1point3acres.com/bbs/forum.php?mod=viewthread&tid=302027&ctid=456
思路：首先判断点分数是否相同 不同返回false 相同固定第一个图里面的一个点 不断以第二个图某个点为起点知道找到完全相同的结果 对第一个点每个点需要n^2的时间 然后再对第一个图里剩下没有被访问到的点继续进行同样的操作 最坏n^3的代价?

26. 200和463的综合：给定条件与前者相同，但不找个数了，要找出周长最大的岛的周长，比那道要找最大面积的hard题都麻烦
链接： http://www.1point3acres.com/bbs/forum.php?mod=viewthread&tid=302027&ctid=456
注意：数neighbor数量时易错！！
思路： 对每个island求parimeter即可， 因为不能确定形状，所以要用dfs和借用global instance来求周长
代码：

```
final int[] b = new int[] {0, 1, 0, -1, 0};
    int cell = 0, neighbor = 0;
    public int maxPerimeter(char[][] grid) {
        int max = 0;
        for(int i = 0; i < grid.length; i++) {
            for(int j = 0; j < grid[0].length; j++) {
                if(grid[i][j] == '1') {
                    getPerimeter(grid, i, j);
                    max = Math.max(max, cell*4-neighbor*2);
                    System.out.println(cell + " " + neighbor);
                    cell = 0;
                    neighbor = 0;
                }
            }
        }
        return max;
    }

    public void getPerimeter(char[][] grid, int i, int j) {
        if(i < 0 || j < 0 || i >= grid.length || j >= grid[0].length || grid[i][j] == '0') return;
        grid[i][j] = '0';
        cell++;
        for(int k = 0; k < 4; k++) {if(isValidNeighbor(grid, i + b[k], j + b[k + 1])) neighbor++;}
        for(int k = 0; k < 4; k++) getPerimeter(grid, i + b[k], j + b[k + 1]);
    }

    public boolean isValidNeighbor(char[][] grid, int i, int j) {
        if(i < 0 || j < 0 || i >= grid.length || j >= grid[0].length || grid[i][j] == '0') return false;
        return true;
    }
```

11.23
27. 两个BST，从小到大输出成一个list，讨论空间复杂度，最坏和最好的情况。

链接：http://www.1point3acres.com/bbs/forum.php?mod=viewthread&tid=301997&ctid=456

思路：参考BST iterator

相似题：173

代码：

```
public List<Integer> mergeBST(TreeNode root1, TreeNode root2) {
    List<Integer> res = new LinkedList<>();
    Stack<TreeNode> stack1 = new Stack<>();
    Stack<TreeNode> stack2 = new Stack<>();
    while(root1 != null) {
        stack1.push(root1);
        root1 = root1.left;
    }
    while(root2 != null) {
        stack2.push(root2);
        root2 = root2.left;
    }
    while(!stack1.isEmpty() && !stack2.isEmpty()) {
        TreeNode cur1 = stack1.peek();
        TreeNode cur2 = stack2.peek();
        if(cur1.val <= cur2.val) {
            stack1.pop();
            res.add(cur1.val);
            cur1 = cur1.right;
            while(cur1 != null) {
                stack1.push(cur1);
                cur1 = cur1.left;
            }
        } else {
            stack2.pop();
            res.add(cur2.val);
            cur2 = cur2.right;
            while(cur2 != null) {
                stack2.push(cur2);
                cur2 = cur2.left;
            }
        }
    }
    while(!stack1.isEmpty()) {
        TreeNode cur = stack1.pop();
        res.add(cur.val);
        cur = cur.right;
        while(cur != null) {
            stack1.push(cur);
            cur = cur.left;
        }
    }

    while(!stack2.isEmpty()) {
        TreeNode cur = stack2.pop();
        res.add(cur.val);
        cur = cur.right;
        while(cur != null) {
            stack2.push(cur);
            cur = cur.left;
        }
```

```
        }
        return res;
    }
```

测试：

```
TreeNode root1 = new TreeNode(13);
        root1.left = new TreeNode(8);
        root1.right = new TreeNode(18);
        root1.left.left = new TreeNode(2);
        root1.left.right = new TreeNode(10);
        root1.right.left = new TreeNode(14);
        root1.right.right = new TreeNode(20);

        TreeNode root2 = new TreeNode(11);
        root2.left = new TreeNode(4);
        root2.right = new TreeNode(12);
        root2.left.left = new TreeNode(2);
        root2.left.right = new TreeNode(5);
    //    root2.right.left = new TreeNode(14);
        root2.right.right = new TreeNode(13);
```

28. given 2 list of interval representing users online offline timestamp e.g (already sorted), find all intervals that both of users are online.

链接： http://www.1point3acres.com/bbs/forum.php?mod=viewthread&tid=301979&ctid=456

e.g A= [(3, 5), (7, 11)] B=[(2, 4), (9, 10)] --> [(3, 4), (9, 10)].

