DEC 9

1. PartitionLabels
   1. 做法: 主要是先建一个int[26]来记录每个字母最后出现的位置, 再建一个anchor记录这一段字符串的起点, 用curMaxLast记录last[c – ‘a’]和curMaxLast比较下来当前能找到的最靠后的位置, 然后顺着string去浏览每一个char,如果i == curMaxLast的话这一段就是最长, 在res里加入curMaxLast – anchor + 1; (r – l +1)是字符串的长度
   2. Runtime: 字符串的长度是n,每个都只浏览了一次所以O(n)
2. kClosesetPointToOrigin
   1. 做法: 有直接Arrays.sort(points, Comparator.comparing( a -> a[0]\*a[0] + a[1]\*a[1])排序方法, 再用forloop排出k个来 (我自己是新建了一个pair的class)
   2. Runtime: 用了排序所以是O(n\*log(n))
3. copyListWithRandomPointer(linkedlist的题)
   1. 做法:这题还有问题,主要做法就是设置一个prev = null, cur = head, 然后设置一个whileloop在cur != null 之前用一个curnext储存cur.next,然后cur.next = prev, 然后prev = cur, cur = cur.next, 这样curnext就被移动到原来prev的位置上去了,其他两个往后移了一位
   2. Runtime: 每个浏览了一次所以是O(n), space应该O(1)

Dec 10

1. topKFrequentWords
   1. 做法:用一个map来储存string和频率, 新建一个pair和comparator来放在一个queue里面,如果频率一样则return b.compareTo(a);来按字母从z到a排列,最后再k次把queue的东西poll()出来再reverse res这个arraylist
   2. Runtime: 有排序应该是O(nlog(n)), space应该是O(n), map的大小而不是queue的大小;
2. palindromeLinkedList
   1. 做法:中心就是建立一个公用的pointer，然后用一个recursive helper去检查能不能找到跟pointer的val相等的listnode， 公用的pointer一开始等于head,所以在找到第一个相等的开始得一直做recursion才能找到所有的相等的
   2. Runtime: O(n),每个浏览了一遍, space应该也是O(n), 因为solution里面说在每次做recursive call的时候一个listnode会把整条linkedlist都call出来,所以是O(n),要叫出来整个stackframe

Dec 12

1. friendCircles:
   1. 做法: (dfs)主要就把这个matrix的纵轴横轴看作一个各人关系的交会关系图, 用一个forloop以行数开始先去浏览每一个人跟其他人的关系是不是1, 如果关系是1(就是朋友)的话,那么朋友圈的个数++, 那么我们现在进入跟以这个人(j值)为i值的helper,这个helper里就是把他看作是i,然后forloop所有的j看有没有关系是1的,如果有是1的那么我们再继续recursion一直到找到这个已经确定好的朋友圈的所有人为止,当然我们需要一个int[n]去记录这个人是不是被浏览过,n是matrix边长的长度, 这个第i个人被浏览过肯定就不会出现在别人的朋友圈里了
   2. Runtime: 每个人都去浏览了他的n个朋友关系,即便是dfs如果浏览过了也不会再浏览了, 所以是O(n^2), space是O(n)
2. RottingOranges
   1. 做法: (bfs) 答案拿bfs写的,主要就是按bfs的格式用一个queue去承载所有的rotten oranges，特别注意bfs的格式，在poll queue里面的东西之前  
      //一定要拿一个int去确定好在这次清理queue之前queue的size，然后再根据每一个poll出来的rottenorange进入一个新循环感染别人
   2. Runtime: 我们所有的位置都指挥检查一次,所以是O(n), space 是O(n), n是矩阵的大小,因为在每一次whileloop里我们最坏的情况就是浏览了matrix所有的位置

Dec 13

1. binaryZigZagLevelOrderTraversal
   1. 做法: (bfs)主要就是建一个queue去承载所有的TreeNode, 然后其实每一层填入的方向是一样的(先root.right再root.left或者反过来), 只要设置一个boolean去来回颠倒填入到temp arraylist的顺序就好了
   2. Runtime: 每一
   3. 每个node都浏览了一遍,所以O(n), space O(k), 也就是queue的大小
2. allNodesDistanceKInBinaryTree
   1. 做法:设置一个map去记录每层有哪几个node, 然后k的距离肯定是往下k距离或者往上k的距离, 如果往上k已经超过root了就反过来从root找第Math.abs(targetLevel – k)的那一层
   2. Runtime: 每个node浏览了一次所以是O(n), 同理space也是map的大小所以是O(n)

Dec 14

1. findPivotIndex:
   1. 做法: 这个题就是找到第一个除了自己本身之外左右两边的和都相等的数字，我们先找到所有数字的和，int leftSum就是左边,sum - leftSum - nums[i]就是右边的， forloop找到第一个相等的就行
   2. Runtime: 每个浏览了一遍,所以是O(n), 每次承载了最多一个数字, 所以space是O(1)
2. maximalSquare
   1. 做法: 这个题最好用dp, 建立一个dp = new int[w][l]; 边上等于1的dp都等于1,然后其他的都是根据左、左上、上三个位置dp最小值+1来判断的,也就是dp[i][j] = Math.min(dp[i-1][j], Math.min(dp[i-1][j-1], dp[i][j-1])) + 1
   2. Runtime: 每个浏览了一遍,所以是O(n), space也是O(n), n 是matrix里面所有单位的总和(写成O(mn)也行)
3. courseSchedule
   1. 做法: 主要就是加了个new int[] indegree; 先用indegree记录所有课程的prerequisites数量（入数）， 然后再用一个forloop, 去找出所有prerequisite是0的课程的名字，也就是数字， 然后都加入到bfs的一个queue里面； bfs的逻辑就是每一个queue里面的int课程被poll()出来代表这这门课被上了，那么用forloop找出所有要求他是prerequisite的课并且indegree[toBeTaken]--,如果indegree变为0那么他也就被加入到bfs里了
   2. Runtime: 最坏的情况就是每门课都互相是prerequisite,所以是O(|E| + |V|), |E|是课程的总数, |V|是所要求上的课的总数, space也是O(|E| + |V|).
4. courseScheduleII
   1. 做法:跟courseSchdule一样, 只是每当indegree[taking]等于0的时候要设置res[pos++]= taking; 这里pos是先使用了再++
   2. Runtime: 最坏的情况就是每门课都互相是prerequisite,所以是O(|E| + |V|), |E|是课程的总数, |V|是所要求上的课的总数, space也是O(|E| + |V|).

Dec 21

1.最短路径的三大算法：  
 a. Dijkstra:  
 条件：一般是去找起始点到某个点的最短的路径，然后所有的点之间距离都是正值  
 核心算法：  
 while(!stops.isEmpty()){  
 int curStop = stops.poll();  
 visited[curStop] = true;  
 if(map.containsKey(curStop){  
 for(int[] time : map.get(curStop)){  
 if(distances.get(time[0]) > distances.get(curStop) + time[1]){  
 distances.put(time[0], distances.get(curStop) + time[1]);  
 }  
 if(!visited[time[0]] stops.offer(time[0]);  
 }  
 }  
  
 Runtime: O(Vlog(V) + Elog(v)), 这是pq的解法的复杂度，V是节点的个数，E是每个节点分别能去到的最多的其他节点的个数， 如果是forloop的解法就是O(V^2)  
  
  
 b. Floyd-warshall：  
 条件：可以有负的权重，其实就是Dijkstra循环了n次并放在一个dp[][]的数组里  
 核心算法：  
 //a就是n个stop循环一下  
 for(int a = 0; a < n; a++){  
 for(int i = 0; i < n; i++){  
 for(int j = 0; j < n; j++){  
 dist[i][j] = Math.min(dist[i][j], dist[i][a] + dist[a][j];  
 }  
 }  
 }  
 Runtime：O(n^3)，相当于所有的node循环了立方次  
  
 c. Bellman-ford  
 条件：也可以有负权重，  
 核心算法：  
 对每条边进行n-1次松弛，如果距离有变化就是闭环  
 Runtime: O(V\*E)

Dec 23

1. spfa简单来说就是加了一个pq的bellman ford
2. Floyd的程序: 743
   1. 先建一个int[N][N]的数组, 如果i==j, 则该位置为Integer.Max\_ Value, 如果不是则为0
   2. 把题目里的数据放进去
   3. dist[i][j] = Math.min(dist[i][j], dist[i][k]+dist[k][j]); 在k,i,j的三个forloop里
3. Prim和kruskal第一步粗略理解:
   1. Prim主要是以点为单位找距离最短的点来生成最小生成树, kruskal主要是新加不闭环的边来生成最小生成树

