German University in Cairo
Media Engineering and Technology
Prof. Dr. Slim Abdennadher
Dr. Nada Sharaf

Concepts of Programming Languages, Spring term 2020
Project Description
"Automatic Text Generation"
Deadline: 30.04.2020 - 11:59 pm

The project will put your knowledge in Haskell to the test. Before proceeding, make sure that you read each section carefully. Enjoy.

Instructions

Please read and follow the instructions carefully:

a) The project should be implemented using Haskell (Hugs98) based on the syntax discussed in class.

b) This is a team project. You can work in groups of maximum 3 members. The groups for this project are the same as project 1. In case of any changes, please report it by sending an email to your TA. You must send complete information about your new team including full names and unique GUC application numbers. The deadline for sending group change requests is Wednesday 15.04.2020 (11:59 pm).

c) You are allowed to use built-in Haskell functions, like `length` or other functions in `Hugs.Prelude` without explanation: https://www.haskell.org/onlinereport/prelude-index.html

d) Every team has to submit their project through the MET Website's submission link:
   - your Haskell project **source file** named after your team code, e.g. `T003.hs`
   - it is **your** responsibility to ensure that you have submitted the correct files and saved the correct content before uploading.

e) Cheating and plagiarism will be strictly punished with a grade of 0 in the project.

f) Please respect the submission deadline marked at the beginning of the document (**Thursday 30.04.2020 - 11:59 pm**). Any delay will result in a **rejection** of the submission.

Project Description

You are required to implement a simple algorithmic automatic text generator. It should be able to learn some statistics from a number of given documents and then use these statistics to generate text.

Example given a list of documents

```
> generateText "the man" 3
"the man saw the saw"
```

The file `DataFile.hs` will contain these documents. Your code should use the list `docs` from that file in order to learn. Make sure you have that file in the same folder as your solution `.hs` file. Then, you can write in your file:

```
import DataFile
```

In order to implement the generator, you are required to implement the following functions:

Note that:

- Types of some of the following functions can be more generic and might contain Eq, type does not have to be a string

1) **wordToken**

   Splits the text so that each word/punctuation is a separate item in the list. You can find the list of punctuations in the `DataFile.hs`. The type of `wordToken` will be:

   ```
    wordToken:: String -> [String]
   ```

   Note that: a String (String) is equivalent to a list of characters ([Char]) so the above type can be re-written as

   ```
   wordToken :: [Char] -> [[Char]]

   > wordToken "the sun is shining. the wind is blowing"

   ["the","sun","is","shining",".","the","wind","is","blowing"]
   ```

2) **wordTokenList**
   Given a list of texts, it generates all tokens (separated words/punctuation). The type of wordTokenList will be:

   ```
   wordTokenList :: [String] -> [String]

   > wordTokenList ["the man is the man. he is great","the man saw the saw"]
   ["the","man","is","the","man",".","he","is","great","the","man","saw","the","saw"]
   ```

3) **uniqueBigrams**
   The input to this function is a list of items and the output is a list of unique pairs of each two consecutive items in the input list. The type of bigrams will be:

   ```
   uniqueBigrams :: [String] -> [(String,String)]

   > uniqueBigrams ["the","man","is","the","man","."]
   [("man","is"),("is","the"),("the","man"),("man",".")]
   ```

4) **uniqueTrigrams**
   The input to this function is a list of items and the output is a list of unique triples of each three consecutive items in the input list. . The type of trigrams will be:

   ```
   trigrams :: [String] -> [(String,String,String)]

   > uniqueTrigrams ["the","man","is","the","man","."]
   [("the","man","is"),("man","is","the"),("is","the","man"),("the","man",".")]
   ```

5) **bigramsFreq**
The input to this function is a list of words and the output is a list of Bigram frequencies which is a list of each two consecutive words (as a pair) and their count as a pair (The order of the output does not matter). The type of bigramsFreq will be:

```
bigramsFreq :: Num a => [String] -> [((String,String),a)]

> bigramsFreq ["the","man","is","the","man","."]
[(("man","is"),1),(("is","the"),1),(("the","man"),2),(("man","."),1)]
```

