# Set 1a: Queue using 2 Stacks [LeetCode #232]

import java.util.Scanner;

import java.util.Stack;

class DynamicQueueUsingStacks {

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

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

public void enqueue(int data) {

stack1.push(data);

System.out.println("Enqueued: " + data);

}

public int dequeue() {

if (isEmpty()) {

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

return -1;

}

if (stack2.isEmpty()) {

while (!stack1.isEmpty()) {

stack2.push(stack1.pop());

}

}

return stack2.pop();

}

public boolean isEmpty() {

return stack1.isEmpty() && stack2.isEmpty();

}

public void display() {

if (isEmpty()) {

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

return;

}

if (stack2.isEmpty()) {

while (!stack1.isEmpty()) {

stack2.push(stack1.pop());

}

}

System.out.println("Queue contents (front to rear):");

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

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

}

System.out.println();

}

public static void main(String[] args) {

DynamicQueueUsingStacks queue = new DynamicQueueUsingStacks();

Scanner sc = new Scanner(System.in);

int choice, value;

do {

System.out.println("\n1. Enqueue\n2. Dequeue\n3. Display\n4. Exit");

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

choice = sc.nextInt();

switch (choice) {

case 1:

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

value = sc.nextInt();

queue.enqueue(value);

break;

case 2:

int removed = queue.dequeue();

if (removed != -1) {

System.out.println("Dequeued: " + removed);

}

break;

case 3:

queue.display();

break;

case 4:

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

break;

default:

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

}

} while (choice != 4);

sc.close();

}

}

# Set 1b: Minimum element in rotated sorted array [LeetCode #153]

class MinInRotatedSortedArray {  
 public static int findMin(int[] nums) {  
 int low = 0, high = nums.length - 1;  
 while (low < high) {  
 int mid = low + (high - low) / 2;  
 if (nums[mid] > nums[high]) {  
 low = mid + 1;  
 } else {  
 high = mid;  
 }  
 }  
 return nums[low];  
 }  
  
 public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

System.out.println("Enter the size of 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.println("Minimum element is: " + findMin(nums));

}

}

Here are the solutions to the problems:

2) a) Pairs with larger element

import java.util.Arrays;

import java.util.Scanner;

public class LargerPairs {

public static void findPairs(int[] arr) {

Arrays.sort(arr);

int count = 0;

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

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

System.out.println("(" + arr[i] + "," + arr[j] + ")");

count++;

}

}

System.out.println("Total pairs: " + count);

}

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

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

}

findPairs(arr);

}

}

B) Stack using 2 queues

import java.util.LinkedList;

import java.util.Queue;

import java.util.Scanner;

class Program2b{

Queue<Integer> q1 = new LinkedList<>();

Queue<Integer> q2 = new LinkedList<>();

public void push(int x) {

q2.add(x);

while (!q1.isEmpty()) {

q2.add(q1.remove());

}

Queue<Integer> temp = q1;

q1 = q2;

q2 = temp;

System.out.println("Pushed: " + x);

}

public int pop() {

if (q1.isEmpty()) {

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

return -1;

}

return q1.remove();

}

public int top() {

if (q1.isEmpty()) {

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

return -1;

}

return q1.peek();

}

public boolean isEmpty() {

return q1.isEmpty();

}

public void display() {

if (q1.isEmpty()) {

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

return;

}

System.out.println("Stack contents (top to bottom): " + q1);

}

public static void main(String[] args) {

StackUsingQueues stack = new StackUsingQueues();

Scanner sc = new Scanner(System.in);

int choice, value;

do {

System.out.println("\n1. Push\n2. Pop\n3. Top\n4. Display\n5. Exit");

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

choice = sc.nextInt();

switch (choice) {

case 1:

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

value = sc.nextInt();

stack.push(value);

break;

case 2:

int popped = stack.pop();

if (popped != -1)

System.out.println("Popped: " + popped);

break;

case 3:

int top = stack.top();

if (top != -1)

System.out.println("Top element: " + top);

break;

case 4:

stack.display();

break;

case 5:

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

break;

default:

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

}

} while (choice != 5);

sc.close();

}

}

# Set 3a: Remove duplicates in a linked list [LeetCode #83]

import java.util.HashSet;

import java.util.Scanner;

class Node {

int data;

Node next;

Node(int data) {

this.data = data;

next = null;

}

}

public class Program3a {

public static Node removeDuplicates(Node head) {

HashSet<Integer> seen = new HashSet<>();

Node current = head, prev = null;

while (current != null) {

if (seen.contains(current.data)) {

prev.next = current.next;

} else {

seen.add(current.data);

prev = current;

}

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 sc = new Scanner(System.in);

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

int n = sc.nextInt();

if (n == 0) {

System.out.println("Empty list.");

return;

}

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

Node head = new Node(sc.nextInt());

Node current = head;

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

current.next = new Node(sc.nextInt());

current = current.next;

}

System.out.print("Original List: ");

printList(head);

head = removeDuplicates(head);

System.out.print("List after removing duplicates: ");

printList(head);

}

}

# Set 3b: Return index number of target value in a rotated sorted array [LeetCode #33]

class SearchInRotatedSortedArray {  
 public static int search(int[] nums, int target) {  
 int left = 0, right = nums.length - 1;  
 while (left <= right) {  
 int mid = left + (right - left) / 2;  
 if (nums[mid] == target) return mid;  
  
 if (nums[left] <= nums[mid]) {  
 if (nums[left] <= target && target < nums[mid]) {  
 right = mid - 1;  
 } else {  
 left = mid + 1;  
 }  
 } else {  
 if (nums[mid] < target && target <= nums[right]) {  
 left = mid + 1;  
 } else {  
 right = mid - 1;  
 }  
 }  
 }  
 return -1;  
 }  
  
 public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

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

int target = scanner.nextInt();

int result = search(nums, target);

if (result != -1) {

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

} else {

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

}

}  
}  
===============================================================================

4) Linked list delete nth node from end

4a)

import java.util.Scanner;

class Node {

int data;

Node next;

Node(int data) {

this.data = data;

next = null;

}

}

public class Program4a{

public static Node deleteNthFromEnd(Node head, int n) {

if (head == null) return null;

Node first = head;

Node second = head;

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

if (first == null) return head;

first = first.next;

}

if (first == null) {

head = head.next;

return head;

}

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

first = first.next;

second = second.next;

}

second.next = second.next.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 sc = new Scanner(System.in);

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

int n = sc.nextInt();

if (n == 0) {

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

return;

}

System.out.print("Enter the elements of the LinkedList: ");

Node head = new Node(sc.nextInt());

Node current = head;

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

current.next = new Node(sc.nextInt());

current = current.next;

}

System.out.print("Enter the position from the end to delete: ");

int positionFromEnd = sc.nextInt();

System.out.print("Original List: ");

printList(head);

head = deleteNthFromEnd(head, positionFromEnd);

System.out.print("List after deleting " + positionFromEnd + "th node from the end: ");

printList(head);

