Programming Task 4

Write a program **for the TC computer, in TCL programming language** (**make sure** to have read their description in Edux, in "Materials" section), which will check whether the given sequence of numbers contains the specified *subsequence*.

By a *subsequence* we mean here choosing some elements of the full sequence, preserving their order, but not necessarily consecutively.
For instance, (1, 2, 3) is a subsequence in (7, 6, 1, 4, 5, 2, 1, 3, 7) but not in (4, 2, 3, 5, 1, 2, 6).

Input

The input consists of numbers, each in a separate line, in the following order:

- The first group of lines specifies the desired subsequence, in reversed order
  (reversing may seem strange but it's purposed to make the task simpler – see "Hints" below)
- Then, one line containing the number -1 – denoting the end of the desired subsequence
- Then, a group of lines specifying the full sequence to search in (this time, in the non-reversed order)
- Then, one line containing the number -1 – denoting the end of the full sequence

Output

Your program should print out exactly one line:

- 0 – if the provided sequence contains the desired subsequence;
- -1 – in the opposite case.

Examples

Example 1

The following input:

```
3
2
1
-1
7
6
1
4
5
2
1
3
7
-1
```

corresponds to the first example from the introduction to this task: the program should search for an occurrence of the subsequence (1, 2, 3) in the sequence (7, 6, 1, 4, 5, 2, 1, 3, 7). Since in this case an occurrence exists, for this input your program should print the following to the output:

```
0
```

Example 2

The following input:

```
3
2
1
-1
4
2
3
5
1
2
6
-1
```

corresponds to the second example from the introduction to this task: the program should search for an occurrence of (1, 2, 3) as a subsequence of (4, 2, 3, 5, 1, 2, 6). Since such occurrence does not exist, for this input your program should print the following:

```
-1
```

Example 3

The following input:

```
5
3
4
3
2
1
-1
6
1
1
2
1
3
4
6
3
5
```

```
3
5
-1
```

corresponds to searching for an occurrence of (1, 2, 3, 4, 3, 5) as a subsequence of (6, 1, 1, 2, 1, 3, 4, 6, 3, 5, 3, 5). Since an occurrence exists, the program should print out:

```
0
```

Example 4

The following input:

```
1
-1
-1
```

is correct, and describes searching for an occurrence of a single-item sequence (1) as a subsequence in an *empty* sequence (). Such an occurrence does not exist, so in this case the program should print out:

```
-1
```

Bonus (20%)

You can gain **additional 20% of points** by sending – besides the standard solution – an additional program (also written in the TCL language) which:

- just like in the standard variant, reads a desired subsequence and a sequence to search in (in particular, the same input format as in the standard version still applies);
- unlike in the standard variant, assumes that the desired subsequence has length exactly 6;
- finds the occurrence of the desired subsequence with the minimal *span*, i.e. distance between the first and the last number in that occurrence;
- prints out:
  - the number -1 – in the case when the given sequence has no occurrence of the desired subsequence;
  - two numbers (each in its own line), describing the positions of the beginning and the end of the minimal-span occurrence of the desired subsequence – in the case when occurrences of the subsequence exist (positions are counted starting from zero).
  - if the minimal span is achieved by several occurrences, your program may pick any of them.

For example, for the input from Example 3 above, i.e. for the desired subsequence (1, 2, 3, 4, 3, 5) and the full sequence being (6, 1, 1, 2, 1, 3, 4, 6, 3, 5, 3, 5), there are 6 occurrences:

1. (6, 1, 1, 2, 1, 3, 4, 6, 3, 5, 3, 5) – with span 8
2. (6, 1, 1, 2, 1, 3, 4, 6, 3, 5, 3, 5) – with span 10
3. (6, 1, 1, 2, 1, 3, 4, 6, 3, 5, 3, 5) – with span 10
4. (6, 1, 1, 2, 1, 3, 4, 6, 3, 5, 3, 5) – **with span 7**
5. (6, 1, 1, 2, 1, 3, 4, 6, 3, 5, 3, 5) – with span 9
6. (6, 1, 1, 2, 1, 3, 4, 6, 3, 5, 3, 5) – with span 9

