**Fewest Coins**

class Result { static final int MAX\_CHARS = 256; public static int fewestCoins(String coins) { // Write your code here int dcount=0; boolean[] visited = new boolean[MAX\_CHARS]; Arrays.fill(visited, false); for (int i = 0; i < coins.length(); i++) { if (visited[coins.charAt(i)] == false) { visited[coins.charAt(i)] = true; dcount+=1; } } int start = 0; int minlen = Integer.MAX\_VALUE; int count = 0; int[] ccount = new int[MAX\_CHARS]; for (int j = 0; j < coins.length(); j++){ ccount[coins.charAt(j)]++; if (ccount[coins.charAt(j)] == 1) count++; if (count == dcount) { while (ccount[coins.charAt(start)] > 1) { if (ccount[coins.charAt(start)] > 1) ccount[coins.charAt(start)]--; start++; } int len = j - start + 1; if (minlen > len) { minlen = len; } } } return minlen; }

}

## ****Fun with Vowels****

class Result { public static int longestVowelSubsequence(String s) { // Write your code here if (s.length() <= 0) throw new IllegalArgumentException();

Map<Character, Character> precCharMap = new HashMap<>(); precCharMap.put('u', 'o'); precCharMap.put('o', 'i'); precCharMap.put('i', 'e'); precCharMap.put('e', 'a');

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

for (char currChar : s.toCharArray()) { String currCharLongestSub; String precCharLongestSub = null; if (currChar == 'a') { currCharLongestSub = runningLongestSubMap.getOrDefault(currChar, ""); } else { currCharLongestSub = runningLongestSubMap.get(currChar); char precChar = precCharMap.get(currChar); precCharLongestSub = runningLongestSubMap.get(precChar); } if (precCharLongestSub == null && currCharLongestSub != null) { updateRunningLongestSubMap(currCharLongestSub, currChar, runningLongestSubMap); } else if (currCharLongestSub == null && precCharLongestSub != null) { updateRunningLongestSubMap(precCharLongestSub, currChar, runningLongestSubMap); } else if (currCharLongestSub != null && precCharLongestSub != null) { if (currCharLongestSub.length() < precCharLongestSub.length()) { updateRunningLongestSubMap(precCharLongestSub, currChar, runningLongestSubMap); } else { updateRunningLongestSubMap(currCharLongestSub, currChar, runningLongestSubMap); } } }

if (runningLongestSubMap.get('u') == null) { return 0; } return runningLongestSubMap.get('u').length();

}

private static void updateRunningLongestSubMap(String longestSub, char currChar, Map<Character, String> runningLongestSubMap) { String currCharLongestSub = longestSub + currChar; runningLongestSubMap.put(currChar, currCharLongestSub); }

}

## ****Keyboard solution****

class Result { public static int find(char[] a, char target) { int i; for (i = 0; i < 9; i++) if (a[i] == target) break; return i; }

public static int entryTime(String s, String keypad) { char[] keyboard = keypad.toCharArray(); char[] str = s.toCharArray(); int index = find(keyboard,str[0]); int pos\_i,pos\_j; pos\_i = index/3; pos\_j = index%3; int curr\_index,curr\_i,curr\_j; int time=0,curr; for(int i=1;i < str.length; i++){ curr\_index = find(keyboard,str[i]); curr\_i = curr\_index/3; curr\_j = curr\_index%3; //System.out.println(curr\_i+" "+curr\_j); curr = Math.max(Math.abs(pos\_i-curr\_i),Math.abs(pos\_j-curr\_j)); time = time + curr; pos\_i = curr\_i; pos\_j = curr\_j;

} return time;

}

}

## ****Palindrome counter****

class Result { public static int countPalindromes(String s) { // Write your code here int count = 0; if (s == null || s.length() == 0) { return count; } boolean[][] dp = new boolean[s.length()][s.length()]; for (int i = 0; i < s.length(); i++) { dp[i][i] = true; count++; } for (int i = 1; i < s.length(); i++) { if (s.charAt(i - 1) == s.charAt(i)) { dp[i - 1][i] = true; count++; } } for (int j = 2; j < s.length(); j++) { for (int i = 0; i < j; i++) { if (dp[i + 1][j - 1] && s.charAt(i) == s.charAt(j)) { dp[i][j] = true; count++; } } } return count; } }

## Price Check

class Result { public static int countPalindromes(String s) { // Write your code here int count = 0; if (s == null || s.length() == 0) { return count; } boolean[][] dp = new boolean[s.length()][s.length()]; for (int i = 0; i < s.length(); i++) { dp[i][i] = true; count++; } for (int i = 1; i < s.length(); i++) { if (s.charAt(i - 1) == s.charAt(i)) { dp[i - 1][i] = true; count++; } } for (int j = 2; j < s.length(); j++) { for (int i = 0; i < j; i++) { if (dp[i + 1][j - 1] && s.charAt(i) == s.charAt(j)) { dp[i][j] = true; count++; } } } return count; } }

## ****Product Defect Java****

class Result { public static int largestArea(List<List> samples) { // Write your code here int i,j; int R = samples.size(); int C = samples.get(0).size(); int S[][] = new int[R][C]; int M[][]=new int[R][C]; for(int d=0;d<R;d++) { for(int b=0;b<C;b++) { M[d][b]=samples.get(d).get(b); } } int max\_of\_s, max\_i, max\_j; for(i = 0; i < R; i++) S[i][0] = M[i][0]; for(j = 0; j < C; j++) S[0][j] = M[0][j]; for(i = 1; i < R; i++) { for(j = 1; j < C; j++) { if(M[i][j] == 1) S[i][j] = Math.min(S[i][j-1], Math.min(S[i-1][j], S[i-1][j-1])) + 1; else S[i][j] = 0; }

} max\_of\_s = S[0][0]; max\_i = 0; max\_j = 0; for(i = 0; i < R; i++) { for(j = 0; j < C; j++) { if(max\_of\_s < S[i][j]) { max\_of\_s = S[i][j]; max\_i = i; max\_j = j; } } } return Math.abs((max\_i-max\_of\_s)-max\_i); }

}

## ****Product Sort****

class Result { public static List itemsSort(List items) { // Write your code here int size=items.size(); int[][] arr=new int[size][2]; int i=0,index=0; while(i<size){ int temp=0,x=0; for(int j=0;j<index;j++){ if(arr[j][0]==items.get(i)){ temp=1; x=j; } } if(temp==0){ arr[index][0]=items.get(i); arr[index][1]=1; index++; } else{ arr[x][1]++; } i++; } for(i=0;i<index-1;i++){ for(int j=i+1;j<index;j++){ if(arr[i][1]>arr[j][1]){ int temp1=arr[i][0]; int temp2=arr[i][1]; arr[i][0]=arr[j][0]; arr[i][1]=arr[j][1]; arr[j][0]=temp1; arr[j][1]=temp2; } if(arr[i][1]==arr[j][1]){ if(arr[i][0]>arr[j][0]){ int temp1=arr[i][0]; int temp2=arr[i][1]; arr[i][0]=arr[j][0]; arr[i][1]=arr[j][1]; arr[j][0]=temp1; arr[j][1]=temp2; } } } } List list=new ArrayList(); for(i=0;i<index;i++){ for(int j=0;j<arr[i][1];j++){ list.add(arr[i][0]); } } return list; } }