}

}

4b)

import java.util.Arrays;

import java.util.Scanner;

public class Program4b {

public static float findCeil(float[] arr, float x) {

Arrays.sort(arr);

for (float num : arr) {

if (num >= x) {

return num;

}

}

return -1;

}

public static float findFloor(float[] arr, float x) {

Arrays.sort(arr);

float floor = -1;

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

if (arr[i] <= x) {

floor = arr[i];

break;

}

}

return floor;

}

public static void main(String[] args) {

Scanner sc = new Scanner(System.in);

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

int n = sc.nextInt();

float[] arr = new float[n];

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

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

arr[i] = sc.nextFloat();

}

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

float x = sc.nextFloat();

float ceil = findCeil(arr, x);

float floor = findFloor(arr, x);

if (ceil != -1) {

System.out.println("Ceil of " + x + " is: " + ceil);

} else {

System.out.println("No Ceil found for " + x);

}

if (floor != -1) {

System.out.println("Floor of " + x + " is: " + floor);

} else {

System.out.println("No Floor found for " + x);

}

}

}

# Set 5a: In sorted array, find element that occurs only once [LeetCode #540]

5a)

import java.util.Scanner;

public class Program5a{

public static int findUnique(int[] arr) {

int n = arr.length;

for (int i = 0; i < n - 1; i += 2) {

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

return arr[i];

}

}

return (n > 0) ? arr[n - 1] : 2;

}

public static void main(String[] args) {

Scanner sc = new Scanner(System.in);

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

int n = sc.nextInt();

int[] arr = new int[n];

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

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

arr[i] = sc.nextInt();

}

int result = findUnique(arr);

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

}

}

# Set 5b: Reverse a single linked list [LeetCode #206]

import java.util.Scanner;

class LinkedList {

Node head;

static class Node {

int data;

Node next;

Node(int data) {

this.data = data;

this.next = null;

}

}

public void append(int data) {

Node newNode = new Node(data);

if (head == null) {

head = newNode;

return;

}

Node last = head;

while (last.next != null) {

last = last.next;

}

last.next = newNode;

}

public void reverse() {

Node prev = null;

Node current = head;

Node next = null;

while (current != null) {

next = current.next;

current.next = prev;

prev = current;

current = next;

}

head = prev;

}

public void printList() {

Node temp = head;

while (temp != null) {

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

temp = temp.next;

}

System.out.println();

}

public static void main(String[] args) {

Scanner sc = new Scanner(System.in);

LinkedList list = new LinkedList();

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

int n = sc.nextInt();

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

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

int data = sc.nextInt();

list.append(data);

}

System.out.print("Original Linked List: ");

list.printList();

list.reverse();

System.out.print("Reversed Linked List: ");

list.printList();

}

}

# Set 6a: Two numbers sum problem [LeetCode #1]

import java.util.\*;  
  
class TwoSum {  
 public static int[] twoSum(int[] nums, int target) {  
 Map<Integer, Integer> map = new HashMap<>();  
 for (int i = 0; i < nums.length; i++) {  
 int comp = target - nums[i];  
 if (map.containsKey(comp)) {  
 return new int[]{map.get(comp), i};  
 }  
 map.put(nums[i], i);  
 }  
 return new int[0];  
 }

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

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

int target = scanner.nextInt();

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

System.out.println("Indices of the two numbers: " + res[0] + " " + res[1]);

}

}  
}

# Set 6b: Search an element in linked list (iteration and recursion) [GeeksforGeeks]

import java.util.Scanner;

public class Program6b {

static class Node {

int data;

Node next;

Node(int d) { data = d; next = null; }

}

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

Node curr = head;

while (curr != null) {

if (curr.data == target) return true;

curr = curr.next;

}

return false;

}

public static Node buildList(int[] vals) {

if (vals.length == 0) return null;

Node head = new Node(vals[0]), tail = head;

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

tail.next = new Node(vals[i]);

tail = tail.next;

}

return head;

}

public static void main(String[] args) {

Scanner sc = new Scanner(System.in);

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

int n = sc.nextInt();

int[] vals = new int[n];

System.out.print("Enter list elements: ");

for (int i = 0; i < n; i++) vals[i] = sc.nextInt();

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

int target = sc.nextInt();

Node head = buildList(vals);

boolean found = searchIterative(head, target);

System.out.println("Element " + target +

(found ? " found" : " not found") + " using Iteration.");

}

}

Recursion

import java.util.Scanner;

public class SearchRecursive {

static class Node {

int data;

Node next;

Node(int d) { data = d; next = null; }

}

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

if (head == null) return false;

if (head.data == target) return true;

return searchRecursive(head.next, target);

}

public static Node buildList(int[] vals) {

if (vals.length == 0) return null;

Node head = new Node(vals[0]), tail = head;

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

tail.next = new Node(vals[i]);

tail = tail.next;

}

return head;

}

public static void main(String[] args) {

Scanner sc = new Scanner(System.in);

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

int n = sc.nextInt();

int[] vals = new int[n];

System.out.print("Enter list elements: ");

for (int i = 0; i < n; i++) vals[i] = sc.nextInt();

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

int target = sc.nextInt();

Node head = buildList(vals);

boolean found = searchRecursive(head, target);

System.out.println("Element " + target +

(found ? " found" : " not found") + " using Recursion.");

}}

# Set 7a: Remove duplicates from an array - O(log n) [LeetCode #26]

class RemoveDuplicatesSortedArray {  
 public static int removeDuplicates(int[] nums) {  
 if (nums.length == 0) return 0;  
 int i = 0;  
 for (int j = 1; j < nums.length; j++) {  
 if (nums[j] != nums[i]) {  
 i++;  
 nums[i] = nums[j];  
 }  
 }  
 return i + 1;  
 }  
  
 public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

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

}

int len = removeDuplicates(nums);

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

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

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

}

}

}

# Set 7b: Reversing a queue using stack [GeeksforGeeks]

import java.util.LinkedList;

import java.util.Queue;

import java.util.Scanner;

public class Program7b {

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

if (q.isEmpty()) {

return;

}

int front = q.poll();

reverseQueue(q);

q.offer(front);

}

public static void main(String[] args) {

Scanner sc = new Scanner(System.in);

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

int n = sc.nextInt();

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

System.out.print("Enter queue elements: ");

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

queue.offer(sc.nextInt());

}

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

System.out.println(queue);

reverseQueue(queue);

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

System.out.println(queue);

}

}

### ****Set 8a: Remove consecutive duplicates in a string [LeetCode #1047]****

import java.util.Scanner;

public class RemoveConsecutiveDuplicates {

public static String removeConsecutiveDuplicates(String str) {

if (str.length() == 0) return "";

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 sc = new Scanner(System.in);

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

String input = sc.nextLine();

String output = removeConsecutiveDuplicates(input);

System.out.println("After removing consecutive duplicates: " + output);

}

}

### ✅ ****Set 8b: Stack sort using another stack [GeeksforGeeks]****

import java.util.Scanner;

import java.util.Stack;

public class SortStack {

public static Stack<Integer> sortStack(Stack<Integer> input) {

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

while (!input.isEmpty()) {

int temp = input.pop();