Dec 25

1. ConnectingCitiesWithMinCost(1135, Kruskal)
   1. 做法: Kruskal主要是去用一个unf去链接connection的各个点， 并且各个vertice中间应该是双向的， 我们用一个unf来记录各个点的祖宗和他们所对应的祖宗的叠加距离， 如果找到两个点祖宗一样， 直接continue， 反之， 如果哪个祖宗的叠加距离大， 那这个祖宗就成为另外一个祖宗的祖宗， 并把他们的叠加距离继续叠加，而且我们一定要记得把connections从大到小先排列了再去forloop。
   2. Runtime: O(Elog(v)) = O(log(v)(E+V)), 然而space也是O(E), parents的大小
2. CheapestFlightWithinKStops(787, Prim)
   1. 做法: 主要就是前半部分建立map和狄杰斯特拉一样，而在建立pq时里面每个stop单元int[] curStop应该是三维的, curStop[0]应该当前的下一个中转站（邻居），curStop[1]应该是一个一直在叠加的price， 而curStop[2]应该是当前叠加的已经经过的中转站的数量, 跟dijkstra之类的方法不同的是我们不是在更新start的最短路径， 所以我们不用建立一个dist[]来比较src到各个点的距离，我们也不需要一直用松弛方法去更新src到各个点的距离， 反而我们要做的是去叠加每一个邻居对应的不同的叠加price， 并且检查已经经过的stop有没有>= K+1; 这个一般是单向的
   2. Runtime: 首先我们有pq,也有他们的邻居, 所以应该是O(vlog(v) + Elog(v)), space应该是O(E+V)

Dec27

1. KthLargestElement(215)
   1. 做法: 直接把每一个数加在一个k大小的pq里, 如果pq.size() > k 就poll()一个出来
   2. Runtime: O(log(k)N), 排序每次poll()出来肯定是log(k) 然后执行N次, space就是O(log(k))
2. uglyNumbersII(264)
   1. 做法: 往一个pq里面加n-1次当前pq.poll() \* prime(所有prime)的值， 记得Long和int之间的转换
   2. Runtime: O(log(k)n), space O(k)

Dec28

1. Topological Sort
   1. 基本数据结构:
      1. 建立一个indegree[]
         1. Int[] indegree = new int[nodeNum];
      2. 先根据给的数据建一个map或者matrix, 把对应的条件参数++
         1. For(int i = 0; i < connections.length; i++){

Map.put(…, …);

Indegree[prerequisite[i][0]]++;

}

* + 1. 建立一个priorityQueue, 把indegree里面是0的参数都加进去
       1. PriorityQueue<> pq = new …;
       2. For(int I =0 ; I < indegree.length; i++){

If(indegree[i] == 0) pq.offer(i);

}

* + 1. Whileloop pq和map, 把变为0的参数都bfs一下, numsTaken记录一下
       1. While(!pq.isEmpty()){

Int cur = pq.poll();

numsTaken++;

For(int I : map.get(cur)){

Indegree[i] --;

If(indegree[i] == 0) pq.offer(i);

}

* + 1. 比较一下numsTaken和所有node的总和

Dec29

* + - 1. SuperUglyNumber:
         1. 解法: 已知所有的prime, 那我们先把所有的prime都放到pq里,这代表了1乘以所有的prime, 然后用当前的pq的poll() (cur)来分别乘所有的prime一次, 最后再把pq里面当前还剩下的元素还跟cur一样的用whileloop全部poll()掉, 这样从1到n-1循环
         2. Runtime: 如果k是prime的个数的话是O(n^2log(n)), space是O(n(logn))
      2. TopKFrequentWords
         1. 解法: 拿一个map去记录所有数的出现频率， 然后用一个size为k的pq由大到小记录各个数的，因为pq永远是尾巴出去， 所以不能是由小到大，这个要注意
         2. Runtime: 我们这个PriorityQueue的大小永远是k, 所以pq里poll()出来一次应该是O(log(k)), 然而一共有N个数我们要poll出来(N-k)次以保证小于前k个数的都不在pq里, 所以runtime应该是O((N-k)log(k)) = O(Nlog(k)), space自然是O(klog(k))

Dec30

Find K Pairs with Smallest Sums(373)

