import java.util.Stack;

public class Main {

    public static void main(String args[]){

        String infix = "(a+b)/(c-d)";

        InfixToPostFix inf = new InfixToPostFix();

        System.out.println(inf.InfixToPostFix(infix));

    }

}

class InfixToPostFix {

    boolean isOperator(char c){

        return c=='+' || c=='-' || c=='/' || c=='\*';

    }

    int getPrecedence(char c){

        switch(c) {

            case '+':

            case '-':

                return 1;

            case '\*':

            case '/':

                return 2;

        }

        return -1;

    }

    String InfixToPostFix(String infix){

        String postfix="";

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

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

            char c = infix.charAt(i);

            if(Character.isLetterOrDigit(c)){

                postfix+=c;

            }else if(c=='('){

                stack.push(c);

            }

            else if(c==')'){

                while(!stack.isEmpty() && stack.peek()!='('){

                    postfix += stack.pop();

                }

                if(!stack.isEmpty() && stack.peek()!='('){

                    return "Invalid Expression";

                }

                stack.pop();

            }

            else if(isOperator(c)){

                while(!stack.isEmpty() && getPrecedence(c)<=getPrecedence(stack.peek())){

                    postfix += stack.pop();

                }

                stack.push(c);

            }

        }

        while(!stack.isEmpty()){

            postfix += stack.pop();

        }

    return postfix;

    }

}

result = k + 1;

List<Integer> elements = new ArrayList<>(Arrays.asList(Arrays.stream(A).boxed().toArray(Integer[]::new)));

while (elements.contains(result)) {

result++;

}

**BACKUP..**

class Node {

int data;

Node left, right;

Node(int data){

this.data = data;

this.left = null;

this.right = null;

}

}

public class Main {

static void leftBoundary(Node root){

if(root==null || root.left==null && root.right==null){

return;

}

System.out.println(root.data);

if(root.left!=null){

leftBoundary(root.left);

}

return;

}

static void leafNodes(Node root){

if(root==null){

return;

}

if(root.left==null && root.right==null){

System.out.println(root.data);

return;

}

leafNodes(root.left);

leafNodes(root.right);

}

static void rightBoundary(Node root){

if(root.right==null){

return;

}

System.out.println(root.data);

if(root.right!=null){

rightBoundary(root.right);

}

}

public static void main(String args[]){

Node root = new Node(10);

root.left = new Node(5);

root.left.left = new Node(3);

root.left.right = new Node(7);

root.right = new Node(15);

root.right.left = new Node(12);

root.right.right = new Node(17);

System.out.println(root.data);

leftBoundary(root.left);

leafNodes(root.left);

leafNodes(root.right);

rightBoundary(root.right);

}

}

//bellman ford

//start coding a new program

public class Main {

static int bellmanFord(int[][] graph, int source, int destination){

int vertices = graph.length;

int[] distances = new int[vertices];

for(int i=0;i<vertices;i++){

distances[i] = Integer.MAX\_VALUE;

}

distances[source] = 0;

//bellman ford code

for(int i=0;i<vertices-1;i++){

for(int u=0;u<vertices;u++){

for(int v=0;v<vertices;v++){

if(graph[u][v]!=0 && distances[u]!=Integer.MAX\_VALUE && distances[u]+graph[u][v] < distances[v]){

distances[v] = distances[u] + graph[u][v];

}

}

}

}

return distances[destination];

}

public static void main(String args[]){

int[][] graph = {

{0,-2,0,1},

{0,0,4,0},

{0,0,0,0},

{0,-1,2,0}

};

int source=0;

int destination = 2;

int shortestDistance = bellmanFord(graph,source,destination);

System.out.println(shortestDistance);

}

}

//dijkstra

public class Main {

public static int dijkstra(int[][] graph, int source, int destination){

int minDistance;

int vertices = graph.length;

int[] distances = new int[vertices];

boolean[] visited = new boolean[vertices];

for(int i=0;i<vertices;i++){

distances[i] = Integer.MAX\_VALUE;

}

distances[source]=0;

for(int i=0;i<vertices;i++){

int minVertex = findMinDistance(distances,visited);

visited[minVertex] = true;

for(int j=0;j<vertices;j++){

if(!visited[j] && graph[minVertex][j] !=0 && distances[minVertex]!=Integer.MAX\_VALUE

&& distances[minVertex] + graph[minVertex][j]<distances[j]){

distances[j] = distances[minVertex] + graph[minVertex][j];

}

}

}

minDistance = distances[destination];

return minDistance;

}

static int findMinDistance(int[] distances,boolean visited[]){

int minDistance = Integer.MAX\_VALUE;

int minVertex = -1;

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

if(!visited[i] && distances[i]<minDistance){

minDistance = distances[i];

minVertex = i;

}

}

return minVertex;

}

public static void main(String args[]){

int[][] graph = {

{0,1,3,0},

{1,0,2,4},

{3,2,0,6},

{0,4,6,0},

};

int source=0;

int destination = 3;

int shortestDistance = dijkstra(graph,source,destination);

System.out.println(shortestDistance);

}

}

//coin combinations

public class Main {

static void findCoinCombinations(int[] coins,int k, int n){

int[] currentCombination = new int[k];

backtrack(coins,k,n,currentCombination,0,0);

}

static void backtrack(int[] coins, int k, int n, int[] currentCombination, int currentIndex,int currentSum)

