1. 0-1 knapsack problem

public class Knapsack {

public static int knapSack(int capacity, int[] weights, int[] values, int n) {

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

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

for (int w = 0; w <= capacity; w++) {

if (weights[i - 1] <= w) {

dp[i][w] = Math.max(values[i - 1] + dp[i - 1][w - weights[i - 1]], dp[i - 1][w]);

} else {

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

}

}

}

return dp[n][capacity];

}

public static void main(String[] args) {

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

int[] values = {1, 4, 5, 7};

int capacity = 7;

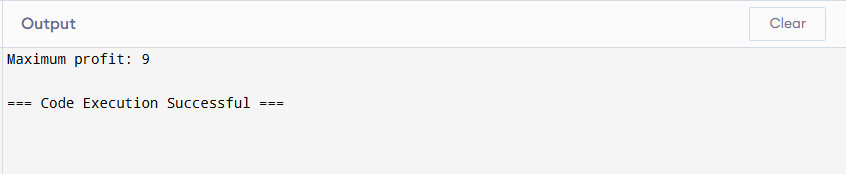
int n = values.length;

int maxProfit = knapSack(capacity, weights, values, n);

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

}

}



1. Floor in sorted array

class Main {

public static int findFloor(int[] arr, int target) {

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

int floor = -1;

while (left <= right) {

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

if (arr[mid] == target) {

return arr[mid];

}

else if (arr[mid] < target) {

floor = arr[mid];

left = mid + 1;

}

else {

right = mid - 1;

}

}

return floor;

}

public static void main(String[] args) {

int[] arr = {1, 2, 4, 6, 10, 12, 14};

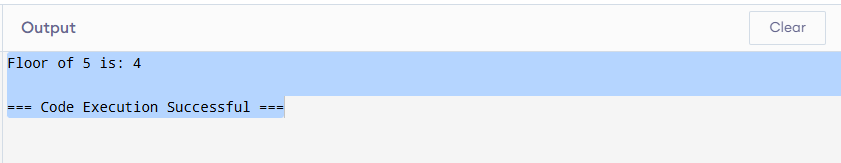
int target = 5;

int floorValue = findFloor(arr, target);

System.out.println("Floor of " + target + " is: " + floorValue);

}

}



3. Check equal arrays

import java.util.HashMap;

class Main {

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

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

return false;

}

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

for (int num : arr1) {

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

}

for (int num : arr2) {

if (!frequencyMap.containsKey(num) || frequencyMap.get(num) == 0) {

return false;

}

frequencyMap.put(num, frequencyMap.get(num) - 1);

}

return true;

}

public static void main(String[] args) {

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

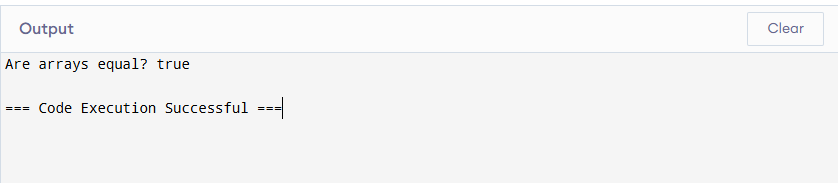
int[] arr2 = {2, 1, 4, 3, 2};

boolean result = areArraysEqual(arr1, arr2);

System.out.println("Are arrays equal? " + result);

}

}



4 . Palindrome linked list

class ListNode {

int val;

ListNode next;

ListNode(int val) {

this.val = val;

this.next = null;

}

}

class Main {

public static boolean isPalindrome(ListNode head) {

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

return true; // A single node or empty list is a palindrome

}

ListNode slow = head;

ListNode fast = head;

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

slow = slow.next;

fast = fast.next.next;

}

ListNode secondHalf = reverseList(slow);

ListNode firstHalf = head;

ListNode secondHalfCopy = secondHalf; // To restore the list later

while (secondHalf != null) {

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

return false;

}

firstHalf = firstHalf.next;

secondHalf = secondHalf.next;

}

reverseList(secondHalfCopy);

return true;

}

private static ListNode reverseList(ListNode head) {

ListNode prev = null;

while (head != null) {

ListNode nextNode = head.next;

head.next = prev;

prev = head;

head = nextNode;

}

return prev;

}

public static void main(String[] args) {

ListNode head = new ListNode(1);

head.next = new ListNode(2);

head.next.next = new ListNode(2);

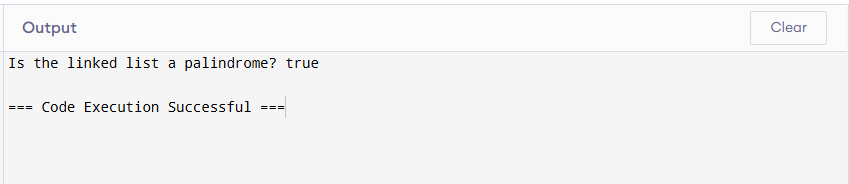
head.next.next.next = new ListNode(1);

boolean result = isPalindrome(head);

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

}

}



5. Balanced tree check

class TreeNode {

int val;

TreeNode left, right;

TreeNode(int val) {

this.val = val;

this.left = null;

this.right = null;

}

}

class Main {

public static boolean isBalanced(TreeNode root) {

return checkHeight(root) != -1;

}

private static int checkHeight(TreeNode node) {

if (node == null) {

return 0; // Base case: empty subtree has height 0

}

int leftHeight = checkHeight(node.left);

if (leftHeight == -1) {

return -1; // Left subtree is unbalanced

}

int rightHeight = checkHeight(node.right);

if (rightHeight == -1) {

return -1; // Right subtree is unbalanced

}

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

return -1; // Current node is unbalanced

}

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

}

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);

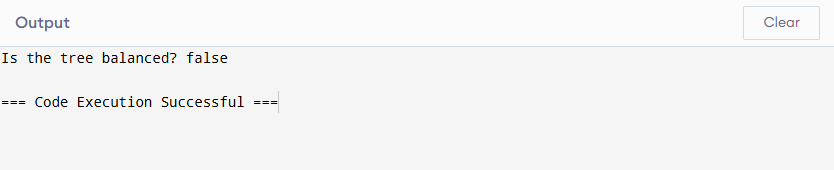
root.left.left.left = new TreeNode(6);

boolean result = isBalanced(root);

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

}

}



6. Triplet sum in array

import java.util.Arrays;

public class Main {

public static void findTriplets(int[] arr, int targetSum) {

Arrays.sort(arr); // Sort the array to simplify finding triplets

int n = arr.length;

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

int left = i + 1;

int right = n - 1;

while (left < right) {

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

if (currentSum == targetSum) {

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

left++;

right--;

} else if (currentSum < targetSum) {

left++;

} else {

right--;

}

}

}

}

public static void main(String[] args) {

int[] arr = {12, 3, 4, 1, 6, 9};

int targetSum = 24;

findTriplets(arr, targetSum);

}

}