思路：用两个指针同时从头开始扫描，如果intersect，将intersect的部分放进res，用剩下的后半部分更新原来end更大的interval，另外一个end小一些的interval指针向后移动一位，继续进行判断。

```
public List<int[]> intersect(List<int[]> A, List<int[]> B) {
        List<int[]> res = new LinkedList<>();
        if(A.size() == 0 || B.size() == 0) return res;
        int i = 0, j = 0;
        while(i < A.size() && j < B.size()) {
            int[] curA = A.get(i), curB = B.get(j);
            int start = Math.max(curA[0], curB[0]), end = Math.min(curA[1], curB[1]);
            if(start < end) {
                res.add(new int[] {start, end});
                if(curA[1] < curB[1]) {
                    curB[0] = curA[1] + 1;
                    i++;
                } else if(curA[1] > curB[1]) {
                    curA[0] = curB[1] + 1;
                    j++;
                } else {
                    i++;
                    j++;
                }
            } else {
                if(curA[1] < curB[1]) i++;
                else j++;
            }
        }
        return res;
    }
```

29.給2個function: getFriends(a)會回傳a的所有好友，getMutualFriends(a,b)會回傳a和b的共同好友。用這2個function寫一個推薦好友的算法
這題比較像是開放式的問題，先和他討論要如何推薦好友，他覺得OK後我再寫出來，follow-up是sort最後的結果使得好的candidates能排在前面
链接： http://www.1point3acres.com/bbs/forum.php?mod=viewthread&tid=301848&ctid=456

30. Remove Invalid Parentheses只返回一个结果（ 他让我空间只用O(n)。我卡在空间就是O(2n)了）
链接：http://www.1point3acres.com/bbs/forum.php?mod=viewthread&tid=301273&ctid=456
思路1：stack1用来存open parenthesis index ，stack2存close的index。如果来了一个是open，直接push 到stack1。 如果是close则pop stack1，如果stack1 是空的，则push 到stack2。最后两个stack上存的是不匹配的open close 的index，然后用一个StringBuilder remove相应不合法的parenthesis。最多有n个同时在两个stack 上，所以space complexity应该是O(n), time complexity 也是O(n).
思路2：one pass 就可以了，一遍之后就知道哪几个index对应的parenthesis是需要移除的，直接移除就可以了
代码1: two pass, O(1) space

```
public String removeInvalid(String input) {
        //two pass
        StringBuilder sb = remove(input, '(');
        return remove(sb.reverse().toString(), ')').reverse().toString();
    }

    public StringBuilder remove(String s, char target) {
        int stack = 0;
        StringBuilder sb = new StringBuilder();
        for(char c: s.toCharArray()) {
            if(c == target) stack++;
            else stack--;
            if(stack >= 0) sb.append(c);
            else stack++;
        }
        return sb;
    }
```

代码2： one pass, O(n) space

```
public String removeInvalid(String input) {
            Stack<Integer> stack = new Stack<Integer>();

        for (int i = 0; i < input.length(); i++) {
            char c = input.charAt(i);
            if (c == '(') {
                stack.push(i);
            } else if (c == ')') {
                if (stack.isEmpty() || input.charAt(stack.peek()) != '(') {
                    stack.push(i);
                } else {
                    stack.pop();
                }
            }
        }
```

```
        StringBuilder sb = new StringBuilder(input);
        while (!stack.isEmpty()) {
            sb.deleteCharAt(stack.pop());
        }

        return sb.toString();
    }
```

31. 给一个树（不是二叉树），找最深叶节点的最近公共祖先。
思路：BFS找 deepest leaf node 的list。然后他们的lowest common ancestor 其实就是第一个leaf和最后一个leaf的lowest common ancestor 然后按类似LC236那么写就可以了。
edge case： 假如最深叶节点只有一个的话，你这个方法是不是还要特殊处理一下?
链接：http://www.1point3acres.com/bbs/forum.php?mod=viewthread&tid=297696&ctid=456
思路：我来说一个思路，先traverse一遍, 把每一个点的深度和父亲存在hash表里面，然后把最深得那些点放在queue里面做个向父亲节点的bfs，队列的size会不断变小，当只有一个点的时候就是答案啦

32. input string, 去除string中重复的character，返回一个新的string； abcad，返回abcd.
如果不确定只有小写字母就用set存就好了

代码：
```
public String removeDuplicate(String input) {
        if(input.length() < 2) return input;
        int[] count = new int[26];
        StringBuilder sb = new StringBuilder();
        for(char c: input.toCharArray()) {
            if(count[c - 'a'] == 0) sb.append(c);
            count[c - 'a']++;
        }
        return sb.toString();
    }
```

33. 给一个tree，返回subtree的最大值；如：
```
     1
   /   \
2     3
      /  \
    -5   -6
```
返回的值应该为2; (就是node 2组成的subtree)；
链接：http://www.1point3acres.com/bbs/thread-302019-1-1.html
代码：
```
int max = Integer.MIN_VALUE;
    public int maxSubtree(TreeNode root) {
        if(root == null) return 0;
        helper(root);
        return max;
    }

    public int helper(TreeNode root) {
        int res = root.val;
```

```
        if(root.left != null) res += helper(root.left);
        if(root.right != null) res += helper(root.right);
        max = Math.max(max, res);
        return res;
    }
```

