KAUNO TECHNOLOGIJOS UNIVERSITETAS

INFORMATIKOS FAKULTETAS

Programavimo kalbų teorija (P175B124)

Laboratorinių darbų ataskaita

Atliko:

IFF-1/9 gr. studentas

Nedas Liaudanskis

2023 m. balandžio 18 d.

Priėmė:

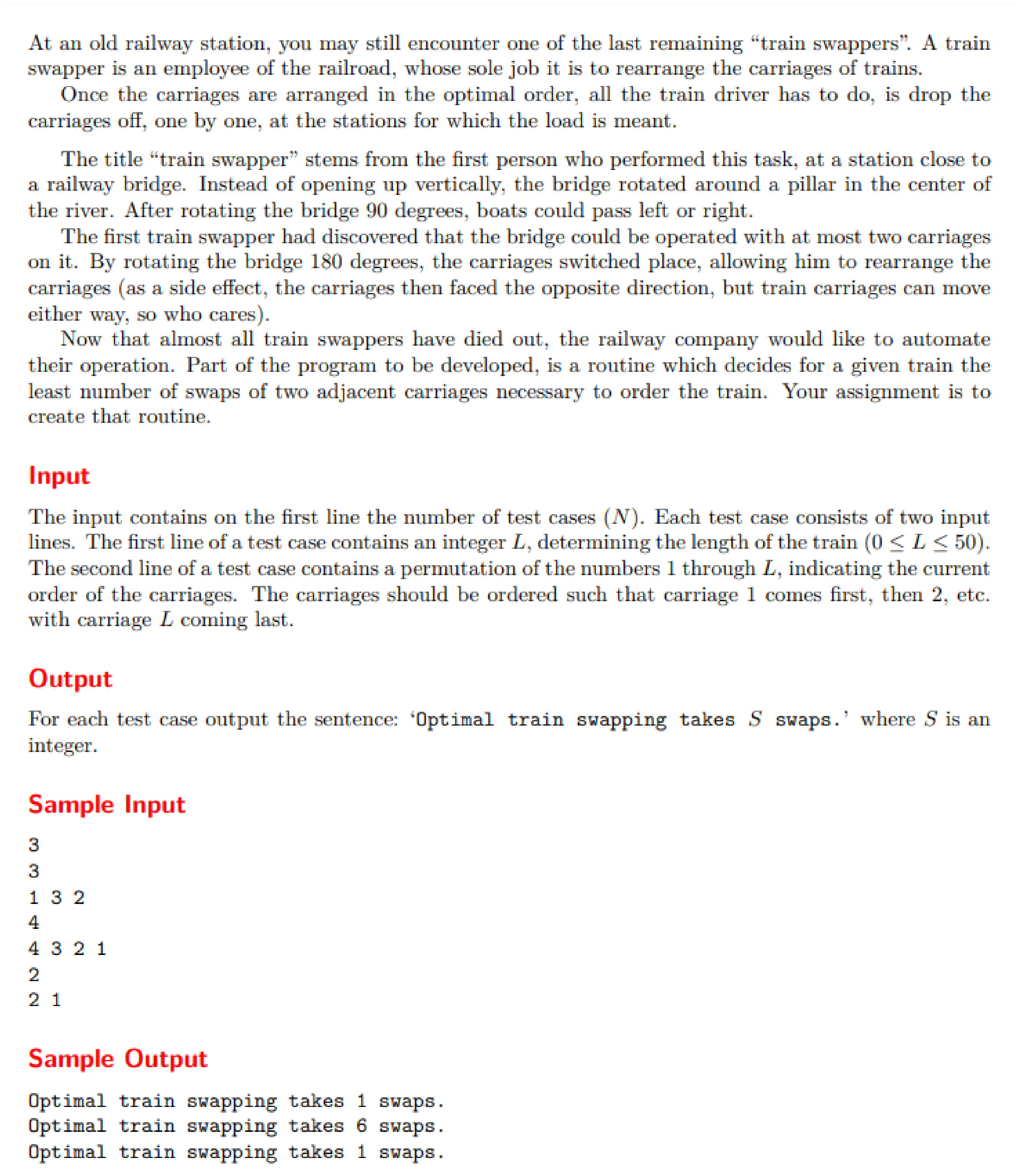
Lekt. Guogis Evaldas

KAUNAS 2023

TURINYS

Haskell(L3)

1.1. *Darbo užduotis*



At an old railway station, you may still encounter one of the last remaining “train swappers”. A train swapper is an employee of the railroad, whose sole job it is to rearrange the carriages of trains.

