DAY1-11/11/24

GEEKSFORGEEKS

0-1 knapsack problem

Code:

class Solution {

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

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

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

} else {

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

}

}

}

return dp[n][capacity];

}

}

Output:

A screenshot of a black screen

Description automatically generated

Time complexity: O(nxcapacity)

1. Floor in sorted array

Code:

class Solution {

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

int low = 0, high = arr.length - 1;

int floorIndex = -1;

while (low <= high) {

int mid = low + (high - low) / 2;

if (arr[mid] <= k) {

floorIndex = mid;

low = mid + 1;

} else {

high = mid - 1;

}

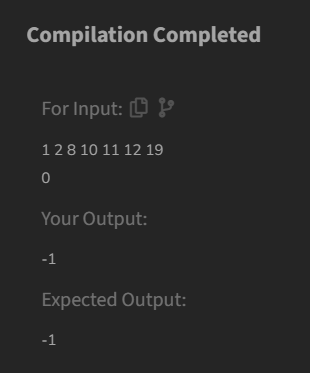
}

return floorIndex;

}

}

Output:



Time complexity: O(logn)

1. Check equal arrays

Code:

import java.util.HashMap;

class Solution {

public static boolean check(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)) {

return false;

}

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

if (frequencyMap.get(num) == 0) {

frequencyMap.remove(num);

}

}

return frequencyMap.isEmpty();

}

}

1. Palindrome linked list

Code:

class Solution {

// Function to check whether the list is palindrome.

boolean isPalindrome(Node head) {

if (head == null || head.next == null) return true;

// Find the middle of the linked list

Node slow = head, fast = head;

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

slow = slow.next;

fast = fast.next.next;

}

// Reverse the second half of the list

Node secondHalfHead = reverseList(slow);

// Compare the first half and the reversed second half

Node firstHalfHead = head;

Node secondHalfIterator = secondHalfHead;

boolean isPalindrome = true;

while (secondHalfIterator != null) {

if (firstHalfHead.data != secondHalfIterator.data) {

isPalindrome = false;

break;

}

firstHalfHead = firstHalfHead.next;

secondHalfIterator = secondHalfIterator.next;

}

// Restore the list (optional)

reverseList(secondHalfHead);

return isPalindrome;

}

// Helper function to reverse a linked list

private Node reverseList(Node head) {

Node prev = null;

while (head != null) {

Node next = head.next;

head.next = prev;

prev = head;

head = next;

}

return prev;

}

}A screenshot of a black screen

Description automatically generated

Time complexity: O(n)

1. Balanced tree check

Code:

class Tree {

// Function to check whether a binary tree is balanced or not.

boolean isBalanced(Node root) {

return checkHeight(root) != -1;

}

// Helper function to calculate height and check balance

private int checkHeight(Node node) {

if (node == null) {

return 0; // Base case: height of an empty tree is 0

}

// Recursively get heights of left and right subtrees

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

// Check the balance condition for the current node

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

return -1; // Current node is unbalanced

}

// Return the height of the current node

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

}

}

A screenshot of a phone

Description automatically generated

Time complexity: O(n)

1. Triplet sum in array

class Solution {

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

Arrays.sort(arr);

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

int left = i + 1;

int right = n - 1;

while (left < right) {

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

if (sum == x) {

return true;

}

else if (sum < x) {

left++;

}

else {

right--;

}

}

}

return false;

}

}

A screenshot of a black screen

Description automatically generated

Time complexity: O(n^2)