{

if(currentSum==n && currentIndex == k){

printCombination(currentCombination);

return;

}

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

if((currentIndex<k) && ((currentSum + coins[i])<=n)){

currentCombination[currentIndex] = coins[i];

backtrack(coins,k,n,currentCombination,currentIndex+1,currentSum+coins[i]);

currentCombination[currentIndex] = 0;

}

}

}

static void printCombination(int[] combination) {

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

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

}

System.out.print("\n");

}

public static void main(String args[]){

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

int k = 3;

int N = 7;

findCoinCombinations(coins,k,N);

}

}

// Printing Tree like a tre

import java.util.LinkedList;

import java.util.Queue;

public class Main

{

static class Node

{

int data;

Node left;

Node right;

Node(int data){

this.data = data;

left = null;

right =null;

}

}

static int depthOfTree(Node root, int d) {

if(root == null) {

return d;

}

int left = d;

int right = d;

if(root.left != null) {

left = depthOfTree(root.left, d+1);

}

if(root.right != null) {

right = depthOfTree(root.right, d+1);

}

return Math.max(left, right);

}

static void printLevelOrder(Node root, int depth)

{

if(root == null)

return;

Queue<Node> q =new LinkedList<Node>();

q.add(root);

while(true)

{

int nodeCount = q.size();

if(nodeCount == 0)

break;

for(int i=0; i<depth; i++) {

System.out.print(" ");

}

while(nodeCount > 0)

{

Node node = q.peek();

System.out.print("("+node.data + ")");

q.remove();

if(node.left != null)

q.add(node.left);

if(node.right != null)

q.add(node.right);

if(nodeCount>1){

System.out.print(", ");

}

nodeCount--;

}

depth--;

System.out.println();

}

}

public static void main(String[] args)

{

Node root = new Node(6);

root.left = new Node(3);

root.right = new Node(2);

root.left.left = new Node(8);

root.left.right = new Node(9);

root.right.right = new Node(-2);

root.right.left = new Node(3);

int d = depthOfTree(root, 1);

//System.out.println(d);

printLevelOrder(root, d);

}

}

//bubble sort on a linked list

// Online Java Compiler

// Use this editor to write, compile and run your Java code online

class Node {

int data;

Node next;

Node(int n){

data = n;

}

}

class LinkedList {

Node head;

LinkedList(int n){

head = new Node(n);

}

void insert(int n){

Node iterator = head;

while(iterator.next!=null){

iterator = iterator.next;

}

iterator.next = new Node(n);

}

void bubbleSort() {

Node iterator = head;

int temp;

while (iterator != null) {

Node innerIterator = head;

Node prev = null;

boolean swapped = false;

while (innerIterator.next != null) {

prev = innerIterator;

innerIterator = innerIterator.next;

if (prev.data > innerIterator.data) {

temp = prev.data;

prev.data = innerIterator.data;

innerIterator.data = temp;

swapped = true;

}

}

if (!swapped) {

break;

}

iterator = iterator.next;

}

}

void print(){

Node iterator = head;

while(iterator!=null){

System.out.print(iterator.data+"->");

iterator = iterator.next;

}

}

}

public class HelloWorld {

public static void main(String[] args) {

LinkedList ll = new LinkedList(10);

ll.insert(9);

ll.insert(8);

ll.insert(7);

ll.insert(6);

ll.bubbleSort();

ll.print();

}

}

//minimum time to finish a project

class Node {

int duration;

Node left, right;

public Node(int n){

duration = n;

}

}

public class Main {

static int bestTimeRecursive(Node root){

if(root==null){

return 0;

}

int leftTime = bestTimeRecursive(root.left);

int rightTime = bestTimeRecursive(root.right);

return root.duration + Math.min(leftTime,rightTime);

}

public static void main(String args[]){

Node root = new Node(10);

root.left = new Node(8);

root.right = new Node(3);

root.left.left = new Node(1);

root.left.right = new Node(6);

root.right.left = new Node(2);

System.out.println(bestTimeRecursive(root));

}

}

//min distance b/w 2 nodes in a graph

import java.util.Queue;

import java.util.LinkedList;

//start coding a new program

public class Main {

static int shortestDistance(int[][] adjacencyMatrix, int startNode, int endNode){

int numNodes = adjacencyMatrix.length;

boolean[] visited = new boolean[numNodes];

int[] distances = new int[numNodes];

Queue<Integer> queue = new LinkedList<>();

queue.offer(startNode);

visited[startNode] = true;

distances[startNode] = 0;

while(!queue.isEmpty()){

int currNode = queue.poll();

if(currNode == endNode){

return distances[currNode];

}

for(int neighbour=0;neighbour<numNodes;neighbour++){

if(adjacencyMatrix[currNode][neighbour]==1 && !visited[neighbour]){

queue.offer(neighbour);

visited[neighbour]=true;

distances[neighbour] = distances[currNode]+1;

}

}

}

return -1;

}

public static void main(String args[]){

int[][] adjacencyMatrix = {

{0, 1, 1, 0, 0},

{1, 0, 1, 1, 0},

{1, 1, 0, 0, 1},

{0, 1, 0, 0, 1},

{0, 0, 1, 1, 0}

};

int startNode = 0, endNode = 3;

int dist = shortestDistance(adjacencyMatrix,startNode,endNode);