Once the carriages are arranged in the optimal order, all the train driver has to do, is drop the carriages off, one by one, at the stations for which the load is meant.

The title “train swapper” stems from the first person who performed this task, at a station close to a railway bridge. Instead of opening up vertically, the bridge rotated around a pillar in the center of the river. After rotating the bridge 90 degrees, boats could pass left or right.

The first train swapper had discovered that the bridge could be operated with at most two carriages on it. By rotating the bridge 180 degrees, the carriages switched place, allowing him to rearrange the carriages (as a side effect, the carriages then faced the opposite direction, but train carriages can move either way, so who cares).

Now that almost all train swappers have died out, the railway company would like to automate their operation. Part of the program to be developed, is a routine which decides for a given train the least number of swaps of two adjacent carriages necessary to order the train. Your assignment is to create that routine.

Input

The input contains on the first line the number of test cases (N). Each test case consists of two input lines. The first line of a test case contains an integer L, determining the length of the train ($0 \leq L \leq 50$). The second line of a test case contains a permutation of the numbers 1 through L, indicating the current order of the carriages. The carriages should be ordered such that carriage 1 comes first, then 2, etc. with carriage L coming last.

Output

For each test case output the sentence: ‘`Optimal train swapping takes` S `swaps.`’ where S is an integer.

Sample Input

```
3
3
1 3 2
4
4 3 2 1
2
2 1
```

Sample Output

```
Optimal train swapping takes 1 swaps.
Optimal train swapping takes 6 swaps.
Optimal train swapping takes 1 swaps.
```

1.2. *Programos tekstas*

```haskell
import Control.Monad
import System.IO

-- a function to count the number of swaps needed to sort a list.
countSwaps :: Ord a => [a] -> Int
-- makes a look that ates xs, length of the array and 0
countSwaps xs = loop xs (length xs) 0
  where
    -- starts loop
    loop xs n count
    -- if n is 0 then return count, we are done with the recursion
      | n == 0    = count
      -- if not call bubble method spawn the numbers if needed and
      -- recursively go further into the array of integers
      | otherwise = loop swapped (n-1) newCount
      where
        (swapped, newCount) = bubble xs count
    bubble (x1:x2:xs) count
      | x1 > x2   = let (ys, c) = bubble (x1:xs) (count+1)
                    in (x2:ys, c)
      | otherwise = let (ys, c) = bubble (x2:xs) count
                    in (x1:ys, c)
    bubble xs count = (xs, count)


--  A function to parse input in the format specified in the problem statement.
parseInput :: String -> [(Int, [Int])]
--  First we do lines input and then we pass the info into tail using $
parseInput input = go (tail $ lines input)
  where
    go [] = []
    -- read function converts n into and integer, words l splits the sentence into a
list of words
    -- read $ converts all numbers into a list of integers
    -- go function keeps going with the remaining list
    go (n:l:xs) = (read n, map read $ words l) : go xs






solveTestCase :: (Int, [Int]) -> String
solveTestCase (n, xs) = "Optimal train swapping takes " ++ show (countSwaps xs) ++ "
swaps."


--
solveAll :: [(Int, [Int])] -> String
-- it uses the map function to apply the solveTestCase function to each test case in
the list
-- unlines function is used to join the list of strings into a single string,
separated by newline characters
solveAll = unlines . map solveTestCase
```

```haskell
--  The main function that reads input from a file and solves the problem.
main :: IO ()
main = do
  input <- readFile "Duom.txt"
  let testCases = parseInput input
      output = solveAll testCases
  writeFile "Rez.txt" output
```

1.3. Rezultatai

Duom.txt

```
3
3
1 3 2
4
4 3 2 1
2
2 1
3
3 2 1
5
3 4 1 2 5
```

Rez.txt

```
Optimal train swapping takes 1 swaps.
Optimal train swapping takes 6 swaps.
Optimal train swapping takes 1 swaps.
Optimal train swapping takes 3 swaps.
Optimal train swapping takes 4 swaps.
```