Set 1:queue using stack:

import java.util.Scanner;

import java.util.Stack;

public class QueueUsingTwoStacks {

Stack<Integer> stack1 = new Stack<>();

Stack<Integer> stack2 = new Stack<>();

// Enqueue operation

public void enqueue(int value) {

stack1.push(value);

System.out.println(value + " enqueued to queue.");

}

// Dequeue operation

public int dequeue() {

if (stack2.isEmpty()) {

if (stack1.isEmpty()) {

System.out.println("Queue is empty.");

return -1;

}

while (!stack1.isEmpty()) {

stack2.push(stack1.pop());

}

}

int val = stack2.pop();

System.out.println(val + " dequeued from queue.");

return val;

}

// Display queue elements

public void display() {

if (stack1.isEmpty() && stack2.isEmpty()) {

System.out.println("Queue is empty.");

return;}

// Temporarily reverse stack1 into a temp stack to maintain order

Stack<Integer> temp = new Stack<>();

System.out.print("Queue: ");

// First, print elements from stack2 (top to bottom)

for (int i = stack2.size() - 1; i >= 0; i--) {

System.out.print(stack2.get(i) + " ");

}

// Then reverse stack1 to get its elements in queue order

for (int i = 0; i < stack1.size(); i++) {

temp.push(stack1.get(i));

}

while (!temp.isEmpty()) {

System.out.print(temp.pop() + " ");

}

System.out.println();

}

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

QueueUsingTwoStacks queue = new QueueUsingTwoStacks();

int choice;

do {

System.out.println("\nQueue Using Two Stacks Menu:");

System.out.println("1. Enqueue");

System.out.println("2. Dequeue");

System.out.println("3. Display");

System.out.println("4. Exit");

System.out.print("Enter your choice: ");

choice = scanner.nextInt();

switch (choice) {

case 1:

System.out.print("Enter value to enqueue: ");

int value = scanner.nextInt();

queue.enqueue(value);

break;

case 2:

queue.dequeue();

break;

case 3:

queue.display();

break;

case 4:

System.out.println("Exiting program.");

break;

default:

System.out.println("Invalid choice. Try again.");

}

} while (choice != 4);

scanner.close();

}

}

Set1: minimum element in rotated sorted array

import java.util.Scanner;

public class MinimumInRotatedSortedArray {

public static int findMin(int[] nums) {

int left = 0;

int right = nums.length - 1;

// If the array is not rotated

if (nums[left] < nums[right]) {

return nums[left];

}

while (left < right) {

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

if (nums[mid] > nums[right]) {

left = mid + 1;

} else {

right = mid;

}

}

return nums[left];

}

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

// Taking dynamic input

System.out.print("Enter the number of elements in the array: ");

int n = scanner.nextInt();

int[] arr = new int[n];

System.out.println("Enter elements of the rotated sorted array:");

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

arr[i] = scanner.nextInt();

}

int min = findMin(arr);

System.out.println("Minimum element in the rotated sorted array is: " + min);

scanner.close();

}

}

Set3: print the position of the element in the rotated binary search tree

import java.util.Scanner;

class Node {

int data;

Node left, right;

Node(int data) {

this.data = data;

left = right = null;

}

}

class RotatedBSTSearch {

static Node root = null;

public static Node insert(Node root, int data) {

if (root == null) {

return new Node(data);

}

if (data < root.data) {

root.left = insert(root.left, data);

} else {

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

}

return root;

}

public static int searchInRotatedBST(Node root, int target) {

if (root == null) {

return -1;

}

int pos = 1;

Node current = root;

while (current != null) {

if (current.data == target) {

return pos;

}

if (current.left != null && target < current.data && target >= current.left.data) {

current = current.left;

} else if (current.right != null && target > current.data && target <= current.right.data) {

current = current.right;

} else {

break; // not found

}

pos++;

}

return -1; // element not found

}

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

System.out.print("Enter number of elements in the BST: ");

int n = scanner.nextInt();

System.out.println("Enter elements to insert into the BST (sorted order):");

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

int value = scanner.nextInt();

root = insert(root, value);

}

System.out.print("Enter the element to search for: ");

int target = scanner.nextInt();

int position = searchInRotatedBST(root, target);

if (position != -1) {

System.out.println("Element " + target + " found at position: " + position);

} else {

System.out.println("Element " + target + " not found in the BST.");

}

scanner.close();

}

}

Set3: delete the duplicate elements in a sorted linked list

import java.util.Scanner;

class Node {

int data;

Node next;

Node(int data) {

this.data = data;

next = null;

}

}class RemoveDuplicatesSortedList {

static Node head = null;

public static void insert(int val) {

Node newNode = new Node(val);

if (head == null || head.data >= newNode.data) {

newNode.next = head;

head = newNode;

} else {

Node current = head;

while (current.next != null && current.next.data < newNode.data) {

current = current.next;

}

newNode.next = current.next;

current.next = newNode;

}

}

public static Node removeDuplicates(Node head) {

Node current = head;

while (current != null && current.next != null) {

if (current.data == current.next.data) {

current.next = current.next.next;

} else {

current = current.next;

}

}

return head;

}

public static void printList(Node head) {

while (head != null) {

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

head = head.next;

}

System.out.println();

}

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

System.out.print("Enter number of elements in the sorted linked list: ");

int n = scanner.nextInt();

System.out.println("Enter sorted elements:");

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

insert(scanner.nextInt());

}

head = removeDuplicates(head);

System.out.println("Linked list after removing duplicates:");

printList(head); } }

Set5: give sorted array (except 1 element all elements will occurs 2 time) return that one time occurred element otherwise return 2

import java.util.Scanner;

class FindSingleElement {

public static int findSingleElement(int[] nums) {

int left = 0, right = nums.length - 1;

while (left < right) {

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

if (mid % 2 == 1) {

mid--; // Move to the left of the pair

}

if (nums[mid] == nums[mid + 1]) {

left = mid + 2; // The element we are looking for is in the right half

} else {

right = mid; // The element we are looking for is in the left half

}

}

return nums[left]; // Single element will be at the 'left' position

}

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

System.out.print("Enter number of elements in the sorted array: ");

int n = scanner.nextInt();

int[] arr = new int[n];

System.out.println("Enter the sorted elements:");

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

arr[i] = scanner.nextInt();

}

int result = findSingleElement(arr);

System.out.println("The element that occurs only once is: " + result);

} }

Set5: ReverseLinkedList

import java.util.Scanner;

class Node {

int data;

Node next;

Node(int data) {

this.data = data;

next = null;

}

}

class ReverseLinkedList {

static Node head = null;

public static void insert(int val) {

Node newNode = new Node(val);

if (head == null) {

head = newNode;

} else {

Node current = head;

while (current.next != null) {

current = current.next;

}

current.next = newNode;

}

}

public static Node reverse(Node head) {

Node prev = null;

Node current = head;

Node next = null;

while (current != null) {

next = current.next;

current.next = prev;

prev = current;

current = next;

}

head = prev;

return head;

}

public static void printList(Node head) {

Node current = head;

while (current != null) {

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

current = current.next;

}

System.out.println();

}

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

System.out.print("Enter number of elements in the linked list: ");

int n = scanner.nextInt();

System.out.println("Enter elements:");

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

insert(scanner.nextInt());

}

head = reverse(head);

System.out.println("Reversed linked list:");

printList(head);

}

}

Set6: TwoSum

import java.util.\*;

class TwoSum {

public static int[] twoSum(int[] nums, int target) {

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

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

int complement = target - nums[i];

if (map.containsKey(complement)) {

return new int[]{map.get(complement), i};

}

map.put(nums[i], i);

}

return new int[]{-1, -1}; // Return invalid indices if no solution is found

}

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

System.out.print("Enter number of elements in the array: ");

int n = scanner.nextInt();

int[] nums = new int[n];

System.out.println("Enter the elements:");

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

nums[i] = scanner.nextInt();

}

System.out.print("Enter the target sum: ");

int target = scanner.nextInt();

int[] result = twoSum(nums, target);

if (result[0] != -1) {

System.out.println("The indices of the two numbers are: " + result[0] + " and " + result[1]);

} else {

System.out.println("No solution found.");

}

}

}

Set6: SearchInLinkedList

import java.util.Scanner;

class Node {

int data;

Node next;

Node(int data) {

this.data = data;

this.next = null;

}

}

class SearchInLinkedList {

static Node head = null;

public static void insert(int val) {

Node newNode = new Node(val);

if (head == null) {

head = newNode;

} else {

Node current = head;

while (current.next != null) {

current = current.next;

}

current.next = newNode;

}

}

public static boolean iterativeSearch(Node head, int target) {

Node current = head;

while (current != null) {

if (current.data == target) {

return true;

}

current = current.next;

}

return false;

}

public static boolean recursiveSearch(Node head, int target) {

if (head == null) {

return false;

}

if (head.data == target) {

return true;

}

return recursiveSearch(head.next, target);

}

public static void printList(Node head) {

Node current = head;

while (current != null) {

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

current = current.next;

}

System.out.println();

}

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

System.out.print("Enter number of elements in the linked list: ");

int n = scanner.nextInt();

System.out.println("Enter elements:");

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

insert(scanner.nextInt());

}

System.out.print("Enter the element to search for: ");

int target = scanner.nextInt();

boolean iterativeResult = iterativeSearch(head, target);

boolean recursiveResult = recursiveSearch(head, target);

if (iterativeResult) {

System.out.println("Element " + target + " found using iteration.");

} else {

System.out.println("Element " + target + " not found using iteration.");

