1, Reorder Data in Log Files ☆ ☆ ☆

```java
class Solution {
    public String[] reorderLogFiles(String[] logs) {
        Arrays.sort(logs,(log1,log2)->{
            String[] split1 = log1.split(" ",2);
            String[] split2 = log2.split(" ",2);
            boolean isDigit1 = Character.isDigit(split1[1].charAt(0));
            boolean isDigit2 = Character.isDigit(split2[1].charAt(0));
            if(!isDigit1 && !isDigit2){
                int cmp = split1[1].compareTo(split2[1]);
                if(cmp != 0) return cmp;
                return split1[0].compareTo(split2[0]);
            }
            return isDigit1 ? (isDigit2 ? 0 : 1) : -1;
        });
        return logs;
    }
}
```

2,Optimal Utilization ☆ ☆ ☆

```java
private List<int[]> getPairs(List<int[]> a, List<int[]> b, int target) {
    Collections.sort(a, (i,j) -> i[1] - j[1]);
    Collections.sort(b, (i,j) -> i[1] - j[1]);
    List<int[]> result = new ArrayList<>();
    int max = Integer.MIN_VALUE;
    int m = a.size();
    int n = b.size();
    int i =0;
    int j =n-1;
    while(i<m && j >= 0) {
        int sum = a.get(i)[1] + b.get(j)[1];
        if(sum > target) {
            --j;
        } else {
            if(max <= sum) {
                if(max < sum) {
                    max = sum;
                    result.clear();
                }
                result.add(new int[]{a.get(i)[0], b.get(j)[0]});
                int index = j-1;
                while(index >=0 && b.get(index)[1] == b.get(index+1)[1]) {
                    result.add(new int[]{a.get(i)[0], b.get(index--)[0]});
                }
            }
            ++i;
        }
    }
    return result;
}
```

Min Cost to Connect Ropes / Min Time to Merge Files ☆ ☆ ☆

```java
public static int mergeFiles(int[] files){
    PriorityQueue<Integer> q = new PriorityQueue<>();
    for(int file : files) q.offer(file);
    int ans = 0;
    while(q.size() > 1){
        int a = q.poll();///
        int b = q.poll();
        int newEle = a + b;
        ans += newEle;
        q.offer(newEle);
    }
    return ans;
}
```

3,Treasure Island / Min Distance to Remove the Obstacle (BFS) ☆ ☆ ☆

```java
public static int treasureIsland(char[][] island){
    Queue<Point> q = new LinkedList<>();
    int steps = 0;
    q.add(new Point(0,0));
    int[][] dirs = {{0,1},{0,-1},{1,0},{-1,0}};
    boolean[][] isVisited = new boolean[island.length][island[0].length];
    isVisited[0][0] = true;
    while(!q.isEmpty()){
        int size = q.size();
        for(int i = 0; i < size;i++){
            Point p = q.poll();
            int x = p.x;
            int y = p.y;
            if ( island[x][y] == 'X') return steps;
            for(int[] dir : dirs){
                int newX = x + dir[0];
                int newY = y + dir[1];
                if( newX >= 0 && newX < island.length && newY >= 0 && newY < island[0].length
                    && island[newX][newY] != 'D' && !isVisited[newX][newY]){
                    q.add(new Point(newX,newY));
                    isVisited[newX][newY] = true;
                }
            }
        }
        steps++;
    }
    return 0;

}
```

4,Treasure Island II (Multi-source BFS)

```java
public static int treasureIslandTwo(char[][] islands){
    int[][] dirs = {{0,1},{0,-1},{1,0},{-1,0}};
    if(islands.length == 0 || islands[0].length == 0 || islands == null) return -1;
    int R = islands.length;
    int C = islands[0].length;
    Queue<Point> q = new LinkedList<>();
    for(int i = 0; i < R; i++){
        for(int j = 0; j < C ;j++){
            if(islands[i][j]=='S'){
                q.add(new Point(i,j));
                islands[i][j] = 'D';
            }
        }
    }

    int step = 1;
    while(!q.isEmpty()){
        int size = q.size();
        for(int i = 0; i < size;i++){
            Point p = q.poll();
            int x = p.x;
            int y = p.y;
            for(int[] dir : dirs){
                int newX = x + dir[0];
                int newY = y + dir[1];
                if( newX < 0 || newX >= R || newY < 0 || newY >= C
                        || islands[newX][newY] == 'D') continue;
                if(islands[newX][newY] == 'E') return step;
                q.add(new Point(newX,newY));
                islands[newX][newY] = 'D';
            }
        }
        step++;
    }
    return -1;
}
```

