САНКТ-ПЕТЕРБУРГСКИЙ ПОЛИТЕХНИЧЕСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ ПЕТРА ВЕЛИКОГО
Институт компьютерных наук и кибербезопасности
Высшая школа технологий искусственного интеллекта
02.03.01 Математика и компьютерные науки

Отчет по курсовой работе по дисциплине: «Функциональное программирование»

«Парсер и диалог на основе N-граммного словаря»

Студент:
группы 5130201/20102
_____ Салимли А.

Преподаватель:
к.т.н.
_____ Моторин Д.Е.

«____» _____ 20___ г.

Санкт-Петербург 2024

Содержание

Введение

В данном отчете, описан результат работы и реализация двух частей курсовой работы по дисциплине «Функциональное программирование». В котором для каждой части, был реализован проект в stack, где был реализован «парсер» использующий аппликативные функторы и монады. Далее во второй части был реализован диалог на основе N-граммного словаря, где пользователь вводит имена файлов в которых записан текст. На основе этого, текст разбивается на предложения, строится N-граммный словарь, генерируются случайные фразы по словарю, и образуется диалог по последнему слову предложения, если же последнее слово не является ключем в N-граммном словаре, берем N-K слово (где K - позиция слова-ключа с конца), если же ни одно слово не является ключем, завершаем диалог.

Ограничения:

1. Все монадические вычисления должны быть записаны без do-нотации.
2. Все чистые функции записать в библиотеку Lib.hs, и ограничить доступ к вспомогательным
3. Тексты для диалога: «Подводная экспидиция», «Под небом Арктики» - Автор: Александр Романович Беляев

1. Постановка задачи

В курсовой работе необходимо: все монадические вычисление должны быть записаны без использования do-нотаций. Для каждой части задания создать проект в stack. Все чистые функции записать в библиотеку Lib.hs и ограничить доступ к вспомогательным функциям.

Необходимо реализовать две части:

1 - написать синтаксический анализатор (парсер), разбирающий строки, прочитанные из текстового файла .txt. Файл должен содержать значение и бинарные операции.

Значения: строки битов.

Бинарные операции: AND(&), OR(|), XOR(^).

Необходимо вычислить проанализированные выражения. Вывести его и результаты вычисления на экран. Пользователь должен ввести название файла.

2 - написать синтаксический анализатор разбирающий текст и генератор предложения текста по введённому слову. Для этого выполнить следующие задачи:

- Разобрать текст на предложения, предложения разделять символами: .!?:;() , удалить все символы пунктуации и цифры из слов и предложений.
- Составить модель N-грамм. Использовать модель биграмм и триграмм. По списку предложений составить словарь. Ключами являются одно или пару слов. Словарь сохранить в .txt файл.
- Реализовать взаимодействие с пользователем, пользователь вводит одно или пару слов. Программа возвращает строку случайной длинны в диапазоне от 2 до 15 слов, если слово - не ключ, то вывести сообщение. Фраза составляет путём добавления случайного слова или пары слов из списка значений текущего слова - ключа или пары - ключа до тех пор пока либо не будет сформирована предложение нужной длины либо не будет достигнут ключ у которого нет значений.
- Организовать диалог двух моделей N-грамм созданных на двух разных текстах. Пользователь задаёт начальное слово или пару слов и глубину M сообщений, которыми обмениваются модели. Ответ модели основывается на последнем слове из предложения оппонента (если последнее слово отсутствует словаре то предпоследняя и так далее пока не будет найдено подходящее слово или не закончится предложение).

2. Часть первая «Парсер»

Парсер - часть программы, преобразующей входные данные (как правило, текст) в некий структурированный формат, нужный для задач последующего их (данных) анализа и использования. Технически, парсер выполняет синтаксический анализ данных (например, текста).

2.1 Основная идея

Основная идея состоит в реализации парсера с помощью аппликативных функторов и монадических вычислений. Такой подход позволяет строить парсеры так, что бы при ошибки парсинга выводились верные результаты, и сообщения об ошибке, то есть использование класса Alternative предоставляет возможность обработки альтернативных вариантов. Главный принцип заключается в том, что парсер поэтапно преобразует входные данные, проверяет их на соответствие правилам и возвращает результат, если правила выполнены.