Therefore, in this case, we must pick the occurrence #4, which starts at posiition 2 and ends at position 9. This means that, for the input from Example 3, the bonus program should print out:

```
2
9
```

Assumptions

- The user-provided input to the program is a sequence of lines, each of which contains one integer.
- The number -1 appears in the last line, and in exactly one of the other lines.
- Except for two occurrences of value -1, all the other numbers in the input are non-negative and lower than 10.
- The length of the full sequence to search in is lower than 1000.
- The desired subsequence to be searched for is non-empty (i.e. its length is at least 1).
- The length of the desired subsequence to be searched for is lower than 100.
    - In the bonus program, assume in addition that this length is exactly 6.
      (In case of doubt: if you're solving the bonus, the assumption of length 6 applies only to your bonus program; your solution for the standard variant should still support any length lower than 100).
- The program will be run with the following values of the constructional parameters of the TC computer:
    - number of registers (N): 100
    - register size in bits (K): 32
    - number of data memory cells (S): 2000
    - the address of the beginning of stack (B): 1000
    - number of instruction memory cells (T): 1000

  In other words, the program will be run in the TCE emulator with the command:

  ```
      python3 tce.py 100 32 2000 1000 1000 <your_tcl_file>
  ```

Requirements

Your solution **must** be implemented in the **TCL** language described in the "Materials" section!

Caution: Our assembly-level TCL language has **nothing in common** with another language known more widely (e.g. in Wikipedia) under the name "**Tcl**". Do not use that other Tcl; such programs will not be accepted.

Hints

Hint 1 (for the standard variant)

A key part of the solution is a proper use of the **stack**:

- While reading the desired subsequence, put its consecutive elements on the stack; that way, you'll reach the state when the top of the stack contains the *beginning* of the desired subsequence.
- Then, while reading the consecutive numbers from the input, we compare them to the top of the stack. If equal, this means that the element at the top of the stack has been just found, so we may stop caring about it. (That is, pop it off the stack).

- We act in a loop, according to the above rule – but how do we know when to stop? Detecting the end of input is easy, thanks to the “sentinel” value -1 at the end. But how to detect the end of the stack? (An attempt to `POP` on an empty stack will cause an error which we have no way to “catch”). Here, there are at least two approaches: you can separately track stack height, or – as another “sentinel” – put at the bottom of the stack some value which cannot be confused with any of the numbers searched for (e.g. -1 would do well).

Hint 2 (for the bonus part)

Due to TCL limitations, some approaches of coding are not feasible, e.g. there is no way to store the full sequence in memory (essentially, TCL has no “arrays”). Therefore, each element of the sequence needs to be processed “on the fly”, right after reading it. How to do it?

I recommend a strategy which I’ll now describe for the input from Example 3. In every moment of reading the input, we remember:

- the latest begginning of an occurrence of the substring (1) in the part of the sequence which we *have read so far*;
- the latest begginning of an occurrence of the substring (1, 2) in the part of the sequence which we *have read so far*;
- the latest begginning of an occurrence of the substring (1, 2, 3) in the part of the sequence which we *have read so far*;
- the latest begginning of an occurrence of the substring (1, 2, 3, 4) in the part of the sequence which we *have read so far*;
- etc.

For example, after reading the initial 6 numbers from the full sequence to search in, the above data will look as follows: “4, 3, 5, *none*, …”. (I leave it to you to determine how to represent “*none*”). It turns out that such collection of data can be easily updated after reading the next input sequence element, and also that basing on it you can efficiently determine the solution of the bonus part. (Here, again, some details are left for you to figure them out).

Final remark

In the "Materials" section in Edux, I published an **emulator for the TC computer and TCL language**. I strongly recommend using it - it should greatly help in testing your work in progress.

Submitting solutions

Solutions should be sent by **June 28** to the address:
dominika.pawlik+tak21l+prog4@pjwstk.edu.pl .