5,Copy List with Random Pointer ☆

```java
class Solution {
    public Node copyRandomList(Node head) {
        HashMap<Node,Node> map = new HashMap<>();
        Node cur = head;
        while(cur != null){
            map.put(cur, new Node(cur.val));
            cur = cur.next;
        }

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

6,Merge Two Sorted Lists ☆

```
class Solution {
    public ListNode mergeTwoLists(ListNode l1, ListNode l2) {
        ListNode dummy = new ListNode(0);
        ListNode cur = dummy;
        while(l1 != null && l2 != null){
            if(l1.val > l2.val){
                cur.next = l2;
                l2 = l2.next;
            }else{
                cur.next = l1;
                l1 = l1.next;
            }
            cur = cur.next;
        }
        if(l1 != null) cur.next = l1;
        if(l2 != null) cur.next = l2;
        return dummy.next;
    }
}
```

7,Subtree of Another Tree ☆

```
/**
 * Definition for a binary tree node.
 * public class TreeNode {
 *     int val;
 *     TreeNode left;
 *     TreeNode right;
 *     TreeNode(int x) { val = x; }
 * }
 */
class Solution {
    public boolean isSubtree(TreeNode s, TreeNode t) {
        if(t == null) return true;
        if(s == null) return false;
        return isSameTree(s,t) || isSubtree(s.left,t) || isSubtree(s.right,t);
    }
    public boolean isSameTree(TreeNode s, TreeNode t){
        if(s == null && t == null) return true;
        if(s == null || t == null) return false;
        if(s.val != t.val) return false;
        return isSameTree(s.left,t.left) && isSameTree(s.right,t.right);
    }
}
```

8,Search a 2D Matrix II ☆☆

```
class Solution {
    public boolean searchMatrix(int[][] matrix, int target) {
        int x = matrix.length-1;
        int y = 0;
        while(x >= 0 && x < matrix.length && y >= 0 && y < matrix[0].length){
            if(target == matrix[x][y]) return true;
            else if (target > matrix[x][y]) y++;
            else x--;
        }
        return false;
    }
}
```

9,Critical Connections ☆☆

```
static int time = 0;
static final int NIL = -1;
static List<Integer[]> ans1 = new LinkedList<>();
```

```
public static class Graph{
    int v;
    private LinkedList<Integer> adj[];
    Graph(int V){
        v = V;
        adj = new LinkedList[v+1];
        for(int i = 0; i < v+1;i++) {
            adj[i] = new LinkedList<>();
        }
    }
    void addEdge(int v,int w){
        adj[v].add(w);
        adj[w].add(v);
    }
}
public static void bridgeDfs(Graph g, int u, boolean[] visited,int[] disc,int[] low,int[] parent){
    visited[u] = true;
    disc[u] = low[u] = ++time;
    Iterator<Integer> i = (g.adj[u]).iterator();
    while(i.hasNext()){
        int v = i.next();
        if(!visited[v]){
            parent[v] = u;
            bridgeDfs(g,v,visited,disc,low,parent);
            low[u] = Math.min(low[u],low[v]);
            if(low[v] > disc[u]) ans1.add(new Integer[]{u,v});
        } else if(v != parent[u]){
            low[u] = Math.min(low[u],disc[v]);
        }
    }