34. 有个interval stream，顺序是乱的，写个方法，每来一个interval，返回目前为止所有interval合并以后总的长度。讨论了几种做法，写了insert interval的办法（存在arraylist里），聊了用BST的做法，当时觉得要自己写个BST没法现在写。后来结束以后查了下文档，发现其实Java也有办法treeset中返回某个元素的iterator的方法（headSet或者tailSet），虽然比C++麻烦。
链接：http://www.1point3acres.com/bbs/forum.php?mod=viewthread&tid=301291&ctid=456

35. 输入两个array {3,2,4,1,5,0} {D,C,E, B,F,A} 程序结束后得到 {0,1,2,3,4,5} {A,B,C,D,E,F}
思路，每次交换一对儿，保证其中一个放在正确的位置上
代码：

```
public void sort(int[] nums, char[] input) {
        int i = 0;
        while(i < nums.length) {
            if(nums[i] == i) i++;
            else swap(nums, input, i, nums[i]);
        }
    }

    public void swap(int[] nums, char[] input, int i, int j) {
        int temp = nums[i];
        char t = input[i];
        nums[i] = nums[j];
        input[i] = input[j];
        nums[j] = temp;
        input[j] = t;
    }
```

36. 排序string, 输入List, 输出排序的List e.g. a4b3 (a 是产品名称，4 是产品版本), a3b2, b1 => a3b2, a4b3, b1; followup, 没有产品名称只有版本，如 “1”，答这个会被放在返回List最前面
链接：http://www.1point3acres.com/bbs/forum.php?mod=viewthread&tid=300981&ctid=456
代码：
时间O(nklogn)，其中k是string的最大长度

```
public List<String> version(List<String> inputs) {
        if(inputs.size() == 0) return new LinkedList<>();
        Comparator<String> comp = new Comparator<String>() {
            public int compare(String o1, String o2) {
                boolean v1 = noName(o1), v2 = noName(o2);
                if(v1 && v2) return Integer.parseInt(o1) - Integer.parseInt(o2);
                else if(v1) return -1;
                else if(v2) return 1;
                return o1.compareTo(o2);
            }
        };
        PriorityQueue<String> queue = new PriorityQueue<>(inputs.size(), comp);
        for(String input: inputs) queue.offer(input);
        List<String> res = new LinkedList<>();
        for(int i = 0; i < inputs.size(); i++) res.add(queue.poll());
        return res;
```

```
    }

    public boolean noName(String s) {
        for(char c: s.toCharArray()) {
            if(Character.isLetter(c)) return false;
        }
        return true;
    }
```

37. 一堆多米诺骨牌 上半边和下半边分别由数字 让你找是否存在一对儿(两片)使得上半边数字的和和下半边数字的和都等于6
链接：http://www.1point3acres.com/bbs/forum.php?mod=viewthread&tid=306195&ctid=456
思路，上半片的数字作为key，所有上半片相同的下班片数字作为list放在set中，只有key的和为6了才看value是不是有
代码：

```
public boolean domino(List<int[]> inputs) {
        Map<Integer, Set<Integer>> map = new HashMap<>();
        for(int[] pair: inputs) {
            if(!map.containsKey(pair[0])) map.put(pair[0], new HashSet<>());
            map.get(pair[0]).add(pair[1]);
        }
        for(int up: map.keySet()) {
            if(map.containsKey(6 - up)) {
                Set<Integer> target = map.get(6 - up);
                for(int down: map.get(up)) {
                    if(target.contains(6 - down) && (up != 3 || down != 3)) return true;
                }
            }
        }
        return false;
    }
```

11.25
38. 给一些sets，保证每个set内元素不重复，如果两个sets有任何共同元素，则让其合并，直到不能合并为止。输出最后的sets的状态。
栗子，Input: {1,2},{2,3},{3,4}{5,6,7},{7,8},{9},{9}
Output: {1,2,3,4},{5,6,7,8},{9}
着重在分析时间复杂度，说完思路、写代码前要先分析好。请以后大家多注意。
链接：http://www.1point3acres.com/bbs/forum.php?mod=viewthread&tid=298581&ctid=456
类似题323

思路：union find
可以建一个图
点是每个set，两个set有交集就连一条边
然后，用BFS或者DFS走一遍，把连通的几个set合并
若n是set的个数，每个set平均m个元素
1）建图时间复杂度 n^2 * m
2) 判断连通 O(n + e) , e是边的数目

建图的时间复杂度

一个map来记录元素的parent，union-find遍历所有元素并更新，最后根据cluster输出元素就行了。
代码：

```
public List<List<Integer>> sets(List<List<Integer>> input) {
        Map<Integer, List<Integer>> map = new HashMap<>();
        buildGraph(input, map);
        Set<Integer> visited = new HashSet<>();
        List<List<Integer>> res = new LinkedList<>();
        for(Integer item: map.keySet()) {
            if(visited.contains(item)) continue;
            List<Integer> list = new LinkedList<>();
            dfs(map, item, visited, list);
            res.add(new LinkedList<>(list));
        }
        return res;
    }

    public void buildGraph(List<List<Integer>> input, Map<Integer, List<Integer>> map) {
        for(List<Integer> list: input) {
            for(int i = 0; i < list.size(); i++) {
                if(!map.containsKey(list.get(i))) map.put(list.get(i), new LinkedList<>());
                if(i != 0) {
                    map.get(list.get(i)).add(list.get(i - 1));
                    map.get(list.get(i - 1)).add(list.get(i));
                }
            }
        }
    }
```