System.out.println(dist);

}

}

//min distance between the target word in an array

//start coding a new program

public class Main {

public static void main(String args[]){

String[] arr = {"practice", "makes", "perfect", "coding", "makes", "makes", "practice", "makes", "perfect", "coding", "makes"} ;

String target = "makes";

int minDistance=99999;

int pointer = -1, pointer2=-1;

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

if(arr[i].equals(target)){

if(pointer==-1){

pointer = i;

}

else {

pointer2 = i;

if(Math.abs(pointer2-pointer)<minDistance){

minDistance = Math.abs(pointer2-pointer);

}

pointer = pointer2;

}

}

}

System.out.println(minDistance);

}

}

//Rectangle with the largest area

public class Main {

static int calculateArea(int[][] rectangle) {

int area;

int x1 = rectangle[0][0];

int y1 = rectangle[0][1];

int x2 = rectangle[2][0];

int y2 = rectangle[2][1];

area = Math.abs((x2-x1)\*(y2-y1));

return area;

}

public static void main(String args[]){

int[][][] arr = {

{{1,1},{1,5},{5,5},{5,1}}, //rect 1

{{2,3},{3,7},{6,7},{6,3}}, //rect 2

{{3,2},{3,6},{8,6},{8,2}} //rect 3

};

int area = 0,maxArea=0,maxIndex=0;

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

area = calculateArea(arr[i]);

if(area>maxArea)

{

maxArea = area;

maxIndex = i;

}

}

System.out.println("The rectangle with the largest area of "+maxArea+" is "+maxIndex);

}

}

//Imposter Syndrome

class StudentNode {

String name;

int score;

StudentNode nextStudent;

StudentNode(String name, int score){

this.name=name;

this.score = score;

}

}

class StudentList {

StudentNode head;

StudentList(String name, int score){

head = new StudentNode(name,score);

}

void insertStudent(String name, int score){

StudentNode iterator = head;

while(iterator.nextStudent!=null){

iterator = iterator.nextStudent;

}

iterator.nextStudent = new StudentNode(name,score);

}

void findImposters(){

StudentNode iterator = head;

StudentNode prevStudent = null;

if(iterator.score<iterator.nextStudent.score){

System.out.println(iterator.name);

}

prevStudent = iterator;

iterator = iterator.nextStudent;

while(iterator.nextStudent!=null){

if(iterator.score<iterator.nextStudent.score && iterator.score<prevStudent.score){

System.out.println(iterator.name);

}

prevStudent = iterator;

iterator = iterator.nextStudent;

}

if(prevStudent.score > iterator.score){

System.out.println(iterator.name);

}

}

}

public class Main {

public static void main(String args[]) {

StudentList list = new StudentList("A",80);

list.insertStudent("B",85);

list.insertStudent("C",70);

list.insertStudent("D",90);

list.insertStudent("E",87);

list.insertStudent("F",95);

list.insertStudent("G",94);

list.findImposters();

}

}

//checking if a graph is cyclic

public class Main {

static boolean isCyclic(int[][] graph){

int n = graph.length;

boolean[] visited = new boolean[n];

boolean[] dfsStack = new boolean[n];

for(int i=0; i<n;i++){

if(!visited[i] && hasCycle(graph,visited,dfsStack,i)){

return true;

}

}

return false;

}

static boolean hasCycle(int[][] graph,boolean[] visited,boolean[] dfsStack, int node){

visited[node] = true;

dfsStack[node] = true;

for(int i=0;i<graph[node].length;i++){

if(graph[node][i]==1){

if(!visited[i]){

if(hasCycle(graph,visited,dfsStack,i)){

return true;

}

}

else if(dfsStack[i]){

return true;

}

}

}

dfsStack[node] = false;

return false;

}

public static void main(String args[]) {

int v = 4;

int[][] adj = new int[v][v];

adj[0][1] = 1;

adj[1][2] = 1;

adj[0][3] = 1;

System.out.println(isCyclic(adj));

}

}

// first and last occurrence of an element in a sorted array

public class Main {

static int findFirst(int[] arr, int target){

int left=0;

int end = arr.length - 1;

int result = -1;

while(left<=end){

int mid = left + (end-left)/2;

if(arr[mid]==target){

result = mid;

end = mid - 1;

}

else if(arr[mid]<target) {

left = mid+1;

}

else {

right = mid - 1;

}

}

return result;

}

static int findSecond(int[] arr, int target){

int left=0;

int end = arr.length - 1;

int result = -1;

while(left<=end){

int mid = left + (end-left)/2;

if(arr[mid]==target){

result = mid;

left = mid + 1;

}

else if(arr[mid]<target) {

left = mid+1;

}

else {

right = mid - 1;

}

}

return result;

}

public static void main(String args[]) {

}

}

//sorting a binary array

public class Main {

public static void main(String args[]) {

int[] arr = {1,0,0,1,0,0,1,1,0,1,1};

int left = 0;

int right = arr.length - 1;

while(left<right){

while(arr[left]==0 && left<right){

left++;

}

while(arr[right]==1 && left<right){

right--;

}

if(left<right) {

arr[left]=0;

arr[right]=1;

left++;

right--;

}

}

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

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

}

}

}

//depth of a binary tree

class Node {

int data;

Node left, right;

Node(int n){

data = n;

}

}

public class Main {

static int depthRecursive(Node root) {

if(root==null) {

return 0;

