National Undergraduate Programme in Mathematical Sciences

National Graduate Programme in Computer Science

Functional Programming in Haskell

Assignment 3

- **Due date**: October 22, 2023, 2359 hours
- Submit your solution in a single file named `loginid.hs` on Moodle. For example, if I were to submit a solution, the file would be called `spsuresh.hs`. You may define auxiliary functions in the same file, but the solutions should have the function names specified by the problems.
- Remember that function names should start with a lowercase letter, and there should be no indentation for non-local functions.
- Please compile and check that the sample cases are satisfied before submitting!
- I have provided a `template.hs` with the function names and types, and some useful helper functions. Replace the **`undefined`** in the file with the actual definition. You are free to define other helper functions in the same file. Copy/rename this file to `loginid.hs`, complete the definitions, and submit.

---

1. The *Hamming distance* between two sequences of equal length is the number of positions at which the corresponding symbols are different. Define the following function to compute the Hamming distance (satisfying the given sample cases).

```
hamming :: Eq a ⇒ [a] → [a] → Maybe Int

hamming "" ""                  = Just 0
hamming "" "haskell"           = Nothing
hamming "haskell" "haskell"    = Just 0
hamming "Haskell" "haskell"    = Just 1
hamming "Happens" "haskell"    = Just 5
hamming "ocaml" "haskell"      = Nothing
```

2. Define a function which checks whether the number of 1s in the binary representation of n is even or odd. We assume that $n \geqslant 0$. Given below is the function signature and some sample cases.

```
parity :: Int → Bool

parity 0    = True      (binary representation 0)
```

```
parity 10   = True      (1010)
parity 100  = False     (1100100)
parity 1000 = True      (1111101000)
parity 2289 = True      (100011110001)
```

3. The *maximum segment sum* of a list $[x_1,\ldots,x_n]$ is a sublist $[x_i,x_{i+1},\ldots,x_{j-1},x_j]$ such that $x_i^3+\cdots+x_j^3$ is maximum, among all values of i,j s.t. $1 \leqslant i \leqslant j \leqslant n$.

   Given below is the function signature and some sample cases.

```
mss :: [Integer] → (Integer,[Integer])

mss []                                  = (0,[])
mss [2,-4,2,-3,16,-16,0,-13,-6,11]      = (4096,[16])
mss [-10,1,-1,15,7,-20,-12,17,18,-6]    = (10745,[17,18])
mss [-17,6,6,-3,-3,20,-8,-13,-4,13]     = (8378,[6,6,-3,-3,20])
```

4. Knights are chess pieces whose moves are characterized as two and a half squares – they move two squares in one direction and one square in an orthogonal direction. On a chessboard, rows are called *ranks* and columns are called *files*. The square on the x^{th} rank and the y^{th} file is represented by the pair (x,y). The set of squares on an 8×8 chessboard is thus given by

```
squares = [(x,y) | x ← [0..7], y ← [0..7]]
```

   Define a function

```
knightMove :: (Int, Int) → Int → [(Int, Int)]
```

   with the behavior that

```
knightMove (x, y) n
```

   gives the list of squares where the knight could be in after a total of `n` moves, starting from the square `(x,y)`. Make sure that your list has no duplicates, and is lexicographically sorted.

   Here are some sample cases.

```
knightMove (0,0) 0      = [(0,0)]
knightMove (0,0) 1      = [(1,2),(2,1)]
knightMove (0,0) 3      = [(0,1),(0,3),(0,5)
                          ,(1,0),(1,2),(1,4),(1,6)
                          ,(2,1),(2,3),(2,5)
                          ,(3,0),(3,2),(3,4),(3,6)
                          ,(4,1),(4,3),(4,5)
                          ,(5,0),(5,2),(5,4)
```

```
                              ,(6,1),(6,3)
                              ]
knightMove (2,2) 2        = [(0,2),(0,6)
                              ,(1,1),(1,3),(1,5)
                              ,(2,0),(2,2),(2,4),(2,6)
                              ,(3,1),(3,3),(3,5)
                              ,(4,2),(4,6)
                              ,(5,1),(5,3),(5,5)
                              ,(6,0),(6,2),(6,4)
                              ]
knightMove (2,2) 1000     = [(0,0),(0,2),(0,4),(0,6)
                              ,(1,1),(1,3),(1,5),(1,7)
                              ,(2,0),(2,2),(2,4),(2,6)
                              ,(3,1),(3,3),(3,5),(3,7)
                              ,(4,0),(4,2),(4,4),(4,6)
                              ,(5,1),(5,3),(5,5),(5,7)
                              ,(6,0),(6,2),(6,4),(6,6)
                              ,(7,1),(7,3),(7,5),(7,7)
                              ]
```

5. This problem concerns editing a string to arrive at another string. We start at the leftmost letter, and either keep it and skip to the next letter, or modify the letter, or insert a new letter at that position, or delete the letter, and continue till the string is completely transformed. The series of edits involved in the transformation is summarised by an *edit string* consisting of the characters `-`, `i`, `d`, and `m`. For instance, here is an edit string that transforms `kitten` to `sitting`.

```
 kitten
s itting
id---m-i
```

Each edit operation has a cost. Skipping a letter has cost 0, modifying a letter has cost 1, while insert and delete have cost 2. One can see that the above sequence of edits has total cost 7. Given below is a sequence of edits with total cost 4.

```
kitten
sitting
m---m-i
```

One can check that the sequence of edits represented by the edit string `"iiiiiiidddddd"` has cost 26.

The task is to find an edit string with minimum overall cost to go from one string to another. (This cost is called the *edit distance*.) Define a function

```
editDistance :: String → String → (Int, String)
```

such that `editDistance as bs` returns `(n,s)`, where `n` is the edit distance, and `s` is an edit string with cost `n`. For instance, the edit distance between `"kitten"` and `"sitting"` is 4.

A naïve recursive program to compute edit distance would take exponential time in the worst case, because of repeated recursive calls with same arguments. You should compute the result row by row, as seen in class. Or you can use arrays.

Here are some sample cases.

```
editDistance "kitten" "sitting"
    = (4,"m---m-i")
editDistance "Kareem was the leading scorer" "LeBron is the scoring leader"
    = (17,"mmmmmm-md---------mmmm----mmmm--")
editDistance "Jordan was an all-time great" "LeBron is even greater"
    = (28,"mmmmm--md--mmmmmmmmmm-dd-ddd")
editDistance "LeBron is even greater" "Jordan was an all-time great"
    = (28,"mmmmm--mi--mmmmmmmmmm-ii-iii")
editDistance "" "This is a nonempty string"
    = (50,"iiiiiiiiiiiiiiiiiiiiiiii")
editDistance "This is a nonempty string" ""
    = (50,"ddddddddddddddddddddddddd")
editDistance "szwyjbvzflvqtdk" "npagbpcwbptbvrnbifxt"
    = (24,"mmmmmmmmmii-mmmmiii")
editDistance "fittingly" "misfitting"
    = (9,"m-mm-mmmmi")
editDistance "fittingly" "unfitting"
    = (8,"ii------dd")
```