Pavadareni R 11.11.2024

**Dsa Program**

1. **0 - 1 Knapsack Problem**

public class Knapsack {

public static int knapsack(int[] val, int[] wt, int capacity) {

int n = val.length;

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

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

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

// If we do not take the i-th item

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

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

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

}

}

}

return dp[n][capacity];

}

public static void main(String[] args) {

int[] val = {60, 100, 120};

int[] wt = {10, 20, 30};

int capacity = 50;

System.out.println(knapsack(val, wt, capacity));

}

}

**Output**

****

**Time Complexity**

O(n\*capacity)

1. **Floor in sorted array**

**Code**

public class FloorInSortedArray {

    public static int floor(int[] arr, int x) {

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

        int floor = -1;

        while (left <= right) {

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

            if (arr[mid] == x) {

                return arr[mid];

            }

            if (arr[mid] < x) {

                floor = arr[mid];

                left = mid + 1;

            } else {

                right = mid - 1;

            }

        }

        return floor;

    }

    public static void main(String[] args) {

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

        int x = 5;

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

        x = 20;

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

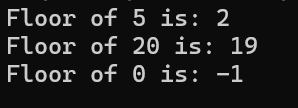
        x = 0;

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

    }

}

**Output**

****

**Time complexity: O(n)**

1. Check equal arrays

import java.util.HashMap;

public class ArrayEquality {

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

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

            return false;

        }

        HashMap<Integer, Integer> freqMap1 = new HashMap<>();

        HashMap<Integer, Integer> freqMap2 = new HashMap<>();

        for (int num : arr1) {

            freqMap1.put(num, freqMap1.getOrDefault(num, 0) + 1);

        }

        for (int num : arr2) {

            freqMap2.put(num, freqMap2.getOrDefault(num, 0) + 1);

        }

        return freqMap1.equals(freqMap2);

    }

    public static void main(String[] args) {

        int[] arr1 = { 1, 2, 5, 4, 0 };

        int[] arr2 = { 2, 4, 5, 0, 1 };

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

        int[] arr3 = { 1, 2, 5 };

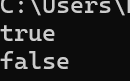
        int[] arr4 = { 2, 4, 15 };

        System.out.println(areArraysEqual(arr3, arr4));

    }

}

Output



Time Complexity: O(n)

1. Palindrome linked list

class ListNode {

    int val;

    ListNode next;

    ListNode(int x) {

        val = x;

    }

}

public class PalindromeLinkedList {

    public static boolean isPalindrome(ListNode head) {

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

            return true;

        }

        ListNode middle = findMiddle(head);

        ListNode secondHalf = reverseList(middle.next);

        ListNode firstHalf = head;

        while (secondHalf != null) {

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

                return false;

            }

            firstHalf = firstHalf.next;

            secondHalf = secondHalf.next;

        }

        return true;

    }

    private static ListNode findMiddle(ListNode head) {

        ListNode slow = head, fast = head;

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

            slow = slow.next;

            fast = fast.next.next;

        }

        return slow;

    }

    private static ListNode reverseList(ListNode head) {

        ListNode prev = null;

        ListNode curr = head;

        while (curr != null) {

            ListNode nextNode = curr.next;

            curr.next = prev;

            prev = curr;

            curr = nextNode;

        }

        return prev;

    }

    public static void main(String[] args) {

        ListNode head = new ListNode(1);

        head.next = new ListNode(2);

        head.next.next = new ListNode(3);

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

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

        System.out.println(isPalindrome(head));

        ListNode head2 = new ListNode(1);

        head2.next = new ListNode(2);

        head2.next.next = new ListNode(3);

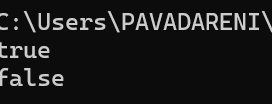
        head2.next.next.next = new ListNode(4);

        System.out.println(isPalindrome(head2));

    }

}

Output



Time Complexity O(n)

1. Balanced tree check

class TreeNode {

int val;

TreeNode left;

TreeNode right;

TreeNode(int x) {

val = x;

}

}

public class BalancedBinaryTree {

public static boolean isBalanced(TreeNode root) {

return checkHeight(root) != -1;

}

private static int checkHeight(TreeNode node) {

if (node == null) {

return 0;

}

int leftHeight = checkHeight(node.left);

if (leftHeight == -1) {

return -1;

}

int rightHeight = checkHeight(node.right);

if (rightHeight == -1) {

return -1;

}

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

return -1;

}

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);

System.out.println(isBalanced(root));

TreeNode unbalancedRoot = new TreeNode(1);

unbalancedRoot.left = new TreeNode(2);

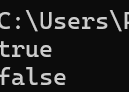
unbalancedRoot.left.left = new TreeNode(3);

System.out.println(isBalanced(unbalancedRoot));

}

}

Output



Time complexity O(n)

1. Triplet sum in array

import java.util.Arrays;

public class TripletSum {

public static boolean hasTripletSum(int[] arr, int sum) {

Arrays.sort(arr);

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

int left = i + 1;

int right = arr.length - 1;

while (left < right) {

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

if (currentSum == sum) {

return true;

}

if (currentSum < sum) {

left++;

} else {

right--;

}

}

}

return false;

}

public static void main(String[] args) {

int[] arr1 = { 1, 4, 6, 8, 10 };

int sum1 = 22;

System.out.println(hasTripletSum(arr1, sum1));

int[] arr2 = { 1, 2, 3, 4, 5 };

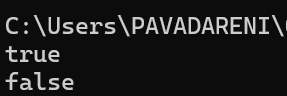
int sum2 = 15;

System.out.println(hasTripletSum(arr2, sum2));

}

}

Output



Time complexity: O(n)