}

int leftDepth = depthRecursive(root.left);

int rightDepth = depthRecursive(root.right);

if(leftDepth>rightDepth) {

return leftDepth + 1;

}

else {

return rightDepth + 1;

}

}

public static void main(String args[]) {

Node root = new Node(19);

root.left = new Node(10);

root.right = new Node(10);

root.left.left = new Node(12);

root.left.right = new Node(8);

root.left.left.right = new Node(3);

root.left.left.right.left = new Node(13);

int depth = depthRecursive(root);

System.out.println(depth);

}

}

//dividing team by age

class Team {

String name;

int age1, age2;

Team(String n, int a1, int a2) {

name = n;

age1 = a1;

age2 = a2;

}

}

public class Main {

static boolean lessThan(Team t1, Team t2) {

if(Math.abs(t1.age2-t1.age1) < Math.abs(t2.age2-t2.age1)) {

return true;

}

else {

return false;

}

}

public static void main(String args[]) {

Team[] teams = new Team[10];

teams[0] = new Team("Team 1",17,20);

teams[1] = new Team("Team 2",27,16);

teams[2] = new Team("Team 3",19,16);

teams[3] = new Team("Team 4",21,23);

teams[4] = new Team("Team 5",19,46);

teams[5] = new Team("Team 6",39,16);

teams[6] = new Team("Team 7",28,16);

teams[7] = new Team("Team 8",11,11);

teams[8] = new Team("Team 9",15,12);

teams[9] = new Team("Team 10",15,12);

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

for(int j=0;j<teams.length-1;j++) {

if(lessThan(teams[j],teams[j+1])){

Team temp = teams[j];

teams[j] = teams[j+1];

teams[j+1] = temp;

}

}

}

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

System.out.println(teams[i].name);

}

}

}

class Team {

String name;

int age1, age2;

Team(String n, int a1, int a2) {

name = n;

age1 = a1;

age2 = a2;

}

boolean lessThan(Team t) {

if(Math.abs(this.age2-this.age1) < Math.abs(t.age2-t.age1)) {

return true;

}

else {

return false;

}

}

}

public class Main {

public static void main(String args[]) {

Team[] teams = new Team[10];

teams[0] = new Team("Team 1",17,20);

teams[1] = new Team("Team 2",27,16);

teams[2] = new Team("Team 3",19,16);

teams[3] = new Team("Team 4",21,23);

teams[4] = new Team("Team 5",19,46);

teams[5] = new Team("Team 6",39,16);

teams[6] = new Team("Team 7",28,16);

teams[7] = new Team("Team 8",11,11);

teams[8] = new Team("Team 9",15,12);

teams[9] = new Team("Team 10",15,12);

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

for(int j=0;j<teams.length-1;j++) {

if(teams[j].lessThan(teams[j+1])){

Team temp = teams[j];

teams[j] = teams[j+1];

teams[j+1] = temp;

}

}

}

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

System.out.println(teams[i]. name);

}

}

}

//numbers

class Node {

int data;

Node next;

Node(int n){

data = n;

}

}

class LinkedList {

Node root;

LinkedList(){

root = null;

}

void addNode(int n){

if(root==null){

root = new Node(n);

return;

}

Node iterator = root;

while(iterator.next!=null){

iterator = iterator.next;

}

iterator.next = new Node(n);

}

void show(){

showRec(root);

}

void showRec(Node root) {

if(root==null){

return;

}

System.out.print(root.data+"->");

showRec(root.next);

}

void genNum(){

Node iterator = root;

int result = 0,k,i=0;

while(iterator!=null){

k = iterator.data;

result = result + k \* (int)Math.pow(10,i);

i++;

iterator = iterator.next;

}

System.out.println();

System.out.println(result);

}

}

public class Main {

public static void main(String args[]){

LinkedList ll = new LinkedList();

ll.addNode(9);

ll.addNode(3);

ll.addNode(7);

ll.addNode(6);

ll.show();

ll.genNum();

}

}

Q3. Which command(s) can be used to check the disk I/O performance of a Linux system?

O iostat

O top

O sar

O all of the above

Q29. Choose the command that is used to find the length of the line in the file.

a) wc -r <file name>

b) wc -w <file name>

c) wc -l <file name>

d) wc -m <file name>

Q30. What is the purpose of the CMOS battery in the computer boot process?

O To provide power to the hard drive

O To store the BIOS settings

O To provide power to the motherboard

O To store the operating system files

Q42. What is the purpose of using SSH keys when accessing a Linux system?

O To allow access to the system without a password

O To bypass the need for an IP address

O To increase the security of the connection.

O To speed up the connection process

044. You are troubleshooting a network connectivity issue on a server and want to check the NIC information using the ethtool command. Which command will you use to display the NIC information for all the available network interfaces?