6) **trigramsFreq**
The input to this function is a list of words and the output is a list of Trigram frequencies which is a list of each three consecutive words (as a pair) and their frequency/count as a pair (The order of the output does not matter). The type of trigramsFreq will be:

```
trigramsFreq:: Num a => [String] -> [((String,String,String),a)]

Example:
> trigramsFreq ["the","man","is","the","man","."]
[(("the","man","is"),1),(("man","is","the"),1),(("is","the","man"),1),(("the","man
","."),1)]
```

7) **getFreq**
Extracts the frequency given an item and a list of (item,frequency) pairs. The type of trigramsFreq will be:

```
getFreq :: (Eq a, Num b) => a -> [(a,b)] -> b

Example:
> getFreq 'a' [('f',1),('a',2),('b',1)]
2
```

8) **generateOneProb**
Generates the probability of a word following two other words given a trigram frequency pair and a list of Bigram frequencies. The probability is calculated as follows:

$$Prob(w3|w1, w2) = \frac{count(w1, w2, w3)}{count(w1, w2)}$$

An example: if we want to know the probability of the word "concepts" coming after the two words "i" and "love" it will be the count of the three words coming after each other "i love concepts" divided by count of the first two words "i love"

$$Prob(concepts|i, love) = \frac{count(i, love, concepts)}{count(i, love)}$$

The type of generateOneProb will be:

```
generateOneProb :: Fractional a => ((String,String,String),a) ->
[((String,String),a)] -> a

Example:
> generateOneProb (("the","man","is"),1)
[(("he","is"),1),(("is","great"),1),(("great","the"),1),(("the","man"),3)]

0.3333333333333333
```

9) **genProbPairs**
Generates a list of probabilities for each and every trigram given a list of Trigram frequencies and another list of Bigram frequencies. The type of genProbPairs will be: genProbPairs :: Fractional a => [((String,String,String),a)] -> [((String,String),a)] -> [((String,String,String),a)]
Every entry in that list is a pair representing statistical information about the probability of a word following other two words which is given by the previously described equation

$$Prob(w3|w1,w2) = \frac{count(w1,w2,w3)}{count(w1,w2)}$$

The type of generateOneProb will be:

```
genProbPairs :: Fractional a => [((String,String,String),a)] ->
[((String,String),a)] -> [((String,String,String),a)]+
```

```
Example:
> genProbPairs [(("the","man","is"),1),(("man","is","the"),1),(("is","the","man"),1
),(("the","man","."),1),(("man",".","the"),1),((".","the","man"),1),(("the","man","
saw"),1)] [(("man","is"),1),(("is","the"),1),(("man","."),1),((".","the"),1),(("the
","man"),3),(("man","saw"),1)]

[(("the","man","is"),0.3333333333333333),(("man","is","the"),1.0),(("is","the","man"
),1.0),(("the","man","."),0.3333333333333333),(("man",".","the"),1.0),((".","the","
man"),1.0),(("the","man","saw"),0.3333333333333333)]
```

10) **generateNextWord**

Given a list of two words and a probability pairs list, it randomly chooses the next word taking into consideration that the probability of this word following those two words is greater than 0.03.
Also, you should handle the case where you can not find the next word or no woord satisfies the condition giving a message `"Sorry, it is not possible to infer from current database"`
Hint: You can use the function randomZeroToX in your solution which you will find in `DataFile.hs`.

The type of generateNextWord will be:

```
generateNextWord :: (Ord a, Fractional a) => [String] ->
[((String,String,String),a)] -> String
```

```
Example:  A possible output to the following
> generateNextWord ["the","man"][(("the","man","is"),0.3333333333333333),(("man","is
","the"),1.0),(("is","the","man"),1.0),(("the","man","."),0.3333333333333333),(("man
",".","the"),1.0),((".","the","man"),1.0),(("the","man","saw"),0.3333333333333333)]
"saw"
```

11) **generateText**
According to the content of the docs list in the DataFile.hs file and given a string of 2 items (words/-punctuation) and number n, you should generate n words/punctuation taking into consideration the statistics mentioned above and return the whole text including the two words you started with.
The type of generateText will be:

```
generateText :: String -> Int -> String
```

```
Example:  A possible output to the following (using the shorter list docs)
> generateText "the man" 2
"the man saw the"
```

Note that: there are two docs lists in the DataFile.hs, a shorter one for testing and the longer one (commented) to try the whole project on at the end. The longer one is a preprocessed one from `https://simple.wikipedia.org/`