测试：

```
List<List<Integer>> inputs = new LinkedList<>();
        inputs.add(Arrays.asList(1,2));
        inputs.add(Arrays.asList(3,2));
        inputs.add(Arrays.asList(3,4));
        inputs.add(Arrays.asList(5,6,7));
        inputs.add(Arrays.asList(7,8));
        inputs.add(Arrays.asList(9));
        inputs.add(Arrays.asList(10,12));
        inputs.add(Arrays.asList(11,12));
```

39. task schedule(621)保留task顺序

```
public int leastInterval(char[] tasks, int n) {
        int[] count = new int[128];
        int[] pos = new int[128];
        Arrays.fill(pos, -n-1);
        Queue<Character> queue = new LinkedList<>();
        char pre = ' ';
        for(char c: tasks) {
            if(c != pre) queue.offer(c);
            pre = c;
            count[c]++;
        }
    //     System.out.println(queue.size());
```

```
        int counter = 0;//exclusive
        while(!queue.isEmpty()) {
            char cur = queue.poll();
            System.out.println(cur);
            if(counter - pos[cur] > n) {
                pos[cur] = counter;
                counter++;
            }
            else {
                pos[cur] = pos[cur] + n + 1;
                counter = pos[cur] + 1;
            }
            count[cur]--;
            if(count[cur] > 0) queue.offer(cur);
        }
        return counter;
    }
```

40. 把html parse成DOM tree，找到target string，并且打印出来path
链接：http://www.1point3acres.com/bbs/forum.php?mod=viewthread&tid=298699&ctid=456
类似：388

41. BST -> doubly linked list
followup: pseudo code for doublely linked list to BST, the simplest method for finding the mid node (oral discussion only), complexity 如果用这个方法来找中点，整个把linked list转回BST的复杂度是什么
链接：http://www.1point3acres.com/bbs/forum.php?mod=viewthread&tid=298894&ctid=456

42. 面经里那个多层链表转单层的题，与面经不一样的是连接下一层的node是个特殊node，要注意不要把它接进去，然后大哥说用全局变量不太好，尽量不用
链接：http://www.1point3acres.com/bbs/thread-294471-1-1.html

43. input是类似 城市名－人口数
LA 10million
NY 7million
SF 3million
返回weighted random
不过input的data structure得自己想，想放map就map，想放list就list...
http://www.1point3acres.com/bbs/forum.php?mod=viewthread&tid=295896&ctid=456
参考解法：https://stackoverflow.com/questions/6737283/weighted-randomness-in-java

44. sorted的array中数target数目
用LC34的方法得到第一个targer的位置和最后一个target位置，时间O(logN)

45. 给定一个数组和一个目标值，数组中均为正数，要求找出数组中是否含有一个连续子数组的和，等于目标值，返回true or false
follow up 数组中包含负数该如何修改
答：全部是正数的时候可以用two pointer做到时空最优，因为sum是递增的；但是如果 有负数的话就只能用hashmap

46. 给一个有正有负的递增数列，返回一个按绝对值大小排列的数列
分为正负两个部分做插入，可以用O(N)时间实现
如果是已经排好序的没必要找0吧，从两侧向中间走就可以吧，两侧绝对值一定最大
代码：

```
public int[] absIncreasing(int[] nums) {
        int[] res = new int[nums.length];
        if(nums.length == 0 || nums[0] >= 0) return nums;
        int start = 0, end = nums.length - 1, index = nums.length - 1;
        while(start <= end) {
            if(Math.abs(nums[start]) >= Math.abs(nums[end])) res[index--] = nums[start++];
            else res[index--] = nums[end--];
        }
        return res;
    }
```

in place的话用quick sort

47. 三姐在墙上写题， 一边写一边介绍。 说给你一个API， 传入的是char， 然后可以将这个char打印出来。 我刚开始以为， 返回值也是char, 跟三姐交谈后她说不是， 调用这个API只是把char在console里面打印出来，没有返回值。 然后她让我写个方法， 把一个integer 打印出来， 这个integer是非负数。 我说也是void么？ 她说是的。 当时感觉这题有点非主流，不过还好 心里一直在默念move fast， 赶紧写码， 我用的是String.valueOf()把int转成string， 然后再把string转成char array， 然后loop 调用API 把char打出来。。 写完之后， 三姐说。 不行， 不能变成String再换成char。 卧槽， 当时有点蒙圈， 再想他到底想要什么。停顿了几分钟， 三姐给我一个hint，说如果输入的integer只是一个single digit 你咋做？ 我说可以用ascii码转换，她说yes, 1在ascii里对应的是几 。。 （尼玛。。。。这个我忘了 回来一查是49) 她看我没说出来， 说先不谈这个，你继续写吧。。 然后我通过她给的hint， 恍然大悟， 就是把一个数字从高位到低位依次输出， 再通过ascii码转换， 调用API打印。 我就写了个从高位到低位打印数字的方法， 之前没写过这个。写完了让我逐行解释，并把每次循环变量的值列出来， 后来她挑出了bug， 100这个case 过不了。 然后我就再改， 背后在冒凉汗， 本来ascii码那个就卡壳了， 现在又有bug, 还只做了一题。 改完到时间了， 三姐都没拍照，，我知道我要再见了。
链接：http://www.1point3acres.com/bbs/forum.php?mod=viewthread&tid=297848&ctid=456

```
List<Character> list = new ArrayList<>();
while (num != null) {
    list.add((char)(num % 10) - '0');
    num /= 10;. from: 1point3acres.com/bbs
}
for (int i = list.size() - 1; i >= 0; i--) {
    API.print(list.get(i));
}

void print(int num)
{
    if (num >=10)
        print(num/10);
    cout << (num%10 + '0');.
}
```