O ethtool eth0

O ethtool -a

O ethtool -i etho

O ethtool -S

//BST

class Node {

int data;

Node left,right;

Node(int n){

data = n;

}

}

class BST {

Node root;

BST(int n){

root = new Node(n);

}

void preOrder(){

preOrderRecursively(root);

}

void preOrderRecursively(Node root){

if(root==null){

return;

}

System.out.println(root.data);

preOrderRecursively(root.left);

preOrderRecursively(root.right);

}

boolean isValid(){

return isValidRec(root);

}

boolean isValidRec(Node root){

if(root==null){

return true;

}

if(root.left !=null ){

if(root.data<root.left.data) {

return false;

}

}

if(root.right !=null ){

if(root.data>root.right.data) {

return false;

}

}

return isValidRec(root.left) && isValidRec(root.right);

}

}

public class Main {

public static void main(String args[]){

BST bst = new BST(10);

bst.root.left = new Node(3);

bst.root.right = new Node(15);

bst.root.left.right = new Node(7);

bst.root.left.right.right = new Node(8);

System.out.println(bst.isValid());

}

}

//BST

class Node {

int data;

Node left,right;

Node(int n){

data = n;

}

}

class BST {

Node root;

BST(int n){

root = new Node(n);

}

void preOrder(){

preOrderRecursively(root);

}

void preOrderRecursively(Node root){

if(root==null){

return;

}

System.out.println(root.data);

preOrderRecursively(root.left);

preOrderRecursively(root.right);

}

/\* boolean isValid(){

}\*/

}

public class Main {

public static void main(String args[]){

BST bst = new BST(10);

bst.root.left = new Node(15);

bst.root.right = new Node(3);

bst.root.left.left = new Node(7);

bst.root.left.right = new Node(8);

bst.preOrder();

}

}

//quick sort

import java.util.Arrays;

public class Main{

static void quickSort(int[] array, int low, int high){

if(low<high) {

int partitionIndex = partition(array, low, high);

quickSort(array,low,partitionIndex-1);

quickSort(array,partitionIndex+1,high);

}

}

static int partition(int[] array, int low, int high) {

int pivotIndex = low + (high-low)/2;

int pivot = array[pivotIndex];

array = swap(array,pivotIndex,high);

int i = low;

int j = high-1;

while(i<=j){

while(array[i]<pivot){

i++;

}

while(array[j]>pivot){

j--;

}

if(i<=j){

swap(array,i,j);

i++;

j--;

}

}

array = swap(array,i,high);

return i;

}

static int[] swap(int[] arr, int i, int j){

int temp = arr[i];

arr[i] = arr[j];

arr[j] = temp;

return arr;

}

public static void main(String args[]) {

int[] arr = {10,8,5,15,3,19,21};

quickSort(arr,0,6);

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

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

}

}

}

//merge sort

public class Main{

public static void mergeSort(int[] arr) {

if(arr.length<=1) {

return;

}

int mid = arr.length/2;

int[] left = new int[mid];

int[] right = new int[arr.length - mid];

for(int i=0;i<mid;i++){

left[i] = arr[i];

}

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

right[i-mid] = arr[i];

}

mergeSort(left);

mergeSort(right);

merge(arr,left,right);

}

static void merge(int[] arr, int[] left, int[] right) {

int i=0,j=0,k=0;

while(i<left.length && j<right.length) {

if(left[i]<=right[j]){

arr[k] = left[i];

i++;

}

else {

arr[k] = right[j];

j++;

}

k++;

}

while(i<left.length){

arr[k] = left[i];

i++;

}

while(j<right.length){

arr[k] = right[j];

j++;

}

}

public static void main(String args[]) {

int[] arr = {10,8,5,15,3};

mergeSort(arr);

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

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

}

}

}

//insertion sort

public class Main{

public static void main(String args[]) {

int[] arr = {10,8,5,15,3};

int temp;

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

int key = arr[i];

int j = i-1;

while(j>=0 && arr[j]>key) {

arr[j+1] = arr[j];

j = j-1;

}

arr[j+1] = key;

}

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

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

}

}

}

//bubble sort

public class Main{

public static void main(String args[]) {

int[] arr = {10,8,5,15,3};

int temp;

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

for(int j=0;j<arr.length-1;j++){

if(arr[j]>arr[j+1]) {

temp = arr[j+1];

arr[j+1] = arr[j];

arr[j]=temp;

}

}

}

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

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

}

}

}

//Selection sort

public class Main{

public static void main(String args[]) {

int[] arr = {10,8,5,15,3};

int temp;

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

int minIndex = i;

for(int j=i+1;j<arr.length;j++) {

if(arr[j]<arr[minIndex]){

minIndex = j;

}

}

temp = arr[minIndex];

arr[minIndex] = arr[i];

arr[i] = temp;

}

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

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

}

}

}

//fibonacci dynamic programming

public class Main{

public static void main(String args[]) {

int[] arr = new int[10];

arr[0]=0;

arr[1]=1;

for(int i=2;i<10;i++) {

arr[i] = arr[i-1] + arr[i-2];

}

for(int i=0;i<10;i++) {

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

}

}

}

//Hash Table

class HashTable {

String[] keys;

int[] values;

int size;

HashTable(int n) {

keys = new String[n];

values = new int[n];

size = n;

}

int hash(String key){

return Math.abs(key.hashCode())%size;

}

void put(String key, int value){

int index = hash(key);

if(keys[index]==null){

keys[index]=key;

values[index]=value;

}

else {

int currIndex = index;

do {

currIndex = (currIndex+1)%size;

if(keys[currIndex]==null){

keys[currIndex] = key;

values[currIndex] = value;

return;

}

} while(currIndex!=index);

System.out.println("Cannot afford new values, hash table is full");

}

}

int get(String key) {

int index = hash(key);

if(keys[index]==key) {

return values[index];

}

else {

int currIndex = index;

do {

currIndex = (currIndex+1)%size;

if(keys[currIndex]==key){

return values[currIndex];

}

} while(currIndex!=index);

return -1;

}

}

}

public class Main{

public static void main(String args[]) {

HashTable ht = new HashTable(10);

ht.put("a",1);

ht.put("b",2);

ht.put("c",3);

ht.put("d",4);

System.out.println(ht.get("a"));

System.out.println(ht.get("d"));

System.out.println(ht.get("k"));

}

}

//Associate Array

class AsscoiateArray {

String[] keys;

int[] values;

int currIndex;

AsscoiateArray(int size){

keys = new String[size];

values = new int[size];

currIndex = 0;

}

void put(String key, int value) {

keys[currIndex] = key;

values[currIndex] = value;

currIndex++;

}

int get(String key) {

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

if(keys[i]==key) {

return values[i];

}

}

return -1;

}

}

public class Main{

public static void main(String args[]) {

AsscoiateArray aa = new AsscoiateArray(10);

aa.put("a",1);

aa.put("b",2);

aa.put("c",3);

aa.put("d",4);

System.out.println(aa.get("b"));

System.out.println(aa.get("d"));

System.out.println(aa.get("f"));

}

}

public class Main {

static void binarySearch(int[] arr, int start, int end, int search) {

int mid = start + (end-start)/2;

if(arr[mid]==search){

System.out.println(mid);

return ;

}

else if(arr[mid]<search){

binarySearch(arr, mid+1, end, search);

}

else {

binarySearch(arr,start,mid-1,search);

}

}

public static void main(String args[]){

int[] arr = {10,20,30,40,50,60,70,80,90,100};

binarySearch(arr,0,arr.length,90);

}

}

class Queue {

int[] arr;

int start = 0, rear = 0;

Queue(int n) {

arr = new int[n];

}

void enqueue(int n) {

arr[rear++] = n;

}

int getLength() {

return rear - start;

}

void viewQueue() {

for (int i = start; i < rear; i++) {

System.out.print(arr[i] + " | ");

}

System.out.println();

}

int dequeue() {

int n = arr[start++];

return n;

}

boolean isEmpty() {

return (start == rear);

}

}

class Stack {

int[] arr;

int top=-1;

Stack(int n){

arr = new int[n];

}

void push(int n) {

arr[++top] = n;

}

int pop(){

return arr[top--];

}

boolean isEmpty(){

return (top==-1);

}

}

class Graph {

int[][] edges;

int vertices;

Queue q;

Stack s;

Graph(int n) {

vertices = n;

edges = new int[n][n];

q = new Queue(vertices);

s = new Stack(vertices);

}

void DFS(int start){

int curVertex;

boolean[] visited = new boolean[vertices];

visited[start]=true;

s.push(start);

while(!s.isEmpty()){

curVertex = s.pop();

System.out.println(curVertex);

for (int i = 0; i < vertices; i++) {

if (edges[curVertex][i] == 1 && (!visited[i])) {

visited[i] = true;

s.push(i);

}

}

}

}

void BFS(int start) {

boolean[] visited = new boolean[vertices];

q.enqueue(start);

visited[start] = true;

int curVertex;

while (!q.isEmpty()) {

curVertex = q.dequeue();

System.out.println(curVertex);

for (int i = 0; i < vertices; i++) {

if (edges[curVertex][i] == 1 && (!visited[i])) {

visited[i] = true;

q.enqueue(i);

}

}

}

}

void addEdge(int source, int destination) {

edges[source][destination] = 1;

// if undirected only

edges[destination][source] = 1;

}

void removeEdge(int source, int destination) {

edges[source][destination] = 0;

// if undirected only

edges[destination][source] = 0;

}

}

public class Main {

public static void main(String args[]) {

Graph g = new Graph(5);

g.addEdge(0, 1);

g.addEdge(0, 2);

g.addEdge(0, 3);

g.addEdge(1, 4);

g.addEdge(1, 3);

g.addEdge(3, 4);

g.addEdge(3, 2);

System.out.println("DFS");

g.DFS(0);

System.out.println("BFS");

g.BFS(0);

}

}

class Queue {

int[] arr;

int start = 0, rear = 0;

Queue(int n) {

arr = new int[n];

}

void enqueue(int n) {

arr[rear++] = n;

}

int getLength() {

return rear - start;

}

void viewQueue() {

for (int i = start; i < rear; i++) {

System.out.print(arr[i] + " | ");

}

System.out.println();

}

int dequeue() {

int n = arr[start++];

return n;

}

boolean isEmpty() {

return (start == rear);

}

}

class Graph {

int[][] edges;

int vertices;

Queue q;

Graph(int n) {

vertices = n;

edges = new int[n][n];

q = new Queue(vertices);

}

void BFS(int start) {

boolean[] visited = new boolean[vertices];

q.enqueue(start);

visited[start] = true;

int curVertex;

while (!q.isEmpty()) {

curVertex = q.dequeue();

System.out.println(curVertex);

for (int i = 0; i < vertices; i++) {

if (edges[curVertex][i] == 1 && (!visited[i])) {

visited[i] = true;

q.enqueue(i);

}

}

}

}

void addEdge(int source, int destination) {

edges[source][destination] = 1;

// if undirected only

edges[destination][source] = 1;

