Date:11/11/2024

Somasundaram M

IT

22IT104

1. **Knapsack Problem**

Given N items where each item has some weight and profit associated with it and also given a bag with capacity W, [i.e., the bag can hold at most W weight in it]. The task is to put the items into the bag such that the sum of profits associated with them is the maximum possible.

Note: The constraint here is we can either put an item completely into the bag or cannot put it at all [It is not possible to put a part of an item into the bag].

import java.util.\*;

class KnapSack{

public static void main(String[] args){

Scanner sc=new Scanner(System.in);

System.out.println("Size Sollra");

int n=sc.nextInt();

int weight=sc.nextInt();

int[] prof=new int[n];

int[] val=new int[n];

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

prof[i]=sc.nextInt();

}

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

val[i]=sc.nextInt();

}

System.out.println(helper(n,weight,prof,val));

}

public static int helper(int n,int cap,int[] prof,int[] wt){

int[][] pri=new int[n+1][cap+1];

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

for(int j=0;j<cap+1;j++){

if(i==0 || j==0) pri[i][j]=0;

else if(wt[i-1]<=j){

pri[i][j]=Math.max(prof[i-1]+pri[i-1][j-wt[i-1]],pri[i-1][j]);

}

else pri[i][j]=pri[i-1][j];

}

}

return pri[n][cap];

}

}

Output:

**Time Complexity: O(n);**

**Space Complexity: O(n);**

1. Given a sorted array and a value **x**, the floor of x is the largest element in the array smaller than or equal to x. Write efficient functions to find the floor of x.

Input: arr[] = {1, 2, 8, 10, 10, 12, 19}, x = 20

Output: 6

Explanation: 19 is the largest element in

arr[] smaller than 20

Input : arr[] = {1, 2, 8, 10, 10, 12, 19}, x = 0

Output : -1

Explanation: Since floor doesn’t exist, output is -1.

**Code:**

class Solution {

public static int Floor(int[] arr, int k) {

int n=arr.length;

int l=0;

int r=n-1;

int ind=-1;

while(l<=r){

int mid=l+(r-l)/2;

if(arr[mid]==k) return mid;

else if(arr[mid]<k){

ind=mid;

l=mid+1;

}

else r=mid-1;

}

return ind;

}

public static void main(String[] ars){

int k=0;

int arr[] = {1, 2, 8, 10, 11, 12, 19};

System.out.println(Floor(arr,k));

}

}

**Output:**

****

1. **Check equal arrays**

Given two arrays, **arr1**and **arr2**of equal length**N**, the task is to determine if the given arrays are equal or not. Two arrays are considered equal if:

* Both arrays contain the same set of elements.
* The arrangements (or permutations) of elements may be different.
* If there are repeated elements, the counts of each element must be the same in both arrays.

Input: arr1[] = {1, 2, 5, 4, 0}, arr2[] = {2, 4, 5, 0, 1}

Output: Yes

Input: arr1[] = {1, 2, 5, 4, 0, 2, 1}, arr2[] = {2, 4, 5, 0, 1, 1, 2}

Output: Yes

Input: arr1[] = {1, 7, 1}, arr2[] = {7, 7, 1}

Output: No

**Code:**

import java.util.\*;

class EqualArrays{

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

// Your code here

if(arr1.length!=arr2.length) return false;

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

for(int i:arr1){

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

}

for(int i:arr2){

if(!hp.containsKey(i)) return false;

hp.put(i,hp.get(i)-1);

if(hp.get(i)==0) hp.remove(i);

}

return hp.isEmpty();

}

public static void main(String[] args){

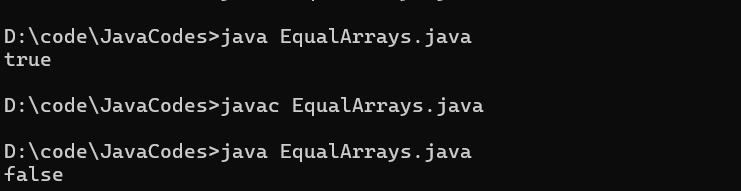
int arr1[] = {1, 7, 1};

int arr2[] = {7, 7, 1};

System.out.println(check(arr1,arr2));

}

}

****

**Time Complexity:O(n)**

**Space Complexity: O(n);**

**4.** **Palindrome Linked List**

Given a singly linked list. The task is to check if the given linked list is palindrome or not.

Examples:

Input: head: 1->2->1->1->2->1

Output: true

Explanation: The given linked list is 1->2->1->1->2->1 , which is a palindrome and Hence, the output is true.

Input: head: 1->2->3->4

Output: false

Explanation: The given linked list is 1->2->3->4, which is not a palindrome and Hence, the output is false.

