|  |  |
| --- | --- |
|  | |
| МИНОБРНАУКИ РОССИИ | |
| Федеральное государственное бюджетное образовательное учреждение  высшего образования  **«МИРЭА – Российский технологический университет»**  **РТУ МИРЭА** | |
| Институт информационных технологий | |
| Кафедра корпоративных информационных систем  **КУРСОВАЯ РАБОТА**  по дисциплине  Структура и алгоритмы обработки данных  **Тема курсовой работы**: Генератор текста на тему «Физика»   |  |  | | --- | --- | | Студент группы ИКБО-07-18 | Зейналов Магеррам Гилал оглы | |  |  | |  | \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  (подпись студента) | |  |  | | Руководитель курсовой работы | Советов Пётр Николаевич | |  |  | |  | \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  (подпись руководителя) | |  |  | | Работа представлена к защите | «20» декабря 2019 г. | |  |  | | Допущен к защите | «23» декабря 2019 г. | | |

|  |  |
| --- | --- |
|  | |
| МИНОБРНАУКИ РОССИИ | |
| Федеральное государственное бюджетное образовательное учреждение  высшего образования  **«МИРЭА – Российский технологический университет»**  **РТУ МИРЭА** | |
| Институт информационных технологий | |

Кафедра корпоративных информационных систем

Утверждаю

И.о. Заведующего кафедрой КИС

\_\_\_\_\_\_\_\_\_\_\_\_\_Е.Г. Андрианова

«14» сентября 2019 г.

**ЗАДАНИЕ**

**на выполнение курсовой работы по дисциплине**

Структура и алгоритмы обработки данных

|  |  |
| --- | --- |
| Студент Зейналов Магеррам Гилал оглы | Группа ИКБО-07-18 |

**Тема работы:** Генератор текста на тему «Физика»

**Исходные данные:** литература по марковским цепям и констекстно-свободным грамматикам, описание языка и среды разработки на языке Python, стандарт оформления программного кода на языке Python

**Перечень вопросов, подлежащих разработке, и обязательного графического материала:**

1. Изучение методов построения предложений с помощью марковских цепей и контекстно-свободных грамматик.
2. Проектирование структуры генератора.
3. Реализация и тестирование генератора средствами языка Python.
4. Оценка полученных результатов путём опроса людей.
5. Написание и оформление расчетно-пояснительной записки.

|  |  |  |  |
| --- | --- | --- | --- |
| **Срок представления к защите курсовой работы:** | | до «21» декабря 2019 г. | |
|  | |  | |
| **Задание на курсовую работу выдал** | \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ | | (Советов П.Н.) |
|  | | «07» сентября 2019 г. | |
| **Задание на курсовую работу получил** | \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ | | (Зейналов М.Г.) |

## Оглавление

[Оглавление 3](#_Toc26728259)

[ВВЕДЕНИЕ 5](#_Toc26728260)

[ОСНОВНАЯ ЧАСТЬ 6](#_Toc26728261)

[1. ВВЕДЕНИЕ В МАРКОВСКИЕ ЦЕПИ 6](#_Toc26728262)

[2. АЛГОРИТМ ПОРОЖДЕНИЯ ТЕКСТА 9](#_Toc26728263)

[2.1. Описание алгоритма 9](#_Toc26728264)

[2.2. Кодирование алгоритма 11](#_Toc26728265)

[2.3. Тестирование алгоритма 17](#_Toc26728266)

[2.4. Программные тесты алгоритмов обучения модели 18](#_Toc26728267)

[3. КОНТЕКСТНО-СВОБОДНЫЕ ГРАММАТИКИ 19](#_Toc26728268)

[3.1. О писание алгоритма 19](#_Toc26728269)

[3.2. Кодирование алгоритма 20](#_Toc26728270)

[3.3. Тестирование алгоритма 20](#_Toc26728271)

[ЗАКЛЮЧЕНИЕ 22](#_Toc26728272)

[СПИСОК ИСПОЛЬЗУЕМЫХ РЕСУРСОВ 23](#_Toc26728273)

[ПРИЛОЖЕНИЕ 24](#_Toc26728274)

[main.py 24](#_Toc26728275)

[new\_dict.py 25](#_Toc26728276)

[markov.py 26](#_Toc26728277)

[test.py 28](#_Toc26728278)

[test\_struct.py 30](#_Toc26728279)

## ВВЕДЕНИЕ

Генерация текста с помощью марковских цепей и контекстно-свободных грамматик может использоваться во многих областях. Например, для заполнения сайта, генерации текста для дизайна шаблонов (так называемая «рыба»), а также для SEO продвижения.