}

void removeEdge(int source, int destination) {

edges[source][destination] = 0;

// if undirected only

edges[destination][source] = 0;

}

}

public class Main {

public static void main(String args[]) {

Graph g = new Graph(5);

g.addEdge(0, 1);

g.addEdge(0, 2);

g.addEdge(0, 3);

g.addEdge(1, 4);

g.addEdge(1, 3);

g.addEdge(3, 4);

g.addEdge(3, 2);

g.BFS(0);

}

}

import java.util.LinkedList;

class Graph {

LinkedList<Integer>[] edges;

int vertices;

Graph(int n){

vertices = n;

edges = new LinkedList[vertices];

for(int i=0;i<vertices;i++){

edges[i] = new LinkedList();

}

}

void addEdge(int source, int destination) {

edges[source].add(destination);

edges[destination].add(source);

}

void removeEdge(int source, int destination) {

edges[source].remove(destination);

edges[destination].remove(source);

}

void allEdges() {

for(int i=0;i<vertices;i++){

for(int j=0;j<edges[i].size();j++) {

System.out.println(i+" - "+edges[i].get(j));

}

}

}

}

public class Main {

public static void main(String args[]) {

Graph g = new Graph(5);

g.addEdge(0,1);

g.addEdge(0,2);

g.addEdge(0,3);

g.addEdge(1,4);

g.addEdge(1,3);

g.addEdge(3,4);

g.addEdge(3,2);

g.allEdges();

}

}

class Graph {

int[][] edges;

int vertices;

Graph(int n){

vertices = n;

edges = new int[n][n];

}

void addEdge(int source, int destination) {

edges[source][destination] = 1;

//if undirected only

edges[destination][source] = 1;

}

void removeEdge(int source, int destination) {

edges[source][destination] = 0;

//if undirected only

edges[destination][source] = 0;

}

void allEdges() {

for(int i=0;i<vertices;i++){

for(int j=0;j<vertices;j++) {

if(edges[i][j]==1) {

System.out.println(i+" - "+j);

}

}

}

}

}

public class Main {

public static void main(String args[]) {

Graph g = new Graph(5);

g.addEdge(0,1);

g.addEdge(0,2);

g.addEdge(0,3);

g.addEdge(1,4);

g.addEdge(1,3);

g.addEdge(3,4);

g.addEdge(3,2);

g.allEdges();

}

}

//start coding a new program

class Node {

int data;

Node left, right;

Node (int n) {

data = n;

}

}

class BST {

Node root;

BST() {

root = null;

}

void insert(int n) {

root = insertRecursively(root,n);

}

Node insertRecursively(Node root, int n){

if(root==null){

root = new Node(n);

return root;

}

if(root.data>n){

root.left = insertRecursively(root.left,n);

}

else if(root.data<n) {

root.right = insertRecursively(root.right,n);

}

return root;

}

void search(int n) {

searchRecursively(root, n);

}

void searchRecursively(Node root, int n) {

if(root==null){

System.out.println("Not found");

return;

}

if(root.data==n) {

System.out.println("Found");

return;

}

if(root.data>n) {

System.out.print("L");

searchRecursively(root.left, n);

}

else {

System.out.print("R");

searchRecursively(root.right, n);

}

}

void inOrder() {

inOrderRecursively(root);

}

void inOrderRecursively(Node root) {

if(root!=null){

inOrderRecursively(root.left);

System.out.println(root.data);

inOrderRecursively(root.right);

}

}

void delete(int value) {

root = deleteRecursively(root,value);

}

Node deleteRecursively(Node root, int value) {

if(root==null) {

return root;

}

if(value<root.data) {

root.left = deleteRecursively(root.left,value);

}

else if(value>root.data) {

root.right = deleteRecursively(root.right,value);

}

else {

if(root.left==null)

{

return root.right;

}

if(root.right==null) {

return root.left;

}

root.data = root.left.data;

root.right = deleteRecursively(root.right, root.data);

}

return root;

}

}

public class Main {

public static void main(String args[]){

BST bst = new BST();

bst.insert(5);

bst.insert(6);

bst.insert(4);

bst.insert(7);

bst.insert(3);

bst.insert(10);

bst.inOrder();

System.out.println("Deleteing...");

bst.delete(6);

bst.inOrder();

}

}

//start coding a new program

class Node {

int data;

Node left, right;

Node (int n) {

data = n;

}

}

class BST {

Node root;

BST() {

root = null;

}

void insert(int n) {

root = insertRecursively(root,n);

}

Node insertRecursively(Node root, int n){

if(root==null){

root = new Node(n);

return root;

}

if(root.data>n){

root.left = insertRecursively(root.left,n);

}

else if(root.data<n) {

root.right = insertRecursively(root.right,n);

}

return root;

}

void search(int n) {

searchRecursively(root, n);

}

void searchRecursively(Node root, int n) {

if(root==null){

System.out.println("Not found");

return;

}

if(root.data==n) {

System.out.println("Found");

return;

}

if(root.data>n) {

System.out.print("L");

searchRecursively(root.left, n);

}

else {

System.out.print("R");

searchRecursively(root.right, n);

}

}

void inOrder() {

inOrderRecursively(root);

}

void inOrderRecursively(Node root) {

if(root!=null){

inOrderRecursively(root.left);

System.out.println(root.data);

inOrderRecursively(root.right);