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

input.push(tempStack.pop());

}

tempStack.push(temp);

}

return tempStack;

}

public static void main(String[] args) {

Scanner sc = new Scanner(System.in);

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

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

int n = sc.nextInt();

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

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

stack.push(sc.nextInt());

}

Stack<Integer> sorted = sortStack(stack);

System.out.print("Sorted Stack (top to bottom): ");

while (!sorted.isEmpty()) {

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

}

}

}

### ✅ ****Set 9a: Equilibrium index [LeetCode #724]****

java

CopyEdit

class EquilibriumIndex {

public static int pivotIndex(int[] nums) {

int total = 0, leftSum = 0;

for (int num : nums) total += num;

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

if (leftSum == total - leftSum - nums[i]) return i;

leftSum += nums[i];

}

return -1;

}

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

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

}

int pivot = pivotIndex(nums);

if (pivot != -1) {

System.out.println("Pivot index is: " + pivot);

} else {

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

}

}

}

### ✅ ****Set 9b: Diameter of BST [LeetCode #543]****

java

CopyEdit

class DiameterOfBinaryTree {

static class TreeNode {

int val;

TreeNode left, right;

TreeNode(int val) { this.val = val; }

}

static int maxDiameter = 0;

public static int diameterOfBinaryTree(TreeNode root) {

depth(root);

return maxDiameter;

}

private static int depth(TreeNode node) {

if (node == null) return 0;

int left = depth(node.left);

int right = depth(node.right);

maxDiameter = Math.max(maxDiameter, left + right);

return 1 + Math.max(left, right);

}

public static void main(String[] args) {

TreeNode root = new TreeNode(1);

root.left = new TreeNode(2);

root.right = new TreeNode(3);

root.left.left = new TreeNode(4);

root.left.right = new TreeNode(5);

System.out.println(diameterOfBinaryTree(root)); // Output: 3

}

}

Set 11

1) Total Cost of Connections

import java.util.PriorityQueue;

import java.util.Scanner;

public class MinConnectionCost {

public static int totalConnectionCost(int[] lengths) {

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

for (int length : lengths) {

minHeap.add(length);

}

int totalCost = 0;

while (minHeap.size() > 1) {

int first = minHeap.poll(); // Smallest

int second = minHeap.poll(); // Second smallest

int cost = first + second;

totalCost += cost;

minHeap.add(cost);

}

return totalCost;

}

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

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

int n = scanner.nextInt();

int[] lengths = new int[n];

System.out.println("Enter the lengths of the ropes:");

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

lengths[i] = scanner.nextInt();

}

int cost = totalConnectionCost(lengths);

System.out.println("Total Cost = " + cost);

} }

2) Construct a BST and Remove All Half Nodes

import java.util.Scanner;

class Node {

int data;

Node left, right;

Node(int item) {

data = item;

left = right = null;

}

}

public class DynamicBSTHalfNodeRemoval {

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

if (root == null) return new Node(key);

if (key < root.data)

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

else

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

return root;

}

public static Node removeHalfNodes(Node root) {

if (root == null) return null;

root.left = removeHalfNodes(root.left);

root.right = removeHalfNodes(root.right);

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

return root.right;

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

return root.left;

return root;

}

public static void inorder(Node root) {

if (root != null) {

inorder(root.left);

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

inorder(root.right);

}

}

public static void main(String[] args) {

Scanner sc = new Scanner(System.in);

Node root = null;

System.out.println("Enter number of nodes to insert:");

int n = sc.nextInt();

System.out.println("Enter " + n + " values:");

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

int val = sc.nextInt();

root = insert(root, val);

}

System.out.print("\nInorder before removing half nodes: ");

inorder(root);

root = removeHalfNodes(root);

System.out.print("\nInorder after removing half nodes: ");

inorder(root);

}

}

### ✅ ****Set 12a: BST print left nodes [GeeksforGeeks]****

java

CopyEdit

class LeftViewOfBST {

static class TreeNode {

int val;

TreeNode left, right;

TreeNode(int val) {

this.val = val;

}

}

static int maxLevel = 0;

public static void printLeftView(TreeNode root) {

printLeftViewUtil(root, 1);

}

private static void printLeftViewUtil(TreeNode node, int level) {

if (node == null) return;

if (maxLevel < level) {

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

maxLevel = level;

}

printLeftViewUtil(node.left, level + 1);

printLeftViewUtil(node.right, level + 1);

public static void main(String[] args) {

TreeNode root = new TreeNode(10);

}

root.left = new TreeNode(12);

root.right = new TreeNode(3);

root.left.left = new TreeNode(4);

root.left.right = new TreeNode(5);

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

root.right.right = new TreeNode(7);

printLeftView(root); // Output: 10 12 4

}

}

### ✅ ****Set 12b: Product of an array without division operator [LeetCode #238]****

java

CopyEdit

import java.util.Arrays;

class ProductOfArrayExceptSelf {

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

int n = nums.length;

int[] left = new int[n];

int[] right = new int[n];

int[] output = new int[n];

left[0] = 1;

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

left[i] = nums[i - 1] \* left[i - 1];

}

right[n - 1] = 1;

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

right[i] = nums[i + 1] \* right[i + 1];

}

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

output[i] = left[i] \* right[i];

}

return output;

}

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

System.out.println("Enter the size of 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.println("Product of array except self: " + Arrays.toString(productExceptSelf(nums)));

}

}

13 a) Assign cookies (gfg)

import java.util.Arrays;

public class AssignCookies {

public static void main(String[] args) {

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

int[] cookies = {1, 1};

System.out.println("Number of children satisfied: " + findContentChildren(greed,

cookies));

}

public static int findContentChildren(int[] greed, int[] cookies) {

Arrays.sort(greed);

Arrays.sort(cookies);

int childIndex = 0;

int cookieIndex = 0;

int satisfiedChildren = 0;

while (childIndex < greed.length && cookieIndex < cookies.length) {

if (cookies[cookieIndex] >= greed[childIndex]) {

satisfiedChildren++;

childIndex++;

}

cookieIndex++;

}

return satisfiedChildren;

}

}

Output:

Greed factors of children: [1, 2, 3]

Cookie sizes: [1, 1]

Number of children satisfied: 1

13 b) Width of binary search tree at given level

import java.util.LinkedList;

import java.util.Queue;

import java.util.Scanner;

class Node {

int data;

Node left, right;

public Node(int item) {

data = item;

left = right = null;

}

}

class BinarySearchTree {

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 widthAtGivenLevel = 0;

while (!queue.isEmpty()) {

int nodeCount = queue.size();

if (currentLevel == level) {

widthAtGivenLevel = nodeCount;

}

while (nodeCount-- > 0) {

Node currentNode = queue.poll();

if (currentNode.left != null) {

queue.add(currentNode.left);

}

if (currentNode.right != null) {

queue.add(currentNode.right);

}

}

currentLevel++;

if (currentLevel > level) {

break;

}

}

return widthAtGivenLevel;

}

public void insert(int data) {

root = insertRec(root, data);