2.2 Исходный код

Lib.hs:

```
{-# LANGUAGE LambdaCase #-}
{-# LANGUAGE InstanceSigs #-}
module Lib (parseAndEvaluateFile) where
import Control.Applicative (Alternative(..))
import Data.Char (digitToInt)
newtype Parser tok a = Parser { runParser :: [tok] -> Maybe ([tok], a) }

instance Functor (Parser tok) where
  fmap f (Parser p) = Parser $ \input ->
    case p input of
      Nothing -> Nothing
      Just (rest, result) -> Just (rest, f result)
instance Applicative (Parser tok) where
  pure x = Parser $ \input -> Just (input, x)
  Parser pf <*> Parser px = Parser $ \input ->
    case pf input of
      Nothing -> Nothing
      Just (rest1, f) -> case px rest1 of
        Nothing -> Nothing
        Just (rest2, x) -> Just (rest2, f x)
instance Monad (Parser tok) where
  (>>=) :: Parser tok a -> (a -> Parser tok b) -> Parser tok b
  Parser p >>= f = Parser $ \input ->
    case p input of
      Nothing -> Nothing
      Just (rest, result) -> runParser (f result) rest

instance Alternative (Parser tok) where
  empty = Parser $ \_ -> Nothing
  Parser p1 <|> Parser p2 = Parser $ \input ->
    case p1 input of
      Nothing -> p2 input
      result -> result

satisfy :: (tok -> Bool) -> Parser tok tok
```

```haskell
satisfy pr = Parser $ \input -> case input of
  (c:cs) | pr c -> Just (cs, c)
  _ -> Nothing

char :: Eq tok => tok -> Parser tok tok
char c = satisfy (== c)

digit :: Parser Char Int
digit = digitToInt <$> satisfy (`elem` "01")

spaces :: Parser Char ()
spaces = () <$ many (satisfy (== ' '))

bitString :: Parser Char [Int]
bitString = spaces *> some digit <* spaces

virovS :: [Int] -> [Int] -> ([Int], [Int])
virovS a b =
    let maxLength = max (length a) (length b)
        padLeft xs = replicate (maxLength - length xs) 0 ++ xs
    in (padLeft a, padLeft b)

bitAnd, bitOr, bitXor :: [Int] -> [Int] -> [Int]
bitAnd a b = let (x, y) = virovS a b in zipWith (\x y -> if x == 1 && y == 1 then 1
      else 0) x y
bitOr  a b = let (x, y) = virovS a b in zipWith (\x y -> if x == 1 || y == 1 then 1
      else 0) x y
bitXor a b = let (x, y) = virovS a b in zipWith (\x y -> if x /= y then 1 else 0) x y

chainl1 :: Parser tok a -> Parser tok (a -> a -> a) -> Parser tok a
chainl1 p op = p >>= rest
  where
    rest x = (op <*> pure x <*> p >>= rest) <|> pure x
eof :: Parser tok ()
eof = Parser $ \input -> if null input then Just (input, ()) else Nothing

expression :: Parser Char [Int]
expression = chainl1 bitString opParser

opParser :: Parser Char ([Int] -> [Int] -> [Int])
opParser =
      (bitAnd <$ char '&')
  <|> (bitOr  <$ char '|')
  <|> (bitXor <$ char '^')

parseAndEvaluate :: String -> Either String [Int]
parseAndEvaluate str =
  case runParser (expression <* eof) str of
    Nothing -> Left "Ошибка парсинга!"
    Just ("", result) -> Right result
    Just _ -> Left "Не удалось разобрать всю строку."

parseAndEvaluateFile :: FilePath -> IO ()
parseAndEvaluateFile filename =
  readFile filename >>= mapM_ putStrLn . processFile . lines
  where
    processFile :: [String] -> [String]
    processFile = map evaluateLine
    evaluateLine :: String -> String
    evaluateLine line = case parseAndEvaluate line of
      Left err -> "Ошибка: " ++ err
      Right result -> line ++ " = " ++ concatMap show result
```

Main.hs:

```haskell
module Main where
import Lib (parseAndEvaluateFile)

main :: IO ()
main =
    putStrLn "Введите имя файла: " >>
    getLine >>= \filename ->
    parseAndEvaluateFile filename
```

2.3 Особенности реализации

newtype Parser tok a:

Представляет парсер, который принимает список токенов [tok] и возвращает результат в виде пары Maybe ([tok], a), где:

- [tok] — остаток нераспознанных токенов.
- a — результат парсинга.

instance Functor (Parser tok):

Позволяет преобразовать результат парсинга, применяя функцию f к результату типа a.

instance Applicative (Parser tok):

Позволяет комбинировать парсеры, сохраняя контекст входных данных.

instance Monad (Parser tok):