类似题：168

48. 给一个质数数组（无重复），输出数组元素所有可能的乘积

答案：由于数组是质数，所以这个问题等同于求subset
注意1的情况，它乘上任何是一样的

```
public List<Integer> primeNumber(List<Integer> nums) {
        List<Integer> res = new LinkedList<>();
        res.add(1);
        boolean hasOne = false;
        for(int num: nums) {
            if(num == 1) {
                hasOne = true;
                continue;
            }
            List<Integer> temp = new LinkedList<>();
            for(int i = 0; i < res.size(); i++) {
                int cur = res.get(i);
                cur *= num;
                temp.add(cur);
            }
            res.addAll(new LinkedList<>(temp));
        }
        if(!hasOne) res.remove(0);
        return res;
    }
```

49. sparse vector dot product

```
public int sparseVector(List<Integer> list1, List<Integer> list2) {
        Map<Integer, Integer> map1 = new HashMap<>();
        Map<Integer, Integer> map2 = new HashMap<>();
        for(int i = 0; i < list1.size(); i++) {
            if(list1.get(i) != 0) map1.put(i, list1.get(i));
        }
        for(int i = 0; i < list2.size(); i++) {
            if(list2.get(i) != 0) map2.put(i, list2.get(i));
        }
        int res = 0;
        for(int item: map1.keySet()) {
            if(map2.containsKey(item)) res += map1.get(item) * map2.get(item);
        }
        return res;
    }
```

50. 奇葩字符串比较（a22<a100这种）
第一个，让实现一个比较两道字符串的方法；前提是让字符串分block比较，相连的字母组成一个block，相连的数字组成一个block，比如“abcd12ef3”由4个block组成，两个字符串变成两组block之后对应的block挨个比较，如果对应的block一个是字母一个是数字，那么字母小于数字；如果对应的block都是字母，用的是String的标准的比较方法；如果对应的两个block都是数字，那么比较数字的绝对值大小。.1point3acres缃
比如“abc12”大于"abc9"（第一个block相等，第二个block 12>9）, "a"小于“1”（字母小于数字），“12abd”小于"12ab"（数字block一样，后面的字母block后者大）。
这道题了解了之后很简单，就是需要沟通很久确定各种情况，而且写代码很麻烦，还有很多边界条件需要确定（比如大小写，数字block会不会overflow之类），再加上当时脑袋有点卡壳，我用了java的String.compareTo，parseInt，substring等自带的方程，都写了半天。
代码：

```
//return true if s1 is larger or euqal to s2
    public boolean compare(String s1, String s2) {
        List<Integer> index1 = new LinkedList<>();
        List<Integer> index2 = new LinkedList<>();
        index1.add(0);
        index2.add(0);
        for(int i = 1; i < s1.length(); i++) {
            if(Character.isDigit(s1.charAt(i))^Character.isDigit(s1.charAt(i - 1))) index1.add(i);
        }
        index1.add(s1.length());
        for(int i = 1; i < s2.length(); i++) {
            if(Character.isDigit(s2.charAt(i))^Character.isDigit(s2.charAt(i - 1))) index2.add(i);
        }
        index2.add(s2.length());
        for(int i = 0; i < Math.min(index1.size(), index2.size()) - 1; i++) {
            char c1 = s1.charAt(index1.get(i)), c2 = s2.charAt(index2.get(i));
            String sub1 = s1.substring(index1.get(i), index1.get(i + 1));
            String sub2 = s2.substring(index2.get(i), index2.get(i + 1));
            if(sub1.compareTo(sub2) == 0) continue;
            if(Character.isDigit(c1) && Character.isDigit(c2)) {
                return Math.abs(Integer.parseInt(sub1)) > Math.abs(Integer.parseInt(sub2));
            }
            else if(!Character.isDigit(c1) && !Character.isDigit(c2)) {
                return sub1.compareTo(sub2) > 0;
            } else {
                return Character.isDigit(c1);
            }
        }
        return true;
    }
```