}

if (recursiveResult) {

System.out.println("Element " + target + " found using recursion.");

} else {

System.out.println("Element " + target + " not found using recursion.");

}

}

}

Set7: RemoveDuplicatesFromArray

import java.util.Scanner;

class RemoveDuplicatesFromArray {

public static int[] removeDuplicates(int[] nums) {

if (nums.length == 0)

return nums;

int uniqueIndex = 1; // First element is always unique

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

if (nums[i] != nums[i - 1]) {

nums[uniqueIndex] = nums[i];

uniqueIndex++;

}

}

int[] result = new int[uniqueIndex];

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

result[i] = nums[i];

}

return result;}

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

System.out.print("Enter the number of elements in the sorted array: ");

int n = scanner.nextInt();

int[] arr = new int[n];

System.out.println("Enter the sorted elements:");

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

arr[i] = scanner.nextInt();

}

int[] result = removeDuplicates(arr);

System.out.println("Array after removing duplicates:");

for (int num : result) {

System.out.print(num + " ");

}

}

}

Set7: ReverseQueueUsingStack

import java.util.\*;

class ReverseQueueUsingStack {

public static void reverseQueue(Queue<Integer> queue) {

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

while (!queue.isEmpty()) {

stack.push(queue.poll());

}

while (!stack.isEmpty()) {

queue.offer(stack.pop());

}

}

public static void printQueue(Queue<Integer> queue) {

for (int num : queue) {

System.out.print(num + " ");}

System.out.println();

}

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

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

System.out.print("Enter the number of elements in the queue: ");

int n = scanner.nextInt();

System.out.println("Enter the elements of the queue:");

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

queue.offer(scanner.nextInt());

}

System.out.println("Queue before reversing:");

printQueue(queue);

reverseQueue(queue);

System.out.println("Queue after reversing:");

printQueue(queue);

}

}

Set8: RemoveConsecutiveDuplicates

import java.util.Scanner;

class RemoveConsecutiveDuplicates {

public static String removeConsecutiveDuplicates(String str) {

if (str == null || str.isEmpty()) {

return str;

}

StringBuilder result = new StringBuilder();

result.append(str.charAt(0));

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

if (str.charAt(i) != str.charAt(i - 1)) {

result.append(str.charAt(i));

}}

return result.toString();

}

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

System.out.print("Enter the string: ");

String input = scanner.nextLine();

String result = removeConsecutiveDuplicates(input);

System.out.println("String after removing consecutive duplicates: " + result);

}

}

Set8: stack sort

import java.util.\*;

class StackSort {

public static void sortStack(Stack<Integer> stack) {

Stack<Integer> tempStack = new Stack<>();

while (!stack.isEmpty()) {

int current = stack.pop();

while (!tempStack.isEmpty() && tempStack.peek() > current) {

stack.push(tempStack.pop());

}

tempStack.push(current);

}

while (!tempStack.isEmpty()) {

stack.push(tempStack.pop());

}

}

public static void printStack(Stack<Integer> stack) {

for (Integer num : stack) {

System.out.print(num + " ");

}

System.out.println();

}

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

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

System.out.print("Enter the number of elements in the stack: ");

int n = scanner.nextInt();

System.out.println("Enter the elements of the stack:");

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

stack.push(scanner.nextInt());

}

System.out.println("Original stack:");

printStack(stack);

sortStack(stack);

System.out.println("Sorted stack:");

printStack(stack);

}

}

Set9: EquilibriumIndex

import java.util.Scanner;

class EquilibriumIndex {

public static int findEquilibriumIndex(int[] arr) {

int totalSum = 0;

int leftSum = 0;

for (int num : arr) {

totalSum += num;

}

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

totalSum -= arr[i];

if (leftSum == totalSum) {

return i;

}

leftSum += arr[i];

}

return -1;

}

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

System.out.print("Enter the number of elements in the array: ");

int n = scanner.nextInt();

int[] arr = new int[n];

System.out.println("Enter the elements of the array:");

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

arr[i] = scanner.nextInt();

}

int result = findEquilibriumIndex(arr);

if (result == -1) {

System.out.println("No equilibrium index found.");

} else {

System.out.println("Equilibrium index is: " + result);

}

}

}

Set9: BinaryTree

import java.util.Scanner;

class BinaryTree {

static class Node {

int data;

Node left, right;

public Node(int item) {

data = item;

left = right = null;

} }

Node root;

public Node insert(Node root, int data) {

if (root == null) {

root = new Node(data);

return root;

}

if (data < root.data) {

root.left = insert(root.left, data);

} else if (data > root.data) {

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

}

return root;

}

private int heightAndDiameter(Node node, int[] diameter) {

if (node == null) {

return 0;

}

int leftHeight = heightAndDiameter(node.left, diameter);

int rightHeight = heightAndDiameter(node.right, diameter);

diameter[0] = Math.max(diameter[0], leftHeight + rightHeight);

return Math.max(leftHeight, rightHeight) + 1;

}

public int diameter() {

int[] diameter = new int[1];

heightAndDiameter(root, diameter);

return diameter[0];

}

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

BinaryTree tree = new BinaryTree();

System.out.print("Enter the number of elements in the tree: ");

int n = scanner.nextInt();

System.out.println("Enter the elements of the tree:");

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

int data = scanner.nextInt();

tree.root = tree.insert(tree.root, data); // Insert the element in the tree

}

System.out.println("Diameter of the BST is: " + tree.diameter());

}

}

Set11: TotalCostOfConnections

import java.util.\*;

class TotalCostOfConnections {

public static int totalCost(int[] costs) {

PriorityQueue<Integer> pq = new PriorityQueue<>();

for (int cost : costs) {

pq.add(cost);

}

int totalCost = 0;

while (pq.size() > 1) {

int first = pq.poll();

int second = pq.poll();

int combinedCost = first + second;

totalCost += combinedCost;

pq.add(combinedCost);

}

return totalCost;

}

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

System.out.print("Enter the number of elements in the array: ");

int n = scanner.nextInt();

int[] costs = new int[n];

System.out.println("Enter the elements of the array:");

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

costs[i] = scanner.nextInt();

}

int result = totalCost(costs);

System.out.println("Total cost of connections: " + result);

}

}

Set11: i act as a java programmer generate Contruct a BST and remove all half nodes Half nodes - nodes with single child by taking dynamic input

import java.util.Scanner;

class BinaryTree {

static class Node {

int data;

Node left, right;

public Node(int item) {

data = item;

left = right = null;

}

}

Node root;

public Node insert(Node root, int data) {

if (root == null) {

root = new Node(data);

return root;

}

if (data < root.data) {

root.left = insert(root.left, data);

} else if (data > root.data) {

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

}

return root;

}

)

public Node removeHalfNodes(Node node) {

if (node == null) {

return null;

}

node.left = removeHalfNodes(node.left);

node.right = removeHalfNodes(node.right);

if (node.left == null && node.right != null) {

return node.right; // Node has only right child

}

if (node.right == null && node.left != null) {

return node.left; // Node has only left child

}

return node;

}

public void inorderTraversal(Node root) {

if (root != null) {

inorderTraversal(root.left);

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

inorderTraversal(root.right);

}

}

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

BinaryTree tree = new BinaryTree();

System.out.print("Enter the number of elements in the tree: ");

int n = scanner.nextInt();

System.out.println("Enter the elements of the tree:");

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

int data = scanner.nextInt();

tree.root = tree.insert(tree.root, data); // Insert the element in the tree

}

System.out.println("Tree before removing half nodes (In-order): ");

tree.inorderTraversal(tree.root);

System.out.println();

tree.root = tree.removeHalfNodes(tree.root);

System.out.println("Tree after removing half nodes (In-order): ");

tree.inorderTraversal(tree.root);

System.out.println();

}

}

Set12: product of array elements except itself without using division operator -O(nlogn)

import java.util.Scanner;

public class ProductArray {

public static void productArray(int[] arr, int n) {

if (n == 1) {

System.out.println(0);

return;

}

int[] leftProduct = new int[n];

int[] rightProduct = new int[n];

int[] result = new int[n];

leftProduct[0] = 1;

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

leftProduct[i] = leftProduct[i - 1] \* arr[i - 1];

}

rightProduct[n - 1] = 1;

for (int i = n - 2; i >= 0; i--) {

rightProduct[i] = rightProduct[i + 1] \* arr[i + 1];

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

result[i] = leftProduct[i] \* rightProduct[i];

}

System.out.print("Product of array elements except itself: ");

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

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

}

System.out.println();

}

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

System.out.print("Enter the number of elements in the array: ");

int n = scanner.nextInt();

int[] arr = new int[n];

System.out.println("Enter the elements of the array:");

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

arr[i] = scanner.nextInt();

}

productArray(arr, n);

}

}

Set12: left view of bst

import java.util.\*;

public class BSTLeftView {

static class Node {

int data;

Node left, right;

public Node(int item) {

data = item;

left = right = null;

}

Node root;

public void leftView(Node root) {

if (root == null) return;

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

queue.add(root);

while (!queue.isEmpty()) {

int n = queue.size();

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

Node temp = queue.poll();

if (i == 1) {

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

}

if (temp.left != null) queue.add(temp.left);

if (temp.right != null) queue.add(temp.right);

}

}

}

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

BSTLeftView tree = new BSTLeftView();

System.out.print("Enter the number of nodes in the tree: ");

int n = scanner.nextInt();

System.out.println("Enter the elements of the tree:");

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

System.out.print("Enter value for node " + (i+1) + ": ");

int value = scanner.nextInt();

tree.root = insert(tree.root, value);

}

System.out.print("Left View of the BST: ");

tree.leftView(tree.root);

}

public static Node insert(Node root, int data) {

if (root == null) {

root = new Node(data);

return root;

}

if (data < root.data) {

root.left = insert(root.left, data);

} else if (data > root.data) {

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

}

return root;

}

}

Set13: Assign cookies (gfg)

import java.util.\*;

public class AssignCookies {

public static int findContentChildren(int[] g, int[] s) {

Arrays.sort(g);

Arrays.sort(s);

int i = 0, j = 0;

while (i < g.length && j < s.length) {

if (g[i] <= s[j]) {

i++;

}

j++;

}

return i;

}

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

System.out.print("Enter the number of children: ");

int n = scanner.nextInt();

int[] g = new int[n];

System.out.println("Enter the greed factor of the children:");

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

g[i] = scanner.nextInt();

}

System.out.print("Enter the number of cookies: ");

int m = scanner.nextInt();

int[] s = new int[m];

System.out.println("Enter the size of the cookies:");

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

s[i] = scanner.nextInt();

}

int result = findContentChildren(g, s);

System.out.println("Number of content children: " + result);

}

}

Set13: Width of binary search tree at given level

import java.util.\*;

public class BSTWidthAtLevel {

static class Node {

int data;

Node left, right;

public Node(int item) {

data = item;

left = right = null; }

}

Node root;

public int widthAtLevel(Node root, int level) {

if (root == null) return 0;

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

queue.add(root);

int currentLevel = 0;

int width = 0;

while (!queue.isEmpty()) {

int n = queue.size();

if (currentLevel == level) {

width = n;

break;

}

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

Node temp = queue.poll();

if (temp.left != null) queue.add(temp.left);

if (temp.right != null) queue.add(temp.right);

}

currentLevel++;

}

return width;

}

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

BSTWidthAtLevel tree = new BSTWidthAtLevel();

System.out.print("Enter the number of nodes in the tree: ");

int n = scanner.nextInt();

System.out.println("Enter the elements of the tree:");

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

System.out.print("Enter value for node " + (i + 1) + ": ");

int value = scanner.nextInt();

tree.root = insert(tree.root, value);

}

System.out.print("Enter the level to find the width: ");

int level = scanner.nextInt();

int width = tree.widthAtLevel(tree.root, level);

System.out.println("Width of the BST at level " + level + " is: " + width);

}

public static Node insert(Node root, int data) {

if (root == null) {

root = new Node(data);

return root;

}

if (data < root.data) {

root.left = insert(root.left, data);

} else if (data > root.data) {

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

}

return root;

}

}

Set14: lemonade. Exchange change of all the queue of customers

import java.util.Scanner;

public class LemonadeChange {

public static boolean lemonadeChange(int[] bills) {

int five = 0, ten = 0;

for (int bill : bills) {

if (bill == 5) {

five++;

} else if (bill == 10) {

if (five > 0) {

five--;

ten++;

} else {

return false;

}

} else if (bill == 20) {

if (ten > 0 && five > 0) {

ten--;

five--;

} else if (five >= 3) {

five -= 3;

} else {

return false;

}

}

}

return true;

}

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

System.out.print("Enter the number of customers: ");

int n = scanner.nextInt();

int[] bills = new int[n];

System.out.println("Enter the bills each customer pays:");

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

bills[i] = scanner.nextInt();

}

if (lemonadeChange(bills)) {

System.out.println("Yes, we can provide change to all customers.");

} else {

System.out.println("No, we cannot provide change to all customers.");

}

}

}

Set14: binary search tree to find height of the tree

import java.util.Scanner;

public class BSTHeight {

static class Node {

int data;

Node left, right;

public Node(int item) {

data = item;

left = right = null;

}

}

Node root;

public int height(Node root) {

if (root == null) {

return -1; // Return -1 for null node, height of empty tree is considered -1.

}

int leftHeight = height(root.left);

int rightHeight = height(root.right);

return Math.max(leftHeight, rightHeight) + 1;

}

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

BSTHeight tree = new BSTHeight();

System.out.print("Enter the number of nodes in the tree: ");

int n = scanner.nextInt();

System.out.println("Enter the elements of the tree:");

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

System.out.print("Enter value for node " + (i + 1) + ": ");

int value = scanner.nextInt();

tree.root = insert(tree.root, value);

}

int treeHeight = tree.height(tree.root);

System.out.println("Height of the BST: " + treeHeight);

}

public static Node insert(Node root, int data) {

if (root == null) {

root = new Node(data);

return root;

}

if (data < root.data) {

root.left = insert(root.left, data);

} else if (data > root.data) {

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

}

return root;

}

}

Set15: Given 3 stacks with positive numbers...Find maximum equal sum among the 3 stacks...

import java.util.\*;

public class MaximumEqualSum {

public static int findMaxEqualSum(Stack<Integer> stack1, Stack<Integer> stack2, Stack<Integer> stack3) {

int sum1 = 0, sum2 = 0, sum3 = 0;

for (int num : stack1) sum1 += num;

for (int num : stack2) sum2 += num;

for (int num : stack3) sum3 += num;

while (sum1 != sum2 || sum2 != sum3) {

if (sum1 > sum2 || sum1 > sum3) {

sum1 -= stack1.pop();

} else if (sum2 > sum1 || sum2 > sum3) {

sum2 -= stack2.pop();

} else {

sum3 -= stack3.pop();

}

if (stack1.isEmpty() || stack2.isEmpty() || stack3.isEmpty()) {

return 0;

}

}

return sum1;

}

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

Stack<Integer> stack1 = new Stack<>();

Stack<Integer> stack2 = new Stack<>();

Stack<Integer> stack3 = new Stack<>();

System.out.print("Enter the number of elements in stack 1: ");

int n1 = scanner.nextInt();

System.out.println("Enter the elements of stack 1:");

for (int i = 0; i < n1; i++) {

stack1.push(scanner.nextInt());

}

System.out.print("Enter the number of elements in stack 2: ");

int n2 = scanner.nextInt();

System.out.println("Enter the elements of stack 2:");

for (int i = 0; i < n2; i++) {

stack2.push(scanner.nextInt());

}

System.out.print("Enter the number of elements in stack 3: ");

int n3 = scanner.nextInt();

System.out.println("Enter the elements of stack 3:");

for (int i = 0; i < n3; i++) {

stack3.push(scanner.nextInt());

}

int maxEqualSum = findMaxEqualSum(stack1, stack2, stack3);

System.out.println("The maximum equal sum among the three stacks is: " + maxEqualSum);

}

}

Set15: Create a BST and find the max width of a BST

import java.util.\*;

public class MaxWidthOfBST {

static class Node {

int data;

Node left, right;

public Node(int item) {

data = item;

left = right = null;

}

}

Node root;

public int findMaxWidth(Node root) {

if (root == null) return 0;

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

queue.add(root);

int maxWidth = 0;

while (!queue.isEmpty()) {

int count = queue.size();

maxWidth = Math.max(maxWidth, count);

while (count > 0) {

Node currentNode = queue.poll();

if (currentNode.left != null) queue.add(currentNode.left);

if (currentNode.right != null) queue.add(currentNode.right);

count--;

}

}

return maxWidth;

}

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

MaxWidthOfBST tree = new MaxWidthOfBST();

System.out.print("Enter the number of nodes in the tree: ");

int n = scanner.nextInt();

System.out.println("Enter the elements of the tree:");

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

System.out.print("Enter value for node " + (i + 1) + ": ");

int value = scanner.nextInt();

tree.root = insert(tree.root, value);

}

int maxWidth = tree.findMaxWidth(tree.root);

System.out.println("The maximum width of the BST is: " + maxWidth);

}

public static Node insert(Node root, int data) {

if (root == null) {

root = new Node(data);

return root;

}

if (data < root.data) {

root.left = insert(root.left, data);

} else if (data > root.data) {

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

}

return root;

}

}

Set17: count all paths

import java.util.Scanner;

public class CountAllPaths {

static class Node {

int data;

Node left, right;

public Node(int item) {

data = item;

left = right = null;

}

}

Node root;

public int countPaths(Node node) {

if (node == null) {

return 0; // No path exists for null node

}

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

return 1;

}

return countPaths(node.left) + countPaths(node.right);

}

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

CountAllPaths tree = new CountAllPaths();

System.out.print("Enter the number of nodes in the tree: ");

int n = scanner.nextInt();

System.out.println("Enter the elements of the tree:");

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

System.out.print("Enter value for node " + (i + 1) + ": ");

int value = scanner.nextInt();

tree.root = insert(tree.root, value);

}

int totalPaths = tree.countPaths(tree.root);

System.out.println("The total number of paths from root to leaf is: " + totalPaths);

}

public static Node insert(Node root, int data) {

if (root == null) {

root = new Node(data);

return root;

}

if (data < root.data) {

root.left = insert(root.left, data);

} else if (data > root.data) {

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

}

return root;

}

}

Set17: breadth first traversal BST

import java.util.\*;

public class BFSOfBST {

static class Node {

int data;

Node left, right;

public Node(int item) {

data = item;

left = right = null;

}

}

Node root;

public void levelOrderTraversal(Node root) {

if (root == null) {

return;

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

queue.add(root);

while (!queue.isEmpty()) {

Node currentNode = queue.poll();

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

if (currentNode.left != null) {

queue.add(currentNode.left);

}

if (currentNode.right != null) {

queue.add(currentNode.right);

}

}

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

BFSOfBST tree = new BFSOfBST();

System.out.print("Enter the number of nodes in the tree: ");

int n = scanner.nextInt();

System.out.println("Enter the elements of the tree:");

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

System.out.print("Enter value for node " + (i + 1) + ": ");

int value = scanner.nextInt();

tree.root = insert(tree.root, value);

}

System.out.println("Level-order (Breadth-First) Traversal of the BST:");

tree.levelOrderTraversal(tree.root); }

public static Node insert(Node root, int data) {

if (root == null) {

root = new Node(data);

return root;

if (data < root.data) {

root.left = insert(root.left, data);

} else if (data > root.data) {

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

}

return root;

}

}

Set18: kadane algorithm

import java.util.Scanner;

public class KadaneAlgorithm {

public static int kadane(int[] arr) {

int maxSoFar = arr[0];

int maxEndingHere = arr[0];

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

maxEndingHere = Math.max(arr[i], maxEndingHere + arr[i]);

maxSoFar = Math.max(maxSoFar, maxEndingHere);

}

return maxSoFar;

}

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

System.out.print("Enter the number of elements in the array: ");

int n = scanner.nextInt();

int[] arr = new int[n];

System.out.println("Enter the elements of the array:");

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

arr[i] = scanner.nextInt();

}

int maxSum = kadane(arr);

System.out.println("The maximum sum of a contiguous subarray is: " + maxSum);

}

}

Set18: traversals -inorder,preorder,postorder

import java.util.Scanner;

public class InorderTraversal {

static class Node {

int data;

Node left, right;

public Node(int item) {

data = item;

left = right = null;

}

}

Node root;

public void inorder(Node node) {

if (node == null) {

return;

}

inorder(node.left); // Visit left subtree

System.out.print(node.data + " "); // Visit root

inorder(node.right); // Visit right subtree

}

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

InorderTraversal tree = new InorderTraversal();

System.out.print("Enter the number of nodes in the tree: ");

int n = scanner.nextInt();

System.out.println("Enter the elements of the tree:");

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

System.out.print("Enter value for node " + (i + 1) + ": ");

int value = scanner.nextInt();

tree.root = insert(tree.root, value);

}

System.out.print("Inorder Traversal of the tree: ");

tree.inorder(tree.root);

}

public static Node insert(Node root, int data) {

if (root == null) {

root = new Node(data);

return root;

}

if (data < root.data) {

root.left = insert(root.left, data);

} else if (data > root.data) {

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

return root;

}

}

Set19: .reverse only string letters(ab-cd) op-(dc-ba)

import java.util.Scanner;

public class ReverseStringLetters {

public static String reverseOnlyLetters(String s) {

char[] arr = s.toCharArray();

int left = 0, right = arr.length - 1;

while (left < right) {

// Skip non-letter characters

if (!Character.isLetter(arr[left])) {

left++;

} else if (!Character.isLetter(arr[right])) {

right--;

} else {

// Swap the letters

char temp = arr[left];

arr[left] = arr[right];

arr[right] = temp;

left++;

right--;

}

}

return new String(arr);

}

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

System.out.print("Enter the string: ");

String input = scanner.nextLine()

String result = reverseOnlyLetters(input);

System.out.println("Result after reversing letters: " + result);

}

}

Set19: implement two stacks using an array(menu driven program with push,pop,display,isempty,top for each stack)

import java.util.Scanner;

public class TwoStacks

static class TwoStack {

int[] arr;

int top1, top2;

int size;

public TwoStack(int size) {

this.size = size;

arr = new int[size];

top1 = -1; // Stack 1 starts from index 0

top2 = size; // Stack 2 starts from the end of the array

}

public void push1(int x) {

if (top1 < top2 - 1) {

top1++;

arr[top1] = x;

System.out.println(x + " pushed to Stack 1");

} else {

System.out.println("Stack 1 Overflow");

}

}

public void push2(int x) {

if (top1 < top2 - 1) {

top2--;

arr[top2] = x;

System.out.println(x + " pushed to Stack 2");

} else {

System.out.println("Stack 2 Overflow");

}

}

public int pop1() {

if (top1 >= 0) {

int x = arr[top1];

top1--;

return x;

} else {

System.out.println("Stack 1 Underflow");

return -1;

}

}

public int pop2() {

if (top2 < size) {

int x = arr[top2];

top2++;

return x;

} else {

System.out.println("Stack 2 Underflow");

return -1;

}

}

public void display1() {

if (top1 >= 0) {

System.out.print("Stack 1: ");

for (int i = 0; i <= top1; i++) {

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

}

System.out.println();

} else {

System.out.println("Stack 1 is empty");

}

}

public void display2() {

if (top2 < size) {

System.out.print("Stack 2: ");

for (int i = size - 1; i >= top2; i--) {

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

}

System.out.println();

} else {

System.out.println("Stack 2 is empty");

}

}

public boolean isEmpty1() {

return top1 == -1;

}

public boolean isEmpty2() {

return top2 == size;

}

public int top1() {

if (top1 >= 0) {

return arr[top1];

}

return -1;

}

public int top2() {

if (top2 < size) {

return arr[top2];

}

return -1;

}

}

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in)

System.out.print("Enter the size of the array: ");

int size = scanner.nextInt();

TwoStack stacks = new TwoStack(size);

int choice;

do {

System.out.println("\nMenu:");

System.out.println("1. Push to Stack 1");

System.out.println("2. Push to Stack 2");

System.out.println("3. Pop from Stack 1");

System.out.println("4. Pop from Stack 2");

System.out.println("5. Display Stack 1");

System.out.println("6. Display Stack 2");

System.out.println("7. Check if Stack 1 is Empty");

System.out.println("8. Check if Stack 2 is Empty");

System.out.println("9. Top of Stack 1");

System.out.println("10. Top of Stack 2");

System.out.println("11. Exit");

System.out.print("Enter your choice: ");

choice = scanner.nextInt();

switch (choice) {

case 1:

System.out.print("Enter element to push to Stack 1: ");

int elem1 = scanner.nextInt();

stacks.push1(elem1);

break;

case 2:

System.out.print("Enter element to push to Stack 2: ");

int elem2 = scanner.nextInt();

stacks.push2(elem2);

break;

case 3:

System.out.println("Popped from Stack 1: " + stacks.pop1());

break;

case 4:

System.out.println("Popped from Stack 2: " + stacks.pop2());

break;

case 5:

stacks.display1();

break;

case 6:

stacks.display2();

break;

case 7:

System.out.println("Stack 1 is empty: " + stacks.isEmpty1());

break;

case 8:

System.out.println("Stack 2 is empty: " + stacks.isEmpty2());

break;

case 9:

System.out.println("Top of Stack 1: " + stacks.top1());

break;

case 10:

System.out.println("Top of Stack 2: " + stacks.top2());

break;

case 11:

System.out.println("Exiting program.");

break;

default:

System.out.println("Invalid choice, try again.");

}

} while (choice != 11);

}

}

Set20: find triplets where the sum of two numbers is equal to the third numbers if it exists return.

Examples : input -{1,2,5,20,50,19,17,21,7}

Output - 5,2,7.

Input -{1,2,5,20,50,19,17,21,0}

Output - no such triplet exists.

import java.util.Arrays;

import java.util.Scanner;

public class TripletsSum {

public static void findTriplets(int[] arr) {

// Sort the array

Arrays.sort(arr);

// Iterate through each element

for (int i = 0; i < arr.length - 2; i++) {

int left = i + 1;

int right = arr.length - 1;

while (left < right) {

int sum = arr[i] + arr[left];

if (sum == arr[right]) {

// Triplet found

System.out.println("Triplet: " + arr[i] + ", " + arr[left] + ", " + arr[right]);

return; // Return after finding the first triplet

} else if (sum < arr[right]) {

left++;

} else {

right--;

}

}

}

// If no triplet is found

System.out.println("No such triplet exists.");

}

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

System.out.print("Enter the number of elements in the array: ");

int n = scanner.nextInt();

int[] arr = new int[n];

System.out.println("Enter the elements of the array:");

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

arr[i] = scanner.nextInt();

}

findTriplets(arr);

}

}

Set20: Implement DFS without recursion

import java.util.\*;

public class DFSWithoutRecursion {

// Method to perform DFS without recursion

public static void dfsIterative(Map<Integer, List<Integer>> graph, int start) {

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

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

// Push the start node onto the stack

stack.push(start);

while (!stack.isEmpty()) {

// Pop the top element from the stack

int node = stack.pop();

// If the node is not visited, process it

if (!visited.contains(node)) {

visited.add(node);

System.out.print(node + " "); // Print the visited node

}

// Push all the neighbors of the current node onto the stack

// In reverse order to maintain the correct DFS traversal order

for (int neighbor : graph.get(node)) {

if (!visited.contains(neighbor)) {

stack.push(neighbor);

}

}

}

}

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

// Read the number of nodes

System.out.print("Enter the number of nodes: ");

int n = scanner.nextInt();

// Initialize the graph

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

// Read the edges

System.out.println("Enter the edges (enter a negative number to stop): ");

while (true) {

System.out.print("Enter an edge (two space-separated nodes, or -1 to stop): ");

int node1 = scanner.nextInt();

if (node1 == -1) {

break;

}

int node2 = scanner.nextInt();

// Add the edge to the graph (undirected graph)

graph.putIfAbsent(node1, new ArrayList<>());

graph.putIfAbsent(node2, new ArrayList<>());

graph.get(node1).add(node2);

graph.get(node2).add(node1);

}

// Read the starting node for DFS

System.out.print("Enter the starting node for DFS: ");

int start = scanner.nextInt();

// Perform DFS traversal

System.out.println("DFS Traversal (without recursion):");

dfsIterative(graph, start);

