EDAF95/EDAN40: Functional Programming Assignment F2 (EDAF95): Sudoku solver Assignment N2 (EDAN40): String alignment

Jacek Malec
Dept. of Computer Science, Lund University, Sweden

May 3rd, 2023

Sudoku (F2)

2	6		3					
5						7		
					1		4	
6			5			2		
		4			8			1
	5		9					
		7						3
					4		1	6

Assignment F2: Sudoku solver

1. Read a file with a set of Sudokus;
2. Solve the current Sudoku and show the result;
3. Present the current unsolved or partially solved Sudoku to the user;
4. Support the user in solving a Sudoku;
5. Wrap it in a nice read-eval-print loop (or REPL).

The style will be important this time!
If you follow the lab4 instructions, you will get a nice monadic solution of the problem.
I/O will take a while to master, so begin early!

Topic of today: memoization

Appropriate reference: section 20.6 in Thompson's textbook "The craft of functional programming", 3rd ed.

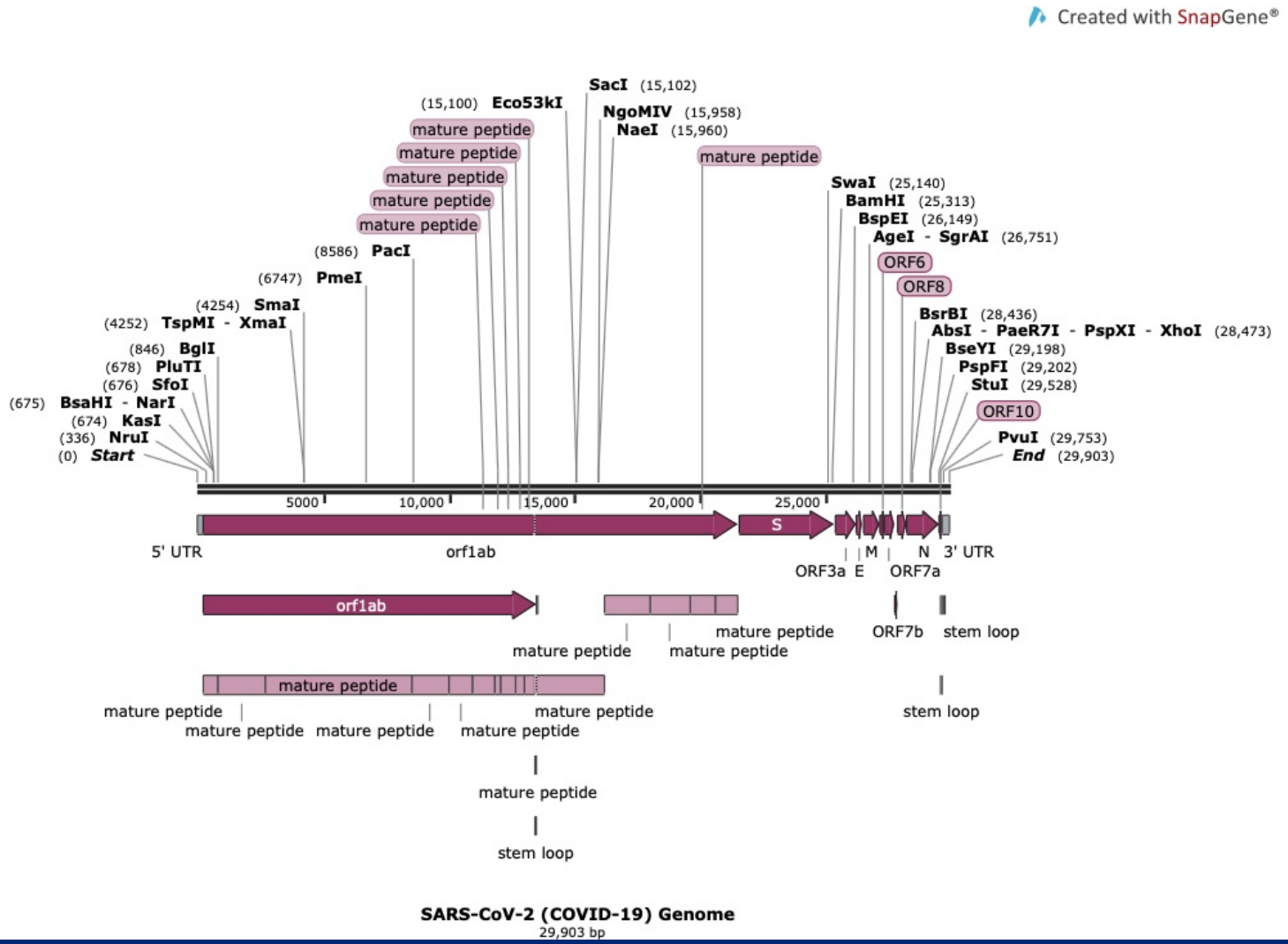
Reminder:

laziness = call-by-name + sharing

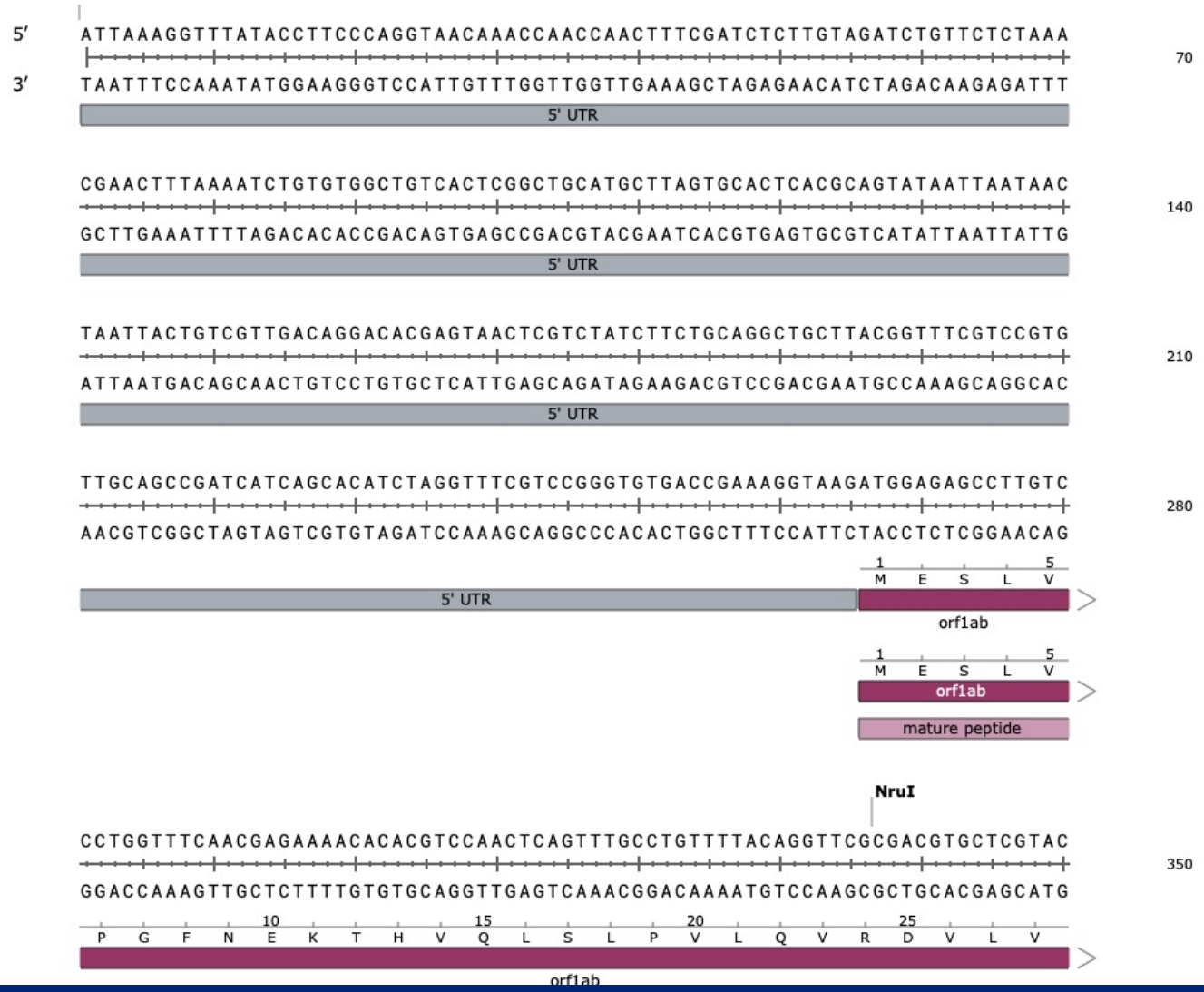
The string alignment problem

An *alignment* of two strings is a way of finding a correspondence between them (e.g. by placing one above the other to illustrate how parts of the strings are related).
Nowadays the most interesting strings consist of only four letters: `A, C, G, T`:
`https://www.snapgene.com/resources/coronavirus-resources/?resource=SARS-CoV-2_(COVID-19)_Genome`

SARS-CoV-2 genome



SARS-CoV-2 genome



The string alignment problem

Given two strings, *s* and *t*, an alignment is obtained by inserting spaces into *s* and *t* so that the characters of the resulting strings may be put in one-to-one correspondence to each other:

```
HASKELL
PASCA-L
```

Spaces may also be added at the beginning and at the end of strings, but a space in one string is not allowed to be aligned with a space in the other string.

```
H-ASKELL
-PASCA-L
```

Optimality of an alignment

The *length* of an alignment is the number of columns it contains, so

```
HASKELL
PASCA-L
```

has length 7, while

```
H-ASKELL
-PASCA-L
```

and

```
H-ASKELL
-PASCAL-
```

have length 8.

Optimality, cont.

Which of the above alignments is better? No definite answer, it depends.

The application decides how mismatches and spaces are penalized and how matches are rewarded.

Below we use three parameters expressing this: `scoreMatch`, `scoreMismatch` and `scoreSpace`.

Difficulty

The combinatorial explosion.

The algorithm: Take two strings, generate all possible alignments, evaluate them and return the ones with maximal score.

For strings of length 1000 each the number of possible alignments is more than 10^{764}.

10^{100} - googol; 10^{80} - # of atoms in the universe

The assignment (N2)

DO SOMETHING SMART ABOUT IT!

The assignment N2, more exactly

Given two strings, `s` and `t`, and values for `scoreMatch`, `scoreMismatch` and `scoreSpace`,

find ALL **optimal** alignments between `s` and `t`.

An optimal alignment is one *with the highest score*. There may be more than one such alignment in general case.

Speaking Haskell

```
optimalAlignments :: Int -> Int -> Int -> String ->
                                    String -> [AlignmentType]

score :: Int -> Int -> Int -> String -> String -> Int
```

Given for example:

```
scoreMatch = 1
scoreMismatch = -1
scoreSpace = -2
```

the score of the first alignment is -2, while of the second and third is -5.

Digression: the MCS problem

MCS: Maximal Common Subsequence

A sequence is a *subsequence* of another sequence if it can be obtained by deleting zero or more elements from that sequence.

The problem: finding maximal (i.e. the longest) common subsequence.

E.g. for lists [3,2,8,2,3,9,4,3,9] and [1,3,2,3,7,9] the MCS is [3,2,3,9] which has length 4.

Digression: the MCS problem

The solution is easy:

```
mcsLength1 :: Eq a => [a] -> [a] -> Int

mcsLength1 _ [] = 0
mcsLength1 [] _ = 0
mcsLength1 (x:xs) (y:ys)
  | x == y     = 1 + mcsLength1 xs ys
  | otherwise = max (mcsLength1 xs (y:ys))
                    (mcsLength1 (x:xs) ys)
```

Digressionˆ2: Fibonacci

```
-- Naive Fibonacci function
fib 0 = 0
fib 1 = 1
fib m = fib (m-2) + fib (m-1)

-- An algorithm which returns a pair
-- of consecutive Fibonacci numbers.

fibP :: Int -> (Int,Int)
fibP 0 = (0,1)
fibP n = (y,x+y)
         where
         (x,y) = fibP (n-1)
```

Digressionˆ2: Fibonacci

```
-- The list of Fibonacci values, defined directly.

fibs :: [Integer]

fibs = 0 : 1 : zipWith (+) fibs (tail fibs)
```

Digression: the MCS problem

	[]	3:[]	2:[3]	3:[2,3]	1:[3,2,3]
[]	0	0	0	0	0
3:[]	0	1	1	1	1
2:[3]	0	1	2	2	2
8:[2,3]	0	1	2	2	2
2:[8,2,3]	0	1	2	2	2
3:[2,8,2,3]	0	1	2	3	3

Digression: the MCS problem

```
mcsLength :: Eq a => [a] -> [a] -> Int
mcsLength xs ys = mcsLen (length xs) (length ys)
  where
     mcsLen i j = mcsTable!!i!!j
     mcsTable = [[ mcsEntry i j | j<-[0..]] | i<-[0..] ]
     mcsEntry :: Int -> Int -> Int
     mcsEntry _ 0 = 0
     mcsEntry 0 _ = 0
     mcsEntry i j
       | x == y     = 1 + mcsLen (i-1) (j-1)
       | otherwise = max (mcsLen i (j-1)) (mcsLen (i-1) j)
       where
          x = xs!!(i-1)
          y = ys!!(j-1)
```

The assignment N2 consists of

1. Answering some questions regarding the problem;
2. Writing some functions:

```
similarityScore :: String -> String -> Int
similarityScore string1 string2
maximaBy :: Ord b => (a -> b) -> [a] -> [a]
maximaBy valueFcn xs
```

   For example, `maximaBy length ["cs", "efd", "lth", "it"]` should return `["efd", "lth"]`.

3. Solving the problem:

```
type AlignmentType = (String,String)
optAlignments :: String -> String -> [AlignmentType]
outputOptAlignments :: String -> String -> IO ()
```

Example

```
scoreMatch = 0
scoreMismatch = -1
scoreSpace = -1
string1 = "writers"
string2 = "vintner"

Main> similarityScore string1 string2
-5
Main> optAlignments string1 string2
[("writ-ers","vintner-"), ("wri-t-ers","-vintner-"),
("wri-t-ers","v-intner-")]
```

Example

```
Main> outputOptAlignments string1 string2
    There are 3 optimal alignments:

    w r i t - e r s
    v i n t n e r -

    w r i - t - e r s
    - v i n t n e r -

    w r i - t - e r s
    v - i n t n e r -

    There are 3 optimal alignments.
```

Optimisation

Your program should be able to handle the following pairs of strings (or even longer ones) within a couple of seconds:

```
newOptAlignments "aferociousmonadatemyhamster"
                 "functionalprogrammingrules"

newOptAlignments "bananrepubliksinvasionsarmestabsadjutant"
                 "kontrabasfiolfodralmakarmästarlärling"
```