Name:Sudheep S Dept:CSE Reg No:22CS!71 Date: 11.11.2024

1. 0-1 knapsack problem

CODE :

class Solution {

    static int max(int[] val,int[] wt,int capacity,int i,int[][] arr){

        if(i==0||capacity==0){

            return 0;

        }

        if(arr[i][capacity]!=0){

            return arr[i][capacity];

        }

        if(wt[i-1]<=capacity){

            int ans1=val[i-1]+max(val,wt,capacity-wt[i-1],i-1,arr);

            int ans2=max(val,wt,capacity,i-1,arr);

            arr[i][capacity]=Math.max(ans1,ans2);

            return arr[i][capacity];

        }

        else{

           arr[i][capacity] =max(val,wt,capacity,i-1,arr);

           return arr[i][capacity];

        }

    }

    static int knapSack(int capacity, int val[], int wt[]) {

        int arr[][]=new int[val.length+1][capacity+1];

        return max(val,wt,capacity,val.length,arr);

    }

}

Output:

4

1 2 3

4 5 1

Your Output:

3

Expected Output:

3

TIME Complexity : O(n\*capacity)

Space Complexity : O(n\*capacity)

1. Floor in sorted array

CODE:

import java.util.Scanner;

public class Main{

    static int findFloor(int[] arr, int k) {

        int n = arr.length;

        for (int i = 0; i < n; i++) {

            if (arr[i] > k)

                return i - 1;

        }

        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 in sorted order:");

        for (int i = 0; i < n; i++) {

            arr[i] = scanner.nextInt();

        }

        System.out.print("Enter the value of k: ");

        int k = scanner.nextInt();

        int result = findFloor(arr, k);

        System.out.println("Output: " + result);

        scanner.close();

    }

}

OUTPUT:

Enter the number of elements in the array: 3

Enter the elements of the array in sorted order:

1 2 3

Enter the value of k: 4

Output: -1

Time Complexity : O(n)

Space Complexity: O(1)

1. Check Equal Arrays

CODE:

import java.util.HashMap;

import java.util.Map;

import java.util.Scanner;

public class Main {

    public static boolean check(int[] arr1, int[] arr2) {

        if (arr1.length != arr2.length) {

            return false;

        }

        Map<Integer, Integer> map1 = new HashMap<>();

        Map<Integer, Integer> map2 = new HashMap<>();

        for (int num : arr1) {

            map1.put(num, map1.getOrDefault(num, 0) + 1);

        }

        for (int num : arr2) {

            map2.put(num, map2.getOrDefault(num, 0) + 1);

        }

        return map1.equals(map2);

    }

    public static void main(String[] args) {

        Scanner scanner = new Scanner(System.in);

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

        int n1 = scanner.nextInt();

        int[] arr1 = new int[n1];

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

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

            arr1[i] = scanner.nextInt();

        }

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

        int n2 = scanner.nextInt();

        int[] arr2 = new int[n2];

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

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

            arr2[i] = scanner.nextInt();

        }

        boolean result = check(arr1, arr2);

        System.out.println("Output: " + result);

        scanner.close();

    }

}

Output:

Enter the number of elements in the first array: 4

Enter the elements of the first array:

1 3 5 2

Enter the number of elements in the second array: 4

Enter the elements of the second array:

1 2 3 4

Output: false

Time Complexity: O(N)

Space Complexity : O(N)

1. Palindrome linked list

CODE:

import java.util.Scanner;

class Node {

    int data;

    Node next;

    Node(int data) {

        this.data = data;

        this.next = null;

    }

}

class Solution {

    Node reverse\_list(Node head) {

        Node prev = null;

        Node curr = head;

        Node next;

        while (curr != null) {

            next = curr.next;

            curr.next = prev;

            prev = curr;

            curr = next;

        }

        return prev;

    }

    boolean is\_identical(Node n1, Node n2) {

        for (; n1 != null && n2 != null; n1 = n1.next, n2 = n2.next)

            if (n1.data != n2.data) return false;

        return true;

    }

    boolean isPalindrome(Node head) {

        int size = 0;

        Node ptr;

        for (ptr = head; ptr != null; ptr = ptr.next) size++;

        if (size < 2) return true;

        ptr = head;

        int mid\_pt = (size - 1) / 2;

        while (mid\_pt > 0) {

            ptr = ptr.next;

            mid\_pt--;

        }

        Node head2 = ptr.next;

        ptr.next = null;

        head2 = reverse\_list(head2);

        boolean result = is\_identical(head, head2);

        head2 = reverse\_list(head2);

        ptr.next = head2;

        return result;

