DSA PRACTISE – 2 – 11/11/2024 – S RAMYA - AIML

1.Palindrome Linked List

package dsa;

import java.util.Stack;

class ListNode{

    int val;

    ListNode next;

    ListNode(int val){

        this.val = val;

    }

}

public class PalindromeLL {

    public static boolean isPalindrome(ListNode head) {

        Stack<Integer> s = new Stack<>();

        ListNode mover = head;

        while (mover != null){

            s.push(mover.val);

            mover = mover.next;

        }

        mover = head;

        while (!s.empty()){

            if (s.pop() != mover.val) return false;

            mover = mover.next;

        }

        return true;

    }

    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(4);

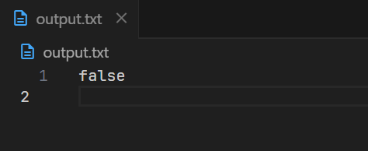
        System.out.println(isPalindrome(head));

    }

}

TC: O(N)

SC: O(N)



2.Check Equal Arrays

package dsa;

public class EqualArrays {

    public static void main(String[] args) {

        int[] arr1 = {1,2,3,4,5};

        int[] arr2 = {1,2,3,4,5};

        int n = arr1.length;

        int m = arr2.length;

        int ptr1 = 0;

        int ptr2 = 0;

        while (ptr1<n && ptr2<m){

            if (arr1[ptr1] != arr2[ptr2]){

                System.out.println("not equal");

                break;

            }

            ptr1++;

            ptr2++;

        }

        if (ptr1 != n || ptr2 != m) System.out.println("not equal");

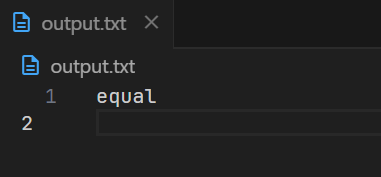
        else System.out.println("equal");

    }

}

TC: O(N)

SC: O(1)



3.Balanced Tree Check

A **height-balanced** binary tree is a binary tree in which the depth of the two subtrees of every node never differs by more than one.

package dsa;

class TreeNode{

    int val;

    TreeNode left;

    TreeNode right;

    TreeNode(int val){

        this.val = val;

        this.left = left;

        this.right = right;

    }

}

public class HeightBalancedTree {

    public static int depth(TreeNode node){

        if (node == null) return 0;

        int l = depth(node.left);

        int r = depth(node.right);

        return Math.max(l,r)+1;

    }

    public static boolean isBalanced(TreeNode root) {

        if (root == null) return true;

        int l = depth(root.left);

        int r = depth(root.right);

        if (Math.abs(l-r) <= 1 && isBalanced(root.left) && isBalanced(root.right)){

            return true;

        }

        return false;

    }

    public static void main(String[] args) {

        TreeNode root = new TreeNode(25);

        root.left = new TreeNode(20);

        root.right = new TreeNode(36);

        root.left.left = new TreeNode(10);

        root.left.left.left = new TreeNode(5);

        root.left.right = new TreeNode(22);

        root.left.left.right = new TreeNode(12);

        root.right.left = new TreeNode(30);

        root.right.left.left = new TreeNode(28);

        root.right.right = new TreeNode(40);

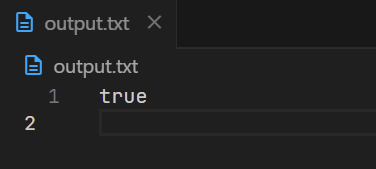
        System.out.println(isBalanced(root));

    }

}

TC: O(N2)

SC: O(H)



4.Triplet Sum

package dsa;

import java.util.\*;

class TripletSum {

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

        Set<List<Integer>> set = new HashSet<>();

        Set<Integer> ele = new HashSet<Integer>();

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

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

                int temp = -(nums[i]+nums[j]);

                if (ele.contains(temp)){

                    List<Integer> lst = Arrays.asList(nums[i], nums[j], temp);

                    lst.sort(null);

                    set.add(lst);

                }

                ele.add(nums[j]);

            }

            ele.clear();

        }

        List<List<Integer>> ans = new ArrayList<>(set);

        return ans;

    }

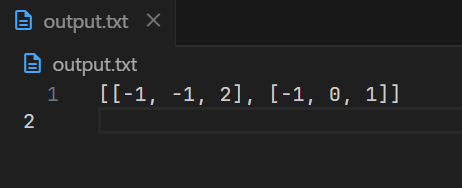
    public static void main(String[] args) {

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

        System.out.println(threeSum(nums));

    }

}



TC: O(N2)

SC: O(N) + O(2\*NO.OF.UNIQUE TRIPLETS)

5.Floor in Sorted Array

package dsa;

public class FloorArray {

    public static int searchInsert(int[] nums, int target) {

        int left = 0;

        int right = nums.length - 1;

        while (left <= right) {

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

            if (nums[mid] == target) {

                return mid;

            } else if (nums[mid] > target) {

                right = mid - 1;

            } else {

                left = mid + 1;

            }

        }

        return left;

    }

    public static void main(String[] args) {

        int[] arr = {1,2,3,4,5};

        int target = 5;

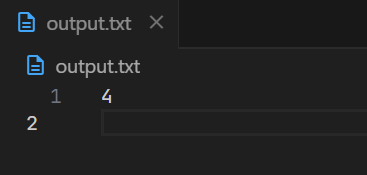
        System.out.println(searchInsert(arr, target));

    }

}

TC: O(logN)

SC: O(1)



6.0/1 Knapsack problem

package dsa;

import java.util.\*;

class Knapsack {

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

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

        for (int i = wt[0]; i <= W; i++) {

            dp[0][i] = val[0];

        }

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

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

                int notTaken = dp[ind - 1][cap];

                int taken = Integer.MIN\_VALUE;

                if (wt[ind] <= cap) {

                    taken = val[ind] + dp[ind - 1][cap - wt[ind]];

                }

                dp[ind][cap] = Math.max(notTaken, taken);

            }

        }

        return dp[n - 1][W];

    }

    public static void main(String args[]) {

        int wt[] = {1, 2, 4, 5};

        int val[] = {5, 4, 8, 6};

        int W = 5;

        int n = wt.length;

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

    }

}

TC: O(N\*W)

SC: O(N\*W)