}

}

Set21: distrubute candies/

import java.util.Scanner;

public class DistributeCandies {

public static int distributeCandies(int[] ratings) {

int n = ratings.length;

if (n == 0) return 0;

int[] candies = new int[n];

// Initially, give each child 1 candy

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

candies[i] = 1;

}

// First pass (Left to Right)

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

if (ratings[i] > ratings[i - 1]) {

candies[i] = candies[i - 1] + 1;

}

}

// Second pass (Right to Left)

for (int i = n - 2; i >= 0; i--) {

if (ratings[i] > ratings[i + 1]) {

candies[i] = Math.max(candies[i], candies[i + 1] + 1);

}

}

// Sum the candies to get the total number of candies

int totalCandies = 0;

for (int candy : candies) {

totalCandies += candy;

}

return totalCandies;

}

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

// Read number of children

System.out.print("Enter the number of children: ");

int n = scanner.nextInt();

// Read ratings of the children

int[] ratings = new int[n];

System.out.println("Enter the ratings of the children:");

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

ratings[i] = scanner.nextInt();

}

// Calculate and print the minimum number of candies

int result = distributeCandies(ratings);

System.out.println("Minimum number of candies required: " + result);

}

}

Set21: Dfs using adjacent list

import java.util.\*;

public class DFSUsingAdjacencyList {

// Recursive DFS function

public static void dfsRecursive(Map<Integer, List<Integer>> graph, int node, Set<Integer> visited) {

// Mark the node as visited and print it

visited.add(node);

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

// Visit all the unvisited neighbors of the current node

for (int neighbor : graph.get(node)) {

if (!visited.contains(neighbor)) {

dfsRecursive(graph, neighbor, visited);

}

}

}

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

// Input the number of nodes

System.out.print("Enter the number of nodes: ");

int n = scanner.nextInt();

// Create the adjacency list representation of the graph

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

// Read the edges and populate the graph

System.out.println("Enter the edges (two space-separated nodes per edge, or -1 to stop):");

while (true) {

int node1 = scanner.nextInt();

if (node1 == -1) {

break;

}

int node2 = scanner.nextInt();

graph.putIfAbsent(node1, new ArrayList<>());

graph.putIfAbsent(node2, new ArrayList<>());

graph.get(node1).add(node2);

graph.get(node2).add(node1); // Undirected graph (bidirectional edges)

}

// Read the starting node for DFS

System.out.print("Enter the starting node for DFS: ");

int start = scanner.nextInt();

// Set to keep track of visited nodes

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

// Call DFS starting from the given node

System.out.println("DFS Traversal (Recursive):");

dfsRecursive(graph, start, visited);

}

}

Set22: stair case problem using dp

import java.util.Scanner;

public class StaircaseProblem {

public static int countWays(int n) {

// Base cases

if (n == 0) {

return 1; // There's 1 way to stay at the ground.

}

if (n == 1) {

return 1; // There's 1 way to reach the first step.

}

// Initialize dp array where dp[i] will store the number of ways to reach the i-th step

int[] dp = new int[n + 1];

dp[0] = 1; // 1 way to stay at the ground (step 0)

dp[1] = 1; // 1 way to reach the first step

// Fill the dp array using the recurrence relation

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

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

}

// The answer will be the number of ways to reach the nth step

return dp[n];

}

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

// Input the number of steps

System.out.print("Enter the number of steps: ");

int n = scanner.nextInt();

// Call the function to calculate the number of ways to reach the nth step

int result = countWays(n);

// Output the result

System.out.println("Number of ways to reach the top: " + result);

}

}

Set22:  dfs

Set23: longest common subsequence

import java.util.Scanner;

public class LongestCommonSubsequence {

public static int lcs(String str1, String str2) {

int m = str1.length();

int n = str2.length();

// Create a 2D array to store lengths of longest common subsequence.

int[][] dp = new int[m + 1][n + 1];

// Build the dp array

for (int i = 1; i <= m; i++) {

for (int j = 1; j <= n; j++) {

// If characters match, add 1 to the diagonal value (dp[i-1][j-1])

if (str1.charAt(i - 1) == str2.charAt(j - 1)) {

dp[i][j] = dp[i - 1][j - 1] + 1;

} else {

// Otherwise take the maximum from the top or left cell

dp[i][j] = Math.max(dp[i - 1][j], dp[i][j - 1]);

}

}

}

// The bottom-right cell contains the length of LCS

return dp[m][n];

}

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

// Input the two strings

System.out.print("Enter the first string: ");

String str1 = scanner.nextLine();

System.out.print("Enter the second string: ");

String str2 = scanner.nextLine();

// Calculate the length of the longest common subsequence

int result = lcs(str1, str2);

// Output the result

System.out.println("Length of Longest Common Subsequence: " + result);

scanner.close();

}

}

Set23: DFS using recursion

import java.util.\*;

public class GraphDFS {

private Map<Integer, List<Integer>> adjList;

// Constructor to initialize the graph

public GraphDFS(int vertices) {

adjList = new HashMap<>();

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

adjList.put(i, new ArrayList<>());

}

}

// Add an edge to the graph (undirected graph)

public void addEdge(int u, int v) {

adjList.get(u).add(v);

adjList.get(v).add(u); // For undirected graph, add edge both ways

}

// Recursive DFS function

public void dfs(int node, Set<Integer> visited) {

// Mark the current node as visited

visited.add(node);

System.out.print(node + " "); // Process the node (here, print it)

// Recur for all the adjacent vertices

for (int neighbor : adjList.get(node)) {

if (!visited.contains(neighbor)) {

dfs(neighbor, visited);

}

}

}

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

// Input the number of vertices and edges

System.out.print("Enter the number of vertices: ");

int vertices = scanner.nextInt();

GraphDFS graph = new GraphDFS(vertices);

System.out.print("Enter the number of edges: ");

int edges = scanner.nextInt();

// Input edges

System.out.println("Enter the edges (u v) format: ");

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

int u = scanner.nextInt();

int v = scanner.nextInt();

graph.addEdge(u, v);

}

// Initialize the set of visited nodes

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

// Perform DFS starting from node 0 (or any node of choice)

System.out.println("DFS Traversal starting from node 0:");

graph.dfs(0, visited);

}

}

Set24: Find triplets with sum zero

import java.util.\*;

public class TripletsWithSumZero {

public static List<List<Integer>> findTriplets(int[] arr) {

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

// Sort the array

Arrays.sort(arr);

// Loop through the array to fix one element

for (int i = 0; i < arr.length - 2; i++) {

// Skip duplicate elements to avoid duplicate triplets

if (i > 0 && arr[i] == arr[i - 1]) {

continue;

}

// Initialize two pointers

int left = i + 1;

int right = arr.length - 1;

while (left < right) {

int sum = arr[i] + arr[left] + arr[right];

// If we found a triplet with sum zero

if (sum == 0) {

result.add(Arrays.asList(arr[i], arr[left], arr[right]));

// Move the left pointer to skip duplicates

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

left++;

}

// Move the right pointer to skip duplicates

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

right--;

}

// Move both pointers after finding a valid triplet

left++;

right--;

} else if (sum < 0) {

left++; // Increase the sum by moving the left pointer

} else {

right--; // Decrease the sum by moving the right pointer

}

}

}

return result;

}

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

// Input array size

System.out.print("Enter the number of elements in the array: ");

int n = scanner.nextInt();

int[] arr = new int[n];

System.out.println("Enter the elements of the array: ");

// Input array elements

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

arr[i] = scanner.nextInt();

}

// Find triplets with sum zero

List<List<Integer>> triplets = findTriplets(arr);

// Print the result

if (triplets.isEmpty()) {

System.out.println("No triplets found with sum zero.");

} else {

System.out.println("Triplets with sum zero are: ");

for (List<Integer> triplet : triplets) {

System.out.println(triplet);

}

} }}

Set24: bFS using adjacency matrix

import java.util.\*;

public class BFSUsingAdjacencyMatrix {

private int[][] adjMatrix;

private int vertices;

// Constructor to initialize the graph

public BFSUsingAdjacencyMatrix(int vertices) {

this.vertices = vertices;

adjMatrix = new int[vertices][vertices];

}

// Add an edge between two vertices

public void addEdge(int u, int v) {

adjMatrix[u][v] = 1;

adjMatrix[v][u] = 1; // For undirected graph, add the reverse edge as well

}

// BFS traversal function

public void bfs(int start) {

boolean[] visited = new boolean[vertices];

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

// Mark the starting node as visited and enqueue it

visited[start] = true;

queue.add(start);

System.out.println("BFS Traversal starting from node " + start + ":");

// Start the BFS loop

while (!queue.isEmpty()) {

int node = queue.poll(); // Dequeue the front element

System.out.print(node + " "); // Process the node (print it)

// Visit all the adjacent nodes of the current node

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

if (adjMatrix[node][i] == 1 && !visited[i]) {

visited[i] = true; // Mark the neighbor as visited

queue.add(i); // Enqueue the unvisited neighbor

}

}

}

}

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

// Input the number of vertices

System.out.print("Enter the number of vertices: ");

int vertices = scanner.nextInt();

BFSUsingAdjacencyMatrix graph = new BFSUsingAdjacencyMatrix(vertices);

// Input the number of edges

System.out.print("Enter the number of edges: ");

int edges = scanner.nextInt();

// Input the edges (u, v) format

System.out.println("Enter the edges (u v) format: ");

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

int u = scanner.nextInt();

int v = scanner.nextInt();

graph.addEdge(u, v);

