1)**Linked Lists  
Add Two Numbers**  
You are given two non-empty linked lists representing two non-negative integers. Add the two numbers and return the sum as a linked list.

class Node{

int data;

Node next;

Node(int data)

{

this.data=data;

this.next=null;

}

}

class LinkedList{

Node head;

public void addlast(int data)

{

Node newnode=new Node(data);

if(head==null)

{

head=newnode;

return ;

}

Node current=head;

while(current.next!=null){

current=current.next;

}

current.next=newnode;

}

public void printlist()

{

Node current=head;

while(current!=null)

{

System.out.print(current.data);

current=current.next;

}

}

public static Node reverselist(Node head)

{

Node current=head;

Node prev=null;

Node temp;

while(current!=null)

{

temp=current.next;

current.next=prev;

prev=current;

current=temp;

}

return prev;

}

public static LinkedList addlinklist(Node l1,Node l2)

{

l1=reverselist(l1);

l2=reverselist(l2);

int carry=0;

int sum=0;

int val1=0;

int val2=0;

LinkedList res=new LinkedList();

while(l1!=null||l2!=null||carry!=0)

{

if(l1!=null)

{

val1=l1.data;

}

else

{

val1=0;

}

if(l2!=null)

{

val2=l2.data;

}

else

{

val2=0;

}

sum=sum+val1+val2+carry;

carry=sum/10;

res.addlast(sum%10);

if(l1!=null)

{

l1=l1.next;

}

if(l2!=null)

{

l2=l2.next;

}

sum=0;

}

res.head=reverselist(res.head);

return res;

}

}

class Main {

public static void main(String[] args) {

LinkedList l1=new LinkedList();

l1.addlast(9);

l1.addlast(9);

l1.addlast(9);

l1.printlist();

System.out.println();

LinkedList l2=new LinkedList();

l2.addlast(9);

l2.addlast(9);

l2.addlast(9);

l2.printlist();

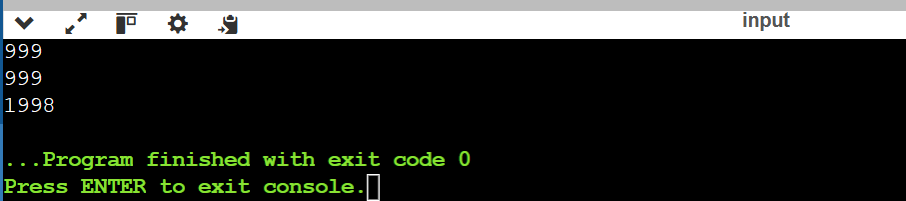
System.out.println();

LinkedList l3=LinkedList.addlinklist(l1.head,l2.head);

l3.printlist();

}

}



**2) Next Greater Element**  
Given a circular array, find the next greater number for every element.  
231 -> 312

public class Main

{

public static void main(String[] args) {

int[] arr={2,3,1};

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

{

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

}

int i=arr.length-2;

while(i>0&&arr[i]>=arr[i+1])

{

i--;

}

int j=arr.length-1;

if(i>=0)

{

while(j>0&&arr[j]<arr[i])

{

j--;

}

int temp=arr[i];

arr[i]=arr[j];

arr[j]=temp;

}

int l=i+1;

int r=arr.length-1;

while(l<r)

{

int temp=arr[l];

arr[l]=arr[r];

arr[r]=temp;

l++;

r--;

}

System.out.println();

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

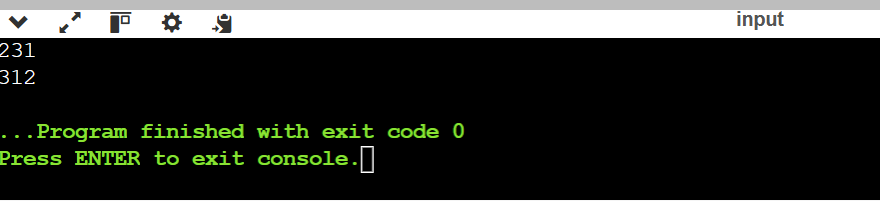
{

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

}

}

}



**5. Reorder List**Reorder a linked list from L0 → L1 → … → Ln-1 → Ln to L0 → Ln → L1 → Ln-1 → L2 → Ln-2 → ….

class Node{

int data;

Node next;

Node(int data)

{

this.data=data;

this.next=null;

}

}

class LinkedList{

Node head;

public void addlast(int data)

{

Node newnode=new Node(data);

if(head==null)

{

head=newnode;

return ;

}

Node current=head;

while(current.next!=null)

{

current=current.next;

}

current.next=newnode;