Позволяет передавать результат одного парсера в другой, что удобно для построения зависимых парсеров.

instance Alternative (Parser tok):

Реализует логику альтернативы: если первый парсер завершился неудачей, используется второй и т.д.

satisfy:

Базовый парсер, проверяющий, соответствует ли текущий токен заданному предикату.

char:

Парсер, который распознаёт конкретный символ.

digit:

Парсер для двоичных цифр 0 и 1, преобразующих символы в числа.

spaces:

Пропускает пробелы.

bitString:

Парсит последовательность двоичных цифр с возможными пробелами в начале и конце.

virovS:

Выравнивает длины двух списков, добавляя нули в начале меньшего списка.

bitAnd, bitOr, bitXor:

Выполняют побитовые операции AND, OR и XOR над двумя выровненными списками.

chainl1:

Реализует парсер для левой ассоциативной цепочки операций.

eof:

Убедится, что входные данные полностью обработаны.

expression:

Главный парсер выражений, поддерживающий операции &, |, ^.

opParser:

Парсер для операторов &, |, ^.

parseAndEvaluate:

Парсит и вычисляет выражение, возвращая результат либо сообщение об ошибке.

parseAndEvaluateFile:

Читает файл, обрабатывает строки как отдельные выражения и выводит результат или сообщение об ошибке для каждой строки.

Main.hs: Выводит сообщение - ввести имя файла, считывает имя файла, введённое пользователем.

Вызывает функцию parseAndEvaluateFile из модуля Lib, которая, описана выше.

3. Часть 2 «Диалог на основе N-граммного словаря»

3.1 Основная идея

Основная идея этого кода заключается в создании диалога на основе генерации предложений с помощью N-граммных моделей, которые извлекаются из текста (текст разбивается на предложения).

Обработка текста:

- Разбить текст на предложения встречая символы .!?;:() , удалить символы пунктуации и привести текст к единому регистру.
- Составить список слов из каждого предложения.

Создание N-граммного словаря:

- Построить словарь, где ключи - это последовательности из N слов, а значения - возможные продолжения этих последовательностей.

Генерация текста:

- Использовать N-граммный словарь для последовательного выбора следующего слова на основе текущего ключа.
- Если ключ не найден, использовать предыдущие слова в убывающем порядке (по количеству слов в ключе).

Диалог:

- Реализовать взаимодействие двух собеседников, каждый из которых использует свой словарь для генерации предложений.
- Передавать результат генерации одного собеседника в качестве входных данных для другого.

3.2 Математическое описание

В нашем случае реализована базовая N-граммная модель, без введения правил грамматик и вероятностей. Описание ниже касается полной N-граммной модели.

N-граммные языковые модели используют предположение Маркова, которое утверждает, что в контексте языкового моделирования вероятность следующего слова в последовательности зависит только от предыдущего(их) слова(слов). Следовательно, приближение вероятности слова, учитывая его контекст в n-граммной модели, можно формализовать следующим образом:

$$P(w_t \mid w_{1:t-1}) \approx P(w_t \mid w_{t-N+1:t-1})$$

где t — это количество слов во всей последовательности, а N — размер контекста (униграмма (1), биграмма (2) и т. д.).

Так же N-грамму при введение вероятности можно изобразить в виде цепи Маркова.

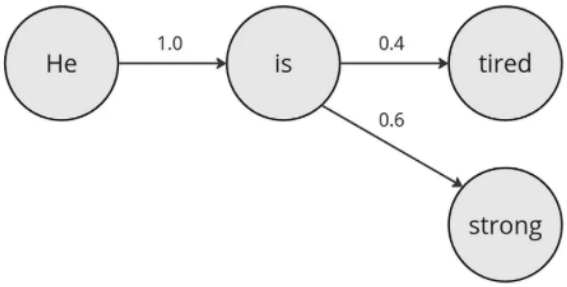

Рис.1 N-граммный словарь в виде цепи Маркова

Для такого представления вероятность можно определить такой формулой:

$$P(w_t \mid w_{t-N+1:t-1}) \approx \frac{C(w_{t-N+1:t-1} w_t)}{C(w_{t-N+1:t-1})}$$

Вероятность следующего слова, учитывая N предыдущих: числитель — сколько раз результирующая последовательность встречается в данных, знаменатель — сколько раз последовательность предыдущих слов встречается в данных.

Такая N-граммная модель реализована в библиотеках LLM.