}

private Node insertRec(Node root, int data) {

if (root == null) {

root = new Node(data);

return root;

}

if (data < root.data) {

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

} else if (data > root.data) {

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

}

return root;

}

}

public class Main {

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

BinarySearchTree tree = new BinarySearchTree();

System.out.print("Enter the number of nodes to insert into the BST: ");

int n = scanner.nextInt();

System.out.println("Enter the values for the BST nodes:");

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

int value = scanner.nextInt();

tree.insert(value);

}

System.out.print("Enter the level at which to calculate 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);

scanner.close();

}

}

Output:

Enter the number of nodes to insert into the BST: 7

Enter the values for the BST nodes:

50 30 20 40 70 60 80

Enter the level at which to calculate the width: 2

Width of the BST at level 2 is: 4

### ✅ ****Set 14a: Lemonade change [LeetCode #860]****

java

CopyEdit

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) return false;

five--; ten++;

} else {

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.println("Enter the number of bills:");

int n = scanner.nextInt();

int[] bills = new int[n];

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

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

bills[i] = scanner.nextInt();

}

System.out.println("Can provide change: " + lemonadeChange(bills));

}

}

### ✅ ****Set 14b: Height of Binary Search Tree [LeetCode #104]****

import java.util.Scanner;

class Node {

int data;

Node left, right;

public Node(int item) {

data = item;

left = right = null;

}

}

class BinarySearchTree {

Node root;

public void insert(int data) {

root = insertRec(root, data);

}

private Node insertRec(Node root, int data) {

if (root == null) {

root = new Node(data);

return root;

}

if (data < root.data) root.left = insertRec(root.left, data);

else if (data > root.data) root.right = insertRec(root.right, data);

return root;

}

public int height() {

return heightRec(root);

}

private int heightRec(Node node) {

if (node == null) return -1;

return Math.max(heightRec(node.left), heightRec(node.right)) + 1;

}

public void inorder() {

inorderRec(root);

}

private void inorderRec(Node root) {

if (root != null) {

inorderRec(root.left);

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

inorderRec(root.right);

}

}

}

public class Main {

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

BinarySearchTree tree = new BinarySearchTree();

while (true) {

System.out.print("Enter a number (or 'exit' to stop): ");

String input = scanner.nextLine();

if (input.equalsIgnoreCase("exit")) break;

try {

int value = Integer.parseInt(input);

tree.insert(value);

} catch (NumberFormatException e) {

System.out.println("Invalid input.");

}

}

System.out.println("\nIn-order traversal of the BST:");

tree.inorder();

System.out.println("\nHeight of the tree: " + tree.height());

scanner.close();

}

}

Output:

Enter a number (or 'exit' to stop): 50

Enter a number (or 'exit' to stop): 30

Enter a number (or 'exit' to stop): 20

Enter a number (or 'exit' to stop): 40

Enter a number (or 'exit' to stop): 70

Enter a number (or 'exit' to stop): exit

In-order traversal of the BST:

20 30 40 50 70

Height of the tree: 2

### ✅ ****Set 17a: Count All Paths (DP) [LeetCode #62]****

class Node {

int data;

Node left, right;

Node(int item) {

data = item;

left = right = null;

}

}

public class CountPathsBinaryTree {

public static int countPaths(Node root) {

return countPathsUtil(root, 0);

}

public static int countPathsUtil(Node root, int currentSum) {

if (root == null) {

return 0;

}

currentSum += root.data;

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

return 1;

}

return countPathsUtil(root.left, currentSum) + countPathsUtil(root.right, currentSum);

}

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

if (root == null) return new Node(key);

if (key < root.data)

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

else

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

return root;

}

public static void main(String[] args) {

Scanner sc = new Scanner(System.in);

Node root = null;

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

int n = sc.nextInt();

System.out.println("Enter " + n + " node values:");

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

int val = sc.nextInt();

root = insert(root, val);

}

int totalPaths = countPaths(root);

System.out.println("Total number of root-to-leaf paths: " + totalPaths);

}

}

Output:

Enter number of nodes: 5

Enter 5 node values:

10 5 20 3 7

Total number of root-to-leaf paths: 4

### ✅ ****Set 17b: BFS Traversal of Binary Tree [LeetCode #102]****

java

CopyEdit

17b) breadth first traversal BST

import java.util.LinkedList;

import java.util.Queue;

import java.util.Scanner;

class Node {

int data;

Node left, right;

Node(int item) {

data = item;

left = right = null;

}

}

public class BFSBinaryTree {

public static void levelOrderTraversal(Node root) {

if (root == null) {

return;

}

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

queue.add(root);

while (!queue.isEmpty()) {

Node current = queue.poll();

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

if (current.left != null) {

queue.add(current.left);

}

if (current.right != null) {

queue.add(current.right);

}

}

}

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

if (root == null) return new Node(key);

if (key < root.data)

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

else

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

return root;

}

public static void main(String[] args) {

Scanner sc = new Scanner(System.in);

Node root = null;

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

int n = sc.nextInt();

System.out.println("Enter " + n + " node values:");

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

int val = sc.nextInt();

root = insert(root, val);

}

System.out.print("\nLevel-Order Traversal: ");

levelOrderTraversal(root);

}

}

Output:

Enter number of nodes: 7

Enter 7 node values:

10 5 20 3 7 15 25 Level-Order Traversal: 10 5 20 3 7 15 25

### ✅ ****Set 18a: Kadane's Algorithm [LeetCode #53]****

java

CopyEdit

class MaximumSubarray {

public static int maxSubArray(int[] nums) {

int maxSum = nums[0];

int currSum = nums[0];

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

currSum = Math.max(nums[i], currSum + nums[i]);

maxSum = Math.max(maxSum, currSum);

}

return maxSum;

}

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

System.out.println("Enter the size of 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.println("Maximum subarray sum: " + maxSubArray(nums));

}

}

### ✅ ****Set 18b: Tree Traversals (Inorder, Preorder, Postorder)****

import java.util.Scanner;

class Node {

int data;

Node left, right;

Node(int item) {

data = item;

left = right = null;

}

}

public class BinaryTreeTraversals {

public static void inorder(Node root) {

if (root != null) {

inorder(root.left);

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

inorder(root.right);

}

}

public static void preorder(Node root) {

if (root != null) {

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

preorder(root.left);

preorder(root.right);

}

}

public static void postorder(Node root) {

if (root != null) {

postorder(root.left);

postorder(root.right);

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

}

}

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

if (root == null) return new Node(key);

if (key < root.data)

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

else

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

return root;

}

public static void main(String[] args) {

Scanner sc = new Scanner(System.in);

Node root = null;

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

int n = sc.nextInt();

System.out.println("Enter " + n + " node values:");

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

int val = sc.nextInt();

root = insert(root, val); // building a BST for simplicity

}

System.out.print("\nInorder Traversal: ");

inorder(root);

System.out.print("\nPreorder Traversal: ");

preorder(root);

System.out.print("\nPostorder Traversal: ");

postorder(root);