}
public static int[][] criticalConnections(int[][] edges, int v ){
    Graph g = new Graph(v);
    for(int i = 0; i < edges.length;i++){
        g.addEdge(edges[i][0],edges[i][1]);
    }
    boolean[] visited = new boolean[v+1];
    int[] disc = new int[v+1];
    int[] low = new int[v+1];
    int parent[] = new int[v+1];

    for(int i = 0; i < v; i++){
        parent[i] = NIL;
        visited[i] = false;
    }

    for(int i = 0; i < v;i++){
        if(!visited[i]) bridgeDfs(g,i,visited,disc,low,parent);
    }

    int[][] res = new int[ans1.size()][2];
    int index=0;
    for(Integer[] temp: ans1)
    {
        res[index][0]=temp[0];
        res[index][1]=temp[1];
        index++;
    }
    return res;
}
```

10,Favorite Genres ☆☆

```
public static Map<String,List<String>> favoritegenre(Map<String,List<String>> userMap, Map<String,List<String>>
genreMap){
    Map<String,List<String>> res = new HashMap<>();
    Map<String,String> songtogenre = new HashMap<>();

    for(String genre : genreMap.keySet()){
        List<String> songs = genreMap.get(genre);
        for(String song : songs){
            songtogenre.put(song,genre);
        }
    }

    Map<String,Map<String,Integer>> usergenrecount = new HashMap<>();
    for(String user : userMap.keySet()) {
        if(!usergenrecount.containsKey(user))
            usergenrecount.put(user, new HashMap<>());
        List<String> songs = userMap.get(user);
        for(String song : songs) {
            String genre = songtogenre.get(song);
            int count = usergenrecount.get(user).getOrDefault(genre, 0) + 1;
            usergenrecount.get(user).put(genre, count);
        }
    }

    for(String user : usergenrecount.keySet()) {
        if(!res.containsKey(user))
            res.put(user, new ArrayList<>());
        Map<String, Integer> pair = usergenrecount.get(user);
        int max = 0;
        List<String> favgenre = new ArrayList<>();
        for(String genre : pair.keySet()) {
            if(favgenre.size() == 0) {
                favgenre.add(genre);
                max = pair.get(genre);
            } else if(pair.get(genre) > max) {
                favgenre.clear();
                favgenre.add(genre);
                max = pair.get(genre);
            } else if(pair.get(genre) == max)
                favgenre.add(genre);
        }
        res.put(user, favgenre);
    }
    return res;

}
```

11,Most Common Word

```
class Solution {
    public String mostCommonWord(String paragraph, String[] banned) {
        HashSet<String> bannedSet = new HashSet<>();
        List<String> list=split(paragraph);
        for(int i=0;i<banned.length;i++) bannedSet.add(banned[i].trim());

        HashMap<String,Integer> map = new HashMap<>();
        int max = 0;
        String ans = "";
        for(String para: list){
            String temp = para.toLowerCase();
            if(!bannedSet.contains(temp)){
                int cnt = map.getOrDefault(temp,0) + 1;
                map.put(temp,cnt);
                if(cnt > max){
                    max = cnt;
                    ans = temp;
                }
            }
        }
        return ans;
    }
    public List<String> split(String str){
        Set<Character> set=new HashSet<>();
        List<String> list=new ArrayList<>();
        set.add(' ');
        set.add('!');
        set.add('?');
        set.add('\'');
        set.add(',');
        set.add(';');
        set.add('.');
        int len=str.length();
        int i=0,j=0;
        while(i<len){
            while(i<len&&set.contains(str.charAt(i)))i++;
            j=i;
            StringBuilder sb=new StringBuilder();
            while(j<len&&!set.contains(str.charAt(j))){
                sb.append(str.charAt(j));
                j++;
            }
            i=j;
            if(sb.length()!=0)
                list.add(sb.toString());
        }
        return list;
    }
}
```

12,K Closest Points to Origin

```
class Solution {
    public int[][] kClosest(int[][] points, int K) {
        int[] dists = new int[points.length];
        for(int i=0;i<points.length;i++){
            dists[i] = caculate(points[i]);
        }

        Arrays.sort(dists);
        int diskK = dists[K-1];

        int[][] ans = new int[K][2];
        int t=0;
        for(int i=0;i<points.length;i++){
            if(caculate(points[i]) <= diskK) ans[t++] = points[i];
        }
        return ans;
    }
    public int caculate(int[] points){
        return points[0]*points[0] + points[1]*points[1];
    }
}
```

13,Longest Palindromic Substring