}

}

}

public class Main {

public static void main(String args[]){

BST bst = new BST();

bst.insert(5);

bst.insert(6);

bst.insert(4);

bst.insert(7);

bst.insert(3);

bst.insert(10);

bst.inOrder();

bst.search(9);

}

}

//start coding a new program

class Node {

int data;

Node left, right;

Node (int n) {

data = n;

}

}

class BST {

Node root;

BST() {

root = null;

}

void insert(int n) {

root = insertRecursively(root,n);

}

Node insertRecursively(Node root, int n){

if(root==null){

root = new Node(n);

return root;

}

if(root.data>n){

root.left = insertRecursively(root.left,n);

}

else if(root.data<n) {

root.right = insertRecursively(root.right,n);

}

return root;

}

void inOrder() {

inOrderRecursively(root);

}

void inOrderRecursively(Node root) {

if(root!=null){

inOrderRecursively(root.left);

System.out.println(root.data);

inOrderRecursively(root.right);

}

}

}

public class Main {

public static void main(String args[]){

BST bst = new BST();

bst.insert(5);

bst.insert(6);

bst.insert(4);

bst.insert(7);

bst.insert(3);

bst.insert(10);

bst.inOrder();

}

}

import java.util.Scanner;

class Node {

int data;

Node next;

public Node(int n){

data = n;

}

}

class LinkedList {

Node head;

void insertAtFirst(int data) {

if(head==null){

head = new Node(data);

}

else {

Node new\_node = new Node(data);

new\_node.next = head;

head = new\_node;

}

}

void deleteAtFirst() {

Node new\_head = head.next;

head = new\_head;

}

}

class Stack {

LinkedList ll;

public Stack()

{

ll = new LinkedList();

}

void push(int n){

ll.insertAtFirst(n);

}

int pop(){

int k = ll.head.data;

ll.deleteAtFirst();

return k;

}

void print(){

Node iterator = ll.head;

while(iterator.next != null){

System.out.println(iterator.data);

iterator = iterator.next;

}

}

}

public class Main {

public static void main(String[] args) {

Stack s = new Stack();

Scanner sc = new Scanner(System.in);

s.push(sc.nextInt());

s.push(sc.nextInt());

s.push(sc.nextInt());

s.push(sc.nextInt());

s.push(sc.nextInt());

s.pop();

s.pop();

s.print();

}

}

class Queue {

int[] arr;

int max\_size, front, rear;

public Queue(int size) {

max\_size = size;

front = 0;

rear = 0;

}

void enque(int n) {

arr[rear] = n;

rear++;

}

int deque() {

int k;

k = arr[front];

for(int i=0;i<rear-1;i++){

arr[i] = arr[i+1];

}

rear--;

System.out.println(k);

return k;

}

boolean isEmpty() {

if(front==rear) {

return true;

}

else {

return false;

}

}

boolean isFull(){

if(rear==max\_size) {

return true;

}

else {

return false;

}

}

}

public class Main {

public static void main(String args[]) {

}

}

-----------------------------------------------------------------------------------

class Stack {

int[] arr;

int max\_size,top;

public Stack(int n) {

arr = new int[n];

max\_size = n;

top = -1;

}

boolean isFull(){

if(top==(max\_size-1)){

return true;

}

else {

return false;

}

}

boolean isEmpty() {

if(top==-1)

{

return true;

}

else {

return false;

}

}

void push(int k){

if(!isFull()){

top++;

arr[top] = k;

System.out.println("Element PUshed");

}

else

{

System.out.println("Cannot push further, stack is full");

}

}

int pop() {

int k;

if(!isEmpty())

{

k = arr[top];

top--;

System.out.println(k);

return k;

}

else {

System.out.println("Stack is empty, nothing to pop");

return -1;

}

}

int peek() {

if(!isEmpty()){

return arr[top];

}

else {

System.out.println("Stack is empty, nothing to peek");

return -1;

}

}

}

public class Main {

public static void main(String args[]) {

Stack st = new Stack(3);

st.push(5);

st.push(8);

st.push(3);

st.push(1);

st.pop();

st.pop();

st.pop();

st.pop();

}

}

---------------------------------------------------------------------------------------------------------------------------------------.

class Car {

String color;

int max\_speed;

int curr\_speed;

public Car(String car\_color, int car\_max\_speed) {

color = car\_color;

max\_speed = car\_max\_speed;

curr\_speed = 0;

}

int accelerate(int accelerate\_by) {

curr\_speed = curr\_speed + accelerate\_by;

System.out.println("Vroom....");

return curr\_speed;

}

void applyBrakes() {

System.out.println("Krrrr......");

}

}

public class Main {

public static void main(String args[]) {

Car c1 = new Car("RED",180);

int new\_speed;

System.out.println("Top Speed : "+(c1.max\_speed));

System.out.println("Color : "+c1.color);

new\_speed = c1.accelerate(10);

System.out.println("Current Speed : "+new\_speed);

new\_speed = c1.accelerate(10);

System.out.println("Current Speed : "+new\_speed);

new\_speed = c1.accelerate(10);

System.out.println("Current Speed : "+new\_speed);

new\_speed = c1.accelerate(10);

System.out.println("Current Speed : "+new\_speed);

new\_speed = c1.accelerate(10);

System.out.println("Current Speed : "+new\_speed);

c1.applyBrakes();

}

}