}

// Input the starting node for BFS

System.out.print("Enter the starting node for BFS: ");

int startNode = scanner.nextInt();

graph.bfs(startNode);

}

}

Set25: Minimum platforms for railway

import java.util.\*;

public class MinimumPlatforms {

public static int findMinimumPlatforms(int[] arrival, int[] departure) {

// Sort arrival and departure arrays

Arrays.sort(arrival);

Arrays.sort(departure);

int n = arrival.length;

int platformsNeeded = 1; // At least one platform is always needed

int result = 1;

// Initialize pointers

int i = 1; // Pointer for arrival

int j = 0; // Pointer for departure

// Traverse through all the events

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

// If the next event is an arrival

if (arrival[i] <= departure[j]) {

platformsNeeded++;

i++;

}

// If the next event is a departure

else {

platformsNeeded--;

j++;

}

// Update the result to store the maximum number of platforms needed

result = Math.max(result, platformsNeeded);

}

return result;

}

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

// Input the number of trains

System.out.print("Enter the number of trains: ");

int n = scanner.nextInt();

int[] arrival = new int[n];

int[] departure = new int[n];

// Input the arrival and departure times for each train

System.out.println("Enter arrival times: ");

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

arrival[i] = scanner.nextInt();

}

System.out.println("Enter departure times: ");

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

departure[i] = scanner.nextInt();

}

int result = findMinimumPlatforms(arrival, departure);

System.out.println("Minimum platforms required: " + result);

}}

Set25: Bfs in adjacent list

import java.util.\*;

public class BFS {

// Graph represented as an adjacency list

private Map<Integer, List<Integer>> adjList;

// Constructor to initialize the graph

public BFS(int vertices) {

adjList = new HashMap<>();

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

adjList.put(i, new ArrayList<>());

}

}

// Add an edge to the graph (undirected)

public void addEdge(int src, int dest) {

adjList.get(src).add(dest);

adjList.get(dest).add(src);

}

// Perform BFS starting from a source node

public void bfs(int start) {

// Create a queue for BFS

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

// Create a set to track visited nodes

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

// Enqueue the start node and mark it as visited

queue.add(start);

visited.add(start);

// Loop while the queue is not empty

while (!queue.isEmpty()) {

// Dequeue a vertex from the queue

int node = queue.poll();

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

// Explore all the adjacent vertices of the dequeued node

for (int neighbor : adjList.get(node)) {

if (!visited.contains(neighbor)) {

// If not visited, mark as visited and enqueue

visited.add(neighbor);

queue.add(neighbor);

}

}

}

}

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

// Input number of vertices

System.out.print("Enter the number of vertices: ");

int vertices = scanner.nextInt();

// Create a BFS object

BFS graph = new BFS(vertices);

// Input number of edges

System.out.print("Enter the number of edges: ");

int edges = scanner.nextInt();

// Input edges

System.out.println("Enter the edges (source destination):");

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

int src = scanner.nextInt();

int dest = scanner.nextInt();

graph.addEdge(src, dest);

}

// Perform BFS starting from vertex 0 (or any other vertex you want)

System.out.print("BFS starting from vertex 0: ");

graph.bfs(0);

}

}

Set26: )given 2 string print them alternatively

import java.util.Scanner;

public class AlternateStringPrint {

public static void printAlternately(String str1, String str2) {

int length1 = str1.length();

int length2 = str2.length();

int i = 0, j = 0;

// Print characters from both strings alternately

while (i < length1 && j < length2) {

System.out.print(str1.charAt(i));

System.out.print(str2.charAt(j));

i++;

j++;

}

// If any characters are left in str1, print them

while (i < length1) {

System.out.print(str1.charAt(i));

i++;

}

// If any characters are left in str2, print them

while (j < length2) {

System.out.print(str2.charAt(j));

j++;

}

}

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

// Input two strings

System.out.print("Enter first string: ");

String str1 = scanner.nextLine();

System.out.print("Enter second string: ");

String str2 = scanner.nextLine();

// Print characters alternatively

System.out.print("Alternating output: ");

printAlternately(str1, str2);

}

}

Set26: balnace parenthesis ex:({) op:false ex2 :({}) op:true

import java.util.Stack;

import java.util.Scanner;

public class BalancedParentheses {

public static boolean isBalanced(String str) {

// Stack to store opening parentheses

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

// Iterate through the string

for (char ch : str.toCharArray()) {

// If the character is an opening parenthesis, push it to the stack

if (ch == '(' || ch == '{' || ch == '[') {

stack.push(ch);

}

// If the character is a closing parenthesis, check for balance

else if (ch == ')' || ch == '}' || ch == ']') {

// If stack is empty, or top of stack doesn't match the closing parenthesis, return false

if (stack.isEmpty()) {

return false;

}

char top = stack.pop();

if ((ch == ')' && top != '(') || (ch == '}' && top != '{') || (ch == ']' && top != '[')) {

return false;

}

}

}

// If the stack is empty, all parentheses are matched

return stack.isEmpty();

}

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

// Input string of parentheses

System.out.print("Enter a string with parentheses: ");

String str = scanner.nextLine();

// Check if the parentheses are balanced

if (isBalanced(str)) {

System.out.println("Output: true");

} else {

System.out.println("Output: false");

}

}

}

Set27: 4-Sum with target x

import java.util.\*;

public class FourSum {

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

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

// Sort the array to make it easier to avoid duplicates and use two-pointer technique

Arrays.sort(nums);

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

// Skip the duplicate numbers for the first index

if (i > 0 && nums[i] == nums[i - 1]) {

continue;

}

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

// Skip the duplicate numbers for the second index

if (j > i + 1 && nums[j] == nums[j - 1]) {

continue;

}

int left = j + 1;

int right = nums.length - 1;

while (left < right) {

int sum = nums[i] + nums[j] + nums[left] + nums[right];

if (sum == target) {

result.add(Arrays.asList(nums[i], nums[j], nums[left], nums[right]));

// Skip duplicates for the third and fourth indices

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

left++;

}

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

right--;

}

left++;

right--;

} else if (sum < target) {

left++;

} else {

right--;

}

}

}

}

return result;

}

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

// Input array and target value

System.out.print("Enter the number of elements in the array: ");

int n = scanner.nextInt();

int[] nums = new int[n];

System.out.println("Enter the elements of the array:");

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

nums[i] = scanner.nextInt();

}

System.out.print("Enter the target sum: ");

int target = scanner.nextInt();

// Get the list of 4-sum quadruplets

List<List<Integer>> result = fourSum(nums, target);

// Output the result

System.out.println("Quadruplets that sum up to " + target + ":");

if (result.isEmpty()) {

System.out.println("No quadruplets found.");

} else {

for (List<Integer> quadruplet : result) {

System.out.println(quadruplet);

}

}

}

}

Set27: delete middle element of stack

import java.util.\*;

public class StackMiddleElement {

// Recursive method to delete the middle element

public static void deleteMiddle(Stack<Integer> stack, int size, int currentIndex) {

// Base case: If we have reached the middle element

if (currentIndex == size / 2) {

stack.pop(); // Remove the middle element

return;

}

// Pop the top element and recurse

int topElement = stack.pop();

// Recurse to the next element

deleteMiddle(stack, size, currentIndex + 1);

// Push the element back to the stack

stack.push(topElement);

}

// Method to remove the middle element from the stack

public static void removeMiddle(Stack<Integer> stack) {

int size = stack.size();

// If the stack is empty, return as there's nothing to delete

if (size == 0) {

System.out.println("Stack is empty!");

return;

}

// Call the recursive function with the stack, its size, and the current index (starting from 0)

deleteMiddle(stack, size, 0);

}

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

// Input: Ask the user for the number of elements in the stack

System.out.print("Enter the number of elements in the stack: ");

int n = scanner.nextInt();

// Create a new stack

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

// Input: Get the elements of the stack from the user

System.out.println("Enter the elements of the stack:");

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

stack.push(scanner.nextInt());

}

// Display the original stack

System.out.println("Original Stack: " + stack);

// Remove the middle element

removeMiddle(stack);

// Display the stack after removing the middle element

System.out.println("Stack after removing middle element: " + stack);

}

}

Set29: Find unique paths in a grid where it has some obstacles

import java.util.Scanner;

public class UniquePathsWithObstacles {

public static int uniquePathsWithObstacles(int[][] obstacleGrid) {

int m = obstacleGrid.length;

int n = obstacleGrid[0].length;

// Create a DP table to store the number of ways to reach each cell

int[][] dp = new int[m][n];

// If the start cell has an obstacle, return 0 (no paths)

if (obstacleGrid[0][0] == 1) {

return 0;

}

// Set the starting point

dp[0][0] = 1;

// Fill the first row (can only come from the left)

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

if (obstacleGrid[0][i] == 0) {

dp[0][i] = dp[0][i - 1];

}

}

// Fill the first column (can only come from the top)

for (int i = 1; i < m; i++) {

if (obstacleGrid[i][0] == 0) {

dp[i][0] = dp[i - 1][0];

}

}

// Fill the rest of the DP table

for (int i = 1; i < m; i++) {

for (int j = 1; j < n; j++) {

if (obstacleGrid[i][j] == 0) {

dp[i][j] = dp[i - 1][j] + dp[i][j - 1];

} else {

dp[i][j] = 0; // If there is an obstacle, set the number of paths to 0

}

}

}

// The answer is the value in the bottom-right corner

return dp[m - 1][n - 1];

}

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

// Input grid dimensions

System.out.print("Enter the number of rows: ");

int m = scanner.nextInt();

System.out.print("Enter the number of columns: ");

int n = scanner.nextInt();

