Reorder log files

Java:

class Solution {

public String[] reorderLogFiles(String[] logs) {

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

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

//Separating letter and digits logs

for(String log:logs){

String[] str=log.split(" ");

String[] split1 = log.split(" ", 2);

System.out.println(Arrays.toString(split1));

String rest=str[1];

if(Character.isDigit(rest.charAt(0))){

digits.add(log);

}

else{

letters.add(log);

}

}

//System.out.println(letters);

//System.out.println(digits);

Collections.sort(letters, new Comparator<String>() {

public int compare(String o1, String o2) {

String s1 = o1.substring(o1.indexOf(" ") + 1);

String s2 = o2.substring(o2.indexOf(" ") + 1);

return s1.equals(s2) ? o1.compareTo(o2) : s1.compareTo(s2);

}

});

String[] output = new String[letters.size() + digits.size()];

for(int i = 0; i < letters.size(); i++) {

output[i] = letters.get(i);

}

for(int i = letters.size(); i < output.length; i++) {

output[i] = digits.get(i-letters.size());

}

return output;

}

}

Python:

class Solution:

def reorderLogFiles(self, logs: List[str]) -> List[str]:

letter\_logs, digit\_logs = [], []

for log in logs:

if log.split()[1].isalpha():

letter\_logs.append(log)

else:

digit\_logs.append(log)

def weirdOrder(log):

all\_parts = log.split()

key, rest = all\_parts[0], all\_parts[1:]

return ' '.join(rest) + ' ' + key

letter\_logs.sort(key=weirdOrder)

return letter\_logs + digit\_logs

Sort Characters By frequency

Java:

class Solution {

public String frequencySort(String s) {

HashMap<Character, Integer> hm=new HashMap<>();

for(int i=0; i<s.length(); i++){

char c=s.charAt(i);

hm.put(c, hm.getOrDefault(c, 0)+1);

}

System.out.println(hm);

//String res="";

List<Character> l=new ArrayList<>(hm.keySet());

Collections.sort(l, (a,b) -> hm.get(b)-hm.get(a)); //descending order

StringBuilder res = new StringBuilder();

for(char c:l){

int freq=hm.get(c);

for(int i=0; i<freq; i++){

res.append(c);

}

}

return res.toString();

}

}

Python:

class Solution:

def frequencySort(self, s: str) -> str:

freq = {}

for ch in s:

freq[ch] = freq.get(ch, 0) + 1

print(freq)

sort\_freq = sorted(freq.items(), key=lambda x: x[1], reverse=True)

print(''.join(x[0]\*x[1] for x in sort\_freq))

return ''.join(x[0]\*x[1] for x in sort\_freq)

Top K frequent elements

Python:

class Solution:

def topKFrequent(self, nums: List[int], k: int) -> List[int]:

freq={}

for l in nums:

freq[l]=freq.get(l, 0) + 1

#print(freq)

ans=[]

ans=sorted([(k,v) for k,v in freq.items()], key=lambda k:(k[1]), reverse=True)

res=[]

for i in range(k):

res.append(ans[i][0])

print(ans)

return res

Java:

class Solution {

public int[] topKFrequent(int[] nums, int k) {

HashMap<Integer, Integer> hm=new HashMap<>();

for(int n:nums){

hm.put(n,hm.getOrDefault(n,0)+1);

}

List<Integer> l=new ArrayList<>(hm.keySet());

Collections.sort(l, (a,b)->hm.get(b)-hm.get(a)); //sort by frequency i.e. reverse order thus b-a

int[] res=new int[k];

for(int i=0; i<k; i++){

res[i]=l.get(i);

}

return res;

}

}

K close points to the origin

https://leetcode.com/problems/k-closest-points-to-origin/discuss/220235/Java-Three-solutions-to-this-classical-K-th-problem.

Java Solution 1: O(nlogn)

class Solution {

public int[][] kClosest(int[][] points, int K) {

double[] dist=new double[points.length];

for(int i=0; i<points.length; i++){

int x=points[i][0];

int y=points[i][1];

dist[i]=Math.sqrt(x\*x+y\*y);

}

Arrays.sort(dist);

double kclose=dist[K-1];

int[][] ans = new int[K][2];

int temp=0;

for(int i=0; i<points.length; i++){

int x=points[i][0];

int y=points[i][1];

double d=Math.sqrt(x\*x+y\*y);

if(d<=kclose){

ans[temp++]=points[i];

}

}

return ans;

