**Syed Nadeem G**

**SDE – Practice Day – 7**

(19/11/2024)

1. **Next Permutation:**

Code:

import java.util.\*;

class Practice {

public void nextPermutation(int[] nums) {

ArrayList<Integer> ls = new ArrayList<>();

for (int num : nums) ls.add(num);

int ind = -1, n = ls.size();

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

if (ls.get(i) < ls.get(i + 1)) {

ind = i;

break;

}

}

if (ind == -1) {

Collections.*reverse*(ls);

} else {

for (int i = n - 1; i > ind; i--) {

if (ls.get(i) > ls.get(ind)) {

int t = ls.get(i);

ls.set(i, ls.get(ind));

ls.set(ind, t);

break;

}

}

ls.subList(ind + 1, n).sort(null);

}

for (int i = 0; i < n; i++) nums[i] = ls.get(i);

}

public static void main(String[] args) {

Scanner sc = new Scanner(System.***in***);

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

int n = sc.nextInt();

int[] nums = new int[n];

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

for (int i = 0; i < n; i++) nums[i] = sc.nextInt();

Practice solution = new Practice();

solution.nextPermutation(nums);

System.***out***.println("Next permutation:");

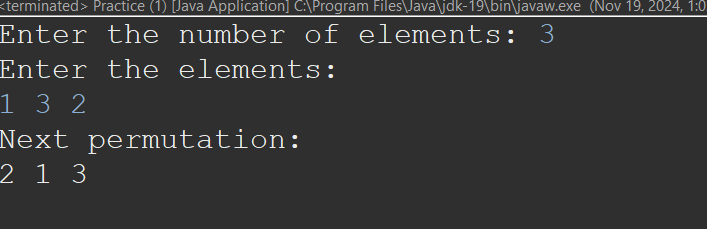
for (int num : nums) System.***out***.print(num + " ");

sc.close();

}

}

Output:



Time Complexity: O(n)

1. **Spiral Matrix:**

Code:

import java.util.ArrayList;

import java.util.List;

import java.util.Scanner;

class Practice {

enum *Dir* {

***Right***, ***Down***, ***Left***, ***Up***

}

public List<Integer> spiralOrder(int[][] matrix) {

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

int n = matrix.length;

int m = matrix[0].length;

boolean[][] vis = new boolean[n][m];

int i = 0, j = 0;

*Dir* dir = *Dir*.***Right***;

while (ls.size() < (n \* m)) {

ls.add(matrix[i][j]);

vis[i][j] = true;

switch (dir) {

case ***Right***:

if ((j + 1) < m && !vis[i][j + 1]) {

j++;

} else {

dir = *Dir*.***Down***;

i++;

}

break;

case ***Down***:

if ((i + 1) < n && !vis[i + 1][j]) {

i++;

} else {

dir = *Dir*.***Left***;

j--;

}

break;

case ***Left***:

if ((j - 1) >= 0 && !vis[i][j - 1]) {

j--;

} else {

dir = *Dir*.***Up***;

i--;

}

break;

case ***Up***:

if ((i - 1) >= 0 && !vis[i - 1][j]) {

i--;

} else {

dir = *Dir*.***Right***;

j++;

}

break;

}

}

return ls;

}

public static void main(String[] args) {

Practice solution = new Practice();

Scanner scanner = new Scanner(System.***in***);

System.***out***.println("Enter the number of rows and columns:");

int n = scanner.nextInt();

int m = scanner.nextInt();

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

int[][] matrix = new int[n][m];

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

for (int j = 0; j < m; j++) {

matrix[i][j] = scanner.nextInt();

}

}

List<Integer> result = solution.spiralOrder(matrix);

System.***out***.println("Spiral Order:");

for (int num : result) {

System.***out***.print(num + " ");

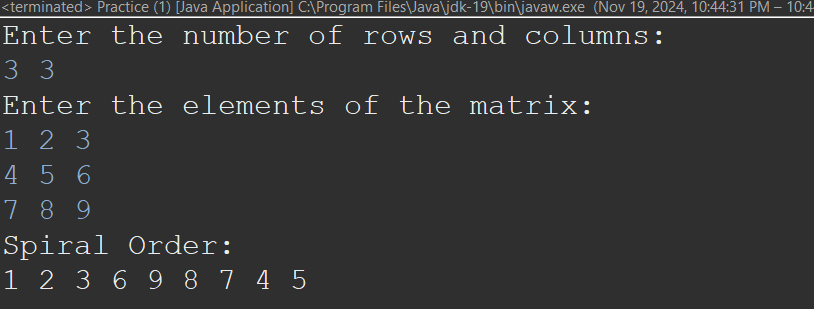
}

scanner.close();

}

}

Output:



Time Complexity: O(n\*m)

1. **Longest Non Repeating Substring:**

Code**:**

import java.util.Arrays;

import java.util.\*;

class Practice {

public int lengthOfLongestSubstring(String s) {

Set<Character> set = new HashSet<>();

int left = 0;

int right = 0;

int max = 0;

int n = s.length();

while(right<n){

while(left < right && set.contains(s.charAt(right))){

set.remove(s.charAt(left));

left++;

}

set.add(s.charAt(right));

max = Math.*max*(max,right-left+1);

right++;

}

return max;

}

public static void main(String[] args) {

Scanner sc = new Scanner(System.***in***);