}

}

**Set 19**

19)a) Reverse characters of the string using two pointers

import java.util.Scanner;

public class ReverseStringTwoPointers {

public static String reverse(String str) {

char[] chars = str.toCharArray();

int left = 0;

int right = chars.length - 1;

while (left < right) {

char temp = chars[left];

chars[left] = chars[right];

chars[right] = temp;

left++;

right--;

}

return new String(chars);

}

public static void main(String[] args) {

Scanner sc = new Scanner(System.in);

System.out.print("Enter a string to reverse: ");

String input = sc.nextLine();

String reversed = reverse(input);

System.out.println("Reversed string: " + reversed);

}

}

Output:

Enter a string to reverse: hello world

Reversed string: dlrow olleh

19)b. Implement two stacks in one array

import java.util.Scanner;

public class TwoStacksDynamic {

int size;

int top1, top2;

int[] arr;

public TwoStacksDynamic(int n) {

size = n;

arr = new int[n];

top1 = -1;

top2 = n;

}

public void push1(int x) {

if (top1 < top2 - 1) {

arr[++top1] = x;

} else {

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

}

}

public void push2(int x) {

if (top1 < top2 - 1) {

arr[--top2] = x;

} else {

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

}

}

public int pop1() {

if (top1 >= 0) {

return arr[top1--];

} else {

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

return -1;

}

}

public int pop2() {

if (top2 < size) {

return arr[top2++];

} else {

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

return -1;

}

}

public void displayStacks() {

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

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

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

}

System.out.println();

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

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

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

}

System.out.println();

}

public static void main(String[] args) {

Scanner sc = new Scanner(System.in);

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

int n = sc.nextInt();

TwoStacksDynamic ts = new TwoStacksDynamic(n);

int choice;

do {

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

System.out.println("6. Exit");

System.out.print("Choose an option: ");

choice = sc.nextInt();

switch (choice) {

case 1:

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

ts.push1(sc.nextInt());

break;

case 2:

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

ts.push2(sc.nextInt());

break;

case 3:

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

break;

case 4:

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

break;

case 5:

ts.displayStacks();

break;

case 6:

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

break;

default:

System.out.println("Invalid choice!");

}

} while (choice != 6);

sc.close();

}

}

Output:

Enter size of the array: 6

1. Push to Stack 1

2. Push to Stack 2

3. Pop from Stack 1

4. Pop from Stack 2

5. Display Stacks

6. Exit

Choose an option: 1

Enter value to push to Stack 1: 10

Choose an option: 2

Enter value to push to Stack 2: 100

Choose an option: 5

Stack 1: 10

Stack 2: 100

**Set 20a: Triplet Sum**

class TripletSum {

public static boolean findTriplet(int[] arr) {

int n = arr.length;

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

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

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

if (arr[i] + arr[j] == arr[k]) return true;

}

}

}

return false;

}

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

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

}

System.out.println("Triplet found: " + findTriplet(arr));

}

}

**Set 20b: DFS without recursion using adjacency list**

build code to implement DFS using adjacency list without recursion

import java.util.\*;

public class DFSWithoutRecursionDynamic {

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

System.out.println("Enter the number of nodes in the graph:");

int nodes = scanner.nextInt();

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

System.out.println("Enter the edges in the graph (as pairs of nodes):");

System.out.println("Type '-1 -1' to stop entering edges.");

while (true) {

int from = scanner.nextInt();

int to = scanner.nextInt();

if (from == -1 && to == -1) break;

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

graph.get(from).add(to);

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

graph.get(to).add(from);

}

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

int startNode = scanner.nextInt();

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

dfs(startNode, graph);

}

public static void dfs(int startNode, Map<Integer, List<Integer>> graph) {

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

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

stack.push(startNode);

visited.add(startNode);

while (!stack.isEmpty()) {

int currentNode = stack.pop();

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

List<Integer> neighbors = graph.getOrDefault(currentNode, new ArrayList<>());

for (int neighbor : neighbors) {

if (!visited.contains(neighbor)) {

stack.push(neighbor);

visited.add(neighbor);

}

}

}

}

}

Output:

Enter the number of nodes in the graph:5

Enter the edges in the graph (as pairs of nodes):

0 1

0 2

1 3

1 4 -1 -1

Enter the starting node for DFS:0

DFS Traversal:0 2 1 4 3

**Set 21a:**

**import java.util.\*;**

**public class Program21a {**

**public static int distributeCandies(int[] rankings) {**

**int n = rankings.length;**

**int[] candies = new int[n];**

**Arrays.fill(candies, 1);**

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

**if (rankings[i] > rankings[i - 1]) {**

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

**}**

**}**

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

**if (rankings[i] > rankings[i + 1]) {**

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

**}**

**}**

**int totalCandies = Arrays.stream(candies).sum();**

**return totalCandies;**

**}**

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

**Scanner scanner = new Scanner(System.in);**

**System.out.print("Enter number of students: ");**

**int n = scanner.nextInt();**

**int[] rankings = new int[n];**

**System.out.println("Enter rankings of students:");**

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

**rankings[i] = scanner.nextInt();**

**}**

**int result = distributeCandies(rankings);**

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

**scanner.close();**

**}**

**}**

**Output:**

**Enter number of students: 6**

**Enter rankings of students:**

**1 2 2 3 4 2**

**Minimum candies required: 10 Distribute Candies**

**Set 21b: DFS with adjacency list (LeetCode #797)**

import java.util.\*;

public class Program21b {

private static void dfs(int node, List<List<Integer>> adjList, boolean[] visited) {

visited[node] = true;

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

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

if (!visited[neighbor]) {

dfs(neighbor, adjList, visited);

}

}

}

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

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

int V = scanner.nextInt();

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

int E = scanner.nextInt();

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

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

adjList.add(new ArrayList<Integer>()); // Explicitly specify Integer type

}

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

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

int u = scanner.nextInt();

int v = scanner.nextInt();

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

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

}

boolean[] visited = new boolean[V];

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

dfs(0, adjList, visited);

scanner.close();

}

}

Output:

Enter number of vertices: 5

Enter number of edges: 4

Enter edges (format: u v):

0 1

0 2

1 3

1 4

DFS Traversal starting from node 0:

0 1 3 4 2

### 22a) ****Staircase Problem using DP (LeetCode #70)****

import java.util.Scanner;

public class Program22a {

public static int countWays(int n) {

if (n < 0) return 0;

if (n == 0) return 1;

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

dp[0] = 1;

if (n >= 1) dp[1] = 1;

if (n >= 2) dp[2] = 2;

for (int i = 3; i <= n; i++) {

dp[i] = dp[i - 1] + dp[i - 2] + dp[i - 3];

}

return dp[n];

}

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

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

int n = scanner.nextInt();

int ways = countWays(n);

System.out.println("Number of ways to climb " + n + " stairs: " + ways);

scanner.close();

}

}

Output:

Enter the number of stairs: 5

Number of ways to climb 5 stairs: 13

### 22b) ****DFS (LeetCode #797)****

import java.util–.\*;

public class Program22b {

private static void dfs(int node, List<List<Integer>> adjList, boolean[] visited) {

visited[node] = true;

