30.03.2022

LONGEST INCREASING SUBSEQUENT PROJECT SOURCE CODE

Prepared by: PRiYANKA DAS

JAVA CODE :

```
package com.simplilearn.LIS;
import java.util.*;
public class LIS {
 public static void main(String[] args) {
        Random random = new Random();
        int[] arr = new int[10];

        // filling the array with random integers
        for (int i = 0; i < arr.length; i++)
            arr[i] = random.nextInt();
 System.out.println("From the array: " + Arrays.toString(arr));
        System.out.println("\nLength of the longest increasing
subsequence: " + findMaxIncreasingSubsequence(arr));
    }

    private static int findMaxIncreasingSubsequence(int[] arr)
{
        // check for null or empty array
        if (arr == null || arr.length == 0)
            return 0;
        // if only one value is present, print it and return 1
        if (arr.length == 1) {
            System.out.println("There is only one value in the list!:
" + arr[0]);
            return 1;
        }
```

```
int currentLen, largestLen;
        currentLen = largestLen = 1;

        /*
           a set is used because a subsequence should not
contain duplicate elements
           for example: [1,2,2,2,3] should be saved as [1,2,3]
           My interpretation of the problem is that only a lesser
integer should break a sequence.
         */
        Set<Integer> currentLongestSubsequenceFound =
new LinkedHashSet<>();
        List<List<Integer>> totalSubsequencesFoundList =
new ArrayList<>();

        // iterate over the array while keeping track of the value
of the previous element
        for (int lastValue = 0, i = 0; i < arr.length; lastValue =
arr[i], i++) {

           if (i == 0)
              continue;
// if the current number is greater than the last, add them to
the current set
           if (arr[i] > lastValue) {
              currentLen += 1;
```

```
  // The largest length so far will either be the current length
or the last length
            largestLen = Math.max(largestLen, currentLen);
currentLongestSubsequenceFound.add(arr[i - 1]);
            currentLongestSubsequenceFound.add(arr[i]);

        } else {
            /*
                In this case the current set isn't increasing any
further,
                so it is added to the total list of subsequences
and the current set is cleared.
             */
            totalSubsequencesFoundList.add(new ArrayList<>
(currentLongestSubsequenceFound));
            currentLongestSubsequenceFound.clear();
            currentLen = 1;
        }
        // Save the current subsequence when we have
reached the end of the array
        if (i == arr.length - 1)
            totalSubsequencesFoundList.add(new ArrayList<>
(currentLongestSubsequenceFound));
    }
    /*
        The largest subsequence is picked from the total list
of sequences.
        In a situation where multiple subsequences are the
largest length, the first subsequence will be chosen.
        for example:
        - input: [1,2,3,0,5,2,3,1,5,6]
        - output: [1,2,3], length 3
      */
```

```
   List<Integer> longestSubsequence =
totalSubsequencesFoundList.stream()
            .max(Comparator.comparing(List::size))
            .get();

      /*
         If all of the numbers in the array are decreasing, then
there is no subsequence of increasing numbers.
         My interpretation is that 1 should still be returned for
the largest length of the sequence.
       */
      if (longestSubsequence.isEmpty())
         System.out.println("No sequence of increasing
numbers found!");
      else
         System.out.println("The longest increasing
subsequence (first of it's length): " + longestSubsequence);

      return largestLen;
   }
}
```

THE END