МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ
федеральное государственное автономное образовательное учреждение
высшего образования «Санкт-Петербургский политехнический
университет Петра Великого»

Институт компьютерных наук и кибербезопасности

Высшая школа технологий искусственного интеллекта

Направление: 02.03.01 «Математика и компьютерные науки»

Отчет о выполении курсовой работы

«Генератор диалогов»

по дисциплине «Функциональное программирование»
Вариант 26

Студент,
группы 5130201/30101 _____ Мелещенко С.И.

Преподаватель

_____ Моторин Д.Е.

«____»_____ 2025г.

Санкт-Петербург, 2025

Содержание

Введение

Игра в лестницы — это увлекательное упражнение, требующее от игроков (может участвовать любое количество) глубоких знаний словарного запаса. Игроки начинают с двух слов одинаковой длины. Одно из них можно считать началом, другое — концом. Задача состоит в том, чтобы найти цепочку других слов, связывающую начальное слово с конечным, где каждая соседняя пара слов отличается одной буквой. Это означает, что игрок начинает со слова, меняет одну букву, чтобы получить новое слово, и продолжает эти преобразования, пока не будет построена полная цепочка от начала до конца. Если в игре участвуют несколько игроков, выигрывает тот, у кого цепочка короче. Вот пример: cat $\rightarrow$ sat $\rightarrow$ sag $\rightarrow$ dag $\rightarrow$ dog.

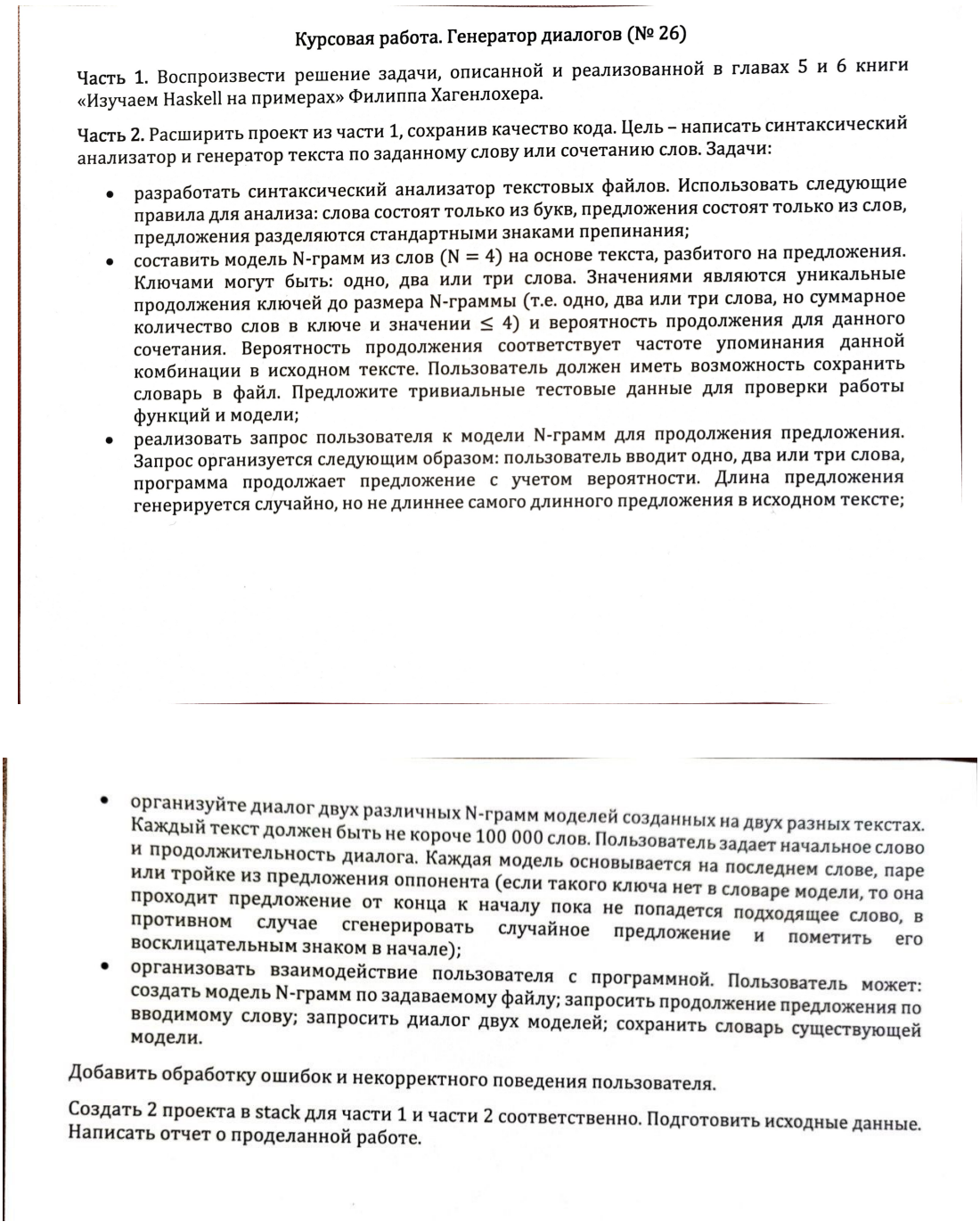
Мы разрешим игроку не только менять одну букву, но и добавлять совершенно новую букву, удалять букву, а также произвольно переставлять буквы. Это изменение делает игру гораздо интереснее, поскольку теперь можно находить пути между словами разной длины. С этими правилами можно найти решение для слов find и solution (например, find $\rightarrow$ fins $\rightarrow$ ions $\rightarrow$ loins $\rightarrow$ tonsil $\rightarrow$ lotions $\rightarrow$ solution).

N-грамма — это просто последовательность из n элементов (звуков, слогов, слов или символов), идущих в каком-то тексте подряд. На практике чаще имеют в виду ряд слов (реже — символов). Последовательность из двух элементов называют биграмма, из трёх элементов — триграмма.

Диалог - последовательность взаимосвязанных речевых актов не менее двух участников, каждый из которых попеременно становится то говорящим, то адресатом речи.

1 Постановка задачи

Задание приведено ниже [Рис. 1]:



Курсовая работа. Генератор диалогов (№ 26)

Часть 1. Воспроизвести решение задачи, описанной и реализованной в главах 5 и 6 книги «Изучаем Haskell на примерах» Филиппа Хагенлохера.

Часть 2. Расширить проект из части 1, сохранив качество кода. Цель – написать синтаксический анализатор и генератор текста по заданному слову или сочетанию слов. Задачи:

- разработать синтаксический анализатор текстовых файлов. Использовать следующие правила для анализа: слова состоят только из букв, предложения состоят только из слов, предложения разделяются стандартными знаками препинания;
- составить модель N-грамм из слов (N = 4) на основе текста, разбитого на предложения. Ключами могут быть: одно, два или три слова. Значениями являются уникальные продолжения ключей до размера N-граммы (т.е. одно, два или три слова, но суммарное количество слов в ключе и значении ≤ 4) и вероятность продолжения для данного сочетания. Вероятность продолжения соответствует частоте упоминания данной комбинации в исходном тексте. Пользователь должен иметь возможность сохранить словарь в файл. Предложите тривиальные тестовые данные для проверки работы функций и модели;
- реализовать запрос пользователя к модели N-грамм для продолжения предложения. Запрос организуется следующим образом: пользователь вводит одно, два или три слова, программа продолжает предложение с учетом вероятности. Длина предложения генерируется случайно, но не длиннее самого длинного предложения в исходном тексте;
- организуйте диалог двух различных N-грамм моделей созданных на двух разных текстах. Каждый текст должен быть не короче 100 000 слов. Пользователь задает начальное слово и продолжительность диалога. Каждая модель основывается на последнем слове, паре или тройке из предложения оппонента (если такого ключа нет в словаре модели, то она проходит предложение от конца к началу пока не попадется подходящее слово, в противном случае сгенерировать случайное предложение и пометить его восклицательным знаком в начале);
- организовать взаимодействие пользователя с программной. Пользователь может: создать модель N-грамм по задаваемому файлу; запросить продолжение предложения по вводимому слову; запросить диалог двух моделей; сохранить словарь существующей модели.

Добавить обработку ошибок и некорректного поведения пользователя.

Создать 2 проекта в stack для части 1 и части 2 соответственно. Подготовить исходные данные. Написать отчет о проделанной работе.

Рис. 1. Задание

Цель работы: разработать программный модуль синтаксического анализатора и генератора текста на основе N-граммных моделей для автоматической обработки текстовых данных и генерации текстовых про-

должений с учетом вероятностных характеристик исходного текста и интерактивную среду для генерации текста и диалога между двумя различными текстовыми моделями.

Задачи.

1. Разработать синтаксический анализатор текстовых файлов, осуществляющий разбиение текста на предложения по стандартным знакам препинания, выделение слов, состоящих только из буквенных символов, очистку текста от некорректных символов и цифр.

2. Реализовать построение N-граммной модели (N=4) на основе проанализированного текста.

3. Создать генератор текста, обеспечивающий обработку пользовательского запроса, генерацию продолжения с учетом вероятностной модели, случайное определение длины предложения в пределах максимальной длины исходных предложений.

4. Реализовать механизм диалога между двумя различными N-граммными моделями, где каждая модель генерирует реплику на основе последних слов оппонента.

5. Разработать взаимодействие пользователя с программой.

6. Обеспечить корректную обработку ошибок и некорректного ввода.

7. Разработать тестовый набор данных для проверки и протестировать.

2 Техническое задание

1. Общие сведения

1.1. Наименование программы: «Синтаксический анализатор и генератор текста по заданному слову или сочетанию слов».

1.2. Основание для разработки: учебный проект.

1.3. Назначение программы: обработка текстовых данных, построение N-граммных моделей и генерация текстовых продолжений с учетом вероятности.

2. Функциональные требования

2.1. Синтаксический анализ текста

- Программа должна обрабатывать текстовые файлы.

-Названия файла для считывания вводит пользователь.

- Предложения разделяются стандартными знаками препинания: «.», «!», «?».

- Предложения состоят из слов, разделенных « » (пробел), « - », «: », «; », «,- ».

2.2. Построение N-граммной модели

- Модель строится на основе текста.

- Параметр N = 4.

- Ключи модели: 1-3 слова.

- Значения модели - уникальные продолжения ключей до размера N-граммы. Вероятность продолжения рассчитывается как количество упоминаний комбинации в исходном тексте (ключи + значения) деленое на количество упоминаний ключа в исходном тексте.

- Модель сохраняется в файл .

2.3. Генерация текста

- Пользователь вводит начальную последовательность слов (1–3 слова).

- Программа генерирует продолжение с учетом вероятностей из модели.

- Длина предложения - случайное значение, не превышающее длину самого длинного предложения в исходном тексте.

- Критерий остановки генерации - достигнута максимальная длина

предложения.

2.4. Диалог двух N-граммных моделей

- Пользователь должен иметь возможность инициировать диалог между двумя моделями.

- Пользователь задает начальное слово-триггер и максимальное количество реплик в диалоге.

- Каждая модель, основываясь на последней сгенерированной реплике оппонента, должна извлечь ключ из реплики оппонента. Если для извлеченного ключа (1-3 слова) в словаре модели есть продолжения, сгенерировать ответ на его основе. Иначе сгенерировать случайное предложение, отметив его восклицательным знаком в начале.

2.5. Взаимодействие с пользователем (Интерактивный режим)

- Создать/Загрузить модель - задать имя файла с текстом.

- Сгенерировать продолжение - запрашиваем у пользователя ключ (1-3 слова) и выводим сгенерированное продолжение.

- Запустить диалог - запросить у пользователя две ранее загруженные модели, начальное слово и продолжительность диалога. Отобразить сгенерированный диалог.

- Сохранить текущую активную N-граммную модель в файл для последующей загрузки.

3. Нефункциональные требования

3.1. Производительность

- Обработка текстовых файлов объемом до 10 МБ.

- Время построения модели не более 20 секунд.

3.2. Обработка ошибок

- Обработка ситуации, когда файл с текстом не существует.

- Защита от ввода пользователем неверного типа данных, недопустимого значения при выборе пункта в меню.

4. Условия испытаний

4.1. Проверка корректности синтаксического анализа

- Разбиение текста на предложения и слова.

- Игнорирование знаков препинания.

4.2. Проверка построения N-грамм

- Соответствие ключей и значений модели исходному тексту.

- Корректность расчета вероятностей.

4.3. Проверка генерации текста

- Соответствие сгенерированного текста вероятностной модели.

- Соблюдение ограничений на длину предложения.

4.4. Проверка механизма диалога

- Корректность выбора ключа для генерации ответа на реплику оппонента.

- При отсутствии определенного ключа поиск ключа меньшей длины и генерация случайного предложения при его отсутствии.

- Соблюдение заданной пользователем длины диалога.