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

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

if (!visited[neighbor]) {

dfs(neighbor, adjList, visited);

}

}

}

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

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

int V = scanner.nextInt();

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

int E = scanner.nextInt();

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

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

adjList.add(new ArrayList<Integer>());

}

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

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

int u = scanner.nextInt();

int v = scanner.nextInt();

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

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

}

boolean[] visited = new boolean[V];

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

dfs(0, adjList, visited);

scanner.close();

}

}

Output:

Enter number of vertices: 5

Enter number of edges: 4

Enter edges:

0 1

0 2

1 3

2 4

Set 23) Strings and DFS using matrix for recursion

public class StringDFS {

public static void dfs(char[][] matrix, int i, int j, String str) {

if (i < 0 || i >= matrix.length || j < 0 || j >= matrix[0].length || matrix[i][j] != str.charAt(0)) {

return;

}

if (str.length() == 1) {

System.out.println("Found string at (" + i + "," + j + ")");

return;

}

char temp = matrix[i][j];

matrix[i][j] = '#';

dfs(matrix, i - 1, j, str.substring(1));

dfs(matrix, i + 1, j, str.substring(1));

dfs(matrix, i, j - 1, str.substring(1));

dfs(matrix, i, j + 1, str.substring(1));

matrix[i][j] = temp;

}

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

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

int m = scanner.nextInt();

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

int n = scanner.nextInt();

char[][] board = new char[m][n];

System.out.println("Enter the characters in the board:");

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

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

board[i][j] = scanner.next().charAt(0);

}

}

System.out.println("Enter the word to search:");

String word = scanner.next();

System.out.println("Word found: " + exist(board, word));

}

}

### ****24****

#### ****A) Find triplets with sum zero****

java

CopyEdit

import java.util.\*;

public class TripletSumZero {

public static void main(String[] args) {

Scanner sc = new Scanner(System.in);

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

int n = sc.nextInt();

int[] arr = new int[n];

System.out.print("Enter array elements: ");

for (int i = 0; i < n; i++) arr[i] = sc.nextInt();

boolean found = false;

Arrays.sort(arr);

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

int left = i + 1, right = n - 1;

while (left < right) {

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

if (sum == 0) {

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

found = true;

left++;

right--;

} else if (sum < 0) left++;

else right--;

}

}

if (!found) System.out.println("No triplet found with sum zero.");

}

}

#### ****B) BFS using adjacency matrix****

java

CopyEdit

import java.util.\*;

public class BFSAdjMatrix {

public static void main(String[] args) {

Scanner sc = new Scanner(System.in);

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

int v = sc.nextInt();

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

System.out.println("Enter adjacency matrix:");

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

for (int j = 0; j < v; j++)

adj[i][j] = sc.nextInt();

System.out.print("Enter starting vertex: ");

int start = sc.nextInt();

boolean[] visited = new boolean[v];

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

visited[start] = true;

q.add(start);

System.out.print("BFS Traversal: ");

while (!q.isEmpty()) {

int node = q.poll();

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

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

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

visited[i] = true;

q.add(i);

}

}

}

}

}

25) Minimum platforms for railway

import java.util.Arrays;

public class MinimumPlatforms {

public static int findPlatform(int[] arr, int[] dep) {

Arrays.sort(arr);

Arrays.sort(dep);

int plat\_needed = 1;

int result = 1;

int i = 1;

int j = 0;

while (i < arr.length && j < dep.length) {

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

plat\_needed++;

i++;

if (plat\_needed > result) {

result = plat\_needed;

}

} else {

plat\_needed--;

j++;

}

}

return result;

}

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

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

int n = scanner.nextInt();

int[] arr = new int[n];

int[] dep = new int[n];

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

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

arr[i] = scanner.nextInt();

}

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

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

dep[i] = scanner.nextInt();

}

System.out.println("Minimum Number of Platforms Required = " + findPlatform(arr, dep));

}

}

}

BFS in Adjacency List

import java.util.\*;

public class Program25b {

public static void bfs(int startNode, List<List<Integer>> adjList, int V) {

boolean[] visited = new boolean[V];

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

visited[startNode] = true;

queue.add(startNode);

System.out.println("BFS Traversal:");

while (!queue.isEmpty()) {

int node = queue.poll();

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

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

if (!visited[neighbor]) {

visited[neighbor] = true;

queue.add(neighbor);

}

}

}

}

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

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

int V = scanner.nextInt();

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

int E = scanner.nextInt();

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

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

adjList.add(new ArrayList<Integer>());

}

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

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

int u = scanner.nextInt();

int v = scanner.nextInt();

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

adjList.get(v).add(u); // Assuming an undirected graph

}

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

int startNode = scanner.nextInt();

bfs(startNode, adjList, V);

scanner.close();

}

}

Output:

Enter number of vertices: 5

Enter number of edges: 4

Enter edges:

0 1

0 2

1 3

2 4

Enter the starting node for BFS: 0

BFS Traversal:

0 1 2 3 4

### 26a) ****Given 2 strings, print them alternatively (LeetCode #1768)****

java

CopyEdit

public class AlternateStrings {

public static String mergeAlternately(String word1, String word2) {

StringBuilder result = new StringBuilder();

int i = 0, j = 0;

while (i < word1.length() || j < word2.length()) {

if (i < word1.length()) result.append(word1.charAt(i++));

if (j < word2.length()) result.append(word2.charAt(j++));

}

return result.toString();

}

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

System.out.println("Enter the first word:");

String word1 = scanner.next();

System.out.println("Enter the second word:");

String word2 = scanner.next();

System.out.println("Merged String: " + mergeAlternately(word1, word2));

}

}

### 26b) ****Balanced Parentheses (LeetCode #20)****

java

CopyEdit

import java.util.Stack;

public class BalancedParentheses {

public static boolean isValid(String s) {

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

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

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

stack.push(ch);

} else {

if (stack.isEmpty()) return false;

char top = stack.pop();

if (ch == ')' && top != '(') return false;

if (ch == '}' && top != '{') return false;

if (ch == ']' && top != '[') return false;

}

}

return stack.isEmpty();

}

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

System.out.println("Enter a string of parentheses:");

String s = scanner.next();

System.out.println("Is balanced: " + isValid(s));

}

}

### 27a) ****4-Sum with Target X (LeetCode #18)****

java

CopyEdit

import java.util.\*;

public class FourSum {

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

Arrays.sort(nums);

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

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

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

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

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

int left = j + 1, 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]));

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);

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

int target = scanner.nextInt();

System.out.println("4-Sum: " + fourSum(nums, target));

}

}

### 27b) ****Delete Middle Element of a Stack (GeeksforGeeks)****

import java.util.\*;

public class Program27b {

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

if (stack.size() == midIndex + 1) {

stack.pop();

return;

}

int temp = stack.pop();

deleteMiddle(stack, midIndex);

stack.push(temp);

}

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

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

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

int n = scanner.nextInt();

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

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

stack.push(scanner.nextInt());