```
class Solution {
    String res="";
    public String longestPalindrome(String s) {
        for(int i = 0; i < s.length();i++){
            center(s,i,i);
            center(s,i,i+1);
        }
        return res;
    }
    public void center(String s,int l,int r){
        while(l>=0 && r<s.length() && s.charAt(l)==s.charAt(r) ){
            l--;
            r++;
        }
        String cur = s.substring(l+1,r);
        if(cur.length() >  res.length()) res = cur;
    }
}
```

14,Two Sum - Unique Pairs

```
public static int newTwoSum(int[] nums, int target) {
        Set<Integer> set1 = new HashSet<>();
        Set<Integer> set2 = new HashSet<>();
        for (int i = 0; i < nums.length; i++) {
            set1.add(nums[i]);
            if (set1.contains(target - nums[i])) {
                set2.add(nums[i]);
            }
        }
        return set2.size();
}
```

15,Movies on Flight (Two Sum Closest)

```
public static int[] twoSumCloset(int[] nums,int target){
    Arrays.sort(nums);
    int[] result = new int[2];
    int left = 0, right = nums.length-1;
    target = target - 30;
    int max = Integer.MIN_VALUE;
    int i = 0, j = 0;

    while(left <= right){
        int sum = nums[left] + nums[right];
        if(sum <= target){
            if(max < sum){
                max = sum;
                i = nums[left];
                j = nums[right];
            }
            left++;
        }else{
            right--;
        }
    }
    result[0] = i;
    result[1] = j;
    return result;
}
```

16,Find Pair With Given Sum (a.k.a. Sort Center)

```
public static int[] findPairWithGivenSum(int[] nums,int target){
    HashMap<Integer,Integer> map = new HashMap<>();
    int m = 0;
    int n = 0;
    int max = Integer.MIN_VALUE;
    for(int i = 0 ; i < nums.length;i++){
        int com = target - 30 - nums[i];
        if(!map.containsKey(com)){
            map.put(nums[i],i);
        }else{
            int sum = i + map.get(com);
            if(sum > max){
                max = sum;
                m = i;
                n = map.get(com);
            }
        }
    }
    return new int[]{m,n};
}
```

17,Find Pair With Max Appeal Sum

```
public static int[] findPairWithMaxAppearSum(int[] arr){
    int m = 0;
    int n = 0;
    int max = Integer.MIN_VALUE;
    for(int i = 0; i < arr.length;i++){
        for(int j = 0; j < arr.length;j++){
            int sum = arr[i]-i+arr[j]+j;
            if(sum > max){
                max = sum;
                m = i;
                n= j;
            }
        }
    }
    return new int[]{m,n};
}
```

18,Min Cost to Connect All Nodes (a.k.a. Min Cost to Add New Roads)

```
public class Main {
    int[] parent;
    int component;

    private int find(int v){
        if(parent[v] == v) return v;
        return parent[v] = find(parent[v]);
    }

    private void connect(int v1, int v2){
        if(find(v1) == find(v2)) return;
        int root = find(v1);
        while(v2 != parent[v2]){
            int temp = parent[v2];
            parent[v2] = root;
            v2 = temp;
        }
        --component;
        parent[v2] = root;
    }

    private boolean isConnected(int v1, int v2){
        return find(v1) == find(v2);
    }

    public int minCosttoConnectAllNodes(int n, int edges[][], int newEdges[][]){
        parent = new int[n + 1]; component = n;
        for(int i = 0; i <= n; ++i) parent[i] = i;
        for(int[] edge: edges) connect(edge[0], edge[1]);
        Arrays.sort(newEdges, (a, b) -> (a[2] - b[2]));
        int cost = 0;
        for(int i = 0; i < newEdges.length; ++i){
            if(!isConnected(newEdges[i][0], newEdges[i][1])){
                connect(newEdges[i][0], newEdges[i][1]);
                cost += newEdges[i][2];
                if(component == 1) return cost;
            }
        }
        return -1;
    }
    public static void main(String[] args) {
        Main main = new Main();
        int[] n_tests = {6};
        int[][][] edges_tests = { {{1, 4}, {4, 5}, {2, 3}}};
        int[][][] newEdges_tests = { {{1, 2, 5}, {1, 3, 10}, {1, 6, 2}, {5, 6, 5}}};
        for(int i = 0; i < n_tests.length; ++i){
            System.out.println(main.minCosttoConnectAllNodes(n_tests[i], edges_tests[i],
                    newEdges_tests[i]));
        }
    }
}
```

19,Min Cost to Repair Edges (MST)