// Input the grid with obstacles

int[][] grid = new int[m][n];

System.out.println("Enter the grid (0 for empty, 1 for obstacle):");

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

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

grid[i][j] = scanner.nextInt();

}

}

// Find the unique paths avoiding obstacles

int result = uniquePathsWithObstacles(grid);

// Output the result

System.out.println("Number of unique paths: " + result);

}

}

Set29: Reverse queue content using stack

import java.util.\*;

public class ReverseQueueUsingStack {

// Method to reverse the queue using a stack

public static void reverseQueue(Queue<Integer> queue) {

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

// Push all elements of the queue into the stack

while (!queue.isEmpty()) {

stack.push(queue.poll());

}

// Pop all elements from the stack and enqueue them back into the queue

while (!stack.isEmpty()) {

queue.offer(stack.pop());

}

}

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

// Input: Get the number of elements in the queue

System.out.print("Enter the number of elements in the queue: ");

int n = scanner.nextInt();

// Create a queue

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

// Input: Add elements to the queue

System.out.println("Enter the elements of the queue:");

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

queue.offer(scanner.nextInt());

}

// Display the original queue

System.out.println("Original Queue: " + queue);

// Reverse the queue using stack

reverseQueue(queue);

// Display the reversed queue

System.out.println("Reversed Queue: " + queue);

}

}

Set30: Remove duplicates from a sorted array

import java.util.Scanner;

public class RemoveDuplicatesFromSortedArray {

public static int removeDuplicates(int[] arr) {

if (arr.length == 0) {

return 0; // No elements to remove

}

// i points to the last unique element in the array

int i = 0;

// Traverse the array starting from the second element

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

// If current element is not the same as the last unique element

if (arr[j] != arr[i]) {

i++; // Move the unique pointer forward

arr[i] = arr[j]; // Update the element at the unique pointer

}

}

// The unique elements will be in the first i+1 positions

return i + 1; // Return the number of unique elements

}

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

// Input: Get the number of elements in the array

System.out.print("Enter the number of elements in the array: ");

int n = scanner.nextInt();

// Create the array and input its elements

int[] arr = new int[n];

System.out.println("Enter the elements of the sorted array:");

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

arr[i] = scanner.nextInt();

}

// Call the removeDuplicates method and get the number of unique elements

int length = removeDuplicates(arr);

// Output: Display the unique elements in the array

System.out.println("Array after removing duplicates:");

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

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

}

}

}

Set 30: Find the minimum element in the stack

import java.util.Scanner;

import java.util.Stack;

public class MinStack {

private Stack<Integer> stack;

private Stack<Integer> minStack;

public MinStack() {

stack = new Stack<>();

minStack = new Stack<>();

}

// Push element onto the stack

public void push(int value) {

stack.push(value);

// If minStack is empty or the value is smaller than or equal to the top of minStack, push it onto minStack

if (minStack.isEmpty() || value <= minStack.peek()) {

minStack.push(value);

}

}

// Pop element from the stack

public void pop() {

if (!stack.isEmpty()) {

int poppedValue = stack.pop();

if (poppedValue == minStack.peek()) {

minStack.pop();

}

}

}

// Get the minimum element in the stack

public int getMin() {

if (!minStack.isEmpty()) {

return minStack.peek();

}

return Integer.MAX\_VALUE; // Return a large value if the stack is empty

}

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

MinStack minStack = new MinStack();

// Menu-driven program for stack operations

int choice;

do {

System.out.println("\n1. Push");

System.out.println("2. Pop");

System.out.println("3. Get Minimum");

System.out.println("4. Exit");

System.out.print("Enter your choice: ");

choice = scanner.nextInt();

switch (choice) {

case 1:

System.out.print("Enter element to push: ");

int value = scanner.nextInt();

minStack.push(value);

System.out.println(value + " pushed to stack.");

break;

case 2:

minStack.pop();

System.out.println("Top element popped.");

break;

case 3:

int min = minStack.getMin();

if (min == Integer.MAX\_VALUE) {

System.out.println("Stack is empty.");

} else {

System.out.println("Minimum element in the stack: " + min);

}

break;

case 4:

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

break;

default:

System.out.println("Invalid choice. Try again.");

}

} while (choice != 4);

}

}

Set 31: Find the missing term in logarithmic sequence

import java.util.Scanner;

public class LogarithmicSequence {

// Method to find the missing term in a logarithmic sequence

public static double findMissingTerm(double[] arr) {

// Find the common difference or ratio based on the progression type

// Check if the sequence follows an arithmetic progression in logarithmic terms

double diff = arr[1] - arr[0]; // Find the difference between first two terms

// Check if the next terms follow the same difference (for arithmetic progression)

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

if (arr[i + 1] - arr[i] != diff) {

// If difference is not constant, it might not be an AP.

System.out.println("The sequence is not an arithmetic progression.");

return -1;

}

}

// Calculate and return the missing term based on the common difference

return arr[arr.length - 1] + diff;

}

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

System.out.print("Enter the number of elements in the sequence (including the missing term): ");

int n = scanner.nextInt();

double[] arr = new double[n];

System.out.println("Enter the logarithmic sequence (leave the missing value as -1): ");

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

arr[i] = scanner.nextDouble();

}

double missingTerm = findMissingTerm(arr);

if (missingTerm != -1) {

System.out.println("The missing term is: " + missingTerm);

}

}

}

Set31: Find the minimum element in stack with O(1) time and O(1) space

import java.util.Scanner;

import java.util.Stack;

public class MinStack {

private Stack<Integer> stack;

private int min;

public MinStack() {

stack = new Stack<>();

min = Integer.MAX\_VALUE; // Initialize the minimum to a very large number

}

// Push element onto the stack

public void push(int value) {

if (stack.isEmpty()) {

stack.push(value);

min = value; // Set the first element as the minimum

} else {

if (value < min) {

stack.push(2 \* value - min); // Store the difference between value and min

min = value; // Update the minimum

} else {

stack.push(value); // Push the value as is

}

}

}

// Pop element from the stack

public void pop() {

if (stack.isEmpty()) {

return; // Stack is empty, nothing to pop

}

int top = stack.pop();

if (top < min) {

min = 2 \* min - top; // Update minimum when a "special" value is popped

}

}

// Get the minimum element in the stack

public int getMin() {

return min;

}

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

MinStack minStack = new MinStack();

// Menu-driven program for stack operations

int choice;

do {

System.out.println("\n1. Push");

System.out.println("2. Pop");

System.out.println("3. Get Minimum");

System.out.println("4. Exit");

System.out.print("Enter your choice: ");

choice = scanner.nextInt();

switch (choice) {

case 1:

System.out.print("Enter element to push: ");

int value = scanner.nextInt();

minStack.push(value);

System.out.println(value + " pushed to stack.");

break;

case 2:

minStack.pop();

System.out.println("Top element popped.");

break;

case 3:

System.out.println("Minimum element in the stack: " + minStack.getMin());

break;

case 4:

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

break;

default:

System.out.println("Invalid choice. Try again.");

}

} while (choice != 4);

}

}

Set34: Ternary search

import java.util.Scanner;

public class TernarySearch {

// Ternary search function

public static int ternarySearch(int[] arr, int left, int right, int target) {

// While there are at least two elements to search between

while (left <= right) {

// Divide the range into three parts

int mid1 = left + (right - left) / 3;

int mid2 = right - (right - left) / 3;

// Compare target with the mid points

if (arr[mid1] == target) {

return mid1; // Target found at mid1

}

if (arr[mid2] == target) {

return mid2; // Target found at mid2

}

// Narrow the search range based on comparisons

if (target < arr[mid1]) {

right = mid1 - 1; // Target lies in the left third

} else if (target > arr[mid2]) {

left = mid2 + 1; // Target lies in the right third

} else {

left = mid1 + 1; // Target lies in the middle third

right = mid2 - 1;

}

}

// If target is not found, return -1

return -1;

}

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

// Input size of the array

System.out.print("Enter number of elements in the array: ");

int n = scanner.nextInt();

// Input the array elements

int[] arr = new int[n];

System.out.println("Enter elements of the array (sorted in ascending order): ");

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

arr[i] = scanner.nextInt();

}

// Input the target element to search for

System.out.print("Enter the target element to search: ");

int target = scanner.nextInt();

// Perform ternary search

int result = ternarySearch(arr, 0, n - 1, target);

// Output the result

if (result == -1) {

System.out.println("Element not found in the array.");

} else {

System.out.println("Element found at index: " + result);

}

}}

set34: palindrone -linked list

import java.util.Scanner;

class ListNode {

int val;

ListNode next;

ListNode(int x) {

val = x;

next = null;

}

}

public class LinkedListPalindrome {

// Function to check if the linked list is a palindrome

public static boolean isPalindrome(ListNode head) {

if (head == null || head.next == null) {

return true; // Empty or single element list is always a palindrome

}

// Step 1: Find the middle of the linked list using slow and fast pointers

ListNode slow = head;

ListNode fast = head;

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

slow = slow.next;

fast = fast.next.next;

}

// Step 2: Reverse the second half of the list

ListNode secondHalf = reverseList(slow);

// Step 3: Compare the first half and the reversed second half

ListNode firstHalf = head;

while (secondHalf != null) {

if (firstHalf.val != secondHalf.val) {

return false; // Not a palindrome if values don't match

}

firstHalf = firstHalf.next;

secondHalf = secondHalf.next;

}

return true; // Palindrome

}

// Helper function to reverse a linked list

public static ListNode reverseList(ListNode head) {

ListNode prev = null;