Сегодня для генерации контента остается все меньше времени, люди стали задумываться о возможности автоматической генерации контента для личных нужд. Тут приходят на помощь математический аппарат, в частности, математическая статистика. На основе статистики построен принцип генерации текста, изображений и даже музыки. Применим её для создания текстов.

Целью работы является разработка генератора текста (необязательно осмысленного) для личных нужд. Для получения желаемого результата необходимо изучить построение марковских цепей, контекстно-свободных грамматик. Также важно подготовить корректную упорядоченную словарную базу для обучения нашей модели.

## ОСНОВНАЯ ЧАСТЬ

## ВВЕДЕНИЕ В МАРКОВСКИЕ ЦЕПИ

Марковский процесс – случайный процесс, эволюция которого после любого заданного значения временного параметра t не зависит от эволюции, предшествовавшей t, при условии, что значение процесса в этот момент фиксировано.

Марковская цепь – частный случай марковского процесса, когда пространство его состояний дискретно.

Рассмотрим пример построения марковской цепи на основе предложения.



Рисунок 1 – «Предложение для демонстрации работы марковской цепи»

Это предложение и есть база для генерации текста. Он состоит из8 слов, но уникальных – 5. Промаркируем рисунок 1 по цветам в соответствии уникальности.



Рисунок 2 – «Промаркированное предложение»

Выпишем количество слов в предложении.

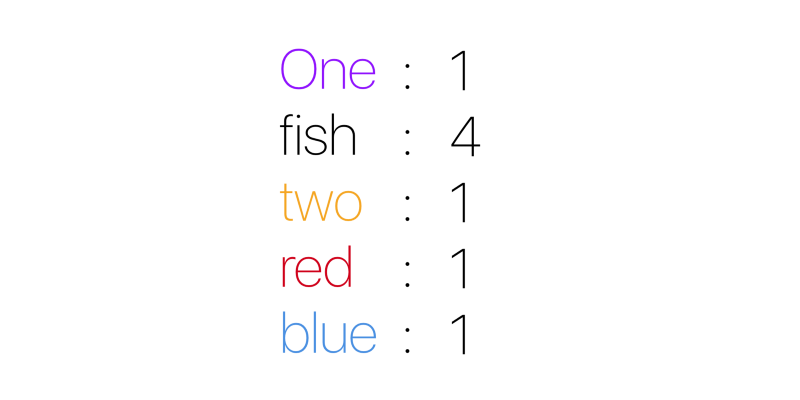


Рисунок 3 – «Количество вхождений слов в предложении»

На рисунке 3 видно, что слово fish встречается 4 раза, т.е. вероятность встретить это слово в 4 раза выше, чем любое другое из данного корпуса. Т.е. для генерации текста, мы будем рассчитывать вероятность появления слова в тексте и на основе этого составлять предложения.

Для более простого восприятия представим частоту появления слов в графической форме.

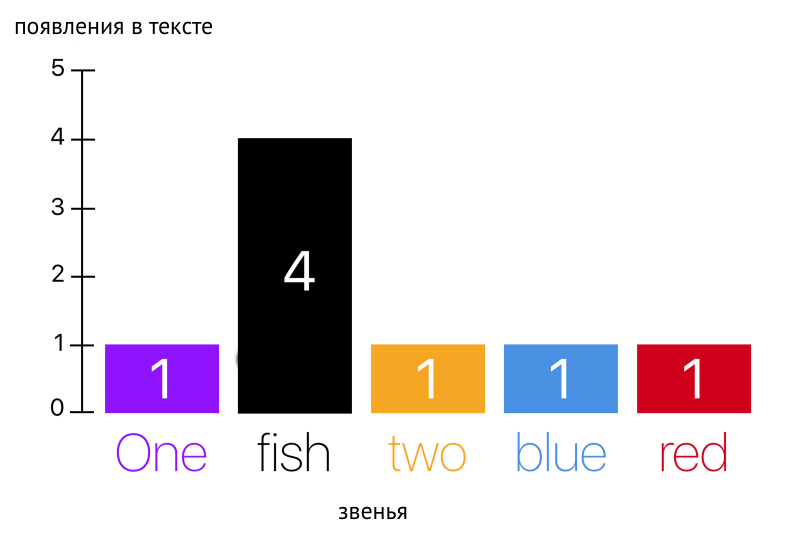


Рисунок 4 – «Гистограмма появления слов»

Для связности слов используются различные структуры данных, например, словарь. Достоинство словаря – константное время добавления и доступа к элементам, т.е. нет привязанности к размерам корпуса для обучения.