}

public void printlist()

{

Node current=head;

while(current!=null)

{

System.out.print(current.data);

current=current.next;

}

}

public static void reorderlist(Node head)

{

Node slow=head;

Node fast=head;

while(fast!=null&&fast.next!=null)

{

slow=slow.next;

fast=fast.next.next;

}

Node prev=null;

Node current=slow.next;

while(current!=null){

Node temp=current.next;

current.next=prev;

prev=current;

current=temp;

}

slow.next=null;

Node first=head;

Node second=prev;

while(second!=null)

{

Node temp1=first.next;

Node temp2=second.next;

first.next=second;

second.next=temp1;

first=temp1;

second=temp2;

}

}

}

public class Main

{

public static void main(String[] args) {

LinkedList l1=new LinkedList();

l1.addlast(3);

l1.addlast(5);

l1.addlast(8);

l1.addlast(7);

l1.addlast(1);

l1.printlist();

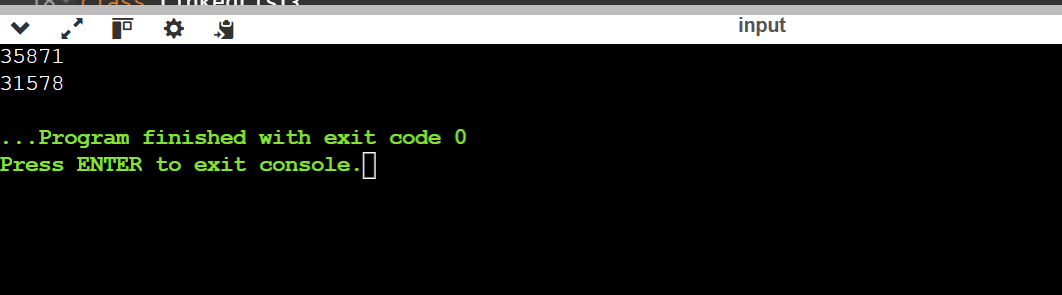
l1.reorderlist(l1.head);

System.out.println();

l1.printlist();

}

}



**6.Remove Nth Node From End of List**Given a linked list, remove the nth node from the end and return its head.class Node{

int data;

Node next;

Node(int data)

{

this.data=data;

this.next=null;

}

}

class LinkedList{

Node head;

public void addlast(int data)

{

Node newnode=new Node(data);

if(head==null)

{

head=newnode;

return ;

}

Node current=head;

while(current.next!=null)

{

current=current.next;

}

current.next=newnode;

}

public void printlist()

{

Node current=head;

while(current!=null)

{

System.out.print(current.data);

current=current.next;

}

}

public static void reorderlist(Node head,int n)

{

int count=0;

Node current=head;

while(current.next!=null)

{

count++;

current=current.next;

}

current=head;

int v=0;

while(current!=null)

{

if(count-n==v)

{

current.next=current.next.next;

}

v++;

current=current.next;

}

}

}

public class Main

{

public static void main(String[] args) {

LinkedList l1=new LinkedList();

l1.addlast(3);

l1.addlast(5);

l1.addlast(8);

l1.addlast(7);

l1.addlast(1);

l1.printlist();

int n=3;

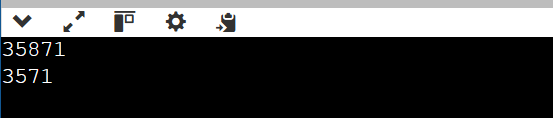
l1.reorderlist(l1.head,n);

System.out.println();

l1.printlist();

}

}



Strings  
**7.Longest Substring Without Repeating Characters**  
Given a string, find the length of the longest substring without repeating characters.

import java.util.HashMap;

public class Main

{

public static void main(String[] args) {

String s="abcdabcde";

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

int right=0;

int max=0;

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

{

char ch=s.charAt(left);

if(map.containsKey(ch))

{

right=map.get(ch)+1;

}

map.put(ch,left);

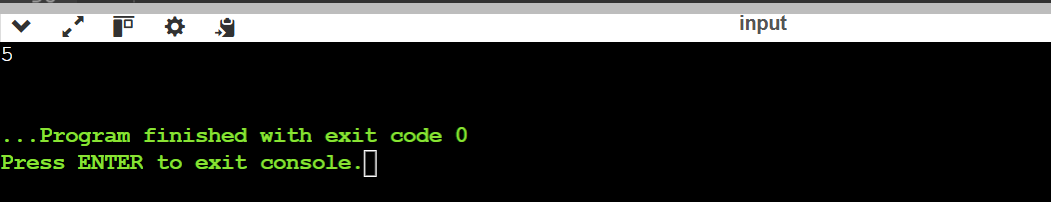
max=Math.max(max,left-right+1);