ListNode current = head;

ListNode next = null;

while (current != null) {

next = current.next;

current.next = prev;

prev = current;

current = next;

}

return prev; // New head of the reversed list

}

// Main method to run the program

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

// Input size of the linked list

System.out.print("Enter the number of elements in the linked list: ");

int n = scanner.nextInt();

// Input the elements of the linked list

System.out.println("Enter the elements of the linked list: ");

ListNode head = null;

ListNode tail = null;

// Create the linked list from the input

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

int value = scanner.nextInt();

ListNode newNode = new ListNode(value);

if (head == null) {

head = newNode;

tail = head;

} else {

tail.next = newNode;

tail = newNode;

}

}

// Perform palindrome check

if (isPalindrome(head)) {

System.out.println("The linked list is a palindrome.");

} else {

System.out.println("The linked list is not a palindrome.");

}

}

}

Set35: sort the elemnts of a given sorted array after squaring the elements

import java.util.Scanner;

public class SortedSquares {

public static int[] sortedSquares(int[] nums) {

int n = nums.length;

int[] result = new int[n];

// Initialize two pointers

int left = 0, right = n - 1;

// Start filling the result array from the back

int index = n - 1;

while (left <= right) {

int leftSquare = nums[left] \* nums[left];

int rightSquare = nums[right] \* nums[right];

// Compare squares and place the larger one in the result array

if (leftSquare > rightSquare) {

result[index--] = leftSquare;

left++;

} else {

result[index--] = rightSquare;

right--;

}

}

return result;

}

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

// Input size of the array

System.out.print("Enter the number of elements in the array: ");

int n = scanner.nextInt();

// Input the elements of the array

System.out.println("Enter the elements of the sorted array: ");

int[] nums = new int[n];

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

nums[i] = scanner.nextInt();

}

// Get the sorted squared array

int[] result = sortedSquares(nums);

// Print the result

System.out.println("Sorted array after squaring the elements: ");

for (int num : result) {

System.out.print(num + " ");

}

}

}

Set35: remove n th element from  linked list

import java.util.Scanner;

class ListNode {

int val;

ListNode next;

ListNode(int x) {

val = x;

next = null;

}

}

public class RemoveNthNode {

public static ListNode removeNthFromEnd(ListNode head, int n) {

if (head == null) return null;

// Create a dummy node to handle edge cases such as removing the head

ListNode dummy = new ListNode(0);

dummy.next = head;

// Initialize two pointers

ListNode first = dummy;

ListNode second = dummy;

// Move first pointer n+1 steps ahead so that the gap between first and second is n

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

first = first.next;

}

// Move first to the end, maintaining the gap

while (first != null) {

first = first.next;

second = second.next;

}

// Remove the n-th node from the end

second.next = second.next.next;

// Return the new head

return dummy.next;

}

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

// Input size of the linked list

System.out.print("Enter the number of elements in the linked list: ");

int n = scanner.nextInt();

// Input the elements of the linked list

System.out.println("Enter the elements of the linked list: ");

ListNode head = null;

ListNode tail = null;

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

int value = scanner.nextInt();

ListNode newNode = new ListNode(value);

if (head == null) {

head = newNode;

tail = head;

} else {

tail.next = newNode;

tail = newNode;

}

}

// Input n for the n-th element to be removed

System.out.print("Enter the value of n (n-th node from the end to be removed): ");

int nFromEnd = scanner.nextInt();

// Remove the n-th node from the end

head = removeNthFromEnd(head, nFromEnd);

// Output the modified linked list

System.out.println("Modified linked list: ");

ListNode temp = head;

while (temp != null) {

System.out.print(temp.val + " ");

temp = temp.next;

}

}

}

Set39: Given an array of N integers, find the length of the longest subsequence of a given sequence such that all elements of the subsequence are sorted in strictly decreasing order.

Examples:

Input: arr[] = [15, 27, 14, 38, 63, 55, 46, 65, 85]

Output: 3

Explanation: The longest decreasing subsequence is {63, 55, 46}

Input: arr[] = {50, 3, 10, 7, 40, 80}

Output: 3

Explanation: The longest decreasing subsequence is {50, 10, 7}

import java.util.Scanner;

public class LongestDecreasingSubsequence {

public static int longestDecreasingSubsequence(int[] arr) {

int n = arr.length;

// Initialize the DP array where dp[i] is the length of the longest decreasing subsequence ending at i

int[] dp = new int[n];

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

dp[i] = 1; // Each element can be a subsequence of length 1

}

// Fill the DP array using the transition

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

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

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

dp[i] = Math.max(dp[i], dp[j] + 1);

}

}

}

// The length of the longest decreasing subsequence will be the maximum value in the dp array

int maxLength = 0;

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

maxLength = Math.max(maxLength, dp[i]);

}

return maxLength;

}

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

// Input the number of elements in the array

System.out.print("Enter the number of elements in the array: ");

int n = scanner.nextInt();

// Input the elements of the array

System.out.println("Enter the elements of the array: ");

int[] arr = new int[n];

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

arr[i] = scanner.nextInt();

}

// Find and print the length of the longest decreasing subsequence

int result = longestDecreasingSubsequence(arr);

System.out.println("The length of the longest decreasing subsequence is: " + result);

}

}

Set39: B)Given an array of strings arr[], the task is to sort the array of strings according to the frequency of each string, in ascending order. If two elements have the same frequency, then they are sorted into lexicographical order.

Examples:

Input: arr[] = {“Geeks”, “for”, “Geeks”}

Output: {“for”, “Geeks”}

Explanation:

As the string “Geeks” has a frequency of 2 in the given array,

Hence, the position of the string “Geeks” will be 2

Input: arr[] = {“abc”, “pqr”, “pqr”, “abc”}

Output: {“abc”, “pqr”}

Explanation:

As both the strings have the same frequency, the array is sorted in the lexicographical order.

import java.util.\*;

public class SortStringsByFrequency {

public static String[] sortByFrequency(String[] arr) {

// Create a HashMap to store the frequency of each string

Map<String, Integer> frequencyMap = new HashMap<>();

// Count the frequency of each string

for (String str : arr) {

frequencyMap.put(str, frequencyMap.getOrDefault(str, 0) + 1);

}

// Create a list of the strings to sort

List<String> strings = new ArrayList<>(frequencyMap.keySet());

// Sort the list using a custom comparator

Collections.sort(strings, new Comparator<String>() {

public int compare(String s1, String s2) {

// First compare by frequency (ascending order)

int freqCompare = frequencyMap.get(s1) - frequencyMap.get(s2);

// If frequencies are the same, compare lexicographically (alphabetical order)

if (freqCompare == 0) {

return s1.compareTo(s2);

}

return freqCompare;

}

});

// Convert the list back to an array

return strings.toArray(new String[0]);

}

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

// Input the number of strings

System.out.print("Enter the number of strings: ");

int n = scanner.nextInt();

scanner.nextLine(); // Consume newline

// Input the strings

System.out.println("Enter the strings: ");

String[] arr = new String[n];

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

arr[i] = scanner.nextLine();

}

// Call the function to sort the strings by frequency

String[] result = sortByFrequency(arr);

// Output the sorted array of strings

System.out.println("Sorted array by frequency and lexicographical order: ");

for (String str : result) {

System.out.print(str + " ");

}

}

}

Set41: best day to sell stocks

import java.util.Scanner;

public class BestDayToSellStocks {

public static int findMaxProfit(int[] prices) {

if (prices.length < 2) {

return 0; // No profit possible if there are fewer than 2 days

}

int minPrice = prices[0]; // Set the first price as the minimum price

int maxProfit = 0; // Initialize max profit to 0

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

// Calculate potential profit if sold on day i

int profit = prices[i] - minPrice;

// Update max profit if the current profit is higher

if (profit > maxProfit) {

maxProfit = profit;

}

// Update the minimum price encountered so far

if (prices[i] < minPrice) {

minPrice = prices[i];

}

}

return maxProfit; // Return the maximum profit

}

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

// Input: number of days (length of prices array)

System.out.print("Enter the number of days: ");

int n = scanner.nextInt();

// Input: stock prices for each day

int[] prices = new int[n];

System.out.println("Enter the stock prices for each day: ");

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

prices[i] = scanner.nextInt();

}

// Find and print the maximum profit

int maxProfit = findMaxProfit(prices);

System.out.println("Maximum Profit: " + maxProfit);

}

}

Set41: number of half nodes in a bst

import java.util.Scanner;

class Node {

int data;

Node left, right;

public Node(int data) {

this.data = data;

left = right = null;

}

}

public class BST {

Node root;

// Function to insert a new node with given data

public Node insert(Node root, int data) {

if (root == null) {

return new Node(data);

}

if (data < root.data) {

root.left = insert(root.left, data);

} else if (data > root.data) {

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

}

return root;

}

// Function to count the number of half nodes in the BST

public int countHalfNodes(Node root) {

if (root == null) {

return 0;

}

int count = 0;

// Check if the current node has exactly one child

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

count = 1;

}

// Recursively count in the left and right subtrees

count += countHalfNodes(root.left);

count += countHalfNodes(root.right);

return count;

}

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

BST bst = new BST();

bst.root = null;

System.out.print("Enter the number of nodes in the BST: ");

int n = scanner.nextInt();

System.out.println("Enter the elements of the BST: ");

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

int data = scanner.nextInt();

bst.root = bst.insert(bst.root, data);

}

// Count the number of half nodes

int halfNodes = bst.countHalfNodes(bst.root);

System.out.println("Number of half nodes in the BST: " + halfNodes);

}

}