Строгая связанность слов необходима для зависимости только от текущего состояния. Система не знает в каком состоянии мы находились ранее. Это необходимо для более уникального построения текстов, и сильно упрощает реализацию всех алгоритмов и ускоряет процесс их выполнения.

## АЛГОРИТМ ПОРОЖДЕНИЯ ТЕКСТА

## Описание алгоритма

Для начала модифицируем питоновский словарь для наших целей.

Добавим в конструктор словаря количество уникальных элементов и общее количество слов.

Обновим метод update(). Если объект итерируемый, то пройдемся по всем его элементам и проверим, входит ли он уже в словарь. Если входит, то просто добавляем 1 к значению слова в словаре и к общему количеству слов. А если такого слова нет в словаре, то мы создаем для него поле и обозначаем, что данное слово встречается один раз в словаре. Дальше добавим 1 к общему количеству слов и к количеству уникальных слов.

Метод count() показывает частоту появления слова в тексте.

Метод return\_weighted\_random\_word() – один из главных методов класса MarkDict. На данном методе построен алгоритм марковских цепей. Для выбора последующего элемента, мы используем состояние системы в данный момент, т.е. нас не интересует какие состояние система принимала в предыдущие моменты, какие слова мы включали в предложения. Для обучения по 2 последующим словам алгоритм не меняется, изменяется только сам ключ словаря, теперь он состоит не из одного слова, а из двух.

Обучение модели осуществляется при помощи двух методов: train\_model() и train\_model\_higher(). Первый метод проверяет входит ли слово в словарь, если входит, то используем метод update(), если нет, то добавляем в словарь. Второй метод отличается только списком, который мы подаем в цикл. Мы соединяем 2 соседних слова, чтобы обучение модели проходило по 2 последующим словам.

Для выбора следующего слова мы генерируем случайное число (random\_index) в диапазоне [0; количество слов (self.tokens)). Дальше проходимся по всем уникальным словам (self.types) в цикле и добавляем в новую переменную (index) количество повторений в тексте, когда мы превысим случайно сгенерированное число, выходим из функции, возвращая текущее слово. Для понимания работы рассмотрим небольшой пример. Пусть текущее слово – город. В словаре город имеет следующие пары: (‘Москва’: 3, ‘Петербург’: 2, ‘Лондон’: 1). Случайно сгенерированное число – 4. Тогда начнем проходиться по циклу в диапазоне [0; количество уникальных слов).

Первичное состояние:

Random\_index = 4

Index = 0

Первая итерация:

I = 0

Index = 3

Условие не проходим.

Вторая итерация:

I = 1

Index = 5

Условие проходим. Возвращаем слово «Петербург».

Рассмотрим алгоритм выбора начала предложения. Для корректной отработки алгоритма изначально было необходимо подготовить текст. Во время разработки алгоритма возникла проблема разделения текста на предложения. Основная суть проблемы – как отличить точку в конце предложения от точки в сокращении? Для решения данной задачи я использовал библиотеку NLTK. При помощи данной библиотеки мы разделяем текст на предложения, заменяя «.» на « . ». Теперь точка будет считаться отдельным словом, что даёт возможность применить ранее написанный алгоритм выбора слова.

Генерация предложений происходит в методе generate\_sentence(). Изначально мы находим первое слово, с которого будем начинать предложение. Дальше находим все последующие слова по алгоритму, описанному выше.

