CODING QUESTIONS SOLUTIONS

1. import java.util.Stack;

class Solution {

public int evalRPN(String[] tokens) {

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

for (String token : tokens) {

if (token.equals("+") || token.equals("-") || token.equals("\*") || token.equals("/")) {

int b = stack.pop();

int a = stack.pop();

int result = 0;

if (token.equals("+")) {

result = a + b;

} else if (token.equals("-")) {

result = a - b;

} else if (token.equals("\*")) {

result = a \* b;

} else if (token.equals("/")) {

result = a / b;

}

stack.push(result);

} else {

stack.push(Integer.parseInt(token));

}

}

return stack.pop();

}

}

1. import java.util.Stack;

class Solution {

public int largestRectangleArea(int[] heights) {

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

int max = 0;

int n = heights.length;

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

int h = (i == n) ? 0 : heights[i];

while (!stack.isEmpty() && h < heights[stack.peek()]) {

int height = heights[stack.pop()];

int width = stack.isEmpty() ? i : i - stack.peek() - 1;

max = Math.max(max, height \* width);

}

stack.push(i);

}

return max;

}

}

1. class Solution {

public ListNode reverseKGroup(ListNode head, int k) {

int count = 0;

ListNode ptr = head;

while (ptr != null && count < k) {

ptr = ptr.next;

count++;

}

if (count == k) {

ListNode prev = reverseKGroup(ptr, k);

while (count-- > 0) {

ListNode temp = head.next;

head.next = prev;

prev = head;

head = temp;

}

head = prev;

}

return head;

}

}

1. class Solution {

public ListNode oddEvenList(ListNode head) {

if (head == null) return head;

ListNode odd = head;

ListNode even = head.next;

ListNode evenHead = even;

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

odd.next = even.next;

odd = odd.next;

even.next = odd.next;

even = even.next;

}

odd.next = evenHead;

return head;

}

}

1. class Solution {

public int minimumRemoval(int[] nums) {

int n = nums.length;

int count = 0;

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

if (nums[i] < nums[i - 1]) {

count++;

}

}

return count;

}

}

1. class Solution {

public int[] finalPrices(int[] prices) {

int n = prices.length;

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

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

if (prices[j] <= prices[i]) {

prices[i] -= prices[j];

break;

}

}

}

return prices;

}

}

1. class Solution {

public ListNode removeZeroSumSublists(ListNode head) {

ListNode dummy = new ListNode(0);

dummy.next = head;

Map<Integer, ListNode> map = new HashMap<>();

int sum = 0;

for (ListNode curr = dummy; curr != null; curr = curr.next) {

sum += curr.val;

map.put(sum, curr);

}

sum = 0;

for (ListNode curr = dummy; curr != null; curr = curr.next) {

sum += curr.val;

curr.next = map.get(sum).next;

}

return dummy.next;

}

}