    }

    public static Node createLinkedList(int[] arr) {

        Node head = null, tail = null;

        for (int val : arr) {

            Node newNode = new Node(val);

            if (head == null) {

                head = newNode;

                tail = newNode;

            } else {

                tail.next = newNode;

                tail = newNode;

            }

        }

        return head;

    }

    public static void displayLinkedList(Node head) {

        Node temp = head;

        while (temp != null) {

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

            temp = temp.next;

        }

        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 linked list: ");

        int n = scanner.nextInt();

        int[] arr = new int[n];

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

        for (int i = 0; i < n; i++) {

            arr[i] = scanner.nextInt();

        }

        Node head = createLinkedList(arr);

        System.out.print("The linked list is: ");

        displayLinkedList(head);

        Solution solution = new Solution();

        boolean result = solution.isPalindrome(head);

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

    }

}

Output:

Enter the number of elements in the linked list: 5 Enter the elements of the linked list: 1 2 3 2 1 The linked list is: 1 2 3 2 1 Is the linked list a palindrome? true

Time Complexity: O(n)

Space Complexity : O(1)

1. . Balanced Tree Check

CODE:

import java.util.Scanner;

import java.util.LinkedList;

import java.util.Queue;

class Node {

    int data;

    Node left, right;

    Node(int d) {

        data = d;

        left = right = null;

    }

}

class Tree {

    boolean isBalanced(Node root) {

        Boolean[] ans = {true};  // Use a Boolean array to pass by reference

        helper(root, ans);

        return ans[0];

    }

    static int helper(Node root, Boolean[] ans) {

        if (root == null) {

            return 0; // Base case: null node has height 0

        }

        int leftHeight = helper(root.left, ans);

        int rightHeight = helper(root.right, ans);

        if (Math.abs(leftHeight - rightHeight) > 1) {

            ans[0] = false;

        }

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

    }

    public static Node insertLevelOrder(int[] arr) {

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

        Node root = new Node(arr[0]);

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

        queue.add(root);

        int i = 1;

        while (!queue.isEmpty() && i < arr.length) {

            Node currNode = queue.poll();

            if (arr[i] != -1) {  // Use -1 to represent a null node

                currNode.left = new Node(arr[i]);

                queue.add(currNode.left);

            }

            i++;

            if (i < arr.length && arr[i] != -1) {

                currNode.right = new Node(arr[i]);

                queue.add(currNode.right);

            }

            i++;

        }

        return root;

    }

    public static void main(String[] args) {

        Scanner scanner = new Scanner(System.in);

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

        int n = scanner.nextInt();

        int[] arr = new int[n];

        System.out.println("Enter the elements (use -1 for null nodes):");

        for (int i = 0; i < n; i++) {

            arr[i] = scanner.nextInt();

        }

        Node root = insertLevelOrder(arr);

        Tree tree = new Tree();

        boolean result = tree.isBalanced(root);

        System.out.println("Is the binary tree balanced? " + result);

        scanner.close();

    }

}

Output:

Enter the number of nodes: 7

Enter the elements (use -1 for null nodes): 1 2 3 4 5 -1 -1

 Is the binary tree balanced? True

Time Complexity : O(N)  
Space Complexity : O(h)

1. Triplet sum

CODE:

import java.util.HashMap;

import java.util.Map;

import java.util.Scanner;

class Main {

    // Function to find if there exists a triplet with sum equal to x

    public static boolean find3Numbers(int arr[], int n, int x) {

        Map<Integer, Integer> s = new HashMap<>();

        for (int i : arr) {

            s.put(i, s.getOrDefault(i, 0) + 1);

        }

        for (int i = 0; i < n; i++) {

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

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

                int k = x - sum;

                if (s.containsKey(k) &&

                    (k != arr[i] || s.get(arr[i]) != 1) &&

                    (k != arr[j] || s.get(arr[j]) != 1) &&

                    (arr[i] != arr[j] || s.get(arr[j]) > 2)) {

                    return true;

                }

            }

        }

        return false;

    }

    public static void main(String[] sasta) {

        Scanner scanner = new Scanner(System.in);

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

        int n = scanner.nextInt();

        int[] arr = new int[n];

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

        for (int i = 0; i < n; i++) {

            arr[i] = scanner.nextInt();

        }

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

        int x = scanner.nextInt();

        boolean result = find3Numbers(arr, n, x);

        System.out.println("Is there a triplet with sum equal to " + x + "? " + result);

    }

}

Output:

Enter the number of elements: 3

Enter the elements:

1 3 5

Enter the target sum x: 4

Is there a triplet with sum equal to 4? False

Time Complexity: O(N\*\*2)

Space Complexity :  O(n)