**Code**:

import java.util.\*;

public class Main {

public static void main(String... argv) {

Scanner scan = new Scanner(System.in);

System.out.println("Enter the Size of the LinkedList :");

int n = scan.nextInt();

System.out.println("Enter the head of the LinkedList :");

int h = scan.nextInt();

Node head = new Node(h);

Node temp = head;

System.out.println("Enter the rem node val :");

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

int num = scan.nextInt();

Node node = new Node(num);

temp.next = node;

temp = temp.next;

}

Node mid = middle(head);

Node secondHead = reverse(mid);

boolean polin = true;

while(head!=null && secondHead!=null){

if(head.val != secondHead.val){

polin = false;

break;

}

head = head.next;

secondHead = secondHead.next;

}

if(polin){

System.out.println("Polindrome");

}else{

System.out.println("Not a Polindrome");

}

}

public static Node middle(Node head){

Node fast = head;

Node slow = head;

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

fast = fast.next.next;

slow = slow.next;

}

return slow;

}

public static Node reverse(Node head){

Node prev = null;

Node temp = head;

while(temp!=null){

Node front = temp.next;

temp.next = prev;

prev = temp;

temp = front;

}

return prev;

}

}

public class Node{

int val;

Node next;

public Node(int val){

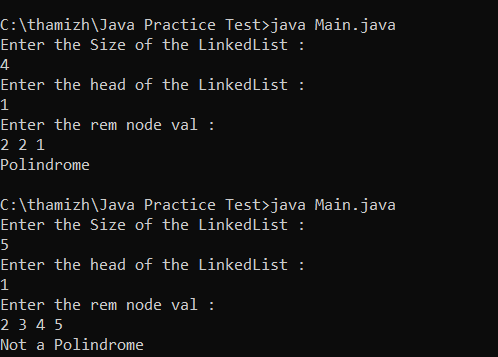
this.val = val;

next = null;

}

}

**Output**:



5. Balanced Tree Check

Given a binary tree, find if it is height balanced or not. A tree is height balanced if difference between heights of left and right subtrees is not more than one for all nodes of tree.

Examples:

Input:

1

/

2

\

3

Output: 0

Explanation: The max difference in height of left subtree and right subtree is 2, which is greater than 1. Hence unbalanced  
  
  
**Code:**

import java.util.\*;

public class Main {

public static void main(String... argv) {

TreeNode root = new TreeNode(1);

TreeNode node2 = new TreeNode(2);

TreeNode node3 = new TreeNode(3);

TreeNode node4 = new TreeNode(4);

TreeNode node5 = new TreeNode(5);

TreeNode node6 = new TreeNode(6);

TreeNode node7 = new TreeNode(7);

root.left = node2;

root.right = node3;

node2.left = node4;

node3.right = node5;

node5.left = node6;

node5.right = node7;

if(helper(root)!=-1){

System.out.println("BALANCED");

}else{

System.out.println("NOT BALANCED");

}

}

public static int helper(TreeNode root){

if(root==null) return 0;

int left = helper(root.left);

int right = helper(root.right);

if(left==-1 || right==-1) return -1;

if(Math.abs(left-right)==-1) return -1;

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

}

}

class TreeNode {

int val;

TreeNode left;

TreeNode right;

TreeNode(int val) {

this.val = val;

left = null;

right = null;

}

}

**Output:**



**6. TRIPLET SUM :**

Given an array **arr[]**of size **n** and an integer **sum**. Find if there’s a triplet in the array which sums up to the given integer **sum**.

**Examples:**

***Input:*** *arr = {12, 3, 4, 1, 6, 9}, sum = 24;****Output:*** *12, 3, 9*

***Explanation:*** *There is a triplet (12, 3 and 9) present   
in the array whose sum is 24.*

***Input:*** *arr = {1, 2, 3, 4, 5}, sum = 9****Output:*** *5, 3, 1*

***Explanation:*** *There is a triplet (5, 3 and 1) present   
in the array whose sum is 9.****Code:***

import java.util.\*;

public class Main {

public static void main(String... argv) {

Scanner scan = new Scanner(System.in);

System.out.println("Enter the Size of the Array :");

int n = scan.nextInt();

int[] arr = new int[n];

System.out.println("Enter the Elements in Array :");

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

arr[i] = scan.nextInt();

}

System.out.println("Enter the Number to find the Triplet ");

int x = scan.nextInt();

Arrays.sort(arr);

boolean found = false;

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

int l = i + 1;

int r = n - 1;

while (l < r) {

int sum = arr[i] + arr[l] + arr[r];

if (sum == x) {

found = true;

break;

} else if (sum < x) {

l++;

} else {

r--;

}

}

}

if(found){

System.out.println("EXIST");

}else{

System.out.println("NOT EXIST");

}

}

}

**Output**:

