X

(https://swayam.gov.in)

(https://swayam.gov.in/nc_details/NPTEL)

vivekdubey74rr@gmail.com

If already registered, click to check your payment status

Week 5: Programming Assignment

Due on 2023-08-31, 23:59 IST

Course outline

1. Define a function subSeq :: String -> String -> Bool which checks whether the first argument is a subsequence of the second. A subsequence is obtained by deleting some letters in a string and retaining the other characters in the same order as in the original string.

Test cases:
subSeq "ab" "abc" = True
subSeq "ab" "acb" = True
subSeq "ab" "bca" = False
subSeq "" "bea" = True
subSeq "ba" "ba" = True

2. Define a function subWord :: String -> String -> Bool which checks whether the first argument is a subword of the second. A subword is obtained by deleting some number (possibly 0) of letters at the left end and right end in a string and retaining the other characters in the same order.

Test cases:
subWord "ab" "abc" = True
subWord "ab" "acb" = False
subWord "ca" "bca" = True
subWord "" "bea" = True
subWord "ba" "ba" = True

3. A two-dimensional matrix can be represented as a list of rows, each row itself being a list of elements. So in general it is of type [[a]]. Not every list of lists is a matrix, though. For instance, [[1,2,3], [], [2,4]] is a list of three lists, each of a different size.

(a) Define a function isMatrix :: [[a]] -> Bool that checks if a list of lists is a valid matrix (nonzero number of rows, each of the same nonzero length).

Test cases:
isMatrix [] = False

isMatrix [[],[],[]] = False
isMatrix [[2,3], [4,5], [6,7]] = True
isMatrix [[2,3,4,5,6,7]] = True

(b) A square matrix is one where the number of rows is equal to the number of columns. Define a function isSquareMatrix :: [[a]] -> Bool that checks if a list of lists is a square matrix.

Test cases:
isSquareMatrix [] = False
isSquareMatrix [[]] = False
isSquareMatrix [[1]] = True
isSquareMatrix [[1,2,3],[4,5,6],[7,8,9]] = True
isSquareMatrix [[1,2,3,4],[5,6,7,8],[9,10,11,12]] = False

(c) Two matrices are addable if they have the same number of rows and same number of columns. Define a function addable :: [[a]] -> [[a]] -> Bool that checks if two matrices are addable.

Test cases:
addable [[1,2],[3,4]] [[1,2],[3,4]] = True
addable [[1,2],[3,4]] [[5,6,7],[8,9,10]] = False
addable [[1,2],[3,4]] [[1,2],[3,4],[3,4]] = False

(d) Define a function addMatrices :: [[Int]] -> [[Int]] -> [[Int]] that computes the sum of the input matrices.

Test cases:
addMatrices [[1,2]] [[3,4]] = [[4,6]]
addMatrices [[1,2],[3,4]] [[1,2],[3,4]] = [[2,4],[6,8]]

(e) Matrix m1 is multiplyable with matrix m2 if the number of columns in m1 is the same as the number of rows in m2. Define a function multiplyable :: [[a]] -> [[a]] -> Bool that checks if matrix m1 is

multiplyable with m2.

Test cases:
multiplyable [[1,2,3],[4,5,6]] [[1,2],[3,4]] = False
multiplyable [[1,2,3],[4,5,6],[1,2,3],[4,5,6]] [[1,2],[3,4],[5,6]] = True

(f) Define a function multiplyMatrices :: [[Int]] -> [[Int]] -> [[Int]] that computes the product of the input matrices.

Test cases:
multiplyMatrices [[1,2],[3,4]] [[1,2,3],[4,5,6]] = [[9,12,15],[19,26,33]]
multiplyMatrices [[1,2,3],[4,5,6]] [[1,2],[3,4],[5,6]] = [[22,28],[49,64]]

Private Test cases used for evaluation	**Input**	**Expected Output**	**Actual Output**	**Status**
Test Case 1	`subSeq "ab" "abc"`	`True`	`True\n`	`Passed`
Test Case 2	`subSeq "ab" "acb"`	`True`	`True\n`	`Passed`
Test Case 3	`subSeq "ab" "bca"`	`False`	`False\n`	`Passed`
Test Case 4	`subSeq ""   "bea"`	`True`	`True\n`	`Passed`
Test Case 5	`subSeq "ba" "ba"`	`True`	`True\n`	`Passed`
Test Case 6	`subWord "ab" "abc"`	`True`	`True\n`	`Passed`
Test Case 7	`subWord "ab" "acb"`	`False`	`False\n`	`Passed`
Test Case 8	`subWord "ca" "bca"`	`True`	`True\n`	

Test Case 9	`subWord ""   "bea"`	`True`	`True\n`	Passed
Test Case 10	`subWord "ba" "ba"`	`True`	`True\n`	Passed
Test Case 11	`isMatrix []`	`False`	`False\n`	Passed
Test Case 12	`isMatrix [[],[],[]]`	`False`	`False\n`	Passed
Test Case 13	`isMatrix [[2,3], [4,5], [6,7]]`	`True`	`True\n`	Passed
Test Case 14	`isMatrix [[2,3,4,5,6,7]]`	`True`	`True\n`	Passed
Test Case 15	`isSquareMatrix []`	`False`	`False\n`	Passed
Test Case 16	`isSquareMatrix [[]]`	`False`	`False\n`	Passed
Test Case 17	`isSquareMatrix [[1]]`	`True`	`True\n`	Passed
Test Case 18	`isSquareMatrix [[1,2,3],[4,5,6],[7,8,9]]`	`True`	`True\n`	Passed
Test Case 19	`isSquareMatrix [[1,2,3,4],[5,6,7,8], [9,10,11,12]]`	`False`	`False\n`	Passed
Test Case 20	`addable [[1,2],[3,4]] [[1,2],[3,4]]`	`True`	`True\n`	Passed
Test Case 21	`addable [[1,2],[3,4]] [[5,6,7],[8,9,10]]`	`False`	`False\n`	Passed
Test Case 22	`addable [[1,2],[3,4]] [[1,2],[3,4], [3,4]]`	`False`	`False\n`	Passed
Test Case 23	`addMatrices [[1,2]] [[3,4]]`	`[[4,6]]`	`[[4,6]]\n`	P

Test Case 24	`addMatrices [[1,2],[3,4]] [[1,2],[3,4]]`	`[[2,4],[6,8]]`	`[[2,4],[6,8]]\n`	Passed
Test Case 25	`multiplyable [[1,2,3],[4,5,6]] [[1,2],[3,4]]`	`False`	`False\n`	Passed
Test Case 26	`multiplyable [[1,2,3],[4,5,6],[1,2,3],[4,5,6]] [[1,2],[3,4],[5,6]]`	`True`	`True\n`	Passed
Test Case 27	`multiplyMatrices [[1,2],[3,4]] [[1,2,3],[4,5,6]]`	`[[9,12,15],[19,26,33]]`	`[[9,12,15],[19,26,33]]\n`	Passed
Test Case 28	`multiplyMatrices [[1,2,3],[4,5,6]] [[1,2],[3,4],[5,6]]`	`[[22,28],[49,64]]`	`[[22,28],[49,64]]\n`	Passed

The due date for submitting this assignment has passed.

28 out of 28 tests passed.

You scored 100.0/100.

Assignment submitted on 2023-08-25, 00:30 IST

Your last recorded submission was :

```haskell
extractArgs :: String -> (String, String)
extractArgs s = (str1, str2)
  where
    (_, _:s1) = break (== '\"') s
    (str1, _:s2) = break (== '\"') s1
    (_, _:s3) = break (== '\"') s2
    (str2, _) = break (== '\"') s3

main = do
    line <- getLine;
    let (func, rest) = break (== ' ') line in
        case func of
            "subSeq" -> let (str1, str2) = extractArgs (tail rest) in
                        putStrLn . show $ subSeq str1 str2
            "subWord" -> let (str1, str2) = extractArgs (tail rest) in
                        putStrLn . show $ subWord str1 str2
            "isMatrix" -> let args = read rest :: [[Int]] in
                putStrLn . show $ isMatrix args
```

```
            "isSquareMatrix" -> let args = read rest :: [[Int]] in
                putStrLn . show $ isSquareMatrix args
            "addable" -> let (rest1, rest2) = break (== ' ') $ tail rest
                             arg1 = read rest1 :: [[Int]]
                             arg2 = read rest2 :: [[Int]] in
                         putStrLn . show $ addable arg1 arg2
            "addMatrices" -> let (rest1, rest2) = break (== ' ') $ tail rest
                                 arg1 = read rest1 :: [[Int]]
                                 arg2 = read rest2 :: [[Int]] in
                             putStrLn . show $ addMatrices arg1 arg2
            "multiplyable" -> let (rest1, rest2) = break (== ' ') $ tail rest
                                  arg1 = read rest1 :: [[Int]]
                                  arg2 = read rest2 :: [[Int]] in
                              putStrLn . show $ multiplyable arg1 arg2
            "multiplyMatrices" -> let (rest1, rest2) = break (== ' ') $ tail rest
                                      arg1 = read rest1 :: [[Int]]
                                      arg2 = read rest2 :: [[Int]] in
                                  putStrLn . show $ multiplyMatrices arg1 arg2
{-

    1. Define a function subSeq :: String -> String -> Bool which checks whether
the first argument is a subsequence of the second. A subsequence is obtained by
deleting some letters in a string and retaining the other characters in the same
order as in the original string.

Test cases:
subSeq "ab" "abc" = True
subSeq "ab" "acb" = True
subSeq "ab" "bca" = False
subSeq ""   "bea" = True
subSeq "ba" "ba"  = True

2. Define a function subWord :: String -> String -> Bool which checks whether
the first argument is a subword of the second. A subword is obtained by deleting
some number (possibly 0) of letters at the left end and right end in a string
and retaining the other characters in the same order.

Test cases:
subWord "ab" "abc" = True
subWord "ab" "acb" = False
subWord "ca" "bca" = True
subWord ""   "bea" = True
subWord "ba" "ba"  = True

3. A two-dimensional matrix can be represented as a list of rows, each row
itself being a list of elements. So in general it is of type [[a]]. Not every
list of lists is a matrix, though. For instance, [[1,2,3], [], [2,4]] is a list
of three lists, each of a different size.

(a) Define a function isMatrix :: [[a]] -> Bool that checks if a list of lists
is a valid matrix (nonzero number of rows, each of the same nonzero length).
```

```
70
71 Test cases:
72 isMatrix [] = False
73 isMatrix [[],[],[]] = False
74 isMatrix [[2,3], [4,5], [6,7]] = True
75 isMatrix [[2,3,4,5,6,7]] = True
76
77 (b) A square matrix is one where the number of rows is equal to the number of
78 columns. Define a function isSquareMatrix :: [[a]] -> Bool that checks if a
79 list of lists is a square matrix.
80
81 Test cases:
82 isSquareMatrix [] = False
83 isSquareMatrix [[]] = False
84 isSquareMatrix [[1]] = True
85 isSquareMatrix [[1,2,3],[4,5,6],[7,8,9]] = True
86 isSquareMatrix [[1,2,3,4],[5,6,7,8],[9,10,11,12]] = False
87
88 (c) Two matrices are addable if they have the same number of rows and same
89 number of columns. Define a function addable :: [[a]] -> [[a]] -> Bool that
90 checks if two matrices are addable.
91
92 Test cases:
93 addable [[1,2],[3,4]] [[1,2],[3,4]] = True
94 addable [[1,2],[3,4]] [[5,6,7],[8,9,10]] = False
95 addable [[1,2],[3,4]] [[1,2],[3,4],[3,4]] = False
96
97 (d) Define a function addMatrices :: [[Int]] -> [[Int]] -> [[Int]] that computes
98 the sum of the input matrices.
99
100 Test cases:
101 addMatrices [[1,2]] [[3,4]] = [[4,6]]
102 addMatrices [[1,2],[3,4]] [[1,2],[3,4]] = [[2,4],[6,8]]
103
104 (e) Matrix m1 is multiplyable with matrix m2 if the number of columns in m1 is
105 the same as the number of rows in m2. Define a function
106 multiplyable :: [[a]] -> [[a]] -> Bool that checks if matrix m1 is
107 multiplyable with m2.
108
109 Test cases:
110 multiplyable [[1,2,3],[4,5,6]] [[1,2],[3,4]] = False
111 multiplyable [[1,2,3],[4,5,6],[1,2,3],[4,5,6]] [[1,2],[3,4],[5,6]] = True
112
113 (f) Define a function multiplyMatrices :: [[Int]] -> [[Int]] -> [[Int]] that
114 computes the product of the input matrices.
115
116 Test cases:
117 multiplyMatrices [[1,2],[3,4]] [[1,2,3],[4,5,6]] = [[9,12,15],[19,26,33]]
118 multiplyMatrices [[1,2,3],[4,5,6]] [[1,2],[3,4],[5,6]] = [[22,28],[49,64]]
119
120
```

```
-}


subSeq :: String -> String -> Bool
subSeq "" xs = True
subSeq s "" = False
subSeq s xs
    | head s == head xs = subSeq (tail s) (tail xs)
    | otherwise = subSeq s ( tail xs )

subWord :: String -> String -> Bool

subWord "" xs = True
subWord s "" = False
subWord s xs
        | head s == head xs = exactmatch s xs
        | otherwise = subWord s (tail xs)
        where exactmatch s xs
                    | s == take (length s) xs = True
                    | otherwise = subWord s (tail xs)


isMatrix :: [[a]] -> Bool
isMatrix [] = False
isMatrix [y]
        | not ( null y) = True
        | otherwise = False
isMatrix x
        | length (head x) == length (x!!1) = isMatrix $ tail x
        | otherwise = False


isSquareMatrix :: [[a]] -> Bool
isSquareMatrix [] = False
isSquareMatrix x = foldl k True x
                   where k acc ele = acc && length ele == length x && not ( null ele)


addable :: [[a]] -> [[a]] -> Bool
addable x y = isMatrix x && isMatrix y && length x == length y && length ( head x) == length ( head y)


addMatrices :: [[Int]] -> [[Int]] -> [[Int]]
addMatrices xs ys = [ [ xs!!i!!j + ys!!i!!j | j<-[ 0 .. length (xs!!i) - 1]] | i <- [0 .. length xs -1]]

multiplyable :: [[a]] -> [[a]] -> Bool
multiplyable xs ys = isMatrix xs && isMatrix ys && length ys == length (head xs)

multiplyMatrices :: [[Int]] -> [[Int]] -> [[Int]]
multiplyMatrices xs ys = [ [  k i j | j <- [0 .. length (head ys)-1]] | i <- [0 .. length xs - 1]]
                             where k i j = sum [ xs!!i!!l * (ys!!l)!!j | l <- [0.. length (xs!!0)-1 ] ]
```