解法: 用一个size为3的int[]记录当前[nums1[i], nums2[0], 0], 然后全部放到一个minheap pq里, 并且k- - > 0 的whileloop(一共取k对sum最小的pair), 当int[2]也就是nums2的index == nums2.length – 1的时候直接continue, 否则就offer当前nums2的index的下一个

Runtime: 假设nums1的length是n的话, pq每个元素的大小 = O((logn+1)) = , 然后再运行k次一直去找当前最小的pair, 跟backtrack差不多, 所以应该是O(k(logn)), space是O(nlog(n)

KthSmallestElementInSortedMatrix(378)

解法: 跟上面373是类似的题, 用一个minheap pq<int[2]>去比较每个坐标的大小, 然后把从i到k行的第一个坐标放到pq里, 在whileloop中运行k次, 每次poll()出来一个cur, 然后如果cur[1] < l -1, 也就是还没到最右边的一个坐标, 就offer他右边的这个 new int[]{cur[0], cur[1]+1}

Runtime: 一开始pq的大小是matrix的宽度w, 也就是row的个数, 然后每次poll()一个又offer进来一个, 一共k次, 所以是O(klog(w))次, space是O(wlog(w))

* + - 1. SplitArrayIntoConsecutiveSequences(659)(tailProblem)
         1. 解法: 用一个int[2]来记录每个不同的sequence的开始的数和结束的数, 因为是完全连续的所以他们之间的长度就是cur[1] – cur[0] + 1, 我们建立一个minHeap pq, 优先比较长度, 如果长度相等则比较cur[0], 然后我们foreach loop每个在nums上的数, 首先, while当前pq不是空的并且当前pq.cur[1] +1不等于当前的num(这里是peek()), 我们poll()当前的cur出来, 那么如果cur的长度还<3, 直接return false, 我们要在第一个whileloop里把不可能跟当前数相连的并且长度大于三的直接poll掉; 第二, 如果当前pq是空的或者当前peek出来的cur与当前的数是一样的, 那么我们offer一个新的int[]{num, num}进去 第三种情况, 如果当下的cur[1]+1 == num, 也就是和num相连, 那么我们直接poll()出来把当前的cur的尾巴加上当前的num并offer回去, 这样一直foreach loop完, 接下来我们再检查一下pq里是不是每个sequence的长度都大于三就好了
         2. Runtime: foreach loop了n次, 而且pq里面最多有n个sequence, 所以O(n(logn))
      2. SortCharactersByFrequency (451) (tailProblem)
         1. 解法: 这跟下面有几道题基本属于一个解法, 先用一个map记录各个元素的频率, 根据题目要求建立一个maxHeap, 然后再在sb上面一直append
         2. Runtime : O(nlog(n)), space也是O(nlog(n))
      3. reorganizeString(767)(tailProblem)
         1. 解法: 跟前面451差不多, 只是加入frequency的map的时候如果当前某个char的频率已经超过了(S.length() + 1)/2的话直接return “”, 主要是拿一个maxheap的pq去承载各个char， 一个map来记录频率， 然后在whileloop里面如果sb的尾巴等于这个poll出来的char， 那么就再poll一个加进去  
            因为现在再poll出来的肯定跟上一个就不一样了， 反之， 如果sb的尾巴和当前poll出来的不相等或者sb是空的， 就直接把当前的这个char append上去， 不管何种方法append上去的char都要看减去一个频率以后还大不大于零再放回去， 如果poll了两个出来第一个也要放回去。
         2. Runtime: O(nlog(n)), space也是O(nlog(n))
      4. distantBarcodes(1054)(tailProblem)
         1. 解法: 跟前面splitIntoSequences差不多, 只是用一个maxHeap记录各个数的频率, 如果res[]是空的或者tail != 当前频率最多的数则加上去当前这个数, 频率-1, 如果当前tail等于这个数则再poll出来一个next加上去,频率-1, 然后把cur也加回去, 而且这个题是保证whileloop能跑完的
         2. Runtime: O(nlog(n), space一样
      5. furthestBuildingReach(1642)(tail+Greedy)
         1. 解法: ladder是无敌的, bricks要慎用, 从0开始forloop并记录每一个位置height[i-1] – heights[i]并放到一个minheap里, pq的size() > ladders的时候把最矮的高度poll()出来用bricks – cur, 加入brick < 0 代表过不去了, 直接return I, 能跑完就return最后一个位置length-1
         2. Runtime : 最多跑到最后一个位置, 所以是O(nlog(n)), space也是一样, 最多放n个高度

