Andrew Januszko,
Morgan Williams-Burrell,
Isabella Boone
Doctor Chen Huo
SWE 300
30 April 2020

Lab 5

Tables:

Before edit						***After edit***				
Method name	**variable name**	**scope**	**live time**	**span**	**status**	**Method name**	**variable name**	**scope**	**live time**	**span**
global	x	12,88	72	13.2	renamed & moved	*main*	subseq	24,30	6	1.5
global	z	12,88	72	13.2	renamed & moved	*main*	seq	23,30	7	2
global	rem	13,88	40	8.5	MOVED	*main*	rem	26,30	4	0.5
global	in	14,88	13	5	renamed & moved	*main*	input	19,31	2	0
global	n	15,88	13	5	MOVED	*main*	n	20,31	11	9
global	ir	16,88	14	2.25	renamed & moved	*main*	buffer	18,31	7	1
main	i	27,31	1	0	DELETED	-	-	-	-	-
answer	x1	35,44	1	0	DELETED	-	-	-	-	-
answer	z1	35,44	1	0	DELETED	-	-	-	-	-
answer	xE	37,44	7	1	DELETED	-	-	-	-	-
answer	zE	38,44	6	0.666	DELETED	-	-	-	-	-
answer	k	40,42	3	1	DELETED	-	-	-	-	-
answer	j	41,42	2	0	DELETED	-	-	-	-	-
-	-	-	-	-	ADDED	*getSubsets*	rem	40,48	8	1.333
cshell	xE	46,53	7	0.5	RENAMED	*getSubsets*	subseq	40,48	8	0.75
cshell	zE	46,53	7	0.75	RENAMED	*getSubsets*	seq	40,48	8	1.333
-	-	-	-	-	ADDED	*calcRem*	rem	57,73	15	3.66
cmeat	xE	60,78	18	1.125	RENAMED	*calcRem*	subseq	57,73	15	1
cmeat	zE	60,78	18	1.833	RENAMED	*calcRem*	seq	57,73	15	1
sameish	end	80,86	3	1	DELETED	-	-	-	-	-
-	-	-	-	-	ADDED	*equalUpTo*	subseq	82,90	4	0.5
-	-	-	-	-	ADDED	*equalUpTo*	seq	82,90	4	2
sameish	i	82,84	2	0	UNCHANGED	equalUpTo	i	80,84	2	0
Averages	**19**	**28.526**	**15.789**	**2.896**		***Averages***	**15**	**9.667**	**7.733**	**1.705**

Method Name Changes	
Original Name	**New Name**
answer	**DELETED**
cshell	getSubsets
cmeat	calcRem
sameish	equalUpTo

Questions:

1. The variables that had the worst original name were xE and zE. They came from the lengths of the strings, however it was difficult to tell that that was their purpose.
2. Memoization works in the problem by storing matching subsets of the strings into an array so that they can be retrieved later. This solution needed memoization so that it did not waste time recomputing some of the subsets.

Code

```
import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;
import java.util.Arrays;

/**
 * @author merlin
 */

public class Lab5 {
  /**
   * Take in 2 strings and check how many subsets exist between them.
   * @param args
   * @throws IOException
   */
  public static void main(String[] args) throws IOException {
    BufferedReader buffer = new BufferedReader(new InputStreamReader(System.in));
    String input = new String(buffer.readLine());
    int n = Integer.parseInt(input);

    do {
      String seq = new String(buffer.readLine());
      String subseq = new String(buffer.readLine());

      int[][] rem = new int[subseq.length()][seq.length()];
      Arrays.stream(rem).forEach(row -> Arrays.fill(row, -1));

      System.out.println(getSubsets(rem, subseq,seq));
    } while (--n > 0);
  }

  /**
   * Get the number of subsets between two strings.
   * @param rem - the remembered subsets.
   * @param subseq - the subsequence to check for.
   * @param seq - the sequence to check.
   * @return - the number of subsets.
   */
  private static int getSubsets(int[][] rem, String subseq, String seq) {
    if (subseq.length() == 0) {
      return 1;
    } else if (rem[subseq.length() - 1][seq.length() - 1] == -1) {
      rem[subseq.length() - 1][seq.length() - 1] = calcRem(rem, subseq, seq);
    }
      return rem[subseq.length() - 1][seq.length() - 1];
```

```
  }

  /**
   * Recursively calculates the subsets and stores them into rem.
   * @param rem - the remembered subsets.
   * @param subseq - the subsequence to check for.
   * @param seq - the sequence to check.
   * @return - the number of subsets to rem.
   */
  private static int calcRem(int[][] rem, String subseq, String seq) {
    if (subseq.length() - 1 > seq.length() - 1) {
      return 1;
    } else if (subseq.length() - 1 == seq.length() - 1) {
      if (equalUpTo(subseq, seq)) {
        return 1;
      } else {
        return 0;
      }
    } else if (subseq.charAt(subseq.length() - 1) == seq.charAt(seq.length() - 1)) {
      return getSubsets(rem, subseq.substring(0, subseq.length() - 1), seq.substring(0, seq.length() - 1))
                                        + getSubsets(rem, subseq, seq.substring(0, seq.length() - 1));
    } else {
      return getSubsets(rem, subseq, seq.substring(0, seq.length() - 1));
    }
  }

  /**
   * Checks if two strings are equal up to a given index.
   * @param end - the index to stop at.
   * @param subseq - the subsequence to be checked.
   * @param seq - the sequence to be checked.
   * @return whether or not the strings match.
   */
  private static boolean equalUpTo(String subseq, String seq) {
    for (int i = 0; i <= subseq.length() - 1; i++) {
      if (subseq.charAt(i) != seq.charAt(i)) {
        return false;
      }
    }
    return true;
  }
}
```