**Coding Practice day 2**

**1.knapsack problem**

**Code**

public class Main {

static int calcMax(int[] v, int[] w, int cap, int i, int[][] dp) {

if (i == 0 || cap == 0) {

return 0;

}

if (dp[i][cap] != 0) {

return dp[i][cap];

}

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

int incl = v[i - 1] + calcMax(v, w, cap - w[i - 1], i - 1, dp);

int excl = calcMax(v, w, cap, i - 1, dp);

dp[i][cap] = Math.max(incl, excl);

return dp[i][cap];

} else {

dp[i][cap] = calcMax(v, w, cap, i - 1, dp);

return dp[i][cap];

}

}

static int knapSack(int cap, int v[], int w[]) {

int dp[][] = new int[v.length + 1][cap + 1];

return calcMax(v, w, cap, v.length, dp);

}

public static void main(String[] args) {

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

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

int capacity = 50;

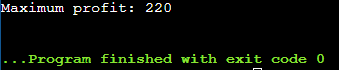
int maxProfit = knapSack(capacity, values, weights);

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

}

}

**Output**

****

TIME Complexity : O(n\*capacity)

Space Complexity : O(n\*capacity)

**2. Floor in sorted array**

**Code**

import java.util.Scanner;

public class Main{

static int findFloor(int[] array, int target) {

int length = array.length;

for (int index = 0; index < length; index++) {

if (array[index] > target)

return index - 1;

}

return -1;

}

public static void main(String[] args) {

Scanner input = new Scanner(System.in);

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

int size = input.nextInt();

int[] array = new int[size];

System.out.println("Enter the elements of the array in sorted order:");

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

array[i] = input.nextInt();

}

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

int target = input.nextInt();

int result = findFloor(array, target);

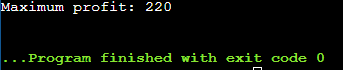
System.out.println("Output: " + result);

input.close();

}

}

**Output**

****

Time Complexity : O(n)

Space Complexity: O(1)

**3.**

**Code**

import java.util.HashMap;

import java.util.Map;

import java.util.Scanner;

public class Main {

public static boolean areArraysEqual(int[] array1, int[] array2) {

if (array1.length != array2.length) {

return false;

}

Map<Integer, Integer> frequencyMap1 = new HashMap<>();

Map<Integer, Integer> frequencyMap2 = new HashMap<>();

for (int element : array1) {

frequencyMap1.put(element, frequencyMap1.getOrDefault(element, 0) + 1);

}

for (int element : array2) {

frequencyMap2.put(element, frequencyMap2.getOrDefault(element, 0) + 1);

}

return frequencyMap1.equals(frequencyMap2);

}

public static void main(String[] args) {

Scanner input = new Scanner(System.in);

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

int size1 = input.nextInt();

int[] array1 = new int[size1];

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

for (int index = 0; index < size1; index++) {

array1[index] = input.nextInt();

}

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

int size2 = input.nextInt();

int[] array2 = new int[size2];

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

for (int index = 0; index < size2; index++) {

array2[index] = input.nextInt();

}

boolean isEqual = areArraysEqual(array1, array2);

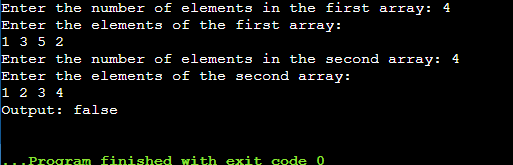
System.out.println("Output: " + isEqual);

input.close();

}

}

**Output**

****

Time Complexity: O(N)

Space Complexity : O(N)

**4.** **Palindrome linked list**

**Code**

import java.util.Scanner;

class Node {

int value;

Node next;

Node(int value) {

this.value = value;

this.next = null;

}

}

class Main {

Node reverseList(Node head) {

Node prevNode = null;

Node currentNode = head;

Node nextNode;

while (currentNode != null) {

nextNode = currentNode.next;

currentNode.next = prevNode;

prevNode = currentNode;

currentNode = nextNode;

}

return prevNode;

}

boolean areIdentical(Node list1, Node list2) {

for (; list1 != null && list2 != null; list1 = list1.next, list2 = list2.next)

if (list1.value != list2.value) return false;

return true;

}

boolean isPalindrome(Node head) {

int length = 0;

Node temp;

for (temp = head; temp != null; temp = temp.next) length++;

if (length < 2) return true;

temp = head;

int midPoint = (length - 1) / 2;

while (midPoint > 0) {

temp = temp.next;

midPoint--;

}

Node secondHalf = temp.next;

temp.next = null;

secondHalf = reverseList(secondHalf);

boolean result = areIdentical(head, secondHalf);

secondHalf = reverseList(secondHalf);

temp.next = secondHalf;

return result;

}

public static Node createLinkedList(int[] arr) {

Node head = null, tail = null;

for (int value : arr) {

Node newNode = new Node(value);

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.value + " ");

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);

Main solution = new Main();

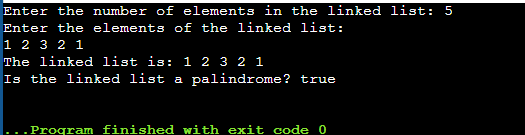
boolean result = solution.isPalindrome(head);

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

}

}

**Output**

****

Time Complexity: O(n)

Space Complexity : O(1)

**5.** **Balanced Tree Check**

**Code**

import java.util.Scanner;

import java.util.LinkedList;

import java.util.Queue;

class TreeNode {

int value;

TreeNode left, right;

TreeNode(int val) {

value = val;

left = right = null;

}

}

class BinaryTree {

boolean checkBalance(TreeNode root) {

Boolean[] result = {true};

evaluateHeight(root, result);

return result[0];

}

static int evaluateHeight(TreeNode node, Boolean[] result) {

if (node == null) {

return 0;

}

int leftHeight = evaluateHeight(node.left, result);

int rightHeight = evaluateHeight(node.right, result);

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

result[0] = false;

}

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

}

public static TreeNode buildTreeFromLevelOrder(int[] values) {

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

TreeNode root = new TreeNode(values[0]);

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

queue.add(root);

int i = 1;

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

TreeNode currentNode = queue.poll();

if (values[i] != -1) {

currentNode.left = new TreeNode(values[i]);

queue.add(currentNode.left);

}

i++;

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

currentNode.right = new TreeNode(values[i]);

queue.add(currentNode.right);

}

i++;

}

return root;

}

}

public class Main {

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

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

int count = scanner.nextInt();

int[] values = new int[count];

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

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

values[i] = scanner.nextInt();

}

TreeNode root = BinaryTree.buildTreeFromLevelOrder(values);

BinaryTree tree = new BinaryTree();

boolean result = tree.checkBalance(root);

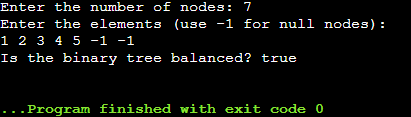
System.out.println("Is the binary tree balanced? " + result);

scanner.close();

}

}

**Output**

****

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

**6.Triplet sum**

**Code**

import java.util.HashMap;

import java.util.Map;

import java.util.Scanner;

class Main {

public static boolean hasTripletWithSum(int[] numbers, int length, int targetSum) {

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

for (int num : numbers) {

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

}

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

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

int currentSum = numbers[i] + numbers[j];

int complement = targetSum - currentSum;

if (frequencyMap.containsKey(complement) &&

(complement != numbers[i] || frequencyMap.get(numbers[i]) != 1) &&

(complement != numbers[j] || frequencyMap.get(numbers[j]) != 1) &&

(numbers[i] != numbers[j] || frequencyMap.get(numbers[j]) > 2)) {

return true;

}

}

}

return false;

}

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

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

int length = scanner.nextInt();

int[] numbers = new int[length];

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

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

numbers[i] = scanner.nextInt();

}

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

int targetSum = scanner.nextInt();

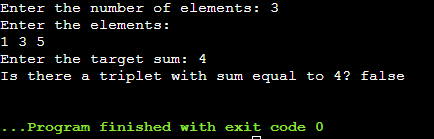
boolean result = hasTripletWithSum(numbers, length, targetSum);

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

}

}

**Output**

****

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

Space Complexity :  O(n)