}

System.out.println(max);

}

}



**ASSIGNED QUESTIONS**

1. **Given an array of strings, group anagrams together.**

import java.util.Arrays;

import java.util.\*;

public class Main

{

public static List<List<String>> group(String[] s1)

{

HashMap<String,List<String>> map=new HashMap<>();

for(int i=0;i<s1.length;i++)

{

char[] ch=s1[i].toCharArray();

Arrays.sort(ch);

String sorted=String.valueOf(ch);

if(!map.containsKey(sorted))

{

map.put(sorted,new ArrayList<>());

}

map.get(sorted).add(s1[i]);

}

return new ArrayList<>(map.values());

}

public static void main(String[] args) {

String[] s1= {"eat", "tea", "tan", "ate", "nat", "bat"};

List<List<String>> res=group(s1);

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

{

System.out.println("["+res.get(i)+"]");

}

}

}



**2)Given a string ss, return the longest palindromic substring in ss.**

public class Main

{

public static boolean palin(String s1)

{

StringBuilder s2=new StringBuilder(s1);

String s3=s2.reverse().toString();

return s1.equals(s3);

}

public static String check(String s1)

{

if(palin(s1))

{

return s1;

}

int left=0;

int right=s1.length()-1;

while(left<right)

{

String str=s1.substring(left,right+1);

if(palin(str))

{

return str;

}

left++;

right--;

}

return "";

}

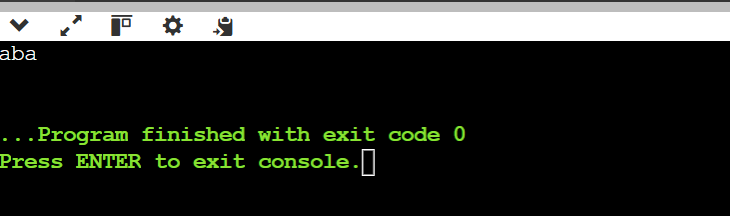
public static void main(String[] args) {

String s1="babad";

System.out.println(check(s1));

}

}



**Problem 3: Given a string containing just the characters '(', ')', '{', '}', '[' and ']', determine if the input string is valid**.

A string is valid if: 1. Open brackets must be closed by the same type of brackets. 2. Open brackets must be closed in the correct order. 3. Every close bracket has a corresponding open bracket of the same type

import java.util.Stack;

public class Main

{

public static boolean valid(String s)

{

Stack<Character> st=new Stack<>();

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

{

char ch=s.charAt(i);

if(ch=='{')

{

st.push('}');

}

else if(ch=='(')

{

st.push(')');

}

else if(ch=='[')

{

st.push(']');

}

else {

if(st.isEmpty()||st.pop()!=ch)

{

return false;

}

}

}

return st.isEmpty();

}

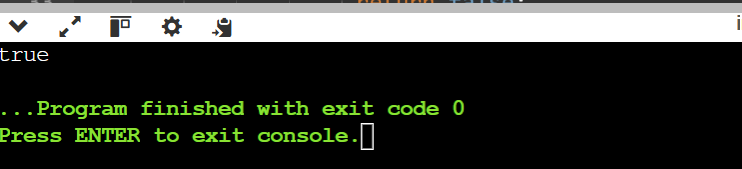
public static void main(String[] args) {

String str="{}()[]";

System.out.print(valid(str));

}

}



import java.util.Arrays;

import java.util.HashMap;

import java.util.Collections;

import java.util.ArrayList;

public class Main

{

public static String sorted(String s)

{

char[] ch=s.toCharArray();

Arrays.sort(ch);

return new String(ch);

}

public static boolean check(String s1,String s2)

{

if(s1.length()!=s2.length())

{

return false;

}

s1=sorted(s1);

s2=sorted(s2);

if(s1.equals(s2))

{

return true;

}

return true;

}

public static void main(String[] args) {

String s1="xxyyzz";

String s2="aabbcc";

System.out.println(check(s1,s2));

}

}

**Given two strings ss and tt, determine if tt is an anagram of a substring of ss. In other words, check if there exists a substring in ss that is an anagram of t**

public class Main

{

public static boolean valid(int[] f1,int[] f2)

{

for(int i=0;i<26;i++)

{if(f1[i]!=f2[i]){return false;}}

return true;

}

public static boolean check(String s1,String s2)

{

int[] f1=new int[26];

int[] f2=new int[26];

for(int i=0;i<s2.length();i++)

{f1[s1.charAt(i)-'a']++;

f2[s2.charAt(i)-'a']++;

}

if(valid(f1,f2)){return true;}

for(int i=s2.length();i<s1.length();i++)

{

f1[s1.charAt(i)-'a']++;

f1[s1.charAt(i-s2.length())-'a']--;

if(valid(f1,f2))

{

return true;

}}

return false;

}

public static void main(String[] args) {

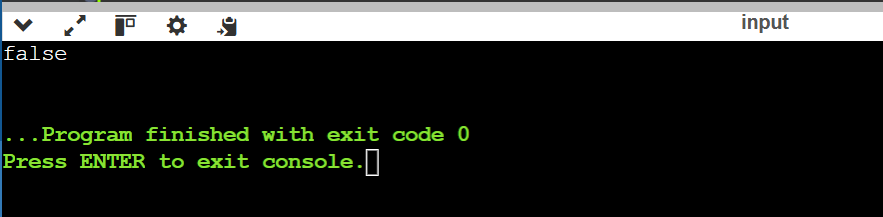
String s1="af";

String s2="be";

System.out.println(check(s1,s2));

}

}



**Write a function to find the longest common prefix string amongst an array of strings. If there is no common prefix, return an empty string "".**

public class Main

{

public static String check(String[] s1)

{

int c=0;

String str1=s1[0];

String str2=s1[s1.length-1];

for(int i=0;i<s1.length;i++)

{

if(str1.charAt(c)==str2.charAt(c))

{

c++;

}

}

return str1.substring(0,c);

}

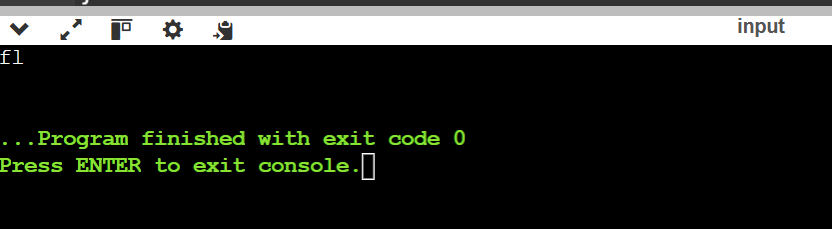
public static void main(String[] args) {

String[] s1={"flower","flow","flight"};

System.out.println(check(s1));

}

}



**Given two strings needle and haystack, return the index of the first occurrence of needle in haystack, or -1 if needle is not part of haystack**

public class Main

{

public static int check(String h,String n)

{

int j=n.length();

for(int i=0;i<h.length();i++,j++)

{

if(h.substring(i,j).equals(n))

{

return i;

}

}

return -1;

}

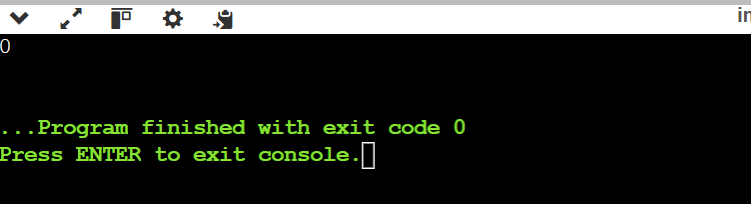
public static void main(String[] args) {

String haystack = "sadbutsad", needle = "sad";

System.out.println(check(haystack,needle));

}

}



**Given a string s consisting of words and spaces, return the length of the last word in the string.**

A word is a maximal

substring

consisting of non-space characters only

public class Main

{

public static int check(String str)

{

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

String out= s[s.length-1];

return out.length();

}

public static void main(String[] args) {

String s= "luffy is still joyboy";

System.out.println(check(s));

}

}

**Given a string paragraph and a string array of the banned words banned, return the most frequent word that is not banned.** It is guaranteed there is at least one word that is not banned, and that the answer is unique. The words in paragraph are case-insensitive and the answer should be returned in lowercase

import java.util.HashMap;

public class Main

{

public static String check(String p,String[] b)

{

Set<String> bb=new HashSet<>(Arrays.asList(b));

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

p=p.toLowerCase().replaceAll("[^a-z]"," ");

String[] para=p.split("\\s+");

int max=0;

String res="";

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

{

if(!bb.contains(para[i]))

{

int c=map.getOrDefault(para[i],0)+1;

map.put(para[i],c);

if(c>=max)

{

max=c;

res=para[i];

}

}

}

return res;

}

public static void main(String[] args) {

String paragraph = "Bob hit a ball, the hit BALL flew far after it was hit.";

String[] banned = {"hit"};

System.out.println(check(paragraph,banned));

}

}