Dec31

七个debug的机经

求奇数位数的数字和偶数位数的数字的乘积是不是相等的

public static void main(String args[]){  
 System.*out*.println(*product*(242));  
}

//用两个int代表偶数和奇数位的product, 先%10一下得奇数位, 然后再/10; 然后重复一遍就是偶数位  
private static boolean product(int num){  
 if (num < 10){  
 return false;  
 }  
 int oddValue = 1, evenValue = 1;  
  
 while(num > 0){  
 int digit = num % 10; //改这个%  
 oddValue \*= digit;  
 num = num / 10;  
 if(num == 0) break;  
 digit = num % 10;//改这个%  
 evenValue \*= digit;  
 num = num / 10;  
 }  
  
 return evenValue == oddValue;  
}

求一个String里面各个连续数字的和

public static void main(String args[]){  
 System.*out*.println(*calculate*("24s2"));  
}

//用一个string temp表示当前的连续数字, forloop string如果isDigit就加上去, 如果不是digit就把temp清空再继续  
private static int calculate(String inputString){  
 String temp = "";  
 int sum = 0;  
 char[] cl = inputString.toCharArray();  
 for(int i = 0; i < cl.length; i++) {  
 char c = inputString.charAt(i);  
 if (Character.*isDigit*(c)) {  
 temp += c;  
 }  
 else{  
 sum += Integer.*parseInt*(temp);  
 temp = "0"; //改这个temp要重置  
 }  
 }  
 return sum + Integer.*parseInt*(temp);  
}

去掉连续的元音字母里的第一个

public static void main(String args[]){  
 System.*out*.println(*removeConsecutiveVowels*("aoosseaowae"));  
}  
//用isVowel去判断当前是不是元音字母，如果当前或者上一个不是就把当前的字符加上去  
private static String removeConsecutiveVowels(String str){  
 String str1 = "";  
 str1 += str.charAt(0);  
 for(int i = 1; i < str.length(); i++){  
 if(!*isVowel*(str.charAt(i)) || !*isVowel*(str.charAt(i - 1))) {  
 char c = str.charAt(i); //改上面这个||, 原题应该是&&  
 str1 += c;  
 }  
 }  
 return str1;  
}  
  
private static boolean isVowel(char c){  
 return c == 'a' || c == 'e' || c == 'i' || c == 'o' || c == 'u';  
}

reverseAlphabeticCharacter

public static void main(String args[]){  
 System.*out*.println(*reverseAlphabetCharsOnly*("ac!d"));  
}  
// 用左右两个pointer来记录， 如果!Character.isAlphabetic(), 就往中间move一个， 如果两个都是的话创建一个temp承载左右任意一个char， 然后再交换  
//然后左右都往中间走一个  
private static String reverseAlphabetCharsOnly(String inputString){  
 char[] inputChar = inputString.toCharArray();  
 int left = 0, right = inputString.length() - 1;  
 while(left < right){  
 if(!Character.*isAlphabetic*(inputChar[right])) right--;  
 else if(!Character.*isAlphabetic*(inputChar[left]))left++;  
 else {  
 char temp = inputChar[left];  
 inputChar[left] = inputChar[right];  
 inputChar[right] = temp;  
 left++;  
 right--; //改这个left++和right--  
 }  
  
 }  
 return new String(inputChar);  
}

checkPairSumExists

//给你一个sum， 建立一个hashset， 如果set里面有sum - arr[i][j], return true, 如果没有则set加上这个arr[i][j]继续  
private static boolean checkPairSumExists(int rows, int cols, int[][] arr, int sum){  
 Set<Integer> set = new HashSet<>();  
 for(int i = 0; i < rows; i++){  
 for(int j = 0; j < cols; j++){  
 if(set.contains(arr[i][j])){  
 return true;  
 }  
 else{  
 set.add(arr[i][j]); //改这个arr[i][j];  
 }  
 }  
  