51. 机器人走房间

52. 给一个数组，把它变成二叉树，该二叉树中序遍历结果和数组一样，同时还要保证这是一个minstack
LeetCode 654换一种说法

53. 链表找环（要求O(1)空间、不破坏链表结构，时间复杂度best case O(1N)，双指针的方法best case是O(1.5N)）。
总的来说都比较简单，只有onsite第三轮的第二道题很麻烦，面试官是做compiler的，对复杂度分析和优化要求得非常严格，要求写出具体的数学公式计算best case、average case和worst case的时间复杂度，LZ说了双指针的方法并算出时间复杂度之后被要求从best caseO(1.5N)优化到O(1N)，然后就开始了一段很懵逼的讨论。
LZ：要不，咱用个hashset？不过这样有局限性（对distinct val的要求，已经较高的空间复杂度）。
面试官：嗯，你说的很有道理，那怎么办呢？
LZ：其实我觉得双指针挺好的，但你非不让用，那我破坏链表结构吧，每访问到一个node，check它的next是不是head，是的话有环，不是的话把它的next指向head，然后继续往后走。这样就是O(1)时间+O(1)空间。
面试官：嗯，不错不错，好方法，不过你这样需要破坏链表结构，能不能不破坏呢？
LZ：（大哥我好累啊你这实在是太mean了，别怪我瞎说了）那这样的话说明我还是得用双指针，因为需要O(1)空间存之前访问过的信息，但是slow指针不能call next来移动。

面试官：（面露满意之色）你说的很有道理，所以怎么办呢?
LZ：（算了我瞎编吧）那我就incremental movement，fast每次走1、2、3、4.....k步，走完之后比较fast有没有追上slow，然后再把slow挪到fast的位置。这样best是O(1N)，但是worst case不如双指针好，而且讲道理我觉得别说worst case了，average case O(1N)都是不可能的啊。

54. A树中有一个path可以到某个节点a, B树中根据相同的路径找到b. 给出A,B,a　求b

55. LC621的变种，给出任务单和同种任务之间的cool down间隔，要求计算每个任务的执行时间列表。比如任务单为[1, 1, 2, 1], cool down间隔为2，那么每个任务的执行时间应该是[0, 3, 4, 6]。这题用hashtable可以得出O(n)的解法
链接：http://www.1point3acres.com/bbs/forum.php?mod=viewthread&tid=297109&ctid=456

```
public List<Integer> leastInterval(char[] tasks, int n) {
        List<Integer> res = new LinkedList<>();
        int[] count = new int[128];
        int[] pos = new int[128];
        Arrays.fill(pos, -n-1);
        int curPos = 0;
        for(char c: tasks) {
            if(curPos - pos[c] > n) {
                pos[c] = curPos;
             //   res.add(pos[c]);
                curPos++;
            } else {
                pos[c] += n + 1;
         //       res.add(pos[c]);
                curPos = pos[c] + 1;
            }
            res.add(pos[c]);
        }
        return res;
    }
```

56. 给出一系列区间，比如{ [1,3], [3,5], [2,4], [4,7], [4,9], [7,12] }。问如何用最少数目的区间来覆盖目标区间（比如[2，9]）。在这个例子中，答案应该是{[2,4], [4,9]}。这道题还要考虑无解的情况。LZ给出的解法是先排序，再扫描一遍出结果。
链接：http://www.1point3acres.com/bbs/forum.php?mod=viewthread&tid=297109&ctid=456

```
public List<Interval> cover(List<Interval> intervals, Interval target) {
    List<Interval> res = new LinkedList<>();
    Comparator<Interval> comp = new Comparator<Interval>() {
      public int compare(Interval a, Interval b) {
        return a.start - b.start;
      }
    };
    Collections.sort(intervals, comp);
    Interval maxEnd = new Interval(-1, -1);
    int i = 0;
    while(i < intervals.size()) {
   //   System.out.println("!");
      while(i < intervals.size() && intervals.get(i).start <= target.start) {
        if(intervals.get(i).end > target.start && intervals.get(i).end > maxEnd.end) {
     //    System.out.println(maxEnd.start + " " + maxEnd.end);
          maxEnd = intervals.get(i);
```

```
      }
      i++;
    }
    /**possible conditions of going out of while:
    1. find a valid intersection: add to res and update target.start
    2. i == intervals.size(): check if length of target becoms non-positive
    3. there is no valid intersection anymore: check if length of target becomes non-pos
    **/
    if(maxEnd.end > target.start) {
      res.add(maxEnd);
      target.start = maxEnd.end;
      if(target.start >= target.end) return res;
    } else {
      return new LinkedList<>();
    }
   }
   if(target.start < target.end) return new LinkedList<>();
   return res;
 }
```

57. Sort an input string using an arbitrary alphabet-google
For example: custom alphabet: xyzabc
input to sort according to custom alphabet: cyxz
output according to custom alphabet: xyzc .
因为alphabet有限，可以用bucket sort，将alph存进map，设alpg长度为m，输入长度n，那么空间O(M), 时间max(m ,n
或者直接存入26个字母表的bucket，然后找找alph的顺序输出。空间26，时间一样
解法1：

```
public String changeAlph(String alph, String word) {
      int[] bucket = new int[26];
      for(char c: word.toCharArray()) {
         bucket[c - 'a']++;
      }
      StringBuilder sb = new StringBuilder();
      for(char c: alph.toCharArray()) {
         while(bucket[c - 'a']-- > 0) sb.append(c);
      }
      return sb.toString();
   }
```

