**DSA PRACTICE – 2 -11/11/24**

1. 0-1 Knapsak Problem

public class KnapsackProblem{

public static int knapSack(int W, int[] wt, int[] val, int n) {

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

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

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

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

dp[i][w] = 0;

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

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

} else {

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

}

}

}

return dp[n][W];

}

public static void main(String[] args) {

int W = 50;

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

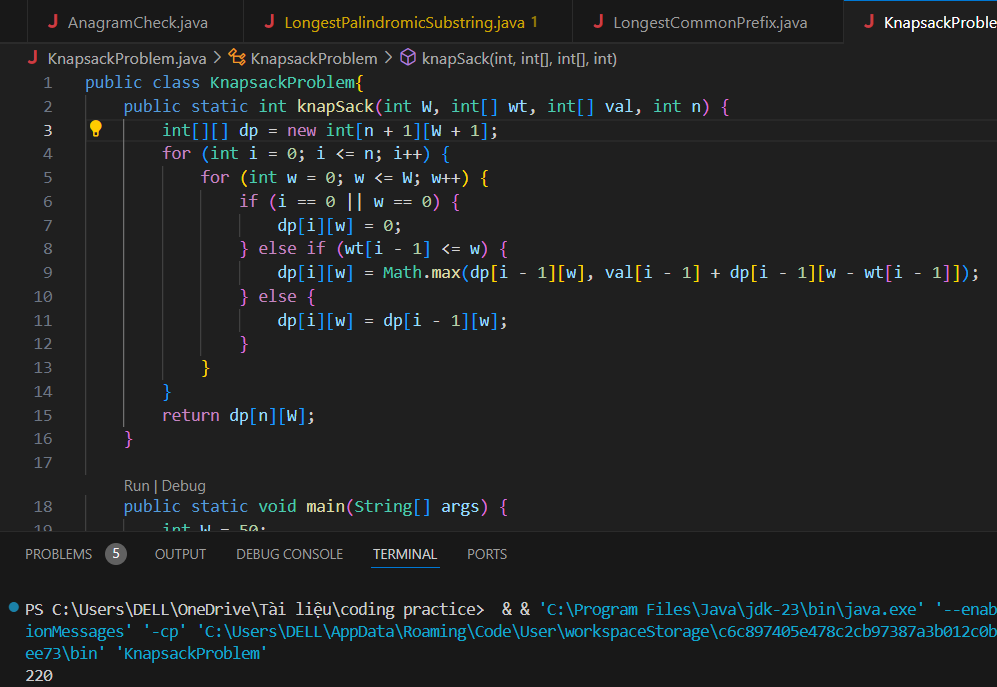
int[] val = {60, 100, 120};

int n = wt.length;

System.out.println(knapSack(W, wt, val, n));

}

}



Time Complexity : O(n\*W)

Space Complexity : O(n\*W)

2.Floor of Sorted Array

import java.util.\*;

public class FloorSortedArray{

    static int check(int[] arr, int target){

        int left=0;

        int right=arr.length -1;

        while(left<=right){

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

            if(arr[mid]==target){

                return mid;

            }

            else if(arr[mid]>target){

                right=mid-1;

            }

            else{

                left=mid+1;

            }

        }

        return left;

    }

    public static void main(String[] args) {

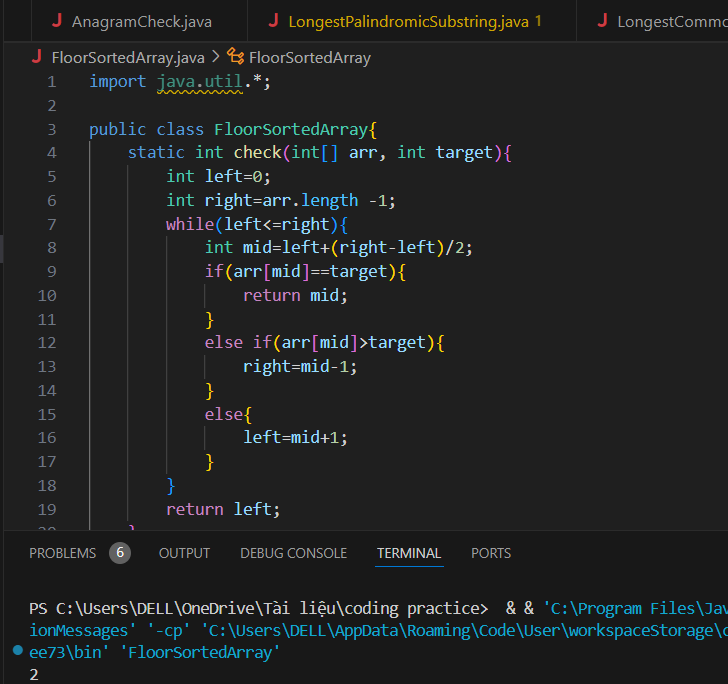
        int[] arr={1,3,5,6};

        int target=4;

        System.out.println(check(arr,target));

    }

}



Time Complexity : O(logn)

Space Complexity : O(1)