 }  
 return false;  
}

countRotations

private static int countRotations(int size, int[] list){  
 int res = *countRotationUntil*(list, 0, size - 1);  
 return res;  
}

private static int countRotationUntil(int[] list, int low, int high){  
 if(high < low){  
 return 0;  
 }  
  
 if(high == low){  
 return low;  
 }  
  
 int mid = low + (low + high)/2;  
  
 if(mid < high && list[mid+1] < list[mid]){  
 return mid+1;  
 }  
  
 if(mid > low && list[mid] < list[mid - 1]){  
 return mid; // 改这里return mid;  
 }  
  
 if(list[high] > list[mid]){  
 return *countRotationUntil*(list, mid, high);  
 }  
  
  
 return *countRotationUntil*(list, mid, high);  
}

countTripletSumPermutation

java 第十行 j = i + 1;

2020 Jan1(DP和第二轮面经)

1. Triangle(dp)
   1. 做法:

从三角形的倒数第一行往上推， 每一行的第i个数字， 等于dp[i]和dp[i+1]的最小值加上当前这个数， 因为我们每次只能往下左下右下两个位置移动,所以对于每一行的每一个index来说， 他最后只会被归入dp[i]和dp[i+1]这两个可能的轨迹中， 因为每个node只能到下面左右两个adjacent的node,那么对于每一层的第i个node来说他所能并入的dp[]也只有dp[i]和dp[i+1], dp[j] = Math.*min*(dp[j], dp[j+1]) + triangle.get(i).get(j);

* 1. Runtime:

triangle的size都浏览了一次， 每次又浏览了每行， 所以是O(n^2), space就是dp的大小O(n)

1. MaxProductSubarray(dp)
   1. 做法:

因为每一个数字都有正负之分， 所以我们需要用一个maxdp和一个mindp来记录当前的乘积， 如果当前的数是<0, maxdp[i]就是当前的数乘以min[i-1]和nums[i]的比较,反之mindp就是当前的数乘以mindp[i-1]和nums[i]的比较， 最后再找出maxdp里面最大的值return. 关系式:

if(nums[i] < 0){  
 max[i] = Math.*max*(min[i-1] \* nums[i], nums[i]);  
 min[i] = Math.*min*(max[i-1] \* nums[i], nums[i]);  
}  
else{  
 max[i] = Math.*max*(max[i-1] \* nums[i], nums[i]);  
 min[i] = Math.*min*(min[i-1]\* nums[i], nums[i]);  
}

Runtime:

所有的都浏览一遍就是O(n), space有两个int[]就是O(2n)也就是O(n)

1. fillTruck(heap)
   1. 做法:第一遍自己写的， 用pq去排序箱子的名字， 依据是他们的容量， 然后每次根据truckSize我们用掉一个容量最多的箱子的个数（peek再--）， 如果这个箱子只剩下0了我们就从pq里 remove掉他，
   2. Runtime: 如果n是truckSize的话就是O(nlog(k))，最坏的情况箱子用不完所以所有的箱子都在pq里， k是箱子的个数， space是O(klog(k))
2. diskSpaceAnalysis(maxSlidingWindow) (DP)
   1. 做法:

答案用的stack过了， 这里stack只能用ArrayDequeue call, 方法是用一个stack来储存当前nums上各个数的index， 用一个pos在res[]上标记当前的位置， 在一个forloop i = 0到 nums.length的过程中来处理stack加上i之前的stack： 用两个whileloop， 第一个whileloop用peek来检查坐标是不是在< i-k+1的范围内，不在的全部poll（）, 第二个whileloop用peekLast（）来检查是不是比当前的nums[i]小， 小的全部pollLast（）； 最后，stack offer一下i，然后因为i是从0开始的，我们需要检查是不是已经>=k-1了, 不然我们是没必要去把运算到i之前的这个stack的peek（）加到res[]里的，

* 1. Runtime:

全部i都只浏览了一次， 自然是O(n),在stack处理的过程中就算每次我们处理空了也只是另一个O(n), 最后一起就是O(n)；  
// 然后Space也是res[]和stack的合， 最多也就是O(2n） = O(n）