3.3 Исходный код

Lib.hs:

```
{-# LANGUAGE LambdaCase #-}
module Lib (prodlojaNorm, nSlovarchik, saveSlovarchik, slovoKomputera, dialog) where
import Data.Char (isLetter, toLower)
import Data.List (nub, intercalate)
import Data.List.Split (splitWhen)
import Data.Map.Strict (Map)
import qualified Data.Map.Strict as Map
import System.Random (randomRIO)

prodlojaNorm :: String -> Maybe [[String]]
prodlojaNorm text
    | null text = Nothing
    | otherwise = Just $ filter (not . null) $ map (words . normPredloja) $ predlojaS text
  where
    razdelit :: String
    razdelit = ".!?;:()"
    predlojaS :: String -> [String]
    predlojaS t = splitWhen (`elem` razdelit) t
    normPredloja :: String -> String
    normPredloja = map toLower . filter (\c -> isLetter c || c == ' ')

nSlovarchik :: Int -> [[String]] -> Map String [String]
nSlovarchik n predlojaS =
  Map.filter (not . null) $
    Map.fromListWith (\new old -> nub (old ++ new)) $
      [ (unwords key, [value])
        | predlojNSlovar <- predlojaS
        , (key, value) <- slovarVPredloj n predlojNSlovar
```

```
      ]
  where
    slovarVPredloj :: Int -> [String] -> [([String], String)]
    slovarVPredloj n words =
      [ (key, unwords value)
        | i <- [0 .. length words - 1]
        , k <- [1 .. min (n - 1) (length words - i)]
        , let key = take k (drop i words)
        , let rest = drop (i + k) words
        , not (null rest)
        , l <- [1 .. min (n - k) (length rest)]
        , let value = take l rest
        , not (null value)
      ]++
      [ (key, "")
        | i <- [max 0 (length words - (n - 1)) .. length words - 1], i >= 0
        , k <- [1 .. min (n - 1) (length words - i)]
        , let key = take k (drop i words)
        , length key == k
        , drop (i + k) words == []
      ]
saveSlovarchik :: FilePath -> Map String [String] -> IO ()
saveSlovarchik filePath dict =
    let vhod = Map.toAscList dict
        filterVhod = filter (not . null . snd) vhod
        formattedvhod = map formatNorm filterVhod
    in writeFile filePath (unlines formattedvhod)
  where
    formatNorm (key, values) =
      let values' = nub values
          nePusto = if all null values' then [] else filter (not . null) values'
          valuesStr = intercalate ", " (map show nePusto)
      in key ++ " : [" ++ valuesStr ++ "]"

slovoKomputera :: Map String [String] -> [String] -> IO (Either String [String])
slovoKomputera dict keyWords = generate (unwords keyWords) [] maxLength
  where
    maxLength = 15
    minLength = 2
    generate _ acc 0
        | length acc >= minLength = return (Right (reverse acc))
        | otherwise = return (Left "Ошибкочка!")
    generate curKey acc n =
        case Map.lookup curKey dict of
            Nothing
                | length acc >= minLength -> return (Right (reverse acc))
                | otherwise -> return (Left $ "Завершен!")
            Just [] -> return (Left $ "Ключ '" ++ curKey ++ "' найден но продолжение
отсутствует(")
            Just nextOptions ->
                randomRIO (0, length nextOptions - 1) >>= \idx ->
                let nextValue = nextOptions !! idx
                in if null nextValue
                   then return (Right (reverse acc))
                   else let continuation = words nextValue
                        in generate (unwords $ drop 1 (words curKey ++ continuation))
(continuation ++ acc) (n - 1)

keyPoUbiv :: [String] -> Map String [String] -> Maybe String
keyPoUbiv [] _ = Nothing
keyPoUbiv (x:xs) dict
    | Map.member x dict = Just x
    | otherwise = keyPoUbiv xs dict

dialog :: Map String [String] -> Map String [String] -> [String] -> Int -> IO ()
dialog dict1 dict2 initialPhrase steps =
    let logDialog prevPhrase d1 d2 remainingSteps sobesNumber
            | remainingSteps == 0 = putStrLn "Диалог завершён"
            | otherwise =
                let currentsobes = if sobesNumber `mod` 2 == 1 then 1 else 2
                    dict = if currentsobes == 1 then d1 else d2
                    otherDict = if currentsobes == 1 then d2 else d1
```

```
                    sobesLabel = "Чел " ++ show currentsobes ++ ": "
                in slovoKomputera dict prevPhrase >>= \case
                    Left err -> putStrLn (sobesLabel ++ err) >> putStrLn "Диалог завершён"
                    Right [] -> putStrLn (sobesLabel ++ "Ошибочка генерации!(") >> putStrLn
"Диалог завершён"
                    Right generatedPhrase ->
                        let phraseWithInitial = unwords prevPhrase ++ " " ++ unwords
generatedPhrase
                            lastWords = reverse (words phraseWithInitial)
                        in putStrLn (sobesLabel ++ "(Ключ: " ++ unwords prevPhrase ++ ") " ++
phraseWithInitial) >>
                           case keyPoUbiv lastWords otherDict of
                               Nothing -> putStrLn "Ключ не найден !!!"
                               Just nextKey -> let nextPhrase = words nextKey
                                               in logDialog nextPhrase d1 d2 (remainingSteps -
1) (sobesNumber + 1)
    in logDialog initialPhrase dict1 dict2 steps 1
```