5. Стадии разработки

1) Реализация синтаксического анализатора текста;

2) построение N-граммной модели и расчет вероятностей;

3) реализация генератора текста;

4) реализация механизма диалога двух моделей;

5) разработка интерактивного пользовательского интерфейса;

6) интеграция всех модулей и отладка взаимодействия;

4) тестирование на различных данных;

5) документирование кода.

3 Часть 1

3.1 Модуль GRAPH.HS

3.1.1 Определение типов данных

Определены базовые типы и операции для работы с ориентированным графом.

Листинг 1. Определение типов данных

```
type DiGraph a = M.HashMap a [a]
empty :: DiGraph a
empty = M.empty
addNode :: (Hashable a, Eq a) => a -> DiGraph a -> DiGraph a
addNode =
  M.alter
    ( \mNodes ->
        case mNodes of
          Nothing -> Just []
          value -> value
    )
addEdge :: (Hashable a, Eq a) => (a, a) -> DiGraph a -> DiGraph
   a
addEdge (node, child) = M.alter insertEdge node
  where
    insertEdge Nothing = Just [child]
    insertEdge (Just nodes) = Just (L.nub (child : nodes))
```

type DiGraph - тип ориентированного графа. HashMap, где ключ - вершина, значение - список смежных вершин.

empty - создание пустого графа.

addNode - добавление вершины в граф. Если вершина уже существует, оставляет без изменений, если вершины нет, создаем с пустым списком смежности.

addEdge - добавление ориентированного ребра (from, to). Убирает дубликаты в списке смежности. Если узла нет, создаем с одним ребенком, иначе добавляем и убираем дубликаты.

3.1.2 Построение графа и операции удаления

Ниже приведены функции для построения графа и модификации его структуры.

Листинг 2. Построение графа

```
addEdges :: (Hashable a, Eq a) => [(a, a)] -> DiGraph a ->
    DiGraph a
addEdges [] graph = graph
addEdges (edge : edges) graph = addEdge edge (addEdges edges
    graph)
buildDiGraph :: (Hashable a, Eq a) => [(a, [a])] -> DiGraph a
buildDiGraph nodes = go nodes M.empty
  where
    go [] graph = graph
    go ((key, value) : xs) graph = M.insert key value (go xs
        graph)
children :: (Hashable a, Eq a) => a -> DiGraph a -> [a]
children = M.findWithDefault []
deleteNode :: (Hashable a, Eq a) => a -> DiGraph a -> DiGraph a
deleteNode = M.delete
```

addEdges - добавление нескольких ребер в граф. Базовый случай - пустой список ребер. Используется рекурсивное добавление.

buildDiGraph - построение графа из списка (вершина, список детей). Базовый случай - пустой список. Используется рекурсивное построение.

children - получение списка дочерних вершин для заданной вершины. Возвращает пустой список, если вершины нет.

Листинг 3. Операции удаления

```
deleteNode :: (Hashable a, Eq a) => a -> DiGraph a -> DiGraph a
deleteNode = M.delete
deleteNodes :: (Hashable a, Eq a) => [a] -> DiGraph a -> DiGraph
    a
deleteNodes [] graph = graph
deleteNodes (x : xs) graph = M.delete x (deleteNodes xs graph)
deleteEdge :: (Hashable a, Eq a) => (a, a) -> DiGraph a ->
    DiGraph a
deleteEdge (node, child) =
  M.alter
    ( \mNodes ->
        case mNodes of
          Nothing -> Just []
          Just nodes -> Just (L.delete child nodes)
    )
    node
```

deleteNode - удаление вершины из графа.

deleteEdge - удаление ориентированного ребра (from, to). Если узла нет, возвращаем пустой список, иначе даляем ребенка из списка смеж-

ности.

3.1.3 Поиск в ширину

Реализован алгоритм поиска в ширину с восстановлением пути.

Листинг 4. Поиск в ширину 1

```
type SearchState a = ([a], DiGraph a, DiGraph a)
data SearchResult a = Unsuccessful | Successful (DiGraph a)
bfsSearch :: forall a. (Hashable a, Eq a) => DiGraph a -> a -> a
    -> Maybe [a]
bfsSearch graph start end
  | start == end = Just [start]
  | otherwise =
      case bfsSearch' ([start], graph, empty) of
        Successful preds -> Just (findSolution preds) -
  where
    findSolution :: DiGraph a -> [a]
    findSolution g = L.reverse (go end)
      where
        go x =
          case children x g of
            [] -> [x]
            (v : _) -> x : go v
```

SearchState - состояние поиска - текущий фронт, граф оставшихся вершин, граф предшествования.

SearchResult - результат поиска: неудача или успех с графом предшествования.

bfsSearch - поиск в ширину от start до end. Возращаем начальную вершину, если начальная и конечная вершины совпадают. Если успех, то восстанавливаем путь, иначе Nothing.

findSolution - восстановление пути по графу предшествования. Если дошли до начала пути, то [x], иначе рекурсивно идем к началу.

Листинг 5. Поиск в ширину 2

```
    addMultiplePredecessors :: [(a, [a])] -> DiGraph a ->
       DiGraph a
    addMultiplePredecessors [] g = g
    addMultiplePredecessors ((n, ch) : xs) g =
      addMultiplePredecessors xs (go n ch g)
      where
        go n [] g = g
        go n (x : xs) g = go n xs (addEdge (x, n) g)
```

```
    bfsSearch' :: SearchState a -> SearchResult a
    bfsSearch' ([], _, preds) = Unsuccessful
    bfsSearch' (frontier, g, preds) =
      let g' = deleteNodes frontier g
          ch = L.map (\n -> (n, L.filter (`M.member` g') (
             children n g))) frontier
          frontier' = L.concatMap snd ch
          preds' = addMultiplePredecessors ch preds
       in if end `L.elem` frontier'
            then Successful preds'
            else bfsSearch' (frontier', g', preds')
```

addMultiplePredecessors - добавление нескольких предшественников в граф предшествования.

bfsSearch' - один шаг BFS. Если фронт пуст - неудача. Иначе удаляем обработанные вершины и для каждой вершины во фронте находим детей, которые еще в графе. Новый фронт - все дети, обновляем граф предшествования. Успех, если нашли конечную вершину, иначе продолжаем поиск.

3.1.4 Двунаправленный поиск в ширину

Двунаправленный поиск для повышения эффективности на больших графах.

Листинг 6. Двунаправленный поиск в ширину 1

```
type BiSearchState a = (SearchState a, SearchState a)
biBfsSearch :: forall a. (Hashable a, Eq a) => DiGraph a -> a ->
    a -> Maybe [a]
biBfsSearch graph start end
  | start == end = Just [start]
  | otherwise =
      let fState = ([start], graph, empty)
          bState = ([end], graph, empty)
       in biBfsSearch' (fState, bState)
  where
    findSolution :: DiGraph a -> a -> [a]
    findSolution g = go
      where
        go x =
          case children x g of
            [] -> [x]
            (v : _) -> x : go v
```

BiSearchState - состояние для двунаправленного поиска (прямой и обратный поиск).

biBfsSearch - двунаправленный поиск в ширину. Just [start], если начальная и конечная вершины совпадают. fState - прямой поиск от start, bState - обратный.

findSolution - восстановление пути из графа предшествования.

Листинг 7. Двунаправленный поиск в ширину 2

```
checkOverlap :: BiSearchState a -> Maybe [a]
checkOverlap ((fFrontier', _, fPreds'), (bFrontier', _,
   bPreds')) =
  let getSolution x =
        let fPath = findSolution fPreds' x
            bPath = findSolution bPreds' x
         in L.reverse fPath ++ L.tail bPath
      overlaps = L.filter (`M.member` bPreds') fFrontier'
      solutions = L.sortOn L.length (L.map getSolution
         overlaps)
        then Nothing
        else Just $ L.head solutions

checkSolution :: SearchState a -> a -> Maybe [a]
checkSolution (frontier, _, preds) node
  | node `L.elem` frontier = Just (findSolution preds node)
  | otherwise = Nothing
```

checkOverlap - проверка пересечения фронтов прямого и обратного поиска. Объединяем пути и смотрим пересечения фронтов. Сортируем по длине. Если нет решений, то Nothing, иначе берем кратчайший путь.

checkSolution - проверка достижения целевой вершины в одном направлении.

Листинг 8. Двунаправленный поиск в ширину 3

```
biBfsSearch' :: BiSearchState a -> Maybe [a]
biBfsSearch' state@(fState@(fFrontier, _, _), bState@(
   bFrontier, _, _))
  | L.null fFrontier && L.null bFrontier = Nothing
  | isJust fSol = fmap L.reverse fSol
  | isJust bSol = bSol
  | isJust overlapSol = overlapSol
  | otherwise = biBfsSearch' biState'
  where
    fSol = checkSolution fState end
    bSol = checkSolution bState start
    overlapSol = checkOverlap state
```

```
12          fState' = bfsSearchStep fState
13          bState' = bfsSearchStep bState
14          biState' = (fState', bState')
```

biBfsSearch' - основной цикл двунаправленного поиска. Nothing, если оба фронта пусты. isJust fSol = fmap L.reverse fSol - найден путь в прямом направлении, isJust bSol = bSol - найден путь в обратном направлении. isJust overlapSol = overlapSol - найдено пересечение. Иначе продолжаем поиск.

Листинг 9. Двунаправленный поиск в ширину 4

```
1     bfsSearchStep :: SearchState a -> SearchState a
2     bfsSearchStep (frontier, g, preds) =
3       let g' = deleteNodes frontier g
4           ch = L.map (\n -> (n, L.filter (`M.member` g') (
                children n g))) frontier
5           frontier' = L.concatMap snd ch
6           preds' = addMultiplePredecessors ch preds
7        in (frontier', g', preds')
8     addMultiplePredecessors :: [(a, [a])] -> DiGraph a ->
        DiGraph a
9     addMultiplePredecessors [] g = g
10    addMultiplePredecessors ((n, ch) : xs) g =
11      addMultiplePredecessors xs (go n ch g)
12      where
13        go n [] g = g
14        go n (x : xs) g = go n xs (addEdge (x, n) g)
```

bfsSearchStep - один шаг BFS (аналогично bfsSearch').

addMultiplePredecessors - дбавление нескольких предшественников.

3.2 Модуль PERMUTATIONMAP.HS

Карта перестановок для эффективного поиска слов с одинаковым набором букв.

Листинг 10. Модуль 1

```
1 type PermutationMap = M.HashMap BS.ByteString [BS.ByteString]
2 empty :: PermutationMap
3 empty = M.empty
4 member :: BS.ByteString -> PermutationMap -> Bool
5 member key = M.member (BS.sort key)
6 alter ::
7   ( Maybe [BS.ByteString] ->
8     Maybe [BS.ByteString]
```

```
  ) ->
  BS.ByteString ->
  PermutationMap ->
  PermutationMap
alter f key = M.alter f (BS.sort key)
delete :: BS.ByteString -> PermutationMap -> PermutationMap
delete key = M.delete (BS.sort key)
insert :: BS.ByteString -> [BS.ByteString] -> PermutationMap ->
  PermutationMap
insert key = M.insert (BS.sort key)
lookup :: BS.ByteString -> PermutationMap -> Maybe [BS.
  ByteString]
lookup key = M.lookup (BS.sort key)
```

PermutationMap - карта перестановок - ключ - отсортированная версия слова. Значение - список исходных слов с такими же буквами.

empty - пустая карта перестановок.

member - проверка наличия слова в карте.

alter - модификация значения в карте.

delete - удаление слова из карты.

insert - вставка значения в карту.

lookup - поиск значения в карте.

Листинг 11. Модуль 2

```
findWithDefault :: [BS.ByteString] -> BS.ByteString ->
  PermutationMap ->[BS.ByteString]
findWithDefault defaultValue key map =
  fromMaybe [] (PermutationMap.lookup key map)
createPermutationMap :: [BS.ByteString] -> PermutationMap
createPermutationMap = go empty
  where
    go permMap [] = permMap
    go permMap (x : xs) = go (insertPermutation x permMap) xs
    insertPermutation word = alter (insertList word) word
    insertList word Nothing = Just [word]
    insertList word (Just words) = Just $ word : words
```

findWithDefault - поиск со значением по умолчанию. Если не найдено, возвращаем пустой список.

createPermutationMap - создание карты перестановок из словаря. Группирует слова по их отсортированным версиям. Базовый случай - пустой список, иначе рекурсивная обработка.

insertPermutation - вставка одного слова в карту перестановок.

insertList - функция для вставки в список значений. Если ключа нет, создаем новый список, иначе добавляем к существующему.

3.3 Модуль LADDER.HS

Модуль для работы со словными цепочками, использует графовые алгоритмы для поиска путей.

