23BAI1117 – BCSE103E – Java

30/09/2024 – Day10 – Theory

Exercises

https://github.com/sriram-s-23BAI1117/javap

1. You are given the heights of consecutive buildings. You can move from the roof of a building to the roof of the next adjacent building. You need to find the maximum number of consecutive steps you can put forward such that you gain an increase in altitude with each step.

Code:

```java
import java.util.Scanner;
//consecutive increase
public class Ex1_30_9{
    public static void main(String[] args) {
        Scanner input = new Scanner(System.in);
        int n = input.nextInt();
        int[] arr = new int[n];
        for (int i=0;i<n;i++){
            arr[i] = input.nextInt();
        }
        int curc=0;
        int maxc=0;
        int i=0;
        while (i<n-1){
            curc=0;
            while (i<n-1 && arr[i]<arr[i+1]){
                curc++;
                i++;
            }
            if (curc > maxc){
                maxc=curc;
            }
            i++;
        }
        System.out.println(maxc);
        input.close();
    }
}
```

Output:

```
PS D:\javap\javap>  & 'C:\Program Files\Java\jdk-23\bin\java.exe' '--
AppData\Roaming\Code\User\workspaceStorage\df070269114bad7e7ee0ccada7
5
1 2 2 3 2
1
PS D:\javap\javap> ^C
PS D:\javap\javap>
PS D:\javap\javap>  d:; cd 'd:\javap\javap'; & 'C:\Program Files\Java
s' '-cp' 'C:\Users\srira\AppData\Roaming\Code\User\workspaceStorage\d
'
4
1 2 3 4
3
PS D:\javap\javap>
```

2. Given a singly linked list of integers. The task is to check if the given linked list is palindrome or not.

Note: You should not use the recursive stack space as well

Code:

```java
import java.util.Scanner;
//palindrome

class Node{
    int data;
    Node next;
    Node(){
        data=0;
        next=null;
    }
    Node(int x){
        data=x;
        next=null;
    }
}

public class Ex2_30_9 {
    public static void main(String[] args) {
        Scanner input = new Scanner(System.in);
        int n = input.nextInt();
        Node head = new Node();
        Node d = head;
        for (int i=0;i<n;i++){
```

```
            Node temp = new Node(input.nextInt());
            d.next = temp;
            d = temp;
        }
        input.close();
        Node copy = new Node();
        for (Node i=head.next; i!=null; i=i.next){
            Node t = new Node();
            t.data=i.data;
            t.next = copy;
            copy = t;
        }
        Node j=head.next;
        Node k=copy;
        while (j!=null && k!=null){
            if (j.data != k.data){
                System.out.println(false);
                return;
            }
            j=j.next;
            k=k.next;
        }
        System.out.println(true);
    }
}
```

Output:

```
PS D:\javap\javap>  & 'C:\Program Files\Java\jdk-23\bin\java.exe
rs\srira\AppData\Roaming\Code\User\workspaceStorage\df070269114b
6
1 2 1 1 2 1
true
PS D:\javap\javap> ^C
PS D:\javap\javap>
PS D:\javap\javap>  d:; cd 'd:\javap\javap'; & 'C:\Program Files
onMessages' '-cp' 'C:\Users\srira\AppData\Roaming\Code\User\work
bb4\bin' 'Ex2_30_9'
4
1 2 3 4
false
```

3. Given two strings **s** and **p**. Find the smallest window in the string **s** consisting of all the

characters(**including duplicates**) of the string **p**. Return "**-1**" in case there is no such window present. In case there are multiple such windows of same length, return the one with the **least starting index**.

Note : All characters are in Lowercase alphabets.

Code:

```
import java.util.Scanner;

public class Ex3_30_9 {
    public static String smallestWindow(String s, String p) {
        if (s.length() < p.length()) {
            return "-1";
        }

        int[] pCnt = new int[26];
        for (char c : p.toCharArray()) {
            pCnt[c - 'a']++;
        }

        int[] windowCnts = new int[26];
        int required = 0;
        for (int count : pCnt) {
            if (count > 0) required++;
        }

        int l = 0, r = 0, formed = 0;
        int minLen = 100001;
        int minLeft = 0;

        while (r < s.length()) {
            char c = s.charAt(r);
            windowCnts[c - 'a']++;

            if (pCnt[c - 'a'] > 0 && windowCnts[c - 'a'] == pCnt[c - 'a']) {
                formed++;
            }

            while (l <= r && formed == required) {
                c = s.charAt(l);
                if (r - l + 1 < minLen) {
                    minLen = r - l + 1;
                    minLeft = l;
                }

                windowCnts[c - 'a']--;
                if (pCnt[c - 'a'] > 0 && windowCnts[c - 'a'] < pCnt[c - 'a'])
{
```

```
                    formed--;
                }
                l++;
            }
            r++;
        }

        return minLen == Integer.MAX_VALUE ? "-1" : s.substring(minLeft,
minLeft + minLen);
    }

    public static void main(String[] args) {
        Scanner input = new Scanner(System.in);
        String s1 = input.next();
        String p1 = input.next();
        input.close();
        System.out.println(smallestWindow(s1, p1));
    }
}
```