System.***out***.println("Enter the String:");

String s = sc.nextLine();

Practice solution = new Practice();

int result = solution.lengthOfLongestSubstring(s);

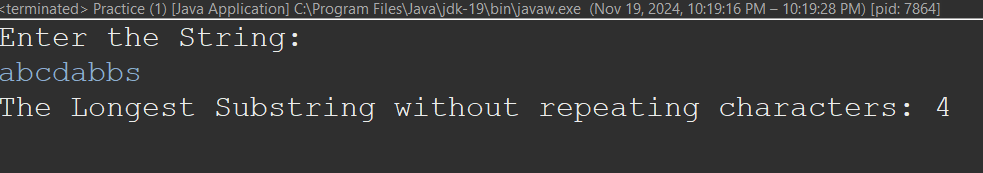
System.***out***.println("The Longest Substring without repeating characters: " + result);

sc.close();

}

}

**OutPut:**



**Time Complexity:** O(n)

1. **Remove Linked List Element:**

Code:

import java.util.Scanner;

class Practice {

static class ListNode {

int val;

ListNode next;

ListNode(int val) {

this.val = val;

}

}

public ListNode removeElements(ListNode head, int val) {

ListNode dummy = new ListNode(-1);

dummy.next = head;

ListNode ptr = dummy;

while (ptr.next != null) {

if (ptr.next.val == val) {

ptr.next = ptr.next.next;

} else {

ptr = ptr.next;

}

}

return dummy.next;

}

public static void main(String[] args) {

Practice solution = new Practice();

Scanner scanner = new Scanner(System.***in***);

System.***out***.println("Enter the no of Nodes: ");

int n = scanner.nextInt();

ListNode head = null, tail = null;

System.***out***.println("Enter the Linked List Elements: ");

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

int value = scanner.nextInt();

ListNode newNode = new ListNode(value);

if (head == null) {

head = newNode;

tail = newNode;

} else {

tail.next = newNode;

tail = newNode;

}

}

System.***out***.println("Enter the element to REMOVE: ");

int val = scanner.nextInt();

ListNode updatedHead = solution.removeElements(head, val);

ListNode current = updatedHead;

System.***out***.print("After Removal ");

while (current != null) {

System.***out***.print(current.val + " ");

current = current.next;

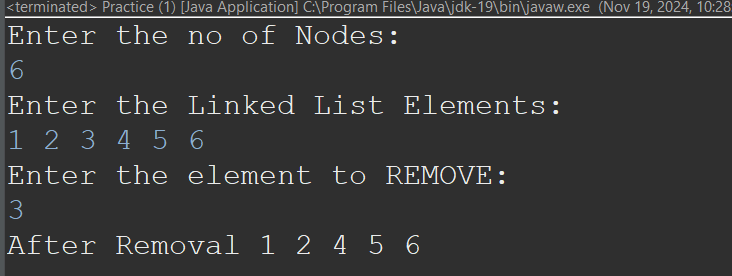
}

scanner.close();

}

}

Output:



Time Complexity: O(n)

1. **Palindromic Linked List:**

Code:

import java.util.Scanner;

class Practice {

static class ListNode {

int val;

ListNode next;

ListNode(int val) {

this.val = val;

}

}

public boolean isPalindrome(ListNode head) {

ListNode slow = head, fast = head;

int len = 0;

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

len++;

}

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

slow = slow.next;

fast = fast.next.next;

}

if (len % 2 != 0) slow = slow.next;

ListNode secondHlfHead = reverse(slow);

while (head != null && secondHlfHead != null) {

if (head.val != secondHlfHead.val) return false;

head = head.next;

secondHlfHead = secondHlfHead.next;

}

return true;

}

private ListNode reverse(ListNode head) {

ListNode curr = head, prev = null;

while (curr != null) {

ListNode nextNode = curr.next;

curr.next = prev;

prev = curr;

curr = nextNode;

}

return prev;

}

public static void main(String[] args) {

Practice solution = new Practice();

Scanner scanner = new Scanner(System.in);

System.out.println("Enter the number of nodes in the linked list:");

int n = scanner.nextInt();

System.out.println("Enter the values of the linked list:");

ListNode head = null, tail = null;

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

int value = scanner.nextInt();

ListNode newNode = new ListNode(value);

if (head == null) {

head = newNode;

tail = newNode;

} else {

tail.next = newNode;

tail = newNode;

}

}

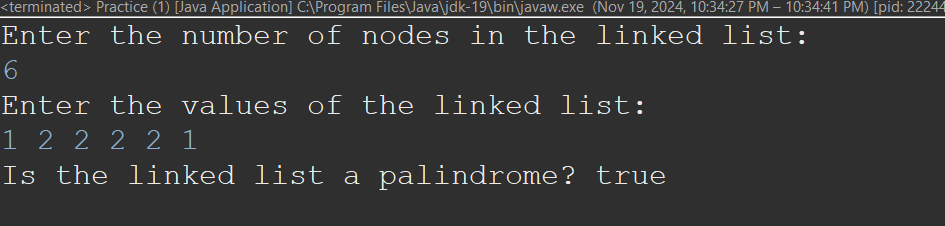
boolean result = solution.isPalindrome(head);

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

scanner.close();

}

}

Output:   


Time Complexity: O(n)