3.Check Equal Arrays

import java.util.HashMap;

class EqualArray {

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

        int N = arr1.length;

        int M = arr2.length;

        if (N != M)

            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[] = { 3, 5, 2, 5, 2 };

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

        if (areEqual(arr1, arr2))

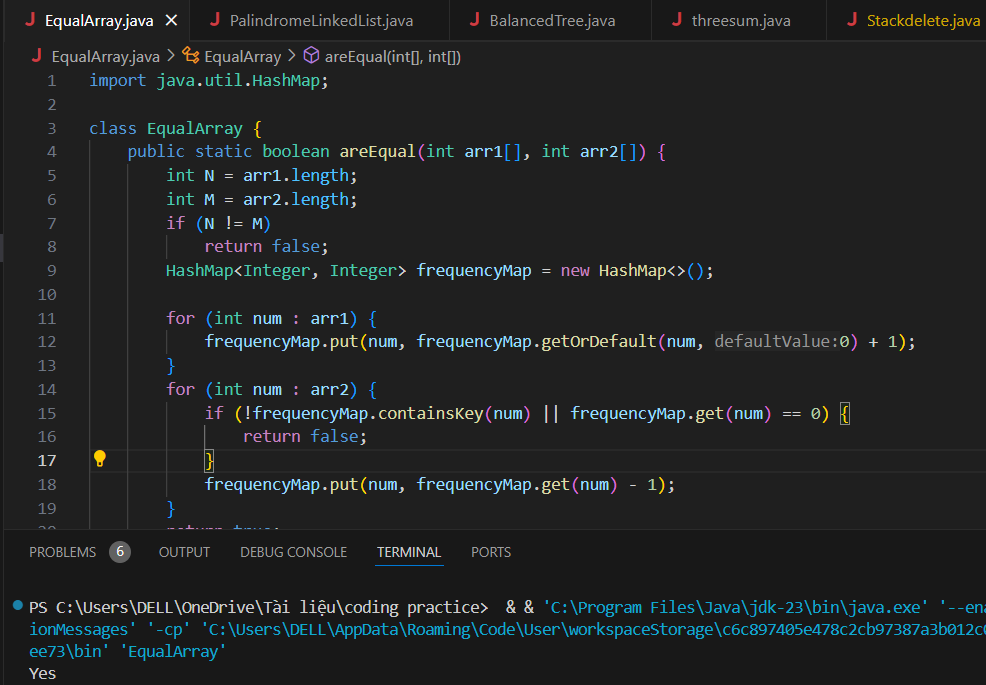
            System.out.println("Yes");

        else

            System.out.println("No");

    }

}



Time Complexity : O(n)

Space Complexity : O(1)

4. Palindrome Linkedlist

import java.util.\*;

class ListNode {

    int val;

    ListNode next;

    ListNode(int x) { val = x; }

}

class PalindromeLinkedList {

    public boolean isPalindrome(ListNode head) {

        List<Integer> values = new ArrayList<>();

        ListNode currentNode = head;

        while (currentNode != null) {

            values.add(currentNode.val);

            currentNode = currentNode.next;

        }

        int left = 0;

        int right = values.size() - 1;

        while (left < right) {

            if (!values.get(left).equals(values.get(right))) {

                return false;

            }

            left++;

            right--;

        }

        return true;

    }

    public static void main(String[] args) {

        ListNode head1 = new ListNode(1);

        head1.next = new ListNode(2);

        head1.next.next = new ListNode(2);

        head1.next.next.next = new ListNode(1);

        PalindromeLinkedList solution = new PalindromeLinkedList();

        System.out.println(solution.isPalindrome(head1));

    }

}

class ListNode {

    int val;

    ListNode next;

    ListNode(int x) { val = x; }

}

class PalindromeLinkedList {

    public boolean isPalindrome(ListNode head) {

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

            return true;

        }

        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;

        boolean isPalindrome = true;

        while (secondHalf != null) {

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

                isPalindrome = false;

                break;

            }

            firstHalf = firstHalf.next;

            secondHalf = secondHalf.next;

        }

        reverseList(secondHalfCopy);

        return isPalindrome;

    }

    private 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 head1 = new ListNode(1);

        head1.next = new ListNode(2);

        head1.next.next = new ListNode(2);

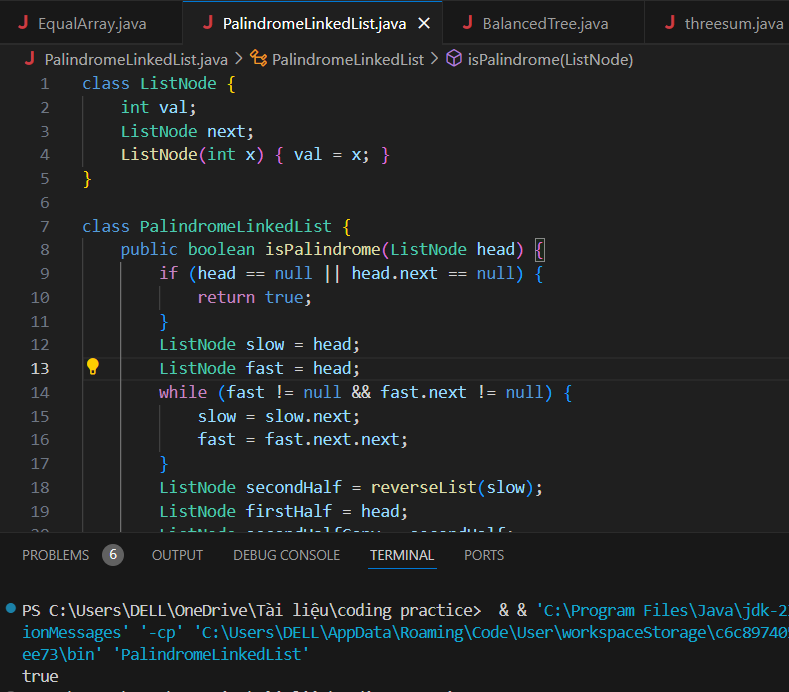
        head1.next.next.next = new ListNode(1);

        PalindromeLinkedList solution = new PalindromeLinkedList();

        System.out.println(solution.isPalindrome(head1));

    }

}



Time Complexity : O(n)

Space Complexity : O(1)

5. Balanced Binary Tree

class TreeNode {

    int val;

    TreeNode left;

    TreeNode right;

    TreeNode(int x) { val = x; }

}

public class BalancedTree {

    private int height(TreeNode node) {

        if (node == null) return 0;

        int leftHeight = height(node.left);

        if (leftHeight == -1) return -1;

        int rightHeight = height(node.right);

        if (rightHeight == -1) return -1;

        if (Math.abs(leftHeight - rightHeight) > 1) return -1;

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

    }

    public boolean isBalanced(TreeNode root) {

        return height(root) != -1;

    }

    public static void main(String[] args) {

        BalancedTree solution = new BalancedTree();

        TreeNode root1 = new TreeNode(3);

        root1.left = new TreeNode(9);

        root1.right = new TreeNode(20);

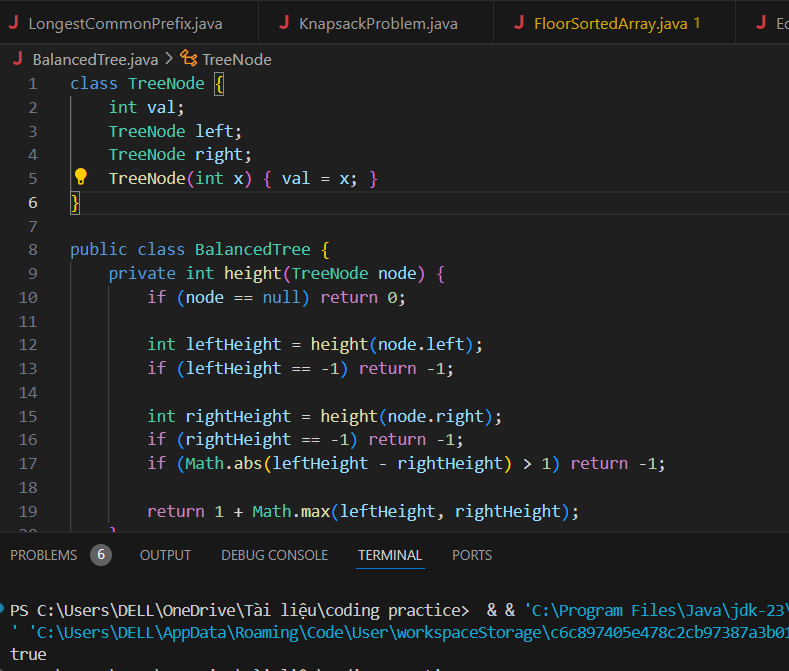
        root1.right.left = new TreeNode(15);

        root1.right.right = new TreeNode(7);

        System.out.println(solution.isBalanced(root1));

    }

}



Time Complexity : O(n)

Space Complexity : O(h)

6. Triplet Sum

import java.util.\*;

public class threesum {

    public static List<List<Integer>> check(int[] nums) {

            List<List<Integer>> result = new ArrayList<>();

            Arrays.sort(nums);

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

                if (i > 0 && nums[i] == nums[i - 1]) continue;

                int left = i + 1;

                int right = nums.length - 1;

                int target = -nums[i];

                while (left < right) {

                    int sum = nums[left] + nums[right];

                    if (sum == target) {

                        result.add(Arrays.asList(nums[i], nums[left], nums[right]));

                        left++;

                        right--;

                        while (left < right && nums[left] == nums[left - 1]) left++;

                        while (left < right && nums[right] == nums[right + 1]) right--;

                    } else if (sum < target) {

                        left++;

                    } else {

                        right--;

                    }

                }

            }

            return result;

        }

        public static void main(String[] args) {

            int[] nums1 = {-1, 0, 1, 2, -1, -4};

            System.out.println(check(nums1));

    }

}



Time Complexity : O(n^2)

Space Complexity : O(n^2)