Output:

```
PS D:\javap\javap>  & 'C:\Program Files\Java\jdk-23\bin\java.exe' '--enabl
AppData\Roaming\Code\User\workspaceStorage\df070269114bad7e7ee0ccada7d6712
timetopractice
toc
toprac
PS D:\javap\javap> ^C
PS D:\javap\javap>
PS D:\javap\javap>  d:; cd 'd:\javap\javap'; & 'C:\Program Files\Java\jdk-
onMessages' '-cp' 'C:\Users\srira\AppData\Roaming\Code\User\workspaceStora
'Ex3_30_9'
zoomlazapzo
oza
apzo
```


4. Given an unsorted array ***arr*** of positive integers. One number '**A**' from set {1, 2,....,n}

is missing and one number '**B**' occurs twice in array. Find numbers **A** and **B**.

Note: The test cases are generated such that there always exists one missing and one repeating number within the range **[1,n]**.

Code:

```java
import java.util.Scanner;

public class Ex4_30_9 {
    public static void findMissingAndRepeating(int[] arr) {
        int n = arr.length;
        int repeating = -1;
        int missing = -1;

        for (int i = 0; i < n; i++) {
            int index = Math.abs(arr[i]) - 1;

            if (arr[index] < 0) {
                repeating = Math.abs(arr[i]);
            } else {
                arr[index] = -arr[index];
            }
        }

        for (int i = 0; i < n; i++) {
            if (arr[i] > 0) {
                missing = i + 1;
                break;
            }
        }

        System.out.println(repeating+" "+missing);
    }

    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        int n = scanner.nextInt();

        int[] arr = new int[n];
        for (int i = 0; i < n; i++) {
            arr[i] = scanner.nextInt();
        }

        findMissingAndRepeating(arr);
        scanner.close();
    }
}
```


Output:

```
PS D:\javap\javap>  d:; cd 'd:\javap\javap'; & 'C:\Program Files\Java\jdk-2
onMessages' '-cp' 'C:\Users\srira\AppData\Roaming\Code\User\workspaceStorag
'Ex4_30_9'
2
2 2
2 1
PS D:\javap\javap> ^C
PS D:\javap\javap>
PS D:\javap\javap>  d:; cd 'd:\javap\javap'; & 'C:\Program Files\Java\jdk-2
s' '-cp' 'C:\Users\srira\AppData\Roaming\Code\User\workspaceStorage\df07026
'
3
1 3 3
3 2
```

5. Given a string of characters, find the length of the longest proper prefix which is also a proper suffix.

NOTE: Prefix and suffix can be overlapping but they should not be equal to the entire string.

Code:

```java
import java.util.Scanner;

public class Ex5_30_9 {
    public static int longestPrefixSuffix(String str) {
        int n = str.length();
        int[] lps = new int[n];
        int length = 0;
        int i = 1;

        while (i < n) {
            if (str.charAt(i) == str.charAt(length)) {
                length++;
                lps[i] = length;
                i++;
            } else {
                if (length != 0) {
                    length = lps[length - 1];
                } else {
                    lps[i] = 0;
                    i++;
                }
            }
        }
```

```
        return lps[n - 1];
    }

    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        String str = scanner.nextLine();

        int result = longestPrefixSuffix(str);
        System.out.println(result);
        scanner.close();
    }
}
```

Output:

```
PS D:\javap\javap>  & 'C:\Program Files\Java\jdk-23\bin\ja
rs\srira\AppData\Roaming\Code\User\workspaceStorage\df0702
abab
2
PS D:\javap\javap> ^C
PS D:\javap\javap>
PS D:\javap\javap>  d:; cd 'd:\javap\javap'; & 'C:\Program
onMessages' '-cp' 'C:\Users\srira\AppData\Roaming\Code\Use
bb4\bin' 'Ex5_30_9'
aaaa
3
```