JAN 2ND

1. Max Average Subarray
   1. 做法: 主要就是用dfs, dfs里面是一个double[3]分别来记录当前的数的val的和、treeNode个数的和、当前subtree从下到上包括跟自己相比最大的average, 总的来说就是一个dfs的方法， helper是一个return double[3]的一个方程， 这个double[]代表着{这个node包括它自己和下面所有node的sum， 这个node和所有它的subtree里的node个数， 当前自己代表的tree的average, 如果root当前到了null了， 那么直接return new double[]{0.0, 0.0, 0.0}， 否则， 因为这个方程不用考虑root是不是null的情况，那么当前回归的double[0]也就是tree里所有的数字之和应该就是root的val加上dfs(root.left)[1], dfs(root.right)[1]; double[1]也就是tree里所有的node的个数也同理然而double[2]也就是当前maxAverage等于当前sum/cnt，dfs(root.left)[2], dfs(root.right)[2]， 也就是它自己和左右subtree average的最大值
   2. Runtime： 每个treeNode都跑到底了， 那么最坏的情况也就是O(n)， space： O(1）
2. NearestCity:
   1. 做法:因为pq跑的时间太长, 用了四个Map, 有两个都是<int, treeset<>()>来记录相同的x所对应的y或者相同的y所对应的x; 有一个map是对应的string和id; 还有最后一个map<int[], String>是坐标和对应的name; 我们把x相同城市的y , y相同的城市的x都放进去, 然后根据query的各个城市的string来找id再找到x, y, 然后再通过treeSet.lower(), treeSet.higher()来找到正好跟当前query的x、y最近的x 、 y坐标, 找到分别在x、y相同的情况下最近的城市, 然后再比较这两个城市的距离, 将最近的那个通过coordinateMap把name找到放在res里
   2. Runtime: 创建这四个map的时候是O(n), 然后query的length是k的话每次又找了k次O(log(n)), 所一共是O(N + klogN) = O(Nlog(N)), space是treeSet最多装n个, O(Nlog(N))

Jan 3rd

1. BreakAPalindrome(1328)
   1. 做法: 这个题其实是让我们修改一个Character所以造成这个对称的String不再对称, 做法是toCharArray()一下这个String, 如果碰到不是‘a’的位置直接改成‘a’并且return String.ValueOf(cl);如果可以forloop完证明所有的字符都是‘a’,那么我们把最后一个改成b然后return String.ValueOf(cl);

* 1. Runtime: O(n), space 也是O(n)

1. WaysToSplitPrimes(dp)
   1. 做法:这个题主要是要稳把这个String拆分成不同的prime最少需要多少次, 首先我们建一个helper来判定一段String是不是prime:
      1. Public Boolean isPrime(String s){

Int cur = Integer.parseInt(s);

For(int I = 2; I \* I <= cur; i++){

If(cur % I == 0) return false;

}

Return true;

}

然后我们再先建立一个dp[s.size()], 里面的数都是-1, 从i=1开始forloop一遍所有的字符, 如果

isPrime(s.subString(0, i)), 则dp[i] = 1;在这个forloop里紧接着我们判断dp[i]是不是1, 如果是的

话, 我们从j开始forloop一下并且 I, j 都应该<= 6, i+j <= n, 如果subString(0, i+j)也是prime, 那

么如果dp[I] == -1, dp[i+j] = 1 ; 如果dp[i] != -1, dp[i] = Math.min(dp[i]+1, dp[i+j]), 最后return

dp[length]

* 1. Runtime: 最多运算O(N^2), space就是dp的大小O[N]

1. FetchItemsToDisplay(pq)
   1. 做法: 主要就是建了一个item的class, 然后建一个pq, 这个pq要注意sortOrder和sortParameter, 我们需要用一个pq来存map里所有的值, 如果sortOrder = 0, 那他需要这个书是从小到大的排序的, 但我们的pq不应该是maxHeap, 我们正好相反要是minHeap, 因为最后一步我们要把所有比目标页数的页数小的页数的item都poll()出来, 然后再把目标页的item的name全部poll()到一个list里才算完成
   2. Runtime: 应该就是O(Nlog(N)), space也是O(Nlog(N)), 因为map也只是N
2. PackagingAutomation
   1. 做法: 主要就是先sort一下这个arr, 然后第一个的value不是1的话我们直接return 0, 否则我们把arr变成int[] l, 往下从1开始每个l[i] = Math.min(l[i-1]+1, l[i])来取极限情况, 然后最后return l[numGroups – 1]
   2. Runtime: O(n), space 就是一个int[]也是O(n)
3. MaximumSubarrayMaximum变种(把sons改成了一个TreeNode[])
   1. 做法: dfs基本思维都一样, 就是在浏览每一个son的num,cnt和average的时候不是简单的加上左右而是加上一个forloop一个个叠加或者用一个另外设置的int maxAverage一个个比较大小
   2. Runtime: 因为里面都加了一个forloop所以是O(N^2), Space是sons的大小最多是O(N)