}

int midIndex = n / 2;

deleteMiddle(stack, midIndex);

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

scanner.close();

}

}

Output:

Enter number of elements in the stack: 5

Enter stack elements:

1 2 3 4 5

Stack after deleting middle element: [1, 2, 4, 5]

### 29a) ****Unique Paths in Grid with Obstacles (LeetCode #63)****

import java.util.\*;

public class Program29a {

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

int m = grid.length, n = grid[0].length;

if (grid[0][0] == 1 || grid[m - 1][n - 1] == 1) return 0;

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

dp[0][0] = 1;

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

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

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

dp[i][j] = 0;

} else {

if (i > 0) dp[i][j] += dp[i - 1][j];

if (j > 0) dp[i][j] += dp[i][j - 1];

}

}

}

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

}

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

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

int m = scanner.nextInt();

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

int n = scanner.nextInt();

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

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

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

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

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

}

}

int result = uniquePathsWithObstacles(grid);

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

scanner.close();

}

}

Output:

Enter number of rows: 3

Enter number of columns: 3

Enter grid:

0 0 0

0 1 0

0 0 0

Number of unique paths: 2

### 29b) ****Reverse Queue Content Using Stack (GeeksforGeeks)****

import java.util.\*;

public class Program29b {

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

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

while (!queue.isEmpty()) {

stack.push(queue.poll());

}

while (!stack.isEmpty()) {

queue.add(stack.pop());

}

}

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

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

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

int n = scanner.nextInt();

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

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

queue.add(scanner.nextInt());

}

reverseQueue(queue);

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

scanner.close();

}

}

Output:

Enter number of elements in the queue: 5

Enter queue elements:

1 2 3 4 5

Queue after reversal: [5, 4, 3, 2, 1]

### 30a) ****Remove Duplicates from Sorted Array (LeetCode #26)****

import java.util.\*;

public class Program30a {

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

LinkedHashSet<Integer> set = new LinkedHashSet<>();

for (int num : nums) {

set.add(num);

}

int[] uniqueArray = new int[set.size()];

int i = 0;

for (int num : set) {

uniqueArray[i++] = num;

}

return uniqueArray;

}

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

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

int n = scanner.nextInt();

int[] nums = new int[n];

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

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

nums[i] = scanner.nextInt();

}

int[] uniqueArray = removeDuplicates(nums);

System.out.println("Unique array: " + Arrays.toString(uniqueArray));

System.out.println("New length: " + uniqueArray.length);

scanner.close();

}

}

Output:

Enter number of elements: 7

Enter array elements:

1 2 2 3 4 4 5

Unique array: [1, 2, 3, 4, 5]

New length: 5

30b)

import java.util.\*;

public class Program30b {

static class MinStack {

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

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

public void push(int x) {

mainStack.push(x);

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

minStack.push(x);

}

}

public void pop() {

if (!mainStack.isEmpty()) {

int popped = mainStack.pop();

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

minStack.pop();

}

}

}

public int getMin() {

return minStack.isEmpty() ? Integer.MAX\_VALUE : minStack.peek();

}

}

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

MinStack stack = new MinStack();

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

int n = scanner.nextInt();

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

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

stack.push(scanner.nextInt());

}

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

scanner.close();

}

}

Output:

Enter number of elements in the stack: 5

Enter stack elements:

3 5 2 1 4

Minimum element in stack: 1

32) A) Intersection of two arrays in log n time complexity

import java.util.Arrays;

public class Intersection {

public static int[] intersection(int[] arr1, int[] arr2) {

Arrays.sort(arr1);

Arrays.sort(arr2);

int i = 0;

int j = 0;

int k = 0;

int[] result = new int[Math.min(arr1.length, arr2.length)];

while (i < arr1.length && j < arr2.length) {

if (arr1[i] < arr2[j]) {

i++;

} else if (arr1[i] > arr2[j]) {

j++;

} else {

result[k++] = arr1[i];

i++;

j++;

}

}

return Arrays.copyOfRange(result, 0, k);

}

public static void main(String[] args) {

int[] arr1 = {1, 2, 3, 4, 5};

int[] arr2 = {4, 5, 6, 7, 8};

int[] result = intersection(arr1, arr2);

System.out.println("Intersection: " + Arrays.toString(result));

}

}

B) Find a string is palindrome or not using stack and queue

import java.util.Stack;

import java.util.Queue;

import java.util.LinkedList;

public class Palindrome {

public static boolean isPalindrome(String str) {

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

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

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

stack.push(c);

queue.add(c);

}

while (!stack.isEmpty()) {

if (stack.pop() != queue.poll()) {

return false;

}

}

return true;

}

public static void main(String[] args) {

String str = "madam";

System.out.println("Is palindrome: " + isPalindrome(str));

}

}

### 34a) ****Ternary Search (GeeksforGeeks)****

import java.util.Scanner;

public class Program34a {

public static int ternarySearch(int[] arr, int low, int high, int target) {

if (low > high) return -1;

int mid1 = low + (high - low) / 3;

int mid2 = high - (high - low) / 3;

if (arr[mid1] == target) return mid1;

if (arr[mid2] == target) return mid2;

if (target < arr[mid1]) {

return ternarySearch(arr, low, mid1 - 1, target);

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

return ternarySearch(arr, mid2 + 1, high, target);

} else {

return ternarySearch(arr, mid1 + 1, mid2 - 1, target);

}

}

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

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

int n = scanner.nextInt();

int[] arr = new int[n];

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

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

arr[i] = scanner.nextInt();

}

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

int target = scanner.nextInt();

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

System.out.println(result != -1 ? "Element found at index: " + result : "Element not found");

scanner.close();

}

}

Output:

Enter number of elements: 6

Enter sorted array elements:

1 2 3 4 5 6

Enter target element: 4

Element found at index: 3

### 34b) ****Palindrome - Linked List (LeetCode #234)****

34b)

import java.util.\*;

class ListNode {

int val;

ListNode next;

ListNode(int val) { this.val = val; this.next = null; }

}

public class Program34b {

public static boolean isPalindrome(ListNode head) {

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

ListNode temp = head;

while (temp != null) {

stack.push(temp.val);

temp = temp.next;

}

temp = head;

while (temp != null) {

if (temp.val != stack.pop()) return false;

temp = temp.next;

}

return true;

}

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

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

int n = scanner.nextInt();

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

ListNode head = null, tail = null;

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

int value = scanner.nextInt();

ListNode newNode = new ListNode(value);

if (head == null) {

head = newNode;

} else {

tail.next = newNode;

}

tail = newNode;

}

System.out.println("Is the linked list a palindrome? " + isPalindrome(head));

scanner.close();

}

}

Output:

Enter number of elements in linked list: 5

Enter linked list elements:

1 2 3 2 1

Is the linked list a palindrome? True

### 35a) ****Sort the Elements of a Sorted Array After Squaring Them (LeetCode #977)****

import java.util.\*;

public class Program35a {

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

int n = arr.length;

int[] result = new int[n];

int left = 0, right = n - 1;

int index = n - 1;

while (left <= right) {

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

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

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);

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

int n = scanner.nextInt();

int[] arr = new int[n];

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

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

arr[i] = scanner.nextInt();

}

int[] result = sortedSquares(arr);

System.out.println("Sorted array after squaring: " + Arrays.toString(result));

scanner.close();

}

}

Output:

Enter number of elements: 6

Enter sorted array elements: -4 -2 0 1 3 5

Sorted array after squaring: [0, 1, 4, 9, 16, 25]

### 35b) ****Remove N-th Node from End in Linked List (LeetCode #19)****

import java.util.\*;

class ListNode {

int val;

ListNode next;

ListNode(int val) { this.val = val; this.next = null; }

}

public class Program35b {

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

ListNode dummy = new ListNode(0);

dummy.next = head;

ListNode slow = dummy, fast = dummy;

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

fast = fast.next;

}

while (fast != null) {

slow = slow.next;

fast = fast.next;

}

slow.next = slow.next.next;

return dummy.next;

}

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

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

int n = scanner.nextInt();

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

ListNode head = null, tail = null;

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

int value = scanner.nextInt();

ListNode newNode = new ListNode(value);

if (head == null) {

head = newNode;

} else {

tail.next = newNode;

}

tail = newNode;

}

System.out.print("Enter Nth element to remove from end: ");

int removePosition = scanner.nextInt();

head = removeNthFromEnd(head, removePosition);

System.out.print("Linked list after deletion: ");

ListNode temp = head;

while (temp != null) {

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

temp = temp.next;

}

scanner.close();

}

}

Output:

Enter number of elements in linked list: 5

Enter linked list elements:

1 2 3 4 5

Enter Nth element to remove from end: 2

Linked list after deletion: 1 2 3 5

36a)

import java.util.\*;

public class Program36a {

public static void paritySort(int[] arr) {

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

while (left < right) {

if (arr[left] % 2 == 0) {

left++;

} else if (arr[right] % 2 != 0) {

right--;

} else {

int temp = arr[left];

arr[left] = arr[right];

arr[right] = temp;

left++;

right--;

}

}

}

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

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

int n = scanner.nextInt();

int[] arr = new int[n];

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

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

arr[i] = scanner.nextInt();

}

paritySort(arr);

System.out.println("Sorted array (Even first, Odd last): " + Arrays.toString(arr));

scanner.close();

}

}

Output:

Enter number of elements: 6

Enter array elements:

3 1 2 4 5 6

Sorted array (Even first, Odd last): [6, 4, 2, 1, 5, 3]

36b)

import java.util.\*;

class TreeNode {

int val;

TreeNode left, right;

TreeNode(int val) { this.val = val; }

}

public class Program36b {

public static TreeNode insert(TreeNode root, int val) {

if (root == null) return new TreeNode(val);

if (val < root.val) root.left = insert(root.left, val);

else root.right = insert(root.right, val);

return root;

}

public static int diameter(TreeNode root) {

int[] diameter = new int[1];

height(root, diameter);

return diameter[0];

}

private static int height(TreeNode node, int[] diameter) {

if (node == null) return 0;

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

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

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

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

}

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

TreeNode root = null;

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

int n = scanner.nextInt();

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

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

int value = scanner.nextInt();

root = insert(root, value);

}

int result = diameter(root);

System.out.println("Diameter of the BST: " + result);

scanner.close();

}

}

Output:

Enter number of nodes in BST: 6

Enter BST elements:

5 3 8 2 4 9

Diameter of the BST: 3

Set 37

A) Find Longest Increasing Subsequence in an Array using Dynamic Programming

import java.util.\*;

public class Program37a {

public static int findLIS(int[] arr) {

int n = arr.length;

int[] dp = new int[n];

Arrays.fill(dp, 1);

int maxLIS = 1;

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);

}

}

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

}

return maxLIS;

}

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

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

int n = scanner.nextInt();

int[] arr = new int[n];

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

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

arr[i] = scanner.nextInt();

}

int result = findLIS(arr);

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

scanner.close();

}

}

Output:

Enter number of elements: 6

Enter array elements:

10 22 9 33 21 50

Length of Longest Increasing Subsequence: 4

37b)

import java.util.\*;

class TrieNode {

Map<Character, TrieNode> children = new HashMap<>();

boolean isEndOfWord;

}

public class Program37b {

private TrieNode root;

public Program37b() {

root = new TrieNode();

}

public void insert(String word) {

TrieNode node = root;

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

node.children.putIfAbsent(ch, new TrieNode());

node = node.children.get(ch);

}

node.isEndOfWord = true;

}

public String longestCommonPrefix() {

TrieNode node = root;

StringBuilder prefix = new StringBuilder();

while (node.children.size() == 1 && !node.isEndOfWord) {

char ch = node.children.keySet().iterator().next();

prefix.append(ch);

node = node.children.get(ch);

}

return prefix.toString();

}

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

Program37b trie = new Program37b();

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

int n = scanner.nextInt();

scanner.nextLine();

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

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

trie.insert(scanner.nextLine());

}

System.out.println("Longest Common Prefix: " + trie.longestCommonPrefix());

scanner.close();

}

}

Output:

Enter number of words: 4

Enter words:

flower

flow

flight

flame

Longest Common Prefix: fl

### 41a) ****Best Day to Sell Stocks (LeetCode #121)****

import java.util.\*;

public class Program41a {

public static int maxProfit(int[] prices) {

if (prices.length == 0) return 0;

int minPrice = Integer.MAX\_VALUE, maxProfit = 0;

for (int price : prices) {

minPrice = Math.min(minPrice, price);

maxProfit = Math.max(maxProfit, price - minPrice);

}

return maxProfit;

}

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

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

int n = scanner.nextInt();

int[] prices = new int[n];

System.out.println("Enter stock prices:");

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

prices[i] = scanner.nextInt();

}

int result = maxProfit(prices);

System.out.println("Maximum profit: " + result);

scanner.close();

}

}

Output:

Enter number of days: 6

Enter stock prices:

7 1 5 3 6 4

Maximum profit: 5

### 41b) ****Number of Half Nodes in BST (GeeksforGeeks)****

java

CopyEdit

import java.util.\*;

class TreeNode {

int val;

TreeNode left, right;

TreeNode(int val) { this.val = val; }

}

public class Program41b {

public static int countHalfNodes(TreeNode root) {

if (root == null) return 0;

int count = 0;

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

count = 1;

}

return count + countHalfNodes(root.left) + countHalfNodes(root.right);

}

public static TreeNode insert(TreeNode root, int val) {

if (root == null) return new TreeNode(val);

if (val < root.val) root.left = insert(root.left, val);

else root.right = insert(root.right, val);

return root;

}

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

TreeNode root = null;

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

int n = scanner.nextInt();

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

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

int value = scanner.nextInt();

root = insert(root, value);

}

int result = countHalfNodes(root);

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

scanner.close();

}

}

Output:

Enter number of nodes in BST: 6

Enter BST elements:

5 3 8 2 4 9

Number of half nodes in BST: 1

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