}

}

|  |  |
| --- | --- |
| Java Solution2.0: O(nlogK)  class Solution {  public int[][] kClosest(int[][] points, int K) {  if(points.length == 0 || points[0].length == 0) return points;  //Add elements in priority queue in ascending order i.e. min to max  PriorityQueue<int[]> pq = new PriorityQueue<>(  (a, b) -> ((a[0]\*a[0] + a[1]\*a[1]) - (b[0]\*b[0] + b[1]\*b[1])));  for (int i = 0; i < points.length; i++) {  pq.add(points[i]);  }  int[][] res = new int[K][2];  for (int i = 0; i < K; i++) {  int[] temp = pq.poll();  res[i][0] = temp[0];  res[i][1] = temp[1];  }  return res;  }  } | Java Solution2.1: O(nlogk)  class Solution {  public int[][] kClosest(int[][] points, int K) {  PriorityQueue<int[]> pq = new  //Add elements in PQ in descending order, so if size >k, poll operation will remove these points  PriorityQueue<int[]>((p1, p2) -> p2[0] \* p2[0] + p2[1] \* p2[1] – p1[0] \* p1[0] – p1[1] \* p1[1]);  for (int[] p : points) {  pq.offer(p);    if (pq.size() > K) {  pq.poll();  }  }  System.out.println(pq);  int[][] res = new int[K][2];  while (K > 0) {  res[--K] = pq.poll();  }  return res;  }  } |

Find Median from data stream

//Median is the middle element after sorting for odd arr and for even arr it is (arr[n/2] + arr[n/2-1])/2

Java:

class MedianFinder {

List<Integer> l;

double median;

/\*\* initialize your data structure here. \*/

public MedianFinder() {

this.l=new ArrayList<>();

this.median=0.0;

}

public void addNum(int num) {

l.add(num);

}

public double findMedian() {

int n=l.size();

Collections.sort(l);

if(n%2!=0){

// int sum=0;

// for(int i=0; i<n; i++){

// sum+=l.get(i);

// }

// median=sum/n;

int medianindex=(n-1)/2;

median=l.get(medianindex);

}

else{

int index1=n/2;

int index2=(n/2-1);

median=((double)l.get(index1)+(double)l.get(index2))/2;

}

return median;

}

}

/\*\*

\* Your MedianFinder object will be instantiated and called as such:

\* MedianFinder obj = new MedianFinder();

\* obj.addNum(num);

\* double param\_2 = obj.findMedian();

\*/

Java solution 2:

<https://www.programcreek.com/2015/01/leetcode-find-median-from-data-stream-java/>

Number of islands

class Solution {

char[][] copyOfgrid;

public int numIslands(char[][] grid) {

copyOfgrid=grid;

int no=0;

for(int i=0; i<grid.length; i++){

for(int j=0; j<grid[0].length; j++){

if(grid[i][j]=='1')

no+=dfs(copyOfgrid,i,j);

}

}

return no;

}

public static int dfs(char[][] grid, int i, int j){

if(i<0 || i>grid.length-1 || j<0 || j>grid[0].length-1 || grid[i][j]=='0') return 0;

grid[i][j]='0';

dfs(grid,i+1,j);

dfs(grid,i-1,j);

dfs(grid,i,j+1);

dfs(grid,i,j-1);

return 1;

}

}

Top K frequent words

class Solution {

//HashMap solution is faster than treemap

public List<String> topKFrequent(String[] words, int k) {

//TreeMap<String, Integer> hm=new TreeMap<>();

HashMap<String, Integer> hm=new HashMap<>();

for(String s:words){

hm.put(s,hm.getOrDefault(s,0)+1);

}

PriorityQueue<Map.Entry<String, Integer>> pq = new PriorityQueue<>(

(a,b) -> a.getValue() == b. getValue() ?

a.getKey().compareTo(b.getKey()) :

b.getValue() - a.getValue());

for(Map.Entry<String, Integer> entry: hm.entrySet()) {

pq.add(entry);

}

//List<String> l=new ArrayList<>(hm.keySet());

//Collections.sort(l, (a,b)->hm.get(b)-hm.get(a));

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

/\*for(int i=0; i<k; i++){

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

}\*/

while(!pq.isEmpty() && k!=0) {

Map.Entry<String, Integer> curr = pq.poll();

res.add(curr.getKey());

k--;

}

return res;

}

}

N-ary level order traversal

/\*

// Definition for a Node.

class Node {

public int val;

public List<Node> children;

public Node() {}

public Node(int \_val) {

val = \_val;

}

public Node(int \_val, List<Node> \_children) {

val = \_val;

children = \_children;

}

};

\*/

class Solution {

//TC: O(n) SC: O(n)

public List<List<Integer>> levelOrder(Node root) {

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

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

q.add(root);

while(!q.isEmpty()){

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

int size = q.size();

for (int i = 0; i < size; i++) {

Node node = q.poll();

level.add(node.val);

q.addAll(node.children);

}

l.add(level);

}

//To have it in reverse order

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

for(int i=l.size()-1; i>=0; i--){

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

}

System.out.println(res);

return l;

}

}

**Symmetric Tree**

/\*\*

\* Definition for a binary tree node.