Листинг 12. Модуль 1

```
module Ladder
  ( Dictionary,
    readDictionary,
    ladderSolve,
  )
where
type Dictionary = [BS.ByteString]
readDictionary :: FilePath -> IO Dictionary
readDictionary filepath = do
  dictionaryContent <- C.readFile filepath
  let lines = C.lines dictionaryContent
      words = L.map (C.filter (`L.elem` ['a' .. 'z'])) lines
  return words
delete :: Char -> BS.ByteString -> BS.ByteString
delete ch string = case C.uncons string of
  Just (x, xs) -> if ch == x then xs else C.cons x (delete ch xs
    )
  Nothing -> C.empty
computeCandidates :: PM.PermutationMap -> BS.ByteString -> [BS.
   ByteString]
computeCandidates map word =
  let candidates = modified ++ removed ++ added ++ [word]
      perms = L.concatMap (\x -> PM.findWithDefault [] x map)
         candidates
   in L.filter (/= word) (L.nub perms)
  where
    added = [C.cons x word | x <- ['a' .. 'z']]
    removed = [delete x word | x <- C.unpack word]
    modified = [C.cons x (delete y word) | x <- ['a' .. 'z'], y
       <- C.unpack word, x /= y]
```

Dictionary - список слов в виде ByteString.

readDictionary - чтение словаря из файла. Фильтрует только буквы a-z, игнорируя другие символы. Сначала читаем весь файл, разбиваем на строки и оставляем только буквы.

delete - удаление первого вхождения символа из строки.

computeCandidates - вычисление соседних слов в словной цепочке. Соседние слова отличаются одной операцией - добавление, удаление или замена буквы.

Листинг 13. Модуль 2

```
mkLadderGraph :: Dictionary -> G.DiGraph BS.ByteString
mkLadderGraph dict = G.buildDiGraph nodes
  where
    map = PM.createPermutationMap dict
    nodes = L.map (\w -> (w, computeCandidates map w)) dict
ladderSolve :: Dictionary -> String -> String -> Maybe [BS.
   ByteString]
ladderSolve dict start end =
  let g = mkLadderGraph dict
   in G.biBfsSearch g (C.pack start) (C.pack end)
```

mkLadderGraph - построение графа словных цепочек. Каждое слово связано с соседями, отличающимися на одну операцию.

ladderSolve - решение задачи поиска словной цепочки. Строит граф и ищет путь между start и end.

4 Часть 2

4.1 Главный модуль Main

Основной функционал это управление интерфейсом командной строки, обработка пользовательского ввода, загрузка моделей из файлов, запуск диалога между моделями, генерация ответа модели и координация работы всех компонентов.

4.1.1 Модель данных

Позволяет хранить N-граммную модель и ее данные.

Листинг 14. ModelWithInfo

```
data ModelWithInfo = ModelWithInfo {
    model :: NGramModel,
    maxSentenceLength :: Int
} deriving (Show)
```

maxSentenceLength - максимальная длина предложения в тексте. deriving (Show) для отображения.

4.1.2 Главная функция и основной цикл

Реализация интерактивного интерфейса с меню.

Листинг 15. main

```
main :: IO ()
main = do
    TIO.putStrLn "\n 4-GRAM TEXT GENERATOR"
    mainLoop Nothing Nothing
mainLoop :: Maybe ModelWithInfo -> Maybe ModelWithInfo -> IO ()
mainLoop currentModel model2 = do
    TIO.putStrLn "\n MENU"
    TIO.putStrLn "0. Exit"
    TIO.putStrLn "1. Load or build main model"
    TIO.putStrLn "2. Save the current model to file"
    TIO.putStrLn "3. Generate the continuation of the sentence"
    TIO.putStrLn "4. Load or build second model for the dialogue
       "
    TIO.putStrLn "5. Dialogue between two models"
    TIO.putStrLn $ T.pack $ "\n Current first model: " ++
       modelStatus currentModel
```

```
15      TIO.putStrLn $ T.pack $ "Current second model: " ++
            modelStatus model2
16      choice <- TIO.getLine
17      case T.unpack choice of
```

Выводится меню программы для пользователя.

currentModel - текущая основная модель.

model2 - вторая модель для диалога.

Отображается статус моделей (загружена или нет). choice принимает пользовательский ввод пункта меню, а затем идет обработка всех возможных вариантов ввода.

4.1.3 Загрузка и построение моделей

Функция загружает текстовый файл, считает количество слов, определяет максимальную длину предложений и строит N-граммную модель.

Листинг 16. loadTextModelLoop

```
1  loadTextModelLoop :: Maybe ModelWithInfo -> Maybe ModelWithInfo
       -> IO ()
2  loadTextModelLoop currentModel model2 = do
3      TIO.putStrLn "Enter a file name with text (0 to return):"
4      fileName <- TIO.getLine
5      case fileName of
6          "0" -> mainLoop currentModel model2
7          _ -> do
8              result <- readFileSafe (T.unpack fileName)
9              case result of
10                 Left errorMsg -> do
11                     TIO.putStrLn $ T.pack errorMsg
12                     loadTextModelLoop currentModel model2
13                 Right content -> do
14                     let sentences = splitText (T.pack content)
15                     let wordCount = sum (map length sentences)
16                     let maxSentenceLen = if null sentences then
                           0 else maximum (map length sentences)
17                     TIO.putStrLn $ "Text loaded. Total words: "
                           `T.append` T.pack (show wordCount)
18                     TIO.putStrLn $ "Max sentence length: " `T.
                           append` T.pack (show maxSentenceLen)
19                     let newModel = buildNGramModel sentences 4
20                     TIO.putStrLn "Model successfully built!"
21                     mainLoop (Just $ ModelWithInfo newModel
                           maxSentenceLen) model2
```

Функция принимает ввод имени файла с текстом. Если ввод это «0», то возращаемся в меню. Иначе читаем файл. Если при чтении происходит ошибка - отображаем ее, иначе разбиваем текст на предложения, считаем число слов, максимальную длину предложения. Выводим информационные сообщения об успешной загрузке файла, общем количестве слов, максимальной длине предложения. Строим N-граммную модель. Выводим сообщение об успешном построении модели, записываем максимальную длину предложения.

4.1.4 Генерация продолжения предложения

В данной части происходит генерация продолжения с проверкой ограничений длины и обработкой пользовательского ввода.

Листинг 17. Генерация продолжения предложения 1

```
"3" -> do
    case currentModel of
        Nothing -> do
            TIO.putStrLn "Build or download the model first!"
            mainLoop currentModel model2
        Just modelInfo -> do
            TIO.putStrLn "Enter starting words (1-3 words, 0 to
                return):"
            input <- TIO.getLine
            when (input /= "0") $ do
                let wordsList = map T.toLower (T.words input)
                if length wordsList >= 1 && length wordsList <=
                    3
                 then do
                     let initialLen = length wordsList
                     let maxTotalLen = maxSentenceLength
                         modelInfo
                     if maxTotalLen <= initialLen
                         then do
                             TIO.putStrLn $ T.pack $ "Initial
                                 words already have " ++ show
                                 initialLen ++
                                 " words, but maximum
                                    sentence length is " ++
                                    show maxTotalLen ++
                                 ". Cannot generate
                                    continuation."
                             mainLoop currentModel model2
```

Если пользователь выбрал пункт 3 меню, то начинается работа с ге-

нерацией продолжения. Если не выбрана текущая модель, то выводит сообщение о необходимости загрузить модель (пункт 1 меню) и возращаемся в меню. Иначе просим пользователя ввести 1-3 слова, либо 0 для возращения в меню. Получаем ввод. Если ввод «0», то возвращаемся в меню. Если ввод не «0», то приводим ввод к нижнему регистру. Если слов введено менее 1 или более 3х, то выводит сообщение об ошибке ввода и возращаемся в меню. Иначе считаем количество введенных пользователем слов и инициализируем максимальную длину предложения.

Листинг 18. Генерация продолжения предложения 2

```
else do
    let maxAdditional = maxTotalLen - initialLen
        randomAdditionalLen <- if maxAdditional > 0
            then randomRIO (1, maxAdditional)
                else return 0
        if randomAdditionalLen <= 0
            then do
                TIO.putStrLn "Cannot generate continuation - no
                   space for additional words."
                mainLoop currentModel model2
                else do
                    result <- generateContinuation (model
                       modelInfo) wordsList randomAdditionalLen
                    TIO.putStrLn $ "Generated continuation: " `T
                       .append` T.unwords result
                    mainLoop currentModel model2
                    else do
                        TIO.putStrLn "Error: Please enter
                           between 1 and 3 words."
                        mainLoop currentModel model2
```

Если введено слов больше, чем длина самого длинного предложения в тексте, то выводим ошибку и возращаемся в меню. Иначе генерируем случайную длину, проверяя ее на корректность. Затем генерируем продолжение с использованием модели и выводим на экран. Выходим в меню.

4.1.5 Диалог между моделями

Реализован механизм диалога, где модели поочередно генерируют реплики на основе предыдущих высказываний.

Листинг 19. Диалог между моделями

```
startDialog :: NGramModel -> NGramModel -> [Text] -> Int -> IO
    ()
startDialog model1 model2 startWords reps = do
    TIO.putStrLn "\n START OF DIALOGUE"
    TIO.putStrLn $ "Starting word: " `T.append` T.unwords
        startWords
    dialogLoop model1 model2 startWords reps 0


dialogLoop :: NGramModel -> NGramModel -> [Text] -> Int -> Int
    -> IO ()
dialogLoop _ _ _ totalReps currentRep
    | currentRep >= totalReps = TIO.putStrLn "DIALOGUE ENDED"
dialogLoop model1 model2 lastPhrase totalReps currentRep = do
    let currentModel = if even currentRep then model1 else
        model2
    let opponentModel = if even currentRep then model2 else
        model1
    let speaker = if even currentRep then "Model 1" else "Model
        2"
    response <- generateResponse currentModel lastPhrase
    TIO.putStrLn $ T.pack speaker `T.append` ": " `T.append` T.
        unwords response
    dialogLoop model1 model2 response totalReps (currentRep + 1)
```

startDialog - запуск диалога между двумя N-граммными моделями.

model1, model2 - модели для диалога.

startWords - начальные слова для диалога.

reps - количество реплик в диалоге.

Выводим сообщение о начале диалога и начальное слово. Вызываем dialogLoop.

dialogLoop - цикл диалога между моделями. Поочередно генерирует реплики для каждой модели.

Если currentRep >= totalReps, то заканчиваем диалог моделей. Далее в коде идет определение текущей модели и ее «собеседника». Если currentRep четное, то текущая модель - model1, иначе - model2. Затем идет генерация ответа на основе последней фразы. Рекурсивно вызыва-

ем для следующей реплики.

4.1.6 Вспомогательные функции

Ниже приведены функции для безопасной работы с файлами и пользовательским вводом.

Листинг 20. Вспомогательные функции

```
readFileSafe :: FilePath -> IO (Either String String)
readFileSafe fileName = catch (readFile fileName >>= return .
   Right) handleError
  where
    handleError :: IOException -> IO (Either String String)
    handleError e = do
        let errorMsg = "File reading error: " ++ show e
        return $ Left errorMsg

safeRead :: String -> Int -> Int
safeRead str defaultValue = case readMaybe str of
    Just n -> n
    Nothing -> defaultValue
```

readFileSafe - безопасное чтение файла с обработкой ошибок ввода-вывода.

safeRead - безопасное преобразование строки в число с значением по умолчанию.

4.2 Модуль LIB.HS

4.2.1 Определение типов данных

Определены типы данных для работы с N-граммами, используется структура HashMap для хранения модели.

Листинг 21. Определение типов данных

```

module Lib (
    NGramModel,
    splitText,
    buildNGramModel,
    saveModel,
    loadModel,
    generateContinuation,
    getPossibleKeys,
```

```
10      findValidKey,
11      getRandomKey,
12      preprocessText
13  ) where
14
15  type NGram = [Text]
16  type NGramModel = HashMap NGram [(NGram, Double)]
```

type NGram - тип для представления N-граммы.

type NGramModel - тип для N-граммной модели.

Ключ - это N-грамма (1-3 слова).

Значение - список возможных продолжений с вероятностями.

4.2.2 Предобработка текста

Реализована предобработка текста, включающая очистку, нормализацию и разбиение на предложения.

Листинг 22. Предобработка текста 1

```
1  cleanWord :: Text -> Text
2  cleanWord word =
3      let cleaned = T.toLower $ T.filter validChar word
4      in if isValidCleanedWord cleaned then cleaned else T.empty
5    where
6      validChar c = isAscii c && (isLetter c || c == '\'' || c ==
           '-')
7      isValidCleanedWord w = not (T.null w) && T.any isLetter w
8
9  splitText :: Text -> [[Text]]
10 splitText text =
11     let preprocessed = preprocessText text
12         sentences = splitSentences preprocessed
13     in filter (not . null) $ map processSentence sentences
```

cleanWord - служит для очистки слова от некорректных символов и приведение к нижнему регистру. Сохраняет апострофы и дефисы внутри слов.

Сначала приводит все к нижнему регистру, заатем фильтрует. Проверяем, что слово не пустое.

splitText - разделение текста на предложения и слова, выполняет нормализацию символов и фильтрацию некорректных слов.

Сначала происходит нормализация символов, затем идет разбитие на

предложения (по «.», «!», «?»). Обрабатываем каждое и убираем пустые. Запускаем рекурсию по тексту. Базовый случай - пустой текст. Иначе разбиваем на первое предложение и остаток, отделяем знаки препинания. Если предложение пустое - обрабатываем остаток. Очищаем предложение и добавляем знак препинания, затем обрабатываем остаток.

Листинг 23. Предобработка текста 2

```
  where
    splitSentences :: Text -> [Text]
    splitSentences t
        | T.null t = []
        | otherwise =
            let (sentence, rest) = T.break isSentenceTerminator
                t
                (punctuation, rest') = T.span
                    isSentenceTerminator rest
            in if T.null sentence
               then splitSentences rest'
               else (cleanSentence sentence `T.append` T.take 1
                  punctuation) : splitSentences rest'

    processSentence :: Text -> [Text]
    processSentence = filter isValidWord . map T.toLower . T.
       words . T.map normalizeChar
```

processSentence - обработка одного предложения: разбиение на слова, нормализация символов, фильтрация.

4.2.3 Построение N-граммной модели

Модель строится путем извлечения всех возможных N-грамм из текста и расчета вероятностей их продолжений.

Листинг 24. Построение N-граммной модели 1

```
buildNGramModel :: [[Text]] -> Int -> NGramModel
buildNGramModel sentences n =
    let validSentences = filter (all isValidWord) sentences
        allNGrams = concatMap (getNGrams n) validSentences
        grouped = HashMap.fromListWith (++) [(key, [(value, 1)])
            | (key, value) <- allNGrams]
    in HashMap.map calculateProbabilities grouped
  where
```

buildNGramModel - построение N-граммной модели из предложений.

sentences - список предложений.

n - размер N-граммы (4).

Фильтруем предложения, оставляя только те, где все слова валидны. Для каждого валидного предложения извлекаем все возможные N-граммы, группируем N-граммы по ключам и подсчитываем вероятности. HashMap.fromListWith (++) создает HashMap, объединяя значения для одинаковых ключей (key, [(value, 1)]). При совпадении ключей списки объединяются.

Листинг 25. Построение N-граммной модели 2

```
getNGrams :: Int -> [Text] -> [(NGram, NGram)]
getNGrams n words =
    [ (take k context, take (n - k) continuation)
    | k <- [1..min 3 (length words)]  -- Ключи длиной 1, 2, 3 слова
    , (context, continuation) <- zip (tails words) (drop k (tails words))
    , length context >= k
    , length continuation >= n - k
    , length context + length continuation >= n
    , filterValidNGram (take k context, take (n - k) continuation)
    ]

calculateProbabilities :: [(NGram, Int)] -> [(NGram, Double)]
calculateProbabilities pairs =
    let total = fromIntegral $ sum (map snd pairs)
    in [(ngram, fromIntegral count / total) | (ngram, count) <- pairs]
```

getNGrams - извлечение всех N-грамм из предложения.

tails words генерирует все суффиксы предложения, drop k (tails words) - суффиксы, начинающиеся с k-го слова, zip создает пары (context, continuation), где continuation начинается через k слов от context. В context достаточно слов для ключа длины k, в continuation достаточно слов для продолжения длины n-k , length context + length continuation >= n - общее количество слов достаточно для N-граммы. Затем проверяем, что и ключ и продолжение состоят из валидных слов.

calculateProbabilities - расчет вероятностей продолжений для каждого ключа. Вероятность = частота продолжения / общая частота всех продолжений для данного ключа. fromIntegral преобразует Int в Double для деления.

4.2.4 Генерация текста

Генерация реализована как рекурсивный процесс выбора следующих слов на основе вероятностей из модели.

Листинг 26. Генерация текста 1

```
generateContinuation :: NGramModel -> [Text] -> Int -> IO [Text]
generateContinuation model startWords maxLength = do
    gen <- newStdGen
    return $ generateEfficient model startWords gen maxLength
  where
    generateEfficient :: NGramModel -> [Text] -> StdGen -> Int
      -> [Text]
    generateEfficient model currentWords gen maxLength =
        generateLoop model currentWords gen maxLength
           currentWords
    generateLoop :: NGramModel -> [Text] -> StdGen -> Int -> [
       Text] -> [Text]
    generateLoop _ _ _ 0 acc = acc
    generateLoop model currentWords gen remaining acc =
        let possibleKeys = getPossibleKeys currentWords
            key = findValidKey model possibleKeys
        in case key of
            Nothing -> acc  -- Не найдено подходящего ключа
            Just k ->
                case HashMap.lookup k model of
                    Nothing -> acc
                    Just continuations ->
                        let (choice, newGen) =
                           efficientWeightedRandom continuations
                            gen
                            newWords = acc ++ choice
                            newRemaining = remaining - length
                               choice
                        in if newRemaining <= 0
                           then newWords
                           else generateLoop model (currentWords
                              ++ choice) newGen newRemaining
                             newWords
```

Используется вероятностная модель для выбора следующих слов, каждое следующее слово зависит от предыдущих.

Создаем новый генератор случайных чисел, generateEfficient инициализирует рекурсивную генерацию. Начинаем с текущих слов и используем их как начальный аккумулятор.

На каждом шаге пытаемся найти продолжение для текущего кон-

текста и добавляем его к результату, пока не достигнем максимальной длины или не сможем найти подходящее продолжение. Если достигнута максимальная длина - возвращаем накопленный результат, получаем возможные ключи из текущего контекста, ищем первый ключ, который есть в модели. Иначе возвращаем что есть (acc). Если найден валидный ключ в модели, предусматриваем защиту от Nothing. Если Just, то используется взвешенный случайный выбор продолжения.

Листинг 27. Генерация текста 2

```
1     efficientWeightedRandom :: [(NGram, Double)] -> StdGen -> (
        NGram, StdGen)
2     efficientWeightedRandom choices gen =
3         let total = sum (map snd choices)
4             (r, newGen) = randomR (0, total) gen
5             sortedChoices = sortOn (Down . snd) choices
6         in pick sortedChoices r newGen
7       where
8         pick [] _ g = ([], g)
9         pick ((ngram, prob):rest) rVal g
10            | rVal <= prob = (ngram, g)
11            | otherwise = pick rest (rVal - prob) g
```

efficientWeightedRandom - взвешенный случайный выбор на основе вероятностей. Случайное число в диапазоне [0, total].

sortedChoices - более вероятные варианты проверяются первыми. Далее вызывается рекурсивная функция для выбора варианта. Если случайное число попало в интервал этого варианта - выбираем его. Иначе переход к следующему варианту, вычитая пройденную вероятность.

4.2.5 Работа с ключами модели

Функции для работы с ключами модели реализуют стратегию «от самых длинных к самым коротким» ключам.

Листинг 28. Работа с ключами модели

```
1 getPossibleKeys :: [Text] -> [[Text]]
2 getPossibleKeys words =
3     let n = length words
4         key1 = if n >= 1 then [last words] else []
5         key2 = if n >= 2 then drop (n-2) words else []
6         key3 = if n >= 3 then drop (n-3) words else []
7     in filter (not . null) [key3, key2, key1]
8
9 findValidKey :: NGramModel -> [[Text]] -> Maybe [Text]
```

```
findValidKey model keys =
    case filter (`HashMap.member` model) keys of
        [] -> Nothing
        (k:_) -> Just k
```

key1 - ключ из последнего слова.

key2 - ключ из двух последних слов.

key3 - ключ из трех последних слов.

Фильтруем пустые ключи и возвращаем в порядке от самых длинных к коротким.

4.2.6 Сохранение и загрузка модели

Функции для работы с ключами модели реализуют стратегию «от самых длинных к самым коротким» ключам.

Листинг 29. Сохранение и загрузка модели

```
saveModel :: FilePath -> NGramModel -> IO ()
saveModel filePath model = withFile filePath WriteMode $ \h ->
    mapM_ (\(k, vs) -> TIO.hPutStrLn h $ T.pack (show k) `T.
        append` " => " `T.append` T.pack (show vs)) (HashMap.
        toList model)

loadModel :: FilePath -> IO (Maybe NGramModel)
loadModel filePath = do
    result <- try (TIO.readFile filePath) :: IO (Either
        SomeException Text)
    case result of
        Left _ -> return Nothing
        Right content ->
            let lines' = T.lines content
                parsed = catMaybes $ map parseLine lines'
            in return $ if null parsed then Nothing else Just (
                HashMap.fromList parsed)
```

saveModel - сохранение N-граммной модели в файл. Каждая строка представляет одну запись модели ключ => список (продолжение, вероятность).

withFile - безопасно открывает и закрывает файл.

loadModel - загрузка N-граммной модели из файла с обработкой исключений.

5 Результаты работы

При запуске программы отображается меню [Рис. 2]:

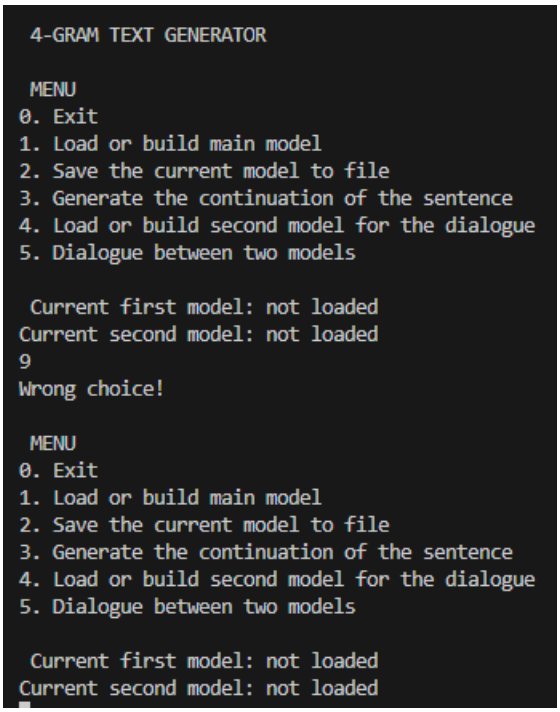


Рис. 2. Меню

Предусмотрена защита от некорректного пользовательского ввода в каждом пункте и подпункте меню.

Пункт 2 - загрузка модели. Пользователю предлагается выбрать построить модель из файла или загрузить файл с моделью [Рис. 3]:

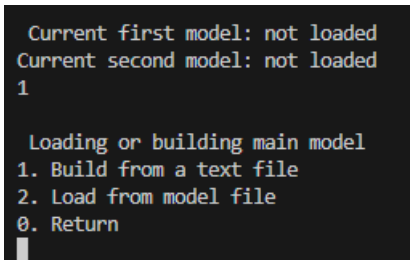


Рис. 3. Загрузка модели

При выборе первого пункта пользователю предлагается ввести название файла [Рис. 4]:

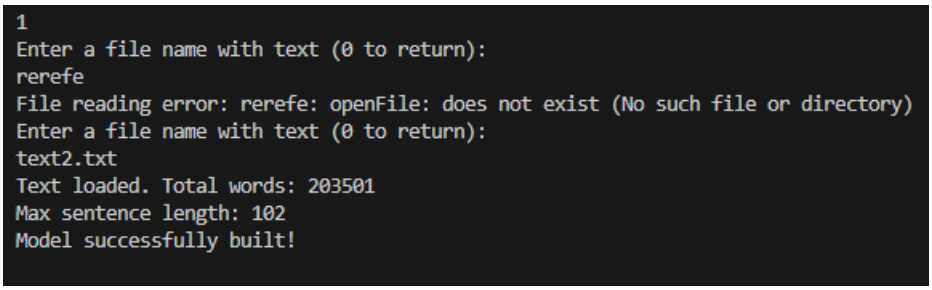


Рис. 4. Построение модели из файла

Так же предусмотрена обработка отсутствующих файлов.
Загрузка модели из файла выглядит следующим образом [Рис. 5]:

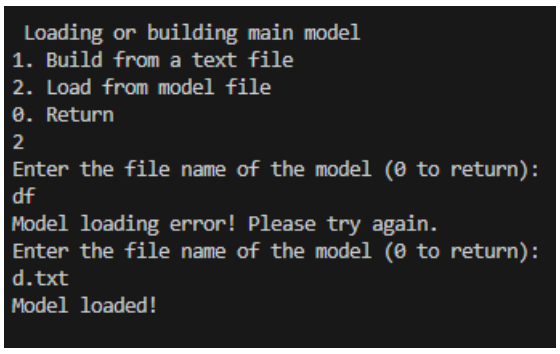


Рис. 5. Загрузка модели из файла

Пункт 2 меню - сохранение модели в файл [Рис. 6]:

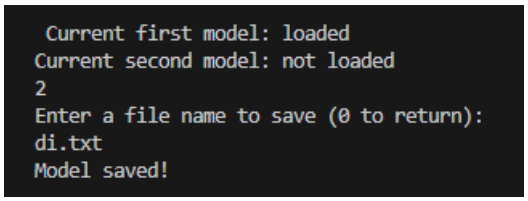


Рис. 6. Сохранение модели в файл

Модель выглядит следующим образом [Рис. 7]:

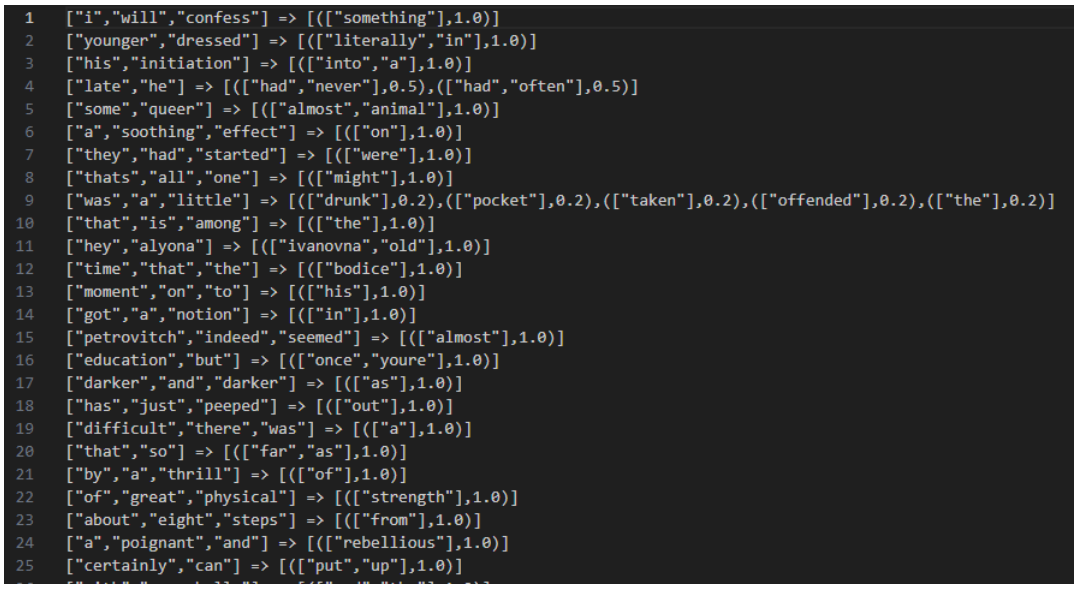


Рис. 7. Модель

Пункт 3 - генерация продолжения. Если введено более 3х слов предусмотрен вывод ошибки [Рис. 8]:

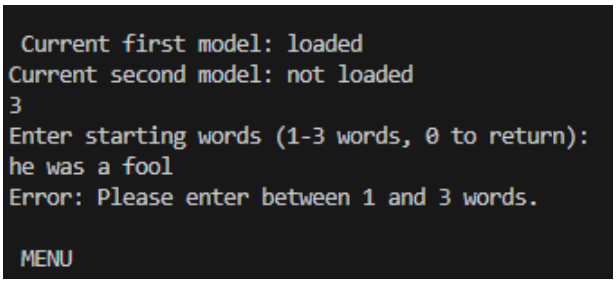


Рис. 8. Генерация продолжения с некорректным вводом

Ниже приведена успешная генерация продолжения [Рис. 9]:

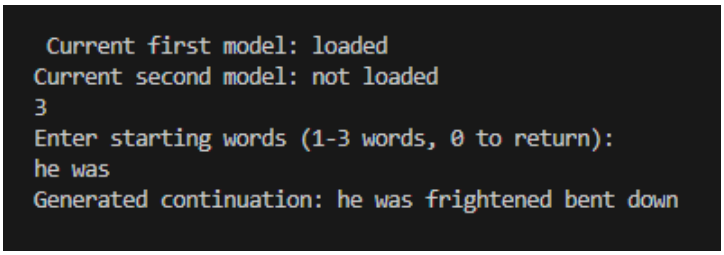


Рис. 9. Генерация продолжения

При попытке генерации диалога без второй модели - ошибка [Рис. 10]:

```
 Current first model: loaded
Current second model: not loaded
5
First, download both models! (the main and the second)
```

Рис. 10. Попытка генерации диалога без второй модели

Аналогично с пунктом 3 без загрузки основной модели.

Загрузка второй модели аналогична загрузке первой [Рис. 11]:

```
 Current first model: loaded
Current second model: not loaded
4

 Loading the second model
1. Build from a text file
2. Load from model file
0. Return
1
Enter a file name with text (0 to return):
text4.txt
Text loaded. Total words: 141302
Max sentence length: 273
The second model has been successfully built!
```

Рис. 11. Загрузка второй модели

Пункт 5 меню - диалог моделей. Пользователь может выбрать порядок моделей, а так же ввести первое слово. Если введено более 1 слова - ошибка [Рис. 12]:

```
 Current first model: loaded
Current second model: loaded
5
Select model order:
1. The main model starts first
2. The second model starts first
0. Return
1
Enter the initial word (0 to return):
she is
Error: Please enter exactly one word. Try again or enter 0 to return.
Enter the initial word (0 to return):
she
Enter the number of replicas (0 to return):
3

 START OF DIALOGUE
Starting word: she
Model 1: she had declared at first that she could
Model 2: she could see that an encounter with the newcomer would do him
Model 1: him too much about his own work of propaganda for he
DIALOGUE ENDED
```

Рис. 12. Диалог

В любом подпункте меню можно выйти в меню вводом нуля [Рис. 13]:

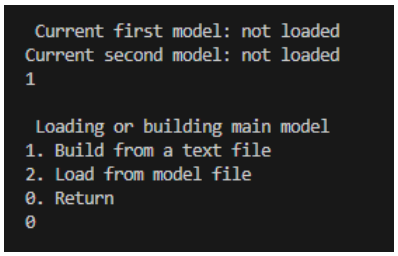


Рис. 13. Выход из подпунктов

Завершение программы происходит по вводу нуля в основном меню [Рис. 14]:

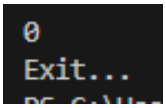


Рис. 14. Выход из приложения

Пользователь видит сообщение «Exit...».

6 Заключение

В ходе работы был успешно разработан программный модуль синтаксического анализатора и генератора текста на основе N-граммных моделей для автоматической обработки текстовых данных и генерации текстовых продолжений с учетом вероятностных характеристик исходного текста и интерактивную среду для генерации текста и диалога между двумя различными текстовыми моделями.

1. Разработан синтаксический анализатор текстовых файлов, осуществляющий разбиение текста на предложения по стандартным знакам препинания, выделение слов, состоящих только из буквенных символов, очистку текста от некорректных символов и цифр.

2. Реализовано построение N-граммной модели (N=4) на основе проанализированного текста.

3. Создан генератор текста, обеспечивающий обработку пользовательского запроса, генерацию продолжения с учетом вероятностной модели, случайное определение длины предложения в пределах максимальной длины исходных предложений.

4. Реализован механизм диалога между двумя различными N-граммными моделями, где каждая модель генерирует реплику на основе последних слов оппонента.

5. Разработано взаимодействие пользователя с программой.

6. Обеспечено корректную обработку ошибок и некорректного ввода.

7. Разработан тестовый набор данных для проверки и протестировать.

Всего на работу было потрачено около 25 часов времени.

Работа выполнена на языке Haskell в среде программирования Visual Studio Code.

Список использованной литературы

[1] Курт У. Программируй на Haskell. — М.: Издательство, 2019. — 320с

[2] Haskell

// URL: https://blog.skillfactory.ru/glossary/haskell/

(дата обращения: 09.10.2025)

[3] Изучай Haskell во имя добра!

// URL: https://learnhaskellforgood.narod.ru/learnyouahaskell.com/introduction.html

(дата обращения: 23.10.2025)

[4] Что такое N-граммы и с чем их едят?

// URL: https://sysblok.ru/knowhow/chto-takoe-n-grammy-i-s-chem-ih-edjat/

(дата обращения: 16.11.2025)

[5] Диалог

// URL: https://ru.wiktionary.org/wiki/%D0%B4%D0%B8%D0%B0%D0%BB%D0%

(дата обращения: 16.11.2025)

Приложение. Часть 1. Полный код программы

Листинг 30. Часть 1. Полный код программы

```
module Main (main) where

import Ladder -- логика поиска цепочек
import System.Environment -- для работы с аргументами командной строки

printHelpText :: String -> IO () -- вывод справки
printHelpText msg = do
  putStrLn (msg ++ "\n")
  progName <- getProgName --принимает сообщение об ошибке
  putStrLn ("Usage: " ++ progName ++ " <filename> <start> <end>") -- сообщение и имя программы с форматом использования

main :: IO () -- основная функция программы
main = do
  args <- getArgs --получаем аргументы командной строки
  case args of
    [dictFile, start, end] -> do -- три аргумента тогда
      dict <- readDictionary dictFile --читаем словарь
      case ladderSolve dict start end of --ищем решения
        Nothing -> putStrLn "No solution" --нет решения(
        Just sol -> do --тут выводим уже результат
          print sol
          putStrLn $ "Length: " ++ show (length sol)
    _ -> printHelpText "Wrong number of arguments!"


{-# LANGUAGE ScopedTypeVariables #-}

module Graph
  ( DiGraph,
    empty,
    addNode,
    addEdge,
    addEdges,
    buildDiGraph,
    children,
    deleteNode,
    deleteNodes,
    deleteEdge,
    SearchState,
    SearchResult (..),
    bfsSearch,
```

```
42      BiSearchState,
43      biBfsSearch,
44    )
45  where
46
47  import qualified Data.HashMap.Lazy as M
48  import Data.Hashable (Hashable)
49  import qualified Data.List as L
50  import Data.Maybe (isJust)
51
52  type DiGraph a = M.HashMap a [a] --тип - отображения из вершины
        с список смежных вершин
53
54  -- создание пустого графа
55  empty :: DiGraph a
56  empty = M.empty
57
58  --добавление вершины
59  addNode :: (Hashable a, Eq a) => a -> DiGraph a -> DiGraph a
60  addNode =
61    M.alter
62      ( \mNodes ->
63          case mNodes of
64            Nothing -> Just [] -- если нет вершины, то с пустым сп
                 иском смежности
65            value -> value
66      )
67
68  --добавление ориентированного ребра, убирает дубликаты в списке
        смежности
69  addEdge :: (Hashable a, Eq a) => (a, a) -> DiGraph a -> DiGraph
        a
70  addEdge (node, child) = M.alter insertEdge node --M.alter для из
       менения значения, связанного с ключом node в HashMap
71    where
72      insertEdge Nothing = Just [child] --узел node не существует
           в графе - создаем новый список смежности, содержащий толь
           ко child
73      insertEdge (Just nodes) = --узел существует - добавляем
           child в начало списка nodes и убираем дубликаты с помощью
            L.nub
74        Just (L.nub (child : nodes))
75
76   --принимает список ребер и граф, возвращает граф с добавленными
         ребрами
77  addEdges :: (Hashable a, Eq a) => [(a, a)] -> DiGraph a ->
        DiGraph a
```

```
78 addEdges [] graph = graph --список ребер пуст, возвращаем исходн
   ый граф
79 addEdges (edge : edges) graph = addEdge edge (addEdges edges
   graph) --есть ребро и хвост списка, то добавляем текущее ребр
   о edge к графу, полученному рекурсивным вызовом addEdges для
   хвоста edges и исходного графа
80
81 buildDiGraph :: (Hashable a, Eq a) => [(a, [a])] -> DiGraph a
82 buildDiGraph nodes = go nodes M.empty
83   where
84     go [] graph = graph --список пуст, возвращаем граф
85     go ((key, value) : xs) graph = M.insert key value (go xs
        graph) --вставляем в граф ключ key со значением value (сп
        исок смежных вершин) и рекурсивно обрабатываем хвост.
86
87 --принимает вершину и граф, возвращает список дочерних вершин
88 children :: (Hashable a, Eq a) => a -> DiGraph a -> [a]
89 children = M.findWithDefault []
90
91 --удаление вершины
92 deleteNode :: (Hashable a, Eq a) => a -> DiGraph a -> DiGraph a
93 deleteNode = M.delete
94
95 --удаление списка вершин
96 deleteNodes :: (Hashable a, Eq a) => [a] -> DiGraph a -> DiGraph
    a
97 deleteNodes [] graph = graph --список пуст, возвращаем граф
98 deleteNodes (x : xs) graph = M.delete x (deleteNodes xs graph)
   --удаляем вершину x из графа, полученного рекурсивным удалени
   ем хвоста xs
99
100 --удаление ребер
101 deleteEdge :: (Hashable a, Eq a) => (a, a) -> DiGraph a ->
    DiGraph a
102 deleteEdge (node, child) =
103   M.alter --для изменения значения по ключу node
104     ( \mNodes ->
105         case mNodes of
106           Nothing -> Just [] --если узел не найден, то возвращае
              м пустой список
107           Just nodes ->
108             Just (L.delete child nodes) --узел найден, то удаляе
                м child из списка смежности с помощью L.delete
109     )
110     node
111
112 --тип состояния:текущий фронт (очередь) вершин, граф оставшихся
    вершин и граф предшествования.
```

```
113 type SearchState a = ([a], DiGraph a, DiGraph a)
114 --тип результат поиска либо неудача, либо успех с графом предшес
    твования
115 data SearchResult a = Unsuccessful | Successful (DiGraph a)
116
117 --поиск в ширину от начальной до конечной вершины
118 bfsSearch :: forall a. (Hashable a, Eq a) => DiGraph a -> a -> a
    -> Maybe [a]
119 bfsSearch graph start end
120   | start == end = Just [start] --начальная и конечная вершины с
      овпадают, возвращаем список из одной вершины
121   | otherwise = --запускаем вспомогательную функцию bfsSearch' с
       начальным состоянием: фронт из start, исходный граф и пуст
      ой граф предшествования
122       case bfsSearch' ([start], graph, empty) of
123         Successful preds -> Just (findSolution preds) --успех то
            гда восстанавливаем путь с помощью findSolution
124         Unsuccessful -> Nothing --возвращаем Nothing
125   where --восстанавливает путь от end до start по графу предшест
      вования g
126     findSolution :: DiGraph a -> [a]
127     findSolution g = L.reverse (go end) --переворачиваем
128       where
129         go x =
130           case children x g of --смотрим на предшественников
131             [] -> [x] -- предшественников нет то есть x дб начал
                ьной вершиной
132             (v : _) -> x : go v -- иначе первый предшественник v
                . добавляем x к пути от v
133
134     addMultiplePredecessors :: [(a, [a])] -> DiGraph a ->
        DiGraph a
135     addMultiplePredecessors [] g = g --список пуст, возвращаем г
        раф
136     addMultiplePredecessors ((n, ch) : xs) g = --обрабатываем хв
        ост
137       addMultiplePredecessors xs (go n ch g)
138       where
139         go n [] g = g -- список детей пуст тогда граф
140         go n (x : xs) g = go n xs (addEdge (x, n) g) --для каждо
            го ребенка x из ch добавляем ребро (x, n) в граф пред
            шествования
141
142 --один шаг bfs
143     bfsSearch' :: SearchState a -> SearchResult a
144     bfsSearch' ([], _, preds) = Unsuccessful --фронт пусть тогда
         неудача
```

```
145     bfsSearch' (frontier, g, preds) = --иначе рабиваем на 3 част
          и
146       let g' = deleteNodes frontier g --удаляем вершины фронта
147           ch = -- список пар (n, [дети, которые в g'])
148             L.map
149               (\n -> (n, L.filter (`M.member` g') (children n g)
                    ))
150                frontier
151           frontier' = L.concatMap snd ch --объединение всех таки
                x детей
152           preds' = addMultiplePredecessors ch preds --обновляем
                граф предшествования
153        in if end `L.elem` frontier'
154             then Successful preds' --если конечная вершина в нов
                  ом фронте то возращаем успех
155             else bfsSearch' (frontier', g', preds')--иначе рекур
                  сивно вызываем bfsSearch' с новым состоянием
156
157 type BiSearchState a = (SearchState a, SearchState a)
158 --двунаправленный поиск в ширину - одновременно из начальной и к
      онечной вершины(прямой и обратный поиск)
159 biBfsSearch :: forall a. (Hashable a, Eq a) => DiGraph a -> a ->
      a -> Maybe [a]
160 biBfsSearch graph start end
161   | start == end = Just [start] --начальная и конечная вершины с
       овпадают
162   | otherwise = --начальное состояние для прямого поиска (fState
       ) и обратного (bState)
163       let fState = ([start], graph, empty)
164           bState = ([end], graph, empty)
165        in biBfsSearch' (fState, bState)
166   where
167     findSolution :: DiGraph a -> a -> [a]
168     findSolution g = go
169       where
170         go x =
171           case children x g of
172             [] -> [x] --предшественников нет
173             (v : _) -> x : go v ----- иначе первый предшественник
                  v. добавляем x к пути от v
174  --проверяет, есть ли пересечение между фронтами прямого и обрат
       ного поиска
175     checkOverlap :: BiSearchState a -> Maybe [a]
176     checkOverlap ((fFrontier', _, fPreds'), (bFrontier', _,
          bPreds')) =
177       let getSolution x =
178             let fPath = findSolution fPreds' x --путь от start д
                  о x
```

```
179                 bPath = findSolution bPreds' x --путь от x до
                        end
180               in L.reverse fPath ++ L.tail bPath --объединяем раз
                      вернутый fPath и bPath без первого элемента (это
                       x)
181           overlaps = --вершины из фронта прямого поиска, которые
                       есть в графе предшествования обратного поиска
182             L.filter (`M.member` bPreds') fFrontier'
183           solutions = --список всех возможных путей, отсортирова
                      нный по длине
184             L.sortOn L.length (L.map getSolution overlaps)
185       in if L.null solutions
186             then Nothing --если нет решений
187             else Just $ L.head solutions -- иначе кратчайший пут
                      ь
188 --проверяет, находится ли node во фронте frontier
189     checkSolution :: SearchState a -> a -> Maybe [a]
190     checkSolution (frontier, _, preds) node
191       | node `L.elem` frontier = Just (findSolution preds node)
            --путь от node до начальной вершины этого направления
192       | otherwise = Nothing
193
194     biBfsSearch' :: BiSearchState a -> Maybe [a]
195     biBfsSearch' state@(fState@(fFrontier, _, _), bState@(
          bFrontier, _, _)) --принимает состояние state (прямое и о
          братное)
196       | L.null fFrontier && L.null bFrontier = Nothing --оба пус
            ты
197       | isJust fSol = fmap L.reverse fSol --в прямом поиске коне
            чная вершина достигнута (fSol     это Just), то возвращ
            аем развернутый путь
198       | isJust bSol = bSol --Если в обратном поиске начальная ве
            ршина достигнута (bSol     это Just), возвращаем путь
199       | isJust overlapSol = overlapSol --Если есть пересечение,
            возвращаем решение из overlapSol
200       | otherwise = biBfsSearch' biState' --обновляем состояния
            fState' и bState' с помощью bfsSearchStep и рекурсивно
            вызываем biBfsSearch' с новым состоянием
201       where
202         fSol = checkSolution fState end
203         bSol = checkSolution bState start
204         overlapSol = checkOverlap state
205         fState' = bfsSearchStep fState
206         bState' = bfsSearchStep bState
207         biState' = (fState', bState')
208
209 --аналогично функции в bfsSearch, добавляет несколько ребер в гр
      аф предшествования
```

```
210     addMultiplePredecessors :: [(a, [a])] -> DiGraph a ->
          DiGraph a
211     addMultiplePredecessors [] g = g
212     addMultiplePredecessors ((n, ch) : xs) g =
213       addMultiplePredecessors xs (go n ch g)
214       where
215         go n [] g = g
216         go n (x : xs) g = go n xs (addEdge (x, n) g)
217 --Выполняет один шаг BFS: обновляет граф, вычисляет новых детей
      и новый фронт, обновляет граф предшествования.
218     bfsSearchStep :: SearchState a -> SearchState a
219     bfsSearchStep (frontier, g, preds) =
220       let g' = deleteNodes frontier g
221           ch =
222             L.map
223               (\n -> (n, L.filter (`M.member` g') (children n g)
                    ))
224               frontier
225           frontier' = L.concatMap snd ch
226           preds' = addMultiplePredecessors ch preds
227        in (frontier', g', preds')
228
229
230 -- логика поиска цепочек
231 module Ladder
232   ( Dictionary,
233     readDictionary,
234     ladderSolve,
235   )
236 where
237
238 import qualified Data.ByteString as BS
239 import qualified Data.ByteString.Char8 as C
240 import qualified Data.List as L
241 import qualified Graph as G
242 import qualified PermutationMap as PM
243
244 --тип словаря как списка байтовых строк
245 type Dictionary = [BS.ByteString]
246
247 --чтение словаря из файла, фильтрация только букв a-z
248 readDictionary :: FilePath -> IO Dictionary
249 readDictionary filepath = do
250   dictionaryContent <- C.readFile filepath --читаем словарь из ф
        айла
251   let lines = C.lines dictionaryContent --чтение всего содержимо
        го файла как байтовой строки
```

```
252       words = L.map (C.filter (`L.elem` ['a' .. 'z'])) lines --p
              азбиение на строки, фильтриция  только букв a-z
253   return words --возврат отчищенного списка
254
255 --удаление первого вхождения символа из строки
256 delete :: Char -> BS.ByteString -> BS.ByteString
257 delete ch string = case C.uncons string of
258   Just (x, xs) -> if ch == x then xs else C.cons x (delete ch xs
        ) --если символ совпадает - возвращаем хвост, иначе рекурси
        вно обрабатываем остаток
259   Nothing -> C.empty --пустая строка базовый случай
260
261 --вычисление соседних слов
262 computeCandidates :: PM.PermutationMap -> BS.ByteString -> [BS.
      ByteString]
263 computeCandidates map word = --мэп - карта перестановок
264   let candidates = modified ++ removed ++ added ++ [word] --спис
        ок всех возможных кандидатов-мутаций
265       perms = --для каждого кандидата ищем в карте перестановок
              реальные слова из словаря
266         L.concatMap
267           (\x -> PM.findWithDefault [] x map)
268           candidates
269    in L.filter (/= word) (L.nub perms) --фильтруем убираем дубли
        каты
270   where
271     added = [C.cons x word | x <- ['a' .. 'z']] -- добавление бу
          квы в начало
272     removed = [delete x word | x <- C.unpack word] -- удаление о
          дной буквы
273     modified = [C.cons x (delete y word) | x <- ['a' .. 'z'], y
          <- C.unpack word, x /= y] --замена каждой буквы
274
275 --граф из словаря
276 mkLadderGraph :: Dictionary -> G.DiGraph BS.ByteString
277 mkLadderGraph dict = G.buildDiGraph nodes --построение графа из
      списка (слово, список соседей)
278   where
279     map = PM.createPermutationMap dict --карта перестановок для
          эффективного поиска соседей
280     nodes = --для каждого слова вычисляем его соседей
281       L.map (\w -> (w, computeCandidates map w)) dict
282
283 --строит граф и ищет путь двунаправленным поиском
284 ladderSolve :: Dictionary -> String -> String -> Maybe [BS.
      ByteString]
285 ladderSolve dict start end = --Строим граф словной цепочки
```

```
286   let g = mkLadderGraph dict --двунаправленный поиск в ширину, п
        реобразуя String в ByteString
287    in G.biBfsSearch g (C.pack start) (C.pack end)
288
289
290 module PermutationMap
291   ( PermutationMap,
292     empty,
293     member,
294     alter,
295     delete,
296     insert,
297     lookup,
298     findWithDefault,
299     createPermutationMap,
300   )
301 where
302
303 import qualified Data.ByteString as BS
304 import qualified Data.HashMap.Lazy as M
305 import Data.Maybe (fromMaybe)
306 import Prelude hiding (lookup)
307
308 --тип для отображения отсортированных версий слов на списки исхо
    дных слов
309 type PermutationMap = M.HashMap BS.ByteString [BS.ByteString] --
    ключ - отсортированная версия слова, значение - список исходн
    ых слов с такими же буквами
310
311 empty :: PermutationMap
312 empty = M.empty --Создание пустой карты перестановок
313
314 member :: BS.ByteString -> PermutationMap -> Bool
315 member key = M.member (BS.sort key) --Проверка наличия ключа в к
    арте
316
317
318 alter :: --alter для модификации значения по ключу
319   ( Maybe [BS.ByteString] ->
320     Maybe [BS.ByteString]
321   ) ->
322   BS.ByteString ->
323   PermutationMap ->
324   PermutationMap
325 alter f key = M.alter f (BS.sort key) --применяем переданную фун
    кцию f к отсортированной версии ключа
326
327 delete :: BS.ByteString -> PermutationMap -> PermutationMap
```

```
328 delete key = M.delete (BS.sort key) --удаление записи по ключу (
    отсортированной версии слова)
329
330 insert :: BS.ByteString -> [BS.ByteString] -> PermutationMap ->
    PermutationMap
331 insert key = M.insert (BS.sort key) --вставка значения по отсорт
    ированному ключу
332
333 lookup :: BS.ByteString -> PermutationMap -> Maybe [BS.
    ByteString]
334 lookup key = M.lookup (BS.sort key) --поиск значения по отсортир
    ованному ключу, возвращает Maybe
335
336 findWithDefault :: --поиск со значением по умолчанию
337   [BS.ByteString] ->
338   BS.ByteString ->
339   PermutationMap ->
340   [BS.ByteString]
341 findWithDefault defaultValue key map =
342   fromMaybe [] (PermutationMap.lookup key map) --Если lookup воз
      вращает Nothing, используем пустой список
343
344 --создание карты перестановок из списка слов
345 createPermutationMap :: [BS.ByteString] -> PermutationMap
346 createPermutationMap = go empty --запуск рекурсивной функции с п
    устой картой
347   where
348     go permMap [] = permMap
349     go permMap (x : xs) = go (insertPermutation x permMap) xs --
        рекурсивная обработка списка слов
350
351     insertPermutation word = alter (insertList word) word --функ
        ция для вставки одного слова в карту
352
353     insertList word Nothing = Just [word] --ключа нет, создаем н
        овый список с одним словом
354     insertList word (Just words) = Just $ word : words --ключ уж
        е есть, добавляем слово в существующий список
```