```
import java.util.*;

public class Min_Cost_to_Repair_Edges {
    //time: O (nodes * edges), space: O(nodes)
    public static int minCost(int n, int[][] edges, int[][] edgesToRepair) {
        if (n == 0) return -1;
        int[] roots = new int[n + 1];// id -> index + 1
        Arrays.fill(roots, -1);

        HashSet<String> broken = new HashSet<>();//record broken edges
        for (int[] edge : edgesToRepair) {
            broken.add(Arrays.toString(Arrays.copyOfRange(edge, 0, 2)));
        }


        for (int[] edge : edges) {
            if (broken.contains(Arrays.toString(edge))) continue;
            int left = find(roots, edge[0]);
            int right = find(roots, edge[1]);
            if (left != right) {
                n--;
                roots[left] = right;
            }
        }
        int res = 0;
        Arrays.sort(edgesToRepair, (a, b) -> a[2] - b[2]);
        for (int[] edge : edgesToRepair) {
            if (n == 1) break;
            int left = find(roots, edge[0]);
            int right = find(roots, edge[1]);
            if (left == right) continue;
            roots[left] = right;
            n--;
            res += edge[2];
        }
        return n == 1 ? res : -1; //return -1 if the graph is not connected, (not possible in the problem)
    }

    private static int find(int[] roots, int i) {
        while (roots[i] != -1) {
            i = roots[i];
        }
        return i;
    }

    public static void main(String[] args) {
        int[][] edges = new int[][]{{1, 2}, {2, 3}, {4, 5}, {3, 5}};
        int[][] repair = new int[][]{{1, 6, 410}, {2, 4, 800}};
        System.out.println(minCost(6, edges, repair));

        edges = new int[][]{{1, 2}, {2, 3}, {3, 4}, {4, 5}, {1, 5}};
        repair = new int[][]{{1, 2, 12}, {3, 4, 30}, {1, 5, 8}};
        System.out.println(minCost(5, edges, repair));
    }
}
```

20,Prison Cells After N Days

```
class Solution {
    public int[] prisonAfterNDays(int[] cells, int N) {
        for (N = (N - 1) % 14 + 1; N > 0; N--) {
            int[] temp = new int[8];
            for (int i = 1; i < 7 ; i++)
                temp[i] = (cells[i - 1] == cells[i + 1]) ? 1 : 0;
            cells = temp;
        }
        return cells;
    }
}
```

21,Substrings of size K with K distinct chars

```
public static void main( String[] args )
{
    String s = "pqpqs";
    int k = 2;

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

    for( int i = 0; i < s.length(); i++ )
    {
        char c = s.charAt( i );
        String tmp = "" + c;
        Set<Character> set = new HashSet<Character>();
        set.add( c );

        for( int j = i+1; j < s.length(); j++ )
        {
            char nc = s.charAt( j );
            set.add( nc );
            tmp += nc;
            if( tmp.length() >= k && set.size() == k ) ans.add( tmp );
        }
    }
    System.out.println( ans );
}
```

22,Count substrings with exactly K distinct chars

```
public static void main( String[] args )
{
    String s = "pqpqs";
    int k = 2;

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

    for( int i = 0; i < s.length(); i++ )
    {
        char c = s.charAt( i );
        String tmp = "" + c;
        Set<Character> set = new HashSet<Character>();
        set.add( c );

        for( int j = i+1; j < s.length(); j++ )
        {
            char nc = s.charAt( j );
            set.add( nc );
            tmp += nc;
            if( tmp.length() >= k && set.size() == k ) ans.add( tmp );
        }
    }
    System.out.println( ans );
}
```

23,Partition Labels