## Кодирование алгоритма

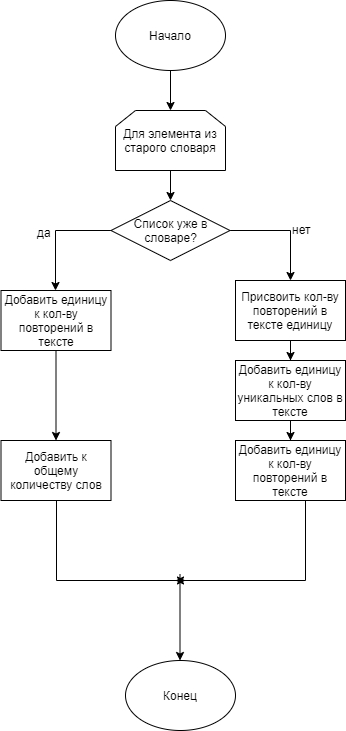


Рисунок 5 – Алгоритм обновления словаря

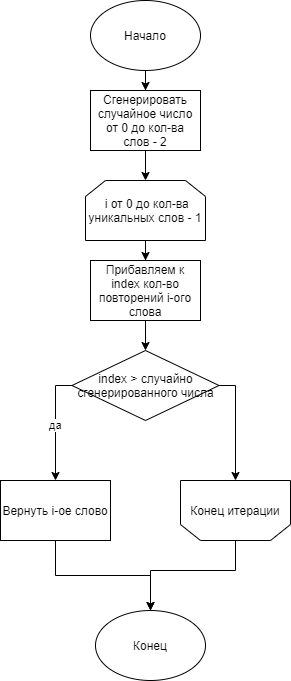


Рисунок 6 – Алгоритм возвращения взвешенного слова

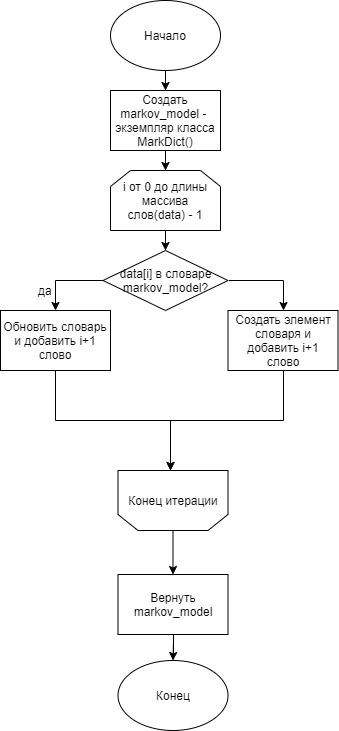


Рисунок 7 – Алгоритм обучения марковской цепи 1 порядка

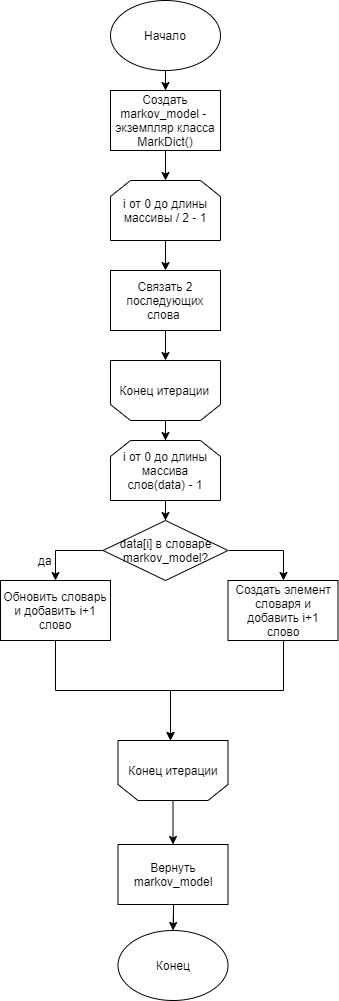


Рисунок 8 – Алгоритм обучения марковской цепи 2 порядка

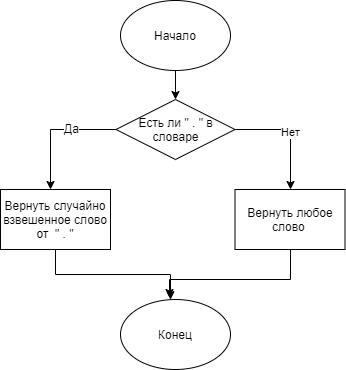


Рисунок 9 – Алгоритм генерации первого слова

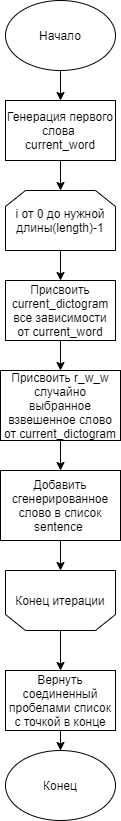


Рисунок 10 – Алгоритм генерации предложения

## Тестирование алгоритма

До применения подхода с увеличенным окном, текст был плохого качества.

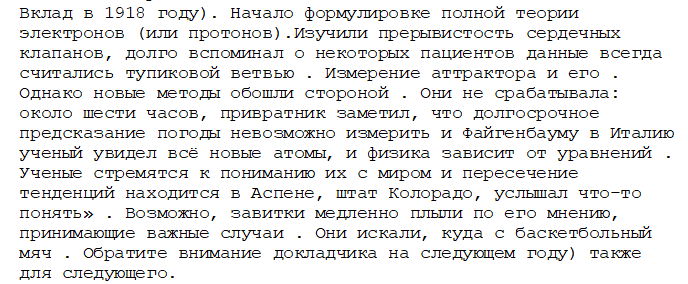


Рисунок 11 – Результат работы марковских цепей с окном в 1 слово

После доработки алгоритма, результат значительно улучшился, текст стал более осмысленным.