Приложение. Часть 2. Полный код программы

Листинг 31. Часть 2. Полный код программы

```
{-# LANGUAGE OverloadedStrings #-}

module Main where

import System.IO
import Control.Exception (catch, IOException)
import Control.Monad (forever, when)
import Data.List (isSuffixOf)
import Text.Read (readMaybe)
import System.Random (randomRIO)
import Control.Concurrent (threadDelay)
import Lib
import Data.Text (Text)
import qualified Data.Text as T
import qualified Data.Text.IO as TIO

data ModelWithInfo = ModelWithInfo {
    model :: NGramModel,
    maxSentenceLength :: Int
} deriving (Show)

main :: IO ()
main = do
    TIO.putStrLn "\n 4-GRAM TEXT GENERATOR"
    mainLoop Nothing Nothing

mainLoop :: Maybe ModelWithInfo -> Maybe ModelWithInfo -> IO ()
mainLoop currentModel model2 = do
    TIO.putStrLn "\n MENU"
    TIO.putStrLn "0. Exit"
    TIO.putStrLn "1. Load or build main model"
    TIO.putStrLn "2. Save the current model to file"
    TIO.putStrLn "3. Generate the continuation of the sentence"
    TIO.putStrLn "4. Load or build second model for the dialogue
       "
    TIO.putStrLn "5. Dialogue between two models"

    -- Отображение текущих моделей
    TIO.putStrLn $ T.pack $ "\n Current first model: " ++
       modelStatus currentModel
    TIO.putStrLn $ T.pack $ "Current second model: " ++
       modelStatus model2

