**LeetCode programs**

26. Remove Duplicates from Sorted Array

class Solution {

public int removeDuplicates(int[] nums) {

Set<Integer> hset=new HashSet<Integer>();

hset.add(1001);

int k=1;

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

{

if(nums[i-1]!=nums[i] && !hset.contains(nums[i]))

{

nums[k]=nums[i];

if(k!=i)

{

hset.add(nums[i]);

nums[i]=1001;

}

k++;

}

else

{

hset.add(nums[i]);

nums[i]=1001;

}

}

return k;

}

}

28. Find the Index of the First Occurrence in a String

public int strStr(String haystack, String needle) {

for (int i=0;i<haystack.length()-needle.length()+1;i++)

{

if(haystack.charAt(i)==needle.charAt(0))

{

if(haystack.substring(i, i+needle.length()).equalsIgnoreCase(needle))

{

return i;

}

}

}

return -1;

}

2138.DivideaStringIntoGroupsofSizek

**package** LeetCode;

**import** java.util.ArrayList;

**public** **class** DivideaStringIntoGroupsofSizek\_2138 {

**public** **static** **void** main(String[] args) {

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

String s = "abcdefghij";

**int** k = 3;

**char** fill = 'x';

// String[] sArr=divideString(s, k, fill);

String[] sArr=*rearrangeString*(s, k, fill);

**for**(String ds:sArr)

{

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

}

}

**private** **static** String[] rearrangeString(String s, **int** k, **char** fill) {

StringBuilder builder=**new** StringBuilder();

ArrayList<String> list=**new** ArrayList<String>();

**for**(**char** ch:s.toCharArray())

{

builder.append(ch);

**if**(builder.length()==k)

{

list.add(builder.toString());

builder.setLength(0);

}

}

**if**(builder.length()>0)

{

**int** count=k-builder.length();

**for**(**int** i=0;i<count;i++)

{

builder.append(fill);

}

list.add(builder.toString());

}

**return** list.toArray(**new** String[list.size()]);

}

**private** **static** String[] divideString(String s, **int** k, **char** fill) {

StringBuilder builder=**new** StringBuilder();

String[] dString;

**int** r=0;

**if**(s.length()%k==0)

{

dString=**new** String[s.length()/k];

}

**else**

{

dString=**new** String[(s.length()/k)+1];

}

**for**(**int** i=0;i<s.length();i=i+k) //i=9

{

**if**(i+k<s.length())

{

**for**(**int** j=i;j<i+k;j++)

{

builder.append(s.charAt(j));

}

dString[r]=builder.toString();

r++;

builder.setLength(0);

}

**else**

{

**for**(**int** j=i;j<i+k;j++)

{

**if**(j<s.length())

builder.append(s.charAt(j));

**else**

builder.append(fill);

}

dString[r]=builder.toString();

}

}

**return** dString;

}

}

1592.RearrangeSpacesBetweenWords\_1592

**package** LeetCode;

**public** **class** RearrangeSpacesBetweenWords\_1592 {

**public** **static** **void** main(String[] args) {

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

// String text=" this is a sentence ";

// String text=" practice makes perfect";

String text="a";

System.***out***.println(*rearrangeString*(text));

}

**public** **static** String rearrangeString(String text)

{

String[] word=text.trim().split("\\s+");

StringBuilder builder=**new** StringBuilder();

**int** spaceCount=text.length()-text.replaceAll("\\s", "").length();

**if**(word.length>1)

{

**int** spaces=spaceCount/(word.length-1);

**int** extraSpace=spaceCount%(word.length-1);

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

{

**if**(i<word.length-1)

{

builder.append(word[i]);

builder.append(" ".repeat(spaces));

}

**else**

{

builder.append(word[i]);

**if**(extraSpace>0)

{

builder.append(" ".repeat(extraSpace));

}

}

}

**return** builder.toString() ;

}

**else**

{

builder.append(word[0]) ;

**return** builder.append(" ".repeat(spaceCount)).toString();

}

}

}

7. Reverse Integer

**package** LeetCode;

**public** **class** ReverseInteger\_7 {

**public** **static** **void** main(String[] args) {

**int** inputNo= 1534236469;

System.***out***.println(*reverseInteger*(inputNo));

}

**private** **static** **int** reverseInteger(**int** x) {

**int** n=0;

**int** intNo;

**if**(x<0)

{

x=0-x;

n++;

}

**long** reversex=0;

**if**(x%10==0)

{

x=x/10;

}

**for** (;x>0;x=x/10)

{

**int** r=x%10;

reversex=reversex\*10+r;

**if**(reversex> 2147483647 || reversex<-2147483647 )

{

**return** 0;

}

}

**if** (n==1)

{

reversex=0-reversex;

}

**return** intNo=(**int**)reversex;

}

}

815. Bus Routes

**if**(source==target) **return** 0;

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

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

{

**for**(**int** j=0;j<routes[i].length;j++)

{

List<Integer> buses=g.containsKey(routes[i][j]) ? g.get(routes[i][j]):**new** ArrayList<>();

buses.add(i);

g.put(routes[i][j], buses);