\* public class TreeNode {

\* int val;

\* TreeNode left;

\* TreeNode right;

\* TreeNode() {}

\* TreeNode(int val) { this.val = val; }

\* TreeNode(int val, TreeNode left, TreeNode right) {

\* this.val = val;

\* this.left = left;

\* this.right = right;

\* }

\* }

\*/

class Solution {

public boolean isSymmetric(TreeNode root) {

Stack<TreeNode> s=new Stack<>();

s.push(root);

s.push(root);

while(!s.isEmpty()){

TreeNode t1=s.pop();

TreeNode t2=s.pop();

if(t1==null && t2==null) continue;

if(t1==null || t2==null) return false;

if(t1.val!=t2.val) return false;

s.push(t1.left);

s.push(t2.right);

s.push(t1.right);

s.push(t2.left);

}

return true;

}

}

public boolean isSymmetric(TreeNode root) {

return isMirror(root, root);

}

public boolean isMirror(TreeNode t1, TreeNode t2) {

if (t1 == null && t2 == null) return true;

if (t1 == null || t2 == null) return false;

return (t1.val == t2.val)

&& isMirror(t1.right, t2.left)

&& isMirror(t1.left, t2.right);

}

Count Univalue Subtree

class Solution {

public int countUnivalSubtrees(TreeNode root) {

Stack<TreeNode> s= new Stack<>();

int count=0;

if(root==null){

return 0;

}

s.push(root);

while(!s.isEmpty()){

TreeNode curr=s.pop();

if(curr.left!=null && curr.right!=null){

if(curr.left.val==curr.right.val){

count++;

}

}

else if(curr.left==null && curr.right==null){

count++;

}

if(curr.left!=null){

s.push(curr.left);

}

if(curr.right!=null){

s.push(curr.right);

}

}

return count;

}

}

Merge Intervals

public int[][] merge(int[][] intervals) {

if(intervals == null || intervals.length == 0)

return new int[0][0];

//Sort the input based on start value of intervals

Arrays.sort(intervals, (a,b) -> a[0] - b[0]);

int start=intervals[0][0];

int end=Integer.MAX\_VALUE;

List<int[]> result = new ArrayList<>();

//Check whether the start value falls inside the previous interval

//if yes, update the end value, as the start remains unchanged

//if not, add the previous interval to result and update the start and end to current values

for(int[] interval: intervals){

if(interval[0] <= end)

end = (end == Integer.MAX\_VALUE) ? interval[1]: Math.max(end,interval[1]);

else{

result.add(new int[]{start,end});

//System.out.println(result.toString());

start = interval[0];

end = interval[1];

}

}

result.add(new int[]{start,end});

//Convert the result list to int[][] and return (Please suggest if there is a better way to do this)

int[][] res = new int[result.size()][];

for(int i = 0; i < result.size(); i++)

res[i] = result.get(i);

return res;

}

Meeting Room

class Solution {

public boolean canAttendMeetings(int[][] intervals) {

Arrays.sort(intervals, (a,b) -> a[0] - b[0]);

for(int i=1; i<intervals.length; i++){

if(intervals[i-1][1]>intervals[i][0]) return false;

}

return true;

}

}

Meeting Room II

class Solution {

public int minMeetingRooms(int[][] intervals) {

if(intervals.length==0) return 0;

int[] start=new int[intervals.length];

int[] end=new int[intervals.length];

for (int i = 0; i < intervals.length; i++) {

start[i] = intervals[i][0];

end[i] = intervals[i][1];

}

Arrays.sort(start);

Arrays.sort(end);

int count=0, startPtr=0, endPtr=0;

while(startPtr<intervals.length){

if(start[startPtr]>=end[endPtr]){

count--;

endPtr++;

}

count++;

startPtr++;

}

return count;

}

}

https://leetcode.com/problems/validate-binary-search-tree/discuss/32112/Learn-one-iterative-inorder-traversal-apply-it-to-multiple-tree-questions-(Java-Solution)