```
class Solution {
    public List<Integer> partitionLabels(String S) {
        int[] last = new int[26];
        for(int i=0;i<S.length();i++) last[S.charAt(i)-'a']=i;

        List<Integer> ans = new ArrayList<>();
        int j = 0;
        int anchor = 0;
        for(int i = 0; i < S.length();i++){
            j = Math.max(j,last[S.charAt(i)-'a']);
            if(i==j){
                ans.add(j-anchor+1);
                anchor = i + 1;
            }
        }

        return ans;
    }
}
```

24,Roll Dice O(n)

```
public static int missingNumber(int[] nums) {
    // initializations
    int min = Integer.MAX_VALUE;
    int temp;
    int[] count = new int[7];

    // counting occurrences of each number in the nums array and placing in count[]
    for (int num : nums) count[num]++;

    // can flip each dice to any number between 1 and 6, so we find the min of each possible top face.
    for (int i = 1; i < 7; i++) {
        /*
         * count twice if compliment of desired (2*count[7-desired]) +
         * total number of dice we have (nums.length) -
         * count of desired occurrences (count[desired] -
         * count of compliments (count[7-desired]).
         * simplify to:
         */
        temp = 2*count[7 - i] + nums.length - count[i] - count[7-i];
        // check if what we calculated for moves is less than something we already found.
        min = temp < min ? temp : min;
    }
    return min;
}
```

25,Subtree with Maximum Average

```
class Solution {

    double max = Integer.MIN_VALUE;
    TreeNode maxNode = null;

    public TreeNode maximumAverageSubtree(TreeNode root) {
        if (root == null) return null;
        helper(root);
        return maxNode;
    }

    private double[] helper(TreeNode root) {
        if (root == null) return new double[] {0, 0};

        double curTotal = root.val;
        double count = 1;
        for (TreeNode child : root.children) {
            double[] cur = helper(child);
            curTotal += cur[0];
            count += cur[1];
        }
        double avg = curTotal / count;
        if (count > 1 && avg > max) { //taking "at least 1 child" into account
            max = avg;
            maxNode = root;
        }
        return new double[] {curTotal, count};
    }
}
```

26,Longest string without 3 consecutive characters

```
public static String generateString(Map<Character, Integer> map) {
    PriorityQueue<Map.Entry<Character, Integer>> maxHeap =
            new PriorityQueue<Map.Entry<Character, Integer>>((a, b) -> b.getValue() - a.getValue());

    int cnt = 0;
    for (Map.Entry<Character, Integer> e: map.entrySet()) {
        cnt += e.getValue();
        maxHeap.add(e);
    }
    // only one char can be on hold
    Map.Entry<Character, Integer> onHold = null;
    StringBuilder sb = new StringBuilder();
    while (!maxHeap.isEmpty()) {
        Map.Entry<Character, Integer> cur = maxHeap.poll();
        sb.append(cur.getKey());
        if (onHold != null) {
            maxHeap.add(onHold);
            onHold = null;
        }
        int curValue = cur.getValue();
        if (curValue > 1) {
            cur.setValue(curValue-1);
            if (sb.length() >= 2 && cur.getKey() == sb.charAt(sb.length()-2)) { // on hold
                onHold = cur;
            } else {  // add back to heap
                maxHeap.add(cur);
            }
        }

    }
    return sb.length() == cnt ? sb.toString(): "";
}
```

27,Longest string made up of only vowels

```
private boolean isVowel(char c){
    return c == 'a' || c == 'e' || c == 'i' || c == 'o' || c == 'u';
}
public int longestString(String s){
    int len = s.length();
    int start = 0, end = len - 1;
    while(start < len && isVowel(s.charAt(start))) ++start;
    while(end >= 0 && isVowel(s.charAt(end))) --end;
    // checking area come to [start, end]
    if(start >= len) return len;
    int res = start + len - 1 - end;
    int longest = 0, sum = 0;
    for(int i = start + 1; i <= end; ++i){
        if(isVowel(s.charAt(i)))
            ++sum;
        else
            sum = 0;
        longest = Math.max(sum, longest);
    }
    return longest + res;
}
```