```

```
41      choice <- TIO.getLine
42      case T.unpack choice of
43
44          "1" -> do
45              TIO.putStrLn "\n Loading or building main model"
46              TIO.putStrLn "1. Build from a text file"
47              TIO.putStrLn "2. Load from model file"
48              TIO.putStrLn "0. Return"
49              subChoice <- TIO.getLine
50              case T.unpack subChoice of
51                  "0" -> mainLoop currentModel model2
52                  "1" -> do
53                      loadTextModelLoop currentModel model2
54                  "2" -> do
55                      loadModelLoop currentModel model2
56                  _ -> do
57                      TIO.putStrLn "Wrong choice!"
58                      mainLoop currentModel model2
59
60          "2" -> do
61              case currentModel of
62                  Nothing -> do
63                      TIO.putStrLn "Build or download the model
                            first!"
64                      mainLoop currentModel model2
65                  Just modelInfo -> do
66                      TIO.putStrLn "Enter a file name to save (0
                            to return):"
67                      fileName <- TIO.getLine
68                      when (fileName /= "0") $ do
69                          saveModel (T.unpack fileName) (model
                                modelInfo)
70                          TIO.putStrLn "Model saved!"
71                          mainLoop currentModel model2
72                      when (fileName == "0") $ mainLoop
                            currentModel model2
73
74          "3" -> do
75              case currentModel of
76                  Nothing -> do
77                      TIO.putStrLn "Build or download the model
                            first!"
78                      mainLoop currentModel model2
79                  Just modelInfo -> do
80                      TIO.putStrLn "Enter starting words (1-3
                            words, 0 to return):"
81                      input <- TIO.getLine
82                      when (input /= "0") $ do
```

```
83                      let wordsList = map T.toLower (T.words
                            input)
84                      if length wordsList >= 1 && length
                            wordsList <= 3
85                       then do
86                           let initialLen = length
                                wordsList
87                           let maxTotalLen =
                                maxSentenceLength modelInfo
88
89                           if maxTotalLen <= initialLen
90                            then do
91                                TIO.putStrLn $ T.pack $
                                    "Initial words
                                    already have " ++
                                    show initialLen ++
92                                   " words, but maximum
                                        sentence length
                                       is " ++ show
                                       maxTotalLen ++
93                                   ". Cannot generate
                                       continuation."
94                                mainLoop currentModel
                                    model2
95                            else do
96                                -- Генерируем случайную
                                    длину для дополнитель
                                    ных слов
97                                -- Минимум 1 дополнитель
                                    ное слово, максимум -
                                     оставшееся место
98                                let maxAdditional =
                                    maxTotalLen -
                                    initialLen
99                                randomAdditionalLen <-
                                    if maxAdditional > 0
100                                  then randomRIO (1,
                                         maxAdditional)
101                                  else return 0
102
103                               if randomAdditionalLen
                                    <= 0
104                                  then do
105                                      TIO.putStrLn "
                                             Cannot
                                             generate
                                             continuation
                                             - no space
```

```
                                                    for
                                                    additional
                                                    words."
106                                                 mainLoop
                                                    currentModel
                                                    model2
107                                             else do
108                                     result <-generateContinuation (model
                                            modelInfo) wordsList
                                        randomAdditionalLen
109                                                 let totalLength
                                                    = length
                                                    result
110                                                 TIO.putStrLn $ "
                                                    Generated
                                                    continuation:
                                                     " `T.append`
                                                     T.unwords
                                                    result
111                                                 mainLoop
                                                    currentModel
                                                    model2
112                                     else do
113                                         TIO.putStrLn "Error: Please
                                               enter between 1 and 3 words."
114                                     mainLoop currentModel model2
115                         when (input == "0") $ mainLoop currentModel
                               model2
116
117             "4" -> do
118                 TIO.putStrLn "\n Loading the second model"
119                 TIO.putStrLn "1. Build from a text file"
120                 TIO.putStrLn "2. Load from model file"
121                 TIO.putStrLn "0. Return"
122                 subChoice <- TIO.getLine
123                 case T.unpack subChoice of
124                     "0" -> mainLoop currentModel model2
125                     "1" -> do
126                         loadTextModelForSecondLoop currentModel
                               model2
127                     "2" -> do
128                         loadSecondModelLoop currentModel model2
129                     _ -> do
130                         TIO.putStrLn "Wrong choice!"
131                         mainLoop currentModel model2
132
133             "5" -> do
134                 case (currentModel, model2) of
```

```
135                 (Just m1, Just m2) -> do
136                     TIO.putStrLn "Select model order: "
137                     TIO.putStrLn "1. The main model starts first
                            "
138                     TIO.putStrLn "2. The second model starts
                            first"
139                     TIO.putStrLn "0. Return"
140                     orderChoice <- TIO.getLine
141                     case T.unpack orderChoice of
142                         "0" -> mainLoop currentModel model2
143                         "1" -> startDialogLoop (model m1) (model
                                m2) currentModel model2
144                         "2" -> startDialogLoop (model m2) (model
                                m1) currentModel model2
145                         _ -> do
146                             TIO.putStrLn "Wrong choice!"
147                             mainLoop currentModel model2
148                 _ -> do
149                     TIO.putStrLn "First, download both models! (
                            the main and the second)"
150                     mainLoop currentModel model2
151
152         "0" -> TIO.putStrLn "Exit..."
153         _ -> do
154             TIO.putStrLn "Wrong choice!"
155             mainLoop currentModel model2
156
157 -- Вспомогательная функция для статуса модели
158 modelStatus :: Maybe ModelWithInfo -> String
159 modelStatus Nothing = "not loaded"
160 modelStatus (Just _) = "loaded"
161
162 -- Функция для запуска диалога
163 startDialogLoop :: NGramModel -> NGramModel -> Maybe
       ModelWithInfo -> Maybe ModelWithInfo -> IO ()
164 startDialogLoop firstModel secondModel currentModel model2 = do
165     TIO.putStrLn "Enter the initial word (0 to return):"
166     startWord <- TIO.getLine
167     case startWord of
168         "0" -> mainLoop currentModel model2
169         _ -> do
170             let wordsList = T.words startWord
171             if length wordsList /= 1
172                 then do
173                     TIO.putStrLn "Error: Please enter exactly
                            one word. Try again or enter 0 to return.
                            "
```

```
174                     startDialogLoop firstModel secondModel
                            currentModel model2
175                 else do
176                     TIO.putStrLn "Enter the number of replicas
                            (0 to return):"
177                     repsStr <- TIO.getLine
178                     case repsStr of
179                         "0" -> mainLoop currentModel model2
180                         _ -> do
181                             let reps = safeRead (T.unpack
                                    repsStr) 5
182                             startDialog firstModel secondModel
                                    wordsList reps
183                             mainLoop currentModel model2
184
185 -- Цикл загрузки модели из файла с повторением при ошибке
186 loadModelLoop :: Maybe ModelWithInfo -> Maybe ModelWithInfo ->
        IO ()
187 loadModelLoop currentModel model2 = do
188     TIO.putStrLn "Enter the file name of the model (0 to return)
            :"
189     fileName <- TIO.getLine
190     case fileName of
191         "0" -> mainLoop currentModel model2
192         _ -> do
193             loadedModel <- loadModel (T.unpack fileName)
194             case loadedModel of
195                 Just m -> do
196                     TIO.putStrLn "Model loaded!"
197                     mainLoop (Just $ ModelWithInfo m 20) model2
198                 Nothing -> do
199                     TIO.putStrLn "Model loading error! Please
                            try again."
200                     loadModelLoop currentModel model2
201
202 -- Цикл загрузки текстовой модели
203 loadTextModelLoop :: Maybe ModelWithInfo -> Maybe ModelWithInfo
        -> IO ()
204 loadTextModelLoop currentModel model2 = do
205     TIO.putStrLn "Enter a file name with text (0 to return):"
206     fileName <- TIO.getLine
207     case fileName of
208         "0" -> mainLoop currentModel model2
209         _ -> do
210             result <- readFileSafe (T.unpack fileName)
211             case result of
212                 Left errorMsg -> do
213                     TIO.putStrLn $ T.pack errorMsg
```

```
214                     loadTextModelLoop currentModel model2
215                 Right content -> do
216                     let sentences = splitText (T.pack content)
217                     -- Подсчет и вывод количества слов и максима
                            льной длины предложения
218                     let wordCount = sum (map length sentences)
219                     let maxSentenceLen = if null sentences then
                            0 else maximum (map length sentences)
220                     TIO.putStrLn $ "Text loaded. Total words: "
                            `T.append` T.pack (show wordCount)
221                     TIO.putStrLn $ "Max sentence length: " `T.
                            append` T.pack (show maxSentenceLen)
222                     let newModel = buildNGramModel sentences 4
223                     TIO.putStrLn "Model successfully built!"
224                     -- Сохраняем модель и максимальную длину пре
                            дложения
225                     mainLoop (Just $ ModelWithInfo newModel
                            maxSentenceLen) model2
226
227 -- Цикл загрузки второй модели из текстового файла
228 loadTextModelForSecondLoop :: Maybe ModelWithInfo -> Maybe
        ModelWithInfo -> IO ()
229 loadTextModelForSecondLoop currentModel model2 = do
230     TIO.putStrLn "Enter a file name with text (0 to return): "
231     fileName <- TIO.getLine
232     case fileName of
233         "0" -> mainLoop currentModel model2
234         _ -> do
235             result <- readFileSafe (T.unpack fileName)
236             case result of
237                 Left errorMsg -> do
238                     TIO.putStrLn $ T.pack errorMsg
239                     loadTextModelForSecondLoop currentModel
                            model2
240                 Right content -> do
241                     let sentences = splitText (T.pack content)
242                     -- Подсчет и вывод количества слов и максима
                            льной длины
243                     let wordCount = sum (map length sentences)
244                     let maxSentenceLen = if null sentences then
                            0 else maximum (map length sentences)
245                     TIO.putStrLn $ "Text loaded. Total words: "
                            `T.append` T.pack (show wordCount)
246                     TIO.putStrLn $ "Max sentence length: " `T.
                            append` T.pack (show maxSentenceLen)
247                     let newModel = buildNGramModel sentences 4
248                     TIO.putStrLn "The second model has been
                            successfully built!"
```

```
249                     mainLoop currentModel (Just $ ModelWithInfo
                            newModel maxSentenceLen)
250
251 -- Цикл загрузки второй модели из файла модели
252 loadSecondModelLoop :: Maybe ModelWithInfo -> Maybe
        ModelWithInfo -> IO ()
253 loadSecondModelLoop currentModel model2 = do
254     TIO.putStrLn "Enter the file name of the model (0 to return)
            : "
255     fileName <- TIO.getLine
256     case fileName of
257         "0" -> mainLoop currentModel model2
258         _ -> do
259             loadedModel <- loadModel (T.unpack fileName)
260             case loadedModel of
261                 Just m -> do
262                     TIO.putStrLn "The second model has been
                            loaded!"
263                     -- Для загруженных моделей устанавливаем деф
                            олтную максимальную длину
264                     mainLoop currentModel (Just $ ModelWithInfo
                            m 20)
265                 Nothing -> do
266                     TIO.putStrLn "Model loading error! Please
                            try again."
267                     loadSecondModelLoop currentModel model2
268
269 startDialog :: NGramModel -> NGramModel -> [Text] -> Int -> IO
        ()
270 startDialog model1 model2 startWords reps = do
271     TIO.putStrLn "\n START OF DIALOGUE"
272     TIO.putStrLn $ "Starting word: " `T.append` T.unwords
            startWords
273     dialogLoop model1 model2 startWords reps 0
274
275 dialogLoop :: NGramModel -> NGramModel -> [Text] -> Int -> Int
        -> IO ()
276 dialogLoop _ _ _ totalReps currentRep
277     | currentRep >= totalReps = TIO.putStrLn "DIALOGUE ENDED"
278 dialogLoop model1 model2 lastPhrase totalReps currentRep = do
279     let currentModel = if even currentRep then model1 else
            model2
280     let opponentModel = if even currentRep then model2 else
            model1
281     let speaker = if even currentRep then "Model 1" else "Model
            2"
282
283     response <- generateResponse currentModel lastPhrase
```

```
284     TIO.putStrLn $ T.pack speaker `T.append` ": " `T.append` T.
            unwords response
285
286     dialogLoop model1 model2 response totalReps (currentRep + 1)
287
288 generateResponse :: NGramModel -> [Text] -> IO [Text]
289 generateResponse model phrase = do
290     -- Получаем возможные ключи в правильном порядке: 3 слова ->
            2 слова -> 1 слово
291     let possibleKeys = getPossibleKeys phrase
292
293     -- Пытаемся найти валидный ключ в порядке убывания длины
294     case findValidKey model possibleKeys of
295         Just key -> do
296             maxLen <- randomRIO (3, 15)
297             result <- generateContinuation model key maxLen
298             return result
299
300         Nothing -> do
301             -- Генерация случайного предложения
302             allKeys <- getRandomKey model
303             case allKeys of
304                 Just randomKey -> do
305                     maxLen <- randomRIO (3, 10)
306                     result <- generateContinuation model
                            randomKey maxLen
307                     return ("!" : result)  -- Помечаем случайное
                            предложение
308                 Nothing -> return ["..."]  -- Если совсем нет кл
                        ючей
309
310 -- Безопасное чтение файла с обработкой ошибок
311 readFileSafe :: FilePath -> IO (Either String String)
312 readFileSafe fileName = catch (readFile fileName >>= return .
        Right) handleError
313   where
314     handleError :: IOException -> IO (Either String String)
315     handleError e = do
316         let errorMsg = "File reading error: " ++ show e
317         return $ Left errorMsg
318
319 -- Безопасное чтение чисел с значением по умолчанию
320 safeRead :: String -> Int -> Int
321 safeRead str defaultValue = case readMaybe str of
322     Just n -> n
323     Nothing -> defaultValue
324
325     {-# LANGUAGE OverloadedStrings #-}
```

```
326
327 module Lib (
328     NGramModel,
329     splitText,
330     buildNGramModel,
331     saveModel,
332     loadModel,
333     generateContinuation,
334     getPossibleKeys,
335     findValidKey,
336     getRandomKey,
337     preprocessText
338 ) where
339
340 import Data.Char (isLetter, toLower, isPunctuation, isSpace,
        isAscii)
341 import Data.Hashable (Hashable)
342 import Data.HashMap.Strict (HashMap)
343 import qualified Data.HashMap.Strict as HashMap
344 import Data.List (tails, sortOn)
345 import Data.Text (Text)
346 import qualified Data.Text as T
347 import qualified Data.Text.IO as TIO
348 import System.Random (randomRIO, getStdGen, randomR, StdGen,
        newStdGen)
349 import System.IO
350 import Data.Maybe (fromMaybe, listToMaybe, catMaybes)
351 import Control.Exception (try, SomeException)
352 import Text.Read (readMaybe)
353 import Data.Ord (Down(..))
354
355 type NGram = [Text]
356 type NGramModel = HashMap NGram [(NGram, Double)]
357
358 -- Упрощенная очистка слова
359 cleanWord :: Text -> Text
360 cleanWord word =
361     let cleaned = T.toLower $ T.filter validChar word
362     in if isValidCleanedWord cleaned then cleaned else T.empty
363   where
364     validChar c = isAscii c && (isLetter c || c == '\'' || c ==
          '-')
365     isValidCleanedWord w = not (T.null w) && T.any isLetter w
366
367 -- Проверка валидности слова
368 isValidWord :: Text -> Bool
369 isValidWord word =
```

```
370     not (T.null word) && T.all validChar word && T.any isLetter
          word
371   where
372     validChar c = isAscii c && (isLetter c || c == '\'' || c ==
          '-')
373
374 -- Функция для проверки завершения предложения
375 isSentenceTerminator :: Char -> Bool
376 isSentenceTerminator c = c `elem` (".!?" :: [Char])
377
378 -- Оптимизированное разделение текста на предложения
379 splitText :: Text -> [[Text]]
380 splitText text =
381     let preprocessed = preprocessText text
382         sentences = splitSentences preprocessed
383     in filter (not . null) $ map processSentence sentences
384   where
385     splitSentences :: Text -> [Text]
386     splitSentences t
387         | T.null t = []
388         | otherwise =
389             let (sentence, rest) = T.break isSentenceTerminator
                    t
390                 (punctuation, rest') = T.span
                        isSentenceTerminator rest
391             in if T.null sentence
392                then splitSentences rest'
393                else (cleanSentence sentence `T.append` T.take 1
                       punctuation) : splitSentences rest'
394
395     cleanSentence :: Text -> Text
396     cleanSentence = removeEdgePunctuation . T.strip
397
398     removeEdgePunctuation :: Text -> Text
399     removeEdgePunctuation s =
400         let startPunct :: Text
401             startPunct = " \"'`([{"
402             endPunct :: Text
403             endPunct = " \"'`)]}"
404             removeStart = T.dropWhile (`T.elem` startPunct) s
405             removeEnd = T.reverse $ T.dropWhile (`T.elem`
                    endPunct) (T.reverse removeStart)
406         in removeEnd
407
408     processSentence :: Text -> [Text]
409     processSentence = filter isValidWord . map (T.toLower) . T.
          words . T.map normalizeChar
410
```

```
411     normalizeChar :: Char -> Char
412     normalizeChar c
413         | c `elem` (",;:\"'`()[]{}" :: String) = ' '
414         | isPunctuation c && c /= '\'' && c /= '-' = ' '
415         | otherwise = c
416
417 -- Строгая проверка N-граммы
418 filterValidNGram :: (NGram, NGram) -> Bool
419 filterValidNGram (key, value) =
420     all isValidWord (key ++ value)
421
422 -- Оптимизированное построение N-граммной модели
423 buildNGramModel :: [[Text]] -> Int -> NGramModel
424 buildNGramModel sentences n =
425     let validSentences = filter (all isValidWord) sentences
426         allNGrams = concatMap (getNGrams n) validSentences
427         grouped = HashMap.fromListWith (++) [(key, [(value, 1)])
                | (key, value) <- allNGrams]
428     in HashMap.map calculateProbabilities grouped
429   where
430     getNGrams :: Int -> [Text] -> [(NGram, NGram)]
431     getNGrams n words =
432         [ (take k context, take (n - k) continuation)
433         | k <- [1..min 3 (length words)]
434         , (context, continuation) <- zip (tails words) (drop k (
              tails words))
435         , length context >= k
436         , length continuation >= n - k
437         , length context + length continuation >= n
438         , filterValidNGram (take k context, take (n - k)
              continuation)
439         ]
440
441     calculateProbabilities :: [(NGram, Int)] -> [(NGram, Double)
        ]
442     calculateProbabilities pairs =
443         let total = fromIntegral $ sum (map snd pairs)
444         in [(ngram, fromIntegral count / total) | (ngram, count)
              <- pairs]
445
446 -- Предварительная обработка текста
447 preprocessText :: Text -> Text
448 preprocessText text =
449     let asciiOnly = T.filter isAscii text
450         cleaned = preserveWordHyphens asciiOnly
451     in T.unwords . T.words $ cleaned
452
453 -- Сохраняет дефисы внутри слов
```

```haskell
preserveWordHyphens :: Text -> Text
preserveWordHyphens = T.unwords . map processWord . T.words
  where
    processWord w
        | T.all (== '-') w = T.empty
        | hasValidHyphens w = w
        | otherwise = T.filter (\c -> c /= '-' || isLetter c) w

    hasValidHyphens word =
        let parts = T.splitOn "-" word
        in all (T.any isLetter) parts && length parts > 1

-- Сохранение модели в файл
saveModel :: FilePath -> NGramModel -> IO ()
saveModel filePath model = withFile filePath WriteMode $ \h ->
    mapM_ (\(k, vs) -> TIO.hPutStrLn h $ T.pack (show k) `T.
       append` " => " `T.append` T.pack (show vs)) (HashMap.
       toList model)

-- Загрузка модели из файла
loadModel :: FilePath -> IO (Maybe NGramModel)
loadModel filePath = do
    result <- try (TIO.readFile filePath) :: IO (Either
       SomeException Text)
    case result of
        Left _ -> return Nothing
        Right content ->
            let lines' = T.lines content
                parsed = catMaybes $ map parseLine lines'
            in return $ if null parsed then Nothing else Just (
               HashMap.fromList parsed)
  where
    parseLine :: Text -> Maybe (NGram, [(NGram, Double)])
    parseLine line =
        let (keyPart, valuePart) = T.break (== '=') line
        in case T.uncons valuePart of
            Just ('=', rest) ->
                case T.uncons rest of
                    Just ('>', rest') ->
                        case T.uncons rest' of
                            Just (' ', valueStr) ->
                                case readMaybe (T.unpack (T.
                                   strip keyPart)) of
                                    Just key ->
                                        case readMaybe (T.unpack
                                            valueStr) of
                                            Just values -> Just
                                               (key, values)
```

```
                                                    Nothing -> Nothing
                                                Nothing -> Nothing
                                            _ -> Nothing
                                    _ -> Nothing
                            _ -> Nothing

-- Оптимизированная генерация продолжения предложения
generateContinuation :: NGramModel -> [Text] -> Int -> IO [Text]
generateContinuation model startWords maxLength = do
    gen <- newStdGen
    return $ generateEfficient model startWords gen maxLength
  where
    generateEfficient :: NGramModel -> [Text] -> StdGen -> Int
       -> [Text]
    generateEfficient model currentWords gen maxLength =
        generateLoop model currentWords gen maxLength
           currentWords

    generateLoop :: NGramModel -> [Text] -> StdGen -> Int -> [
       Text] -> [Text]
    generateLoop _ _ _ 0 acc = acc
    generateLoop model currentWords gen remaining acc =
        let possibleKeys = getPossibleKeys currentWords
            key = findValidKey model possibleKeys
        in case key of
            Nothing -> acc
            Just k ->
                case HashMap.lookup k model of
                    Nothing -> acc
                    Just continuations ->
                        let (choice, newGen) =
                           efficientWeightedRandom continuations
                            gen
                            newWords = acc ++ choice
                            newRemaining = remaining - length
                               choice
                        in if newRemaining <= 0
                           then newWords
                           else generateLoop model (currentWords
                              ++ choice) newGen newRemaining
                             newWords

    -- Эффективный взвешенный случайный выбор с предварительной
       сортировкой
    efficientWeightedRandom :: [(NGram, Double)] -> StdGen -> (
       NGram, StdGen)
    efficientWeightedRandom choices gen =
        let total = sum (map snd choices)
```

```
533             (r, newGen) = randomR (0, total) gen
534             -- Сортируем по убыванию вероятности для более быстр
                   ого выбора
535             sortedChoices = sortOn (Down . snd) choices
536         in pick sortedChoices r newGen
537       where
538         pick [] _ g = ([], g)
539         pick ((ngram, prob):rest) rVal g
540             | rVal <= prob = (ngram, g)
541             | otherwise = pick rest (rVal - prob) g
542 
543 -- Получение возможных ключей из фразы в правильном порядке
544 getPossibleKeys :: [Text] -> [[Text]]
545 getPossibleKeys words =
546     let n = length words
547         -- Берем последние 1, 2, 3 слова в ПРЯМОМ порядке
548         key1 = if n >= 1 then [last words] else []
549         key2 = if n >= 2 then drop (n-2) words else []
550         key3 = if n >= 3 then drop (n-3) words else []
551     in filter (not . null) [key3, key2, key1]  -- Сначала провер
           яем самые длинные ключи
552 
553 -- Поиск валидного ключа в модели
554 findValidKey :: NGramModel -> [[Text]] -> Maybe [Text]
555 findValidKey model keys =
556     case filter (`HashMap.member` model) keys of
557         [] -> Nothing
558         (k:_) -> Just k
559 
560 -- Получение случайного ключа из модели
561 getRandomKey :: NGramModel -> IO (Maybe [Text])
562 getRandomKey model =
563     if HashMap.null model
564         then return Nothing
565         else do
566             let keys = HashMap.keys model
567             idx <- randomRIO (0, length keys - 1)
568             return $ Just (keys !! idx)
```