## areBracketsBalanced

// function to check if brackets are balanced

**static** **boolean** areBracketsBalanced(String expr)

{

// Using ArrayDeque is faster than using Stack class

Deque<Character> stack = **new** ArrayDeque<Character>();

// Traversing the Expression

**for** (**int** i = 0; i < expr.length(); i++)

{

**char** x = expr.charAt(i);

**if** (x == '(' || x == '[' || x == '{')

{

// Push the element in the stack

stack.push(x);

**continue**;

}

System.***out***.println(stack);

// If current character is not opening

// bracket, then it must be closing. So stack

// cannot be empty at this point.

**if** (stack.isEmpty())

**return** **false**;

**char** check;

**switch** (x) {

**case** ')':

check = stack.pop();

**if** (check == '{' || check == '[')

**return** **false**;

**break**;

**case** '}':

check = stack.pop();

**if** (check == '(' || check == '[')

**return** **false**;

**break**;

**case** ']':

check = stack.pop();

**if** (check == '(' || check == '{')

**return** **false**;

**break**;

}

}

// Check Empty Stack

**return** (stack.isEmpty());

}

## findPlatformsRequiredForStation

**static** **int** findPlatformsRequiredForStation(**int** arr[], **int** dep[], **int** n)

{

**int** platform\_needed = 0, maxPlatforms = 0;

Arrays.*sort*(arr);

Arrays.*sort*(dep);

**int** i = 0, j = 0;

// Similar to merge in merge sort

**while** (i < n && j < n)

{

**if** (arr[i] < dep[j])

{

platform\_needed++;

i++;

**if** (platform\_needed > maxPlatforms)

maxPlatforms = platform\_needed;

}

**else**

{

platform\_needed--;

j++;

}

}

**return** maxPlatforms;

}

## getOddTimesElementHashing

**static** **int** getOddTimesElementHashing(**int** ar[])

{

**int** i;

Map<Integer,Integer> elements=**new** HashMap<Integer,Integer>();

**for** (i = 0; i < ar.length; i++)

{

**int** element=ar[i];

**if**(elements.get(element)==**null**)

{

elements.put(element, 1);

}

**else**

elements.put(element, elements.get(element)+1);

}

//List<Integer> list=elements.entrySet().stream().map(entry->{if(entry.getValue()%2==0) return entry.getValue()}).collect(Collectors.toList());

**for** (Entry<Integer,Integer> entry:elements.entrySet()) {

**if**(entry.getValue()%2==1)

{

**return** entry.getKey();

}

}

**return** -1;

}

## convert\_String\_To\_Number

**public** **static** **int** convert\_String\_To\_Number(String numStr) {

**char** ch[] = numStr.toCharArray();

**int** num = 0;

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

num = num \* 10 + ((**int**)numStr.charAt(i) - 48);

}

**return** num;

}

## printPairsUsingSet

// print sum of two numbers equal to given number

**public** **static** **void** printPairsUsingSet() {

// given sum=7

**int** numbers[] = { 2, 4, 3, 5, 6, -2, 4, 7, 8, 9 }, sum = 7;

**if** (numbers.length < 2) {

**return**;

}

Set set = **new** HashSet(numbers.length);

**for** (**int** value : numbers) {

**int** target = sum - value;

// if target number is not in set then add

**if** (!set.contains(target)) {

set.add(value);

} **else** {

System.***out***.printf("(%d, %d) %n", value, target);

}

}

System.***out***.println(set);

}

## maxSumSubArray

**public** **static** **void** maxSumSubArray() {

**int** maxSumSoFar = -2147483648;

**int** curSum = 0, maxSum;

**int** start = 0;

**int** end = 0, s = 0, i;

**int**[] array = { -2, 1, -3, 4, -1, 2, 1, -5, 4 };

// int []array={-6,2,-3,-4,-1,-5,-5};

**for** (i = 0; i < array.length; i++) {

curSum = curSum + array[i];//

**if** (curSum > maxSumSoFar) {

maxSumSoFar = curSum;

start = s;

end = i;

}

**if** (curSum < 0) {

curSum = 0;

s = i + 1;

}

}

**for** (i = start; i <= end; i++)

System.***out***.print(array[i] + " ");

System.***out***.println(start);

}

## revereseArray

**static** **void** rvereseArray(**int** arr1[], **int** start, **int** end) {

**int** temp;

**while**(start<end) {

temp=arr1[start];

arr1[start]=arr1[end];

arr1[end]=temp;

start++;

end--;

}

}

## secondHighest

**private** **static** **int** secondHighest(**int**[] arr) {

**int** secondHighest = Integer.***MIN\_VALUE***;

**int** firstHighest = Integer.***MIN\_VALUE***;

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

**if**(arr[i]>firstHighest) {

secondHighest=firstHighest;

firstHighest=arr[i];

}**else** **if**(secondHighest<arr[i] && arr[i]!=firstHighest) {

secondHighest=arr[i];

}

}

System.***out***.println(secondHighest);

**return** secondHighest;

}

## intersection

**static** **void** intersection(**int** x[],**int** y[]) {

**int**[] z=**new** **int**[x.length+y.length];

**int** c=0;

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

**for**(**int** j=0;j<y.length;j++) {

**if**(x[i]==y[j]) {

z[c]=x[i];

c++;

}

}

}

System.***out***.println(" Common elements in Arrays : "+Arrays.*toString*(z));

}

## printPairsUsingTwoPointers

**public** **static** **void** printPairsUsingTwoPointers(**int**[] numbers, **int** k){

**if**(numbers.length < 2){

**return**;

}

Arrays.*sort*(numbers);

**int** left = 0; **int** right = numbers.length -1,sum=0;

**while**(left < right){

sum=numbers[left]+numbers[right];

**if**(sum==k) {

System.***out***.println( "Pairs are "+numbers[left]+" , "+numbers[right]);

right--;

left++;

}**else** **if**(sum>k)

right--;

**else** **if**(sum<k)

left++;

}

}

## firstRepeating

**public** **static** **void** firstRepeating(**int**[] arr) {

**int** minimum = -1;

HashSet set = **new** HashSet();

**for** (**int** i = arr.length-1 ;i >=0 ;i--) {

**if**(set.contains(arr[i]))

minimum = i;

**else**

set.add(arr[i]);

}

**if**(minimum != -1){

System.***out***.println("first repeating element is : "+ arr[minimum]);

}

}

## largestAndSmallest

**public** **static** **void** largestAndSmallest(**int**[] numbers) {

**int** largest=Integer.***MIN\_VALUE***;

**int** smallest=Integer.***MAX\_VALUE***;

**for**(**int** number:numbers) {

**if**(number>largest)

largest=number;

**if**(number<smallest)

smallest=number;

}

System.***out***.println("Given Array...."+Arrays.*toString*(numbers));

System.***out***.println("smallest : "+smallest);

System.***out***.println("largest : "+largest);

}

## getMissingNumber

**private** **static** **int** getMissingNumber(**int**[] numbers, **int** totalCount) {

**int** expectedSum = totalCount \* ((totalCount + 1) / 2);

**int** actualSum = 0;

**for** (**int** i : numbers) {

actualSum += i;

}

**return** expectedSum - actualSum;

}

## findFirstUniqueChar

**public** **static** **char** findFirstUniqueChar(String str) {

**char** input[] = str.toCharArray();

// char unique=str.charAt(0);

**boolean** found = **true**;

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

found = **true**;

**for** (**int** j = 0; j < input.length; j++) {

**if** (input[i] == input[j] && i != j) {

found = **false**;

**break**;

}

}

**if** (found == **true**)

**return** input[i];

}

**return** 0;

}

## toBinaryArray

//Binary format using Array

public static void toBinaryArray(int decimal) {

int[] binary = new int[25];

int index = 0;

while (decimal != 0) {

binary[index++] = decimal % 2;

decimal = decimal / 2;

}

for (int i = index - 1; i >= 0; i--) {

System.out.print(binary[i]);

}

}

## toBinary

public static void toBinary(int decimal) {

int binaryResult = 0;

while (decimal != 0) {

binaryResult = binaryResult \* 10 + decimal % 2;

decimal = decimal / 2;

}

## isAnagram

public static boolean isAnagram(String first, String second)

{

// remove all whitespaces and convert strings to lowercase

first = first.replaceAll("\\s", "").toLowerCase();

second = second.replaceAll("\\s", "").toLowerCase();

/\* check whether string lengths are equal or not,

if unequal then not anagram \*/

if (first.length() != second.length())

return false;

// convert string to char array

char[] firstArray = first.toCharArray();

char[] secondArray = second.toCharArray();

// sort both the arrays

Arrays.sort(firstArray);

Arrays.sort(secondArray);

// checking whether both strings are equal or not

return Arrays.equals(firstArray,secondArray);

}

Or

public static boolean isAnagram(String first, String second)

{

// remove all whitespaces and convert strings to lowercase

first = first.replaceAll("\\s", "").toLowerCase();

second = second.replaceAll("\\s", "").toLowerCase();

/\* check whether string lengths are equal or not,

if unequal then not anagram \*/

if (first.length() != second.length())

return false;

// convert first string to char array

char[] firstArray = first.toCharArray();

// check whether each character of firstArray is present in second string

for (char c : firstArray)

{

int index = second.indexOf(c);

// indexOf function returns -1 if the character is not found

if (index == -1)

return false;

// if character is present in second string, remove that character from second string

second = second.substring(0,index) + second.substring(index+1, second.length());

}

return true;

}

## find maximum

**public** **static** **void** max(**int**[] input) {

// **TODO** Auto-generated method stub

**int** max = input[0];

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

**if** (input[i] > max)

max = input[i];

}

System.***out***.println(max);

}

## How To Find Duplicates In Array Using Java 8 Streams?

**private** **static** **void** findDuplicatesUsingJava8(**int**[] inputArray) {

Set<Integer> uniqueElements = **new** HashSet<>();

Set<Integer> duplicateElements = Arrays.*stream*(inputArray).filter(i -> !uniqueElements.add(i)).boxed()

.collect(Collectors.*toSet*());

}

**private** **static** **void** findDuplicatesUsingHashMap(**int**[] inputArray) {

HashMap<Integer, Integer> map = **new** HashMap<>();

**for** (**int** element : inputArray) {

**if** (map.get(element) == **null**) {

map.put(element, 1);

} **else** {

map.put(element, map.get(element) + 1);

}

}

Set<Entry<Integer, Integer>> entrySet = map.entrySet();

**for** (Entry<Integer, Integer> entry : entrySet) {

**if** (entry.getValue() > 1) {

System.***out***

.println("Duplicate Element : " + entry.getKey() + " - found " + entry.getValue() + " times.");

}

}