解法2：

```
public String changeAlph(String alph, String word) {
      Map<Character, Integer> map = new HashMap<>();
      for(char c: word.toCharArray()) {
         if(!map.containsKey(c)) map.put(c, 0);
         map.put(c, map.get(c) + 1);
      }
      StringBuilder sb = new StringBuilder();
      for(char c: alph.toCharArray()) {
         if(map.containsKey(c)) {
            for(int i = 0; i < map.get(c); i++) sb.append(c);
         }
      }
      return sb.toString();
```

}

58. 第一题是序列化反序列化树，唯一不同要求序列化输出 list<int>，那么就不能用字符形式表达空节点。确认了下树每个数值范围有限，面试官说INT_MAX可以用
链接：http://www.1point3acres.com/bbs/forum.php?mod=viewthread&tid=296569&ctid=456

59. 给一个数组来表示一捆木头，数组里的每个数表示木头的长度 L 。目标是找到一个maximum length K来切这些木头， 然后让切完所有木头之后长度为 K 的木头的数量等于一个给定的值T。
链接：http://www.1point3acres.com/bbs/forum.php?mod=viewthread&tid=296351&ctid=456

60. 给一个int[], 输出所有 A + B = C + D 的unique index pairs, A B 下标不同，CD 下标不同， A B C D 输出以*f*后，就不能输出C D B A了 但是可以ABDC 和BACD这种
思路:提供一个思路， A+B=C+D ---> A+B+(-C)+(-D)=0。 这就成了一个4sum问题。 而且4sum可以转化成两个2sum. .构造一个 bn = -an 并且merg & sort. 然后求解4sum. 不过解出来的时候需要验证是否合理。 给俩例子1111： -> -1_0,-1_1,-1_2,-1_3,1_0,1_1,1_2,1_3. 计算4sum，我用 -1_x 表示他是由哪个变化来的。 -1_0表示他由index 为0的那个数变化而来。 4SUM输出： -1_0,-1_1,1_0,1_1. 下标集合[0,1,0,1]不符合，排除。 下一个 [-1_0,-1_1,1_0,1_2], 下标[0,1,0,2],如果题目要求ABCD严格不同，也不能输出， 直到[-1_0,-1_1,1_2,1_3],此时输出需要枚举各种组合，[0,1,2,3],[1,0,2,3],[0,1,3,2],[1,0,3,2]加入 一个set. 方便去重。
再来一个 1，3，2，5，4 -> -5_3, -4_4, -3_1, -2_2, -1_0, 1_0, 3_1, 2_2, 5__3, 4_4. . 4sum 输出[-5, -4, 5, 4],这个地方数字唯一就不打下标了， 下标[3, 4, 3, 4]不要。 到了[-5, -2, 3, 4] ->[3, 2, 1, 4]符合，输出4个到set，直到4sum完成。
对于一个n个数，首先扩展到2n, 然后在以2n为base的前提下，有 sort和4sum, 最终复杂度上界是4sum.
链接：http://www.1point3acres.com/bbs/forum.php?mod=viewthread&tid=296345&ctid=456

62. merge two sorted list (not listnode)

63. find the length of shortest path between two nodes in a tree
链接：http://www.1point3acres.com/bbs/forum.php?mod=viewthread&tid=296113&ctid=456
代码：

```
public List<Integer> shortestPaths(TreeNode root, TreeNode p, TreeNode q) {
    if(root == null) return new LinkedList<>();
    LinkedList<Integer> res1 = new LinkedList<>();
    LinkedList<Integer> res2 = new LinkedList<>();
    Map<TreeNode, TreeNode> parent = new HashMap<>();
    dfs(root, parent);
    Set<TreeNode> set = new HashSet<>();
    set.add(p);
    TreeNode scanner = p;
    while(parent.containsKey(scanner)) {
        set.add(parent.get(scanner));
        scanner = parent.get(scanner);
    }
    while(!set.contains(q)) {
        res1.add(q.val);
        q = parent.get(q);
    }
    TreeNode common = q;
    res1.add(q.val);
    while(p != q) {
```

```
            res2.addFirst(p.val);
            p = parent.get(p);
        }
        for(Integer i: res2) res1.add(i);
        return res1;
    }

    public void dfs(TreeNode root, Map<TreeNode, TreeNode> parent) {
        if(root.left != null) {
            parent.put(root.left, root);
            dfs(root.left, parent);
        }
        if(root.right != null) {
            parent.put(root.right, root);
            dfs(root.right, parent);
        }
    }
```

64. Intersection of two sorted interval lists, A=[(1,2), (5,7)..] B=[(2,6)...] return [(5,6)..].

65. 给一个list of intervals,and find the point where maximum intervals overlap. 面试官提示说类似于skyline problem那样用一个stack,但最后也没来得及完全想出来
http://www.1point3acres.com/bbs/forum.php?mod=viewthread&tid=295937&ctid=456

66. 判断一个数是不是完全平方数
http://www.1point3acres.com/bbs/forum.php?mod=viewthread&tid=295937&ctid=456

```
public boolean perfectSquares(int num) {
        if(num < 0) return false;
        if(num == 0) return true;
        int start = 1, end = num;
        while(start <= end) {
            int mid = start + (end - start)/2;
            if(num/mid == mid) return num%mid == 0;
            else if(num/mid < mid) end = mid - 1;
            else start = mid + 1;
        }
        return false;
    }
```

67. 给一个Map<String, List<String>> sub. 其中key是要被替换的字母，list可以替换的一堆字母，输出所有可能
E.G. 'a' -> {'b', 'c', 'd'}
那么一个word例如是abc，所有a都要被替换，那么输出就是{'abc','bbc','dbc'}
代码：

```
public List<String> replaceChars(Map<String, List<String>> map, String word) {
        List<String> res = new LinkedList<>();
        for(int i = 0; i < word.length(); i++) {
            for(int j = 0; j <= i; j++) {
                //[j, i] is the substitude candidate
                String cand = word.substring(j, i + 1);
                if(map.containsKey(cand)) {
                    List<String> subs = map.get(cand);
                    for(String sub: subs) {
```

```
                    res.add(word.substring(0, j) + sub + word.substring(i + 1));
                }
            }
        }
    }
    return res;
}
```

测试：

```
List<String> nums1 = new LinkedList<>(Arrays.asList("aaa", "bbb", "ccc", "ddd", "eee"));
    List<String> nums2 = new LinkedList<>(Arrays.asList("oo", "ff", "ll", "pp", "gg"));
    Map<String, List<String>> inputs = new HashMap<>();
    inputs.put("ab", nums1);
    inputs.put("abc", nums2);
    System.out.println(s.replaceChars(inputs, "abc"));
```

68. 检查一个array，看能不能在one swap以内把它变成ascending order，刷题刷的不够，感觉没有印象

觉得不能冷场，先说用stack（举了个例子后被否决了），然后看着题目中的例子凭直觉说用two pointers，还好work了。。。。

follow up：

时间复杂度

check if can be fixed in ascending order or descending order

思路：我是说用两个int代表index，从左往右扫，比较nums和nums[i - 1]的大小，

如果第一次遇到nums[i - 1]比nums小， 把第一个int设为i - 1，第二次遇到这样的pair把第二个int设为i，第三次遇到直接return false

然后交换这两个int指的数，最后扫一遍，看是不是升序

链接：http://www.1point3acres.com/bbs/forum.php?mod=viewthread&tid=294895&ctid=456

69. given a sorted number [1,1,2,2,3,4,4,5,5,5] return the total unique numbers and put them in the left side. i.e. return 5 and the left 5 numbers will be [1,2,3,4,5,The_rest_doesnt_matter]

two pointers

代码：

```
public int func(int[] nums) {
    int valid = 0;//inclusive
    for(int i = 1; i < nums.length; i++) {
        if(nums[i] != nums[i - 1]) nums[++valid] = nums[i];
    }
    return valid + 1;
}
```

70. 找最长等差数列

71. Solve equation followup 有括号

72. 给一个string 像这样a---a--b---c,两个连着的字母相同的话中间就算桥，就都变成+，就变成++++--b----c. From 1point 3acres bbs

再给个例子a---a-a-b---c => +++++++-b—c

代码：

```
public String bridge(String input) {
    if(input.length() == 0) return "";
    char[] chars = input.toCharArray();
```

```
        char pre = '*';
        int start = -1;
        for(int i = 0; i < input.length(); i++) {
            if(chars[i] == '-') continue;
            if(chars[i] == pre) {

                fillBridge(chars, start, i);
                start = i;
            } else {
                start = i;
                pre = chars[i];
            }

        }
        return String.valueOf(chars);
    }

    public void fillBridge(char[] chars, int start, int end) {
        for(int i = start; i <= end; i++) {
            chars[i] = '+';
        }
    }
```

76. Word Break输出一个结果

```
public boolean wordBreak(String s, List<String> wordDict) {
        Set<String> set = new HashSet<>();
        for(String word: wordDict) set.add(word);
        int dp[] = new int[s.length() + 1];
        Arrays.fill(dp, -1);
        dp[0] = 0;
        for(int i = 0; i < s.length(); i++) {
            for(int j = 0; j <= i; j++) {
                //valid string is [j, i], index if the exclusive end and value is inclusive start
                if(set.contains(s.substring(j, i + 1)) && dp[j] != -1) {
                    dp[i + 1] = j;
                    break;
                }
            }
        }
        int i = s.length() - 1;
        for(int num: dp) System.out.print(num + " ");
        while(true) {

            if(dp[i + 1] == -1) return false;
            System.out.print(s.substring(dp[i + 1], i + 1) + " ");
            if(i == -1) break;
            i = dp[i + 1] - 1;
        //    System.out.println(i);

        }
     //   System.out.print(s.substring(dp[i + 1], i + 1) + " ");
        return true;
    }
```