Main.hs:

```
{-# LANGUAGE LambdaCase #-}
{-# OPTIONS_GHC -Wno-unrecognised-pragmas #-}
{-# HLINT ignore "Use lambda-case" #-}
{-# HLINT ignore "Use >=>" #-}
module Main where
import Lib
import System.IO (hFlush, stdout)
import qualified Data.Map.Strict as Map

main :: IO ()
main =
    putStrLn "Какую задачку рассмотрим?" >>
    putStrLn "1 - Разделить текст на предл." >>
    putStrLn "2 - Словарь N-грамм" >>
    putStrLn "3 - Сгенерировать предложение" >>
    putStrLn "4 - Диалог двух челов" >>
    hFlush stdout >>
    getLine >>= \choice ->
    case choice of
        "1" -> part1
        "2" -> part2
        "3" -> part3
        "4" -> part4
        _   -> putStrLn "Выберите из цифр, которые в меню" >> main
part1 :: IO ()
part1 =
    putStrLn "Введите имя файла:" >>
    hFlush stdout >>
    getLine >>= \filename ->
    readFile filename >>= \content ->
    case prodlojaNorm content of
        Nothing -> putStrLn "Тут пусто ("
        Just sentences -> putStrLn "Результат:" >> mapM_ print sentences
part2 :: IO ()
part2 =
    putStrLn "Введите имя файла для словаря:" >>
    hFlush stdout >>
    getLine >>= \filename ->
    readFile filename >>= \content ->
    case prodlojaNorm content of
        Nothing -> putStrLn "Тут пусто ("
        Just sentences ->
            putStrLn "Введите N-грамму:" >>
            hFlush stdout >>
            getLine >>= \nInput ->
            let n = read nInput :: Int
                dict = nSlovarchik n sentences
                dictFileName = show n ++ "_gram_slovar.txt" ++ filename
            in if Map.null dict
               then putStrLn "Ошибка!("
               else saveSlovarchik dictFileName dict >>
                    putStrLn "Ура работает!" >>
                    putStrLn ("Словарь сохранён: " ++ dictFileName) >>
```

```
                    putStrLn "Оп оп:" >>
                    readFile dictFileName >>= putStrLn
part3 :: IO ()
part3 =
    putStrLn "Введите имя файла для словаря:" >>
    hFlush stdout >>
    getLine >>= \filename ->
    readFile filename >>= \content ->
    case prodlojaNorm content of
        Nothing -> putStrLn "Тут пусто ("
        Just sentences ->
            putStrLn "Введите N-грамму:" >>
            hFlush stdout >>
            getLine >>= \nInput ->
            let n = read nInput :: Int
                dict = nSlovarchik n sentences
            in if Map.null dict
               then putStrLn "Упс ! Невозможно построить так!"
               else putStrLn "Введите слово/ предлож. для генерации:" >>
                    hFlush stdout >>
                    getLine >>= \input ->
                    let keyWords = words input
                        key = unwords keyWords
                    in case Map.lookup key dict of
                        Nothing -> putStrLn ("Упс! Ключа '" ++ key ++ "' нет в словаре ((")
                        Just _ -> slovoKomputera dict keyWords >>= \case
                            Left err -> putStrLn err
                            Right phrase -> putStrLn ("Сгенерированная фраза: " ++ unwords
(keyWords ++ phrase))
part4 :: IO ()
part4 =
    putStrLn "Введите имя первого файла (1):" >>
    hFlush stdout >>
    getLine >>= \file1 ->
    putStrLn "Введите имя второго файла (2):" >>
    hFlush stdout >>
    getLine >>= \file2 ->
    readFile file1 >>= \content1 ->
    readFile file2 >>= \content2 ->
    case (prodlojaNorm content1, prodlojaNorm content2) of
        (Nothing, _) -> putStrLn "Упс! Пусто в первом файле!"
        (_, Nothing) -> putStrLn "Упс! Пусто во втором файле!"
        (Just sentences1, Just sentences2) ->
            putStrLn "Введите N-грамму:" >>
            hFlush stdout >>
            getLine >>= \nInput ->
            let n = read nInput :: Int
                dict1 = nSlovarchik n sentences1
                dict2 = nSlovarchik n sentences2
            in putStrLn "Введите начальное слово или предложение:" >>
               hFlush stdout >>
               getLine >>= \input ->
               let initialKey = words input
               in putStrLn "Введите глубину диалога:" >>
                  hFlush stdout >>
                  getLine >>= \mInput ->
                  let m = read mInput :: Int
                  in dialog dict1 dict2 initialKey m
```

3.4 Особенности реализации

Для Lib.hs:

prodlojaNorm:

Нормализует текст, разбивая его на предложения и удаляя лишние символы (знаки пунктуации).

Каждое предложение преобразуется в список слов в нижнем регистре.

Подготавливает текст для последующего построения N-граммного словаря.

nSlovarchik:

Создаёт N-граммный словарь. Каждому ключу (фрагменту текста длиной до N-1 слов) сопоставляет возможные продолжения.

Обрабатывает предложения из текста и строит словарь в виде Map, где ключ - последовательность слов, а значение - список слов-продолжений.

Это основа для генерации текста и анализа последовательностей слов.

saveSlovarchik:

Сохраняет N-граммный словарь в файл. Каждый ключ и его значения записываются в человекочитаемом формате.

Позволяет сохранить результаты обработки для последующего анализа или использования в других программах.

slovoKomputera:

Генерирует последовательность слов, начиная с указанного ключа.

Последовательно выбирает случайное продолжение из словаря и добавляет его к результату.

Если продолжения нет, генерация заканчивается.

keyPoUbiv:

Рекурсивно ищет ключ в словаре, начиная с последнего слова списка. Если ключ не найден, пробует с предпоследним словом и так далее.

Помогает находить подходящий ключ для продолжения текста, даже если предыдущий ключ не существует в словаре.

dialog:

Реализует диалог между двумя "собеседниками". Каждый собеседник поочередно генерирует текст, используя N-граммный словарь.

Один собеседник берёт ключ из словаря первого текста, другой - из второго. Генерация продолжается до тех пор, пока не будут исчерпаны шаги или не найдётся ключ.

Позволяет создать симуляцию диалога между двумя текстами.

В результате получаем что:

prodlojaNorm сначала нормализует текст, затем nSlovarchik строит N-граммный словарь на основе нормализованных предложений, после чего saveSlovarchik сохраняет словарь. slovoKomputera использует словарь для генерации текста. keyPoUbiv помогает находить ключи для продолжения текста и dialog объединяет всё вместе, создавая взаимодействие между текстами.

Main.hs:

1. Главное меню

Пользователю нужно выбрать одну из четырёх функций (part1, part2, part3, part4).

Выводится список задач.

Запрашивается ввод числа (от 1 до 4) для выбора задачи.

Если введённое число некорректно, меню повторяется.

Основные функции:

part1 — Разделение текста на предложения

Читает текст из файла и разделить его на предложения.

Запрашивает имя файла.

Читает содержимое файла.

Преобразует текст в список предложений с помощью функции prodlojaNorm.

Если текст пустой — сообщает об ошибке. Иначе — выводит полученные предложения.

part2 — Создание N-граммного словаря

Строит N-граммный словарь и сохранить его в файл.

Запрашивает имя файла с текстом.

Читает текст и нормализует его через prodlojaNorm.

Запрашивает значение N (размер N-граммы).

Создаёт словарь N-грамм через nSlovarchik.

Если словарь пуст — сообщает об ошибке. Иначе:

Сохраняет словарь в файл.

Выводит содержимое файла для подтверждения.

part3 — Генерация текста

Использует N-граммный словарь для генерации продолжения текста.

Запрашивает имя файла для создания словаря.

Читает и нормализует текст.

Запрашивает N для создания словаря.

Просит ввести начальные слова для генерации.

Проверяет, есть ли ключ в словаре:

Если ключ отсутствует — сообщает об этом.

Если ключ найден — с помощью slovoKomputera генерирует продолжение и выводит его.

part4 — Диалог между текстами

Создает "диалог" между двумя текстами, используя N-граммные словари.

Запрашивает имена двух файлов с текстами.

Читает и нормализует оба текста.

Запрашивает N для создания словарей.

Создаёт два N-граммных словаря (dict1, dict2).

Просит ввести начальное слово или предложение для начала диалога.

Запрашивает глубину диалога (количество шагов).

Вызывает dialog для генерации последовательности, чередуя словари:

4. Результаты работы

4.1 Результаты работы «Парсер»

Для запуска программы необходимо прописать в пути папки с проектом команду:

```
stack exec KursachHaskell-exe
```

Исходный текст:

1 ^ 1

1 ^ 0

0 ^ 1

0 ^ 0

1 & 1

1 & 0

0 & 1

0 & 0

1 | 1

1 | 0

0 | 1

0 | 0

Результат:

1 ^ 1 = 0

1 ^ 0 = 1

0 ^ 1 = 1

0 ^ 0 = 0

1 & 1 = 1

1 & 0 = 0

0 & 1 = 0

0 & 0 = 0

1 | 1 = 1

1 | 0 = 1

0 | 1 = 1

0 | 0 = 0

Текст с ошибкой:

0101 & 0011

1111 | 0000

1010 ^ 1100

0101 abv 1111

0000 & 1111

Результат:

0101 & 0011 = 0001

1111 | 0000 = 1111

1010 ^ 1100 = 0110

Ошибка: Ошибка парсинга!

0000 & 1111 = 0000

4.2 Результаты работы «Диалога на основе N-граммного словаря»

Для запуска программы необходимо прописать в пути папки с проектом команду:

stack exec kursachPart2-exe

После чего выводится **меню** программы:

Какую задачку рассмотрим?

1 - Разделить текст на предл.

2 - Словарь N-грамм

3 - Сгенерировать предложение

4 - Диалог двух челов

Отрывок из оригинального текста:

Podvodnaya Expediciya!??!
-- Eto mysl!
Otlazhiv knigu v storonu, mister Solli eshche raz povtoril:
-- Da. Eto mysl! -- i pogruzilsya v razdumye.
Mister Solli -- udachlivyy nyu-yorkskiy fabrikan i birzhevik, nazhivshiysya na voennykh postavkakh vo vremya mirovoy voyny.
Voyna -- prekrasnaya veshch dlya takikh lyudey. Chem vyshe roslI gory trupov na polyakh srazheniy, tem bolshe okruglyalsya tekushchiy schet mistra Solli. K kontsu voyny mister Solli "stoil" neskolko milliardov dollarov.
No kogda on dostig vershiny finansovogo mogushchestva, s nim sluchilas nepriyatnaya istoriya. Roskoshnye obedy, obilno oroshayemye tonkimi vinami, vyzvali zabolevanie mozgovykh sosudov. I on sleg ot mozgovogo udara, kogda menee vsego ozhidal etogo. U nego otnyalis pravaya ruka i noga. Udar byl legkiy, i pri tchatelnom meditsinskom ukhode cherez neskolko mesyatsev bolnoy opravilsya. Paralich, kazalos, proshel. No vrach kategoricheski zapretil mistru Solli vozvrashchat'sya k kommercheskoy deyatel'nosti.
-- Dovol'no, porabotali, -- skazal vrach. -- Yesli vy khotite sokhranit' zdorov'ye, vy dolzhny sovershenno izmenit' obraz zhizni. Puteshestvuyte, zanimaytes kollektsionirovaniem, blagotvoritel'nost'yu -- odnim slovom, chem khotite, lish by zapolnit' vashe vremya i razvlech'sya, no izbegayte umstvennogo i nervnogo napryazheniya. Inache ya ne otvechayu za vashu zhizn.
Posle takogo ultimatum pered misterom Solli vstal nelegkiy vopros: chem napolnit' zhizn i kak

Оригинальный текст

Разбитый текст (1):

["salvators","coming","tomorrow"]
["my","fevers","kept","me","away","just","when","there","was","a","lot","for","us","to","talk","about","cristo","was","saying","to","baltasar","in","his","shop"]
["cock","your","ears","brother","and","dont","interrupt","me","sos","i","wont","forget","anything"]
["still","weak","after","the","fever","cristo","paused","marshalling","his","thoughts","then","continued"]
["weve","done","a","hell","of","a","lot","for","zurita","brother"]
["hes","more","brass","than","both","of","us","but","hes","out","to","get","still","more"]
["he","wants","to","catch","the","seadevil","baltasar","made

N-граммный словарь (N = 3) (2):

a : ["daily", "daily appearance", "final", "final snort", "big", "big swell", "visit", "trip", "trip underwater", "lookup", "choppy", "choppy sea", "way", "way through", "biggish", "biggish cave", "stout", "stout iron", "closer", "closer look", "builtin", "builtin lock", "look", "look round", "new", "new search", "high", "high solid", "piece", "piece of", "small", "small steel", "spyhole", "spyhole shut", "regular", "regular fortress", "chink", "chink anywhere", "peep", "peep through", "sign", "sign of", "specially", "specially sharp", "single", "single sound", "faint", "faint smile", "god", "god and", "seadevil", "seadevil down", "god up", "man", "man who", "broken", "broken leg", "mustang", "split", "split skull", "nice", "nice girl", "miracleworker", "man of", "case", "case as", "fortune", "high wall", "house", "house

Случайно сгенерированное предложение (N = 3) (3):

Сгенерированная фраза: afriku i tchatelnom pri meditsinskom ukhode cherez neskolko mesyatsev bolnoy opravilsya

Отрывок второго текста для диалога:

“Salvator’s coming tomorrow. My fever’s kept me away just when there was a lot for us to talk about,” Cristo was saying to Baltasar in his shop. “Cock your ears, brother, and don’t interrupt me, so’s I won’t forget anything.”

Still weak after the fever Cristo paused, marshalling his thoughts, then continued:

“We’ve done a hell of a lot for Zurita, brother. He’s more brass than both of us but he’s out to get still more. He wants to catch the ‘sea-devil’-” Baltasar made to speak.

“Hold it, brother, else I forget something. Zurita wants the ‘sea-devil’ to slave for him. And d’you know what the ‘sea-devil’ is? A regular treasure. Untold riches. The ‘sea-devil’ can pick pearls from the sea-bottom-any amount of ‘em. But that’s not all. On the sea-bottom there’s plenty of

Диалог (N = 19, M = 9, слово: afriku):

Введите имя первого файла (1):
First.txt
Введите имя второго файла (2):
Scd.txt
Введите N-грамму:
19
Введите начальное слово или предложение:
afriku
Введите глубину диалога:
9
Чел 1: (Ключ: afriku) afriku lvov na okhotitsya i eto kak ruzvelt sdelal
Чел 2: (Ключ: kak) kak mnogie i ego tovarishhi
Чел 1: (Ключ: ego) ego litse krasnom zhyrnom grimasy brezglivosti
Чел 2: (Ключ: ego) ego v ego snjarjadili i sberezhenijami nebolshimi dorogu
Чел 1: (Ключ: i) i tsel yego dostoynuyu nayti nadezhdu teryal uzhe on kogda vot pomoshch na prishel sluchay prostoy zhizni
Чел 2: (Ключ: na) na peshchernyj obrazovalsja zavoda razvalinah poselok bezrabotnyh
Чел 1: (Ключ: na) na stole pismennom oblozhki poloskami golubymi i serymi s knizhku pestroyu
Чел 2: (Ключ: s) s ego snjarjadili i sberezhenijami nebolshimi ego dorogu v
Чел 1: (Ключ: v) v tomov bolee soderzhit institute smitsonianskom
Диалог завершён

5. Заключение

В результате выполнения частей заданий из курсовой работы, были реализованы:

- Парсер работающий с аппликативными функторами;
- Парсер разбивающий текст на предложения;
- N-граммный словарь;
- Генератор случайного предложения по N-граммному словарю;
- Диалог работающий по N-граммному словарю.

Была освоена работа с функторами, монадами.

Достоинства: Можно анализировать любой текст, составляя другие случайные предложения и диалоги. В первой части, можно парсить выражения подряд.

Недостатки: Из-за реализации словаря на основи Map, метод автоматически сортирует слова и предложения по алфавиту.

Масштабируемость: При введении дополнительных условий - например грамматику или вероятности преобразовав модель в цепь Маркова, можно получить полноценную LLM - модель.

6. Список источников

[1] W. Kurt. Get Programming with Haskell. Москва: ДМК, 2019. 481 с.

[2] В. Брагилевский. Haskell in Depth. Москва: ДМК, 2019. 340 с.

[3] Computer science center. Д. Н. Москвин. Лекция 9. Использование аппликативных функторов. Электронный ресурс: https://compscicenter.ru/courses/func-prog/2022-spring/classes/8880/. Дата последнего обращения (23.11.24 г.)

7. Приложение А

Ссылка на репозиторий с кодом: https://github.com/MathematicLove/Haskellove