}

}

Queue<Integer> q=**new** LinkedList<>();

**boolean**[] ride=**new** **boolean**[routes.length];

q.offer(source);

**int** step=0;

**while**(!q.isEmpty())

{

++step;

**int** size=q.size();

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

{

**int** cur=q.poll();

**for**(Integer bus:g.get(cur))

{

**if**(ride[bus]) **continue**;

ride[bus]=**true**;

**for**(**int** stop:routes[bus])

{

**if**(stop==target) **return** step;

q.offer(stop);

}

}

}

}

**return** -1;

15. 3Sum

**→Brute force approach (time limit exceeds)**

**public** **static** List<List<Integer>> threeSum1(**int**[] nums)

{

**int** n=0;

List<List<Integer>> a=**null**;

Set<List<Integer>> set=**new** HashSet<>();

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

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

{

**int** num=m.containsKey(nums[i]) ? m.get(nums[i])+1 : 1 ;

m.put(nums[i], num);

}

**for**(**int** i=0;i<nums.length-2;i++)

{

**for**(**int** j=i+1;j<nums.length-1;j++)

{

**for**(**int** k=j+1;k<nums.length;k++)

{

**if**(nums[i]+nums[j]+nums[k]==0)

{

n++;

List<Integer> list=**new** ArrayList<Integer>(Arrays.*asList*(**new** Integer[]{nums[i],nums[j],nums[k]}));

Collections.*sort*(list);

set.add(list);

}

}

}

}

**if**(n>0)

**return** a = **new** ArrayList(set);

**else**

**return** a = **new** ArrayList();

}

**→Two pointers approach**

**public** **static** List<List<Integer>> threeSum1(**int**[] nums)

{

Arrays.*sort*(nums);

List<List<Integer>> list=**null**;

Set<List<Integer>> arrSet=**new** HashSet<>();

**int** lo=0;

**int** hi=0;

**for**(**int** i=0;i<nums.length-2;i++)

{

**if**(i==0 || nums[i]!=nums[i-1])

{

lo=i+1;

hi=nums.length-1;

**while**(lo<hi)

{

**if**(nums[lo]+nums[hi]==(-nums[i]))

{

List<Integer> arrs=**new** ArrayList<Integer>(Arrays.*asList*(nums[i],nums[lo],nums[hi]));

Collections.*sort*(arrs);

arrSet.add(arrs);

lo++;

}

**else** **if**(nums[lo]+nums[hi]<(-nums[i]))

{

**while**(nums[lo+1]==nums[lo] && (lo+1)<hi) lo++;

lo++;

}

**else**

{

**while**(nums[hi-1]==nums[hi] && (hi-1)>lo) hi--;

hi--;

}

}

}

**else** **continue**;

}

**return** list=**new** ArrayList<>(arrSet);

}

16. 3Sum Closest

**public** **static** **int** closestThreeSum(**int**[] nums , **int** target)

{

Arrays.*sort*(nums);

**int** lo=0;

**int** hi=0;

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

**for**(**int** i=0;i<nums.length-2;i++)

{

**if**(i==0 || nums[i]!=nums[i-1])

{

lo=i+1;

hi=nums.length-1;

**while**(lo<hi)

{

**int** numlo=nums[lo];

**int** numhi=nums[hi];

**int** sum=nums[lo]+nums[hi]+nums[i];

**int** sub=Math.*subtractExact*(nums[lo]+nums[hi]+nums[i],target);

**int** abs=Math.*abs*(sub);

**if**(Math.*abs*(sub)<Math.*abs*(num)) num=sub;

**if**(sum==target) **return** num+target;

**if**(sum<target)

{

**while**(nums[lo+1]==nums[lo] && (hi-1)>lo) lo++;

lo++;

}

**else**

{

**while**(nums[hi-1]==nums[hi] && (hi-1)>lo) hi--;

hi--;

}

}

}

**else** **continue**;

}

**return** (num+target);

}

18. 4Sum

**public** **static** List<List<Integer>> fourSum(**int**[] nums, **int** target)

{

Arrays.*sort*(nums);

List<List<Integer>> list=**null**;

Set<List<Integer>> arrSet=**new** HashSet<>();

**int** lo=0;

**int** hi=0;

**for**(**int** i=0;i<nums.length-3;i++)

{

**if**(i==0 || nums[i]!=nums[i-1])

{

**for**(**int** j=i+1;j<nums.length-2;j++)

{

lo=j+1;

hi=nums.length-1;

**while**(lo<hi)

{

**long** sum=(nums[i]+nums[j]);

sum=sum+nums[lo]+nums[hi];

**if**(sum==target)

{

List<Integer> arrs=**new** ArrayList<Integer>(Arrays.*asList*(nums[i],nums[j],nums[lo],nums[hi]));

Collections.*sort*(arrs);

arrSet.add(arrs);

lo++;

}

**else** **if**(nums[i]+nums[j]+nums[lo]+nums[hi]<target)

{

**while**(nums[lo+1]==nums[lo] && (lo+1)<hi) lo++;

lo++;

}

**else**

{

**while**(nums[hi-1]==nums[hi] && (hi-1)>lo) hi--;

hi--;

}

}

}

}

**else** **continue**;

}

**return** list=**new** ArrayList<>(arrSet);

}

Write a program that takes input of a string and return the character which occurs the most in string ex - mississippi i=4,s=4

**public** **static** **void** main(String[] args) {

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

String s="mississippi";

**int** count=1;

**int** maxCout=0;

**char** c=' ';

String temp="";

**for**(**int** i=0;i<s.length()-1;i++)

{

**if** ( temp.indexOf(s.charAt(i))==-1) temp=temp+s.charAt(i);

**else** **continue**;

**for**(**int** j=i+1;j<s.length();j++)

{

**if**(s.charAt(i)==s.charAt(j))

{

count++;

}

}

**if**(count>=maxCout)

{

c=s.charAt(i);

maxCout=count;

hm.put(c, count);

}

count=1;

}

**for**(Map.Entry<Character, Integer> e:hm.entrySet())

{

**if**(e.getValue()==maxCout)

{

System.***out***.println(e.getKey()+"="+e.getValue());

}

}

}

27. Remove Element

public int removeElement(int[] nums, int val) {

int j=nums.length-1;

int i=0;

if(nums.length==1 && nums[0]==val)

{

nums[0]=1000;

return 0;

}

while(i<j)

{

if(nums[i]==val)

{

while(nums[j]==val)

{

nums[j]=1000;

if((j-1)>i) j--;

else break;

}

nums[i]=nums[j];

nums[j]=1000;

j--;

}

i++;

if(i==j & nums[i]==val) nums[i]=1000;

}

i=0;

for(int k=0;k<nums.length;k++)

{

if(nums[k]==1000)

{

break;

}

i++;

}

return i;

}

60. Permutation Sequence

public String getPermutation(int n, int k) {

k=k-1;

StringBuilder builder=new StringBuilder();

int b=0;

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

for(int i=1;i<=n;i++)

{

list.add(i);

}

while(!list.isEmpty())

{

b=fact(n-1);

int index=k/b;

builder.append(list.remove(index));

k=k%b;

n--;

}

return builder.toString();

}

public static int fact(int n)

{

int fact=1;

while(n>0)

{

fact=fact\*n;

n--;

}

return fact;

}

31. Next Permutation

**private** **static** **int**[] nextPermutation(**int**[] nums) {

for(int i=nums.length-1;i>0;i--)

{

if(nums[i-1]<nums[i])

{

int j=nums.length-1;

while(nums[i-1]>nums[j])

{

j--;

}

swap(i-1,j,nums);

reverse(i,nums.length-1,nums);

}

}

}

private static int[] reverse(int i, int j, int[] nums) {

while(i<j)

{

swap(i,j,nums);

i++; j--;

}

return nums;

}

private static int[] swap(int i, int j, int[] nums) {

int d=0;

d=nums[i];

nums[i]=nums[j];

nums[j]=d;

return nums;

}

152. Maximum Product Subarray

public int maxProduct(int[] nums) {

int csum=1;

int maxsum=nums[0];

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

{

csum=csum\*nums[i];

if(csum>maxsum) maxsum=csum;

if(csum==0) csum=1;

}

csum=1;

for(int i=nums.length-1;i>=0;i--)

{

csum=csum\*nums[i];

if(csum>maxsum) maxsum=csum;

if(csum==0) csum=1;

}

return maxsum;

}

**OR**

**private static int maxProducts(int[] nums) {**

**int maP=nums[0];**

**int miP=nums[0];**

**int ans=nums[0];**

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

**{**

**int tmp=0;**

**tmp=*maxInt*(miP\*nums[i],maP\*nums[i], nums[i]);**

**miP=*minInt*(miP\*nums[i],maP\*nums[i], nums[i]);**

**maP=tmp;**

**if (ans<maP) ans=Math.*max*(maP, miP);**

**}**

**return ans;**

**}**

**private static int minInt(int i, int j, int k) {**

**if(i<j)**

**{**

**if(i<k) return i;**

**}**

**else if(j<k)**

**{**

**return j;**

**}**

**return k;**

**}**

**private static int maxInt(int i, int j, int k) {**

**if(i>j)**

**{**

**if(i>k) return i;**

**}**

**else if(j>k)**

**{**

**return j;**

**}**

**return k;**

**}**

53. Maximum Subarray

**private** **static** **int** maxSubArray(**int**[] nums) {

**int** csum=0;

**int** maxsum=nums[0];

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

{

csum=csum+nums[i];

**if**(csum>maxsum) maxsum=csum;

**if**(csum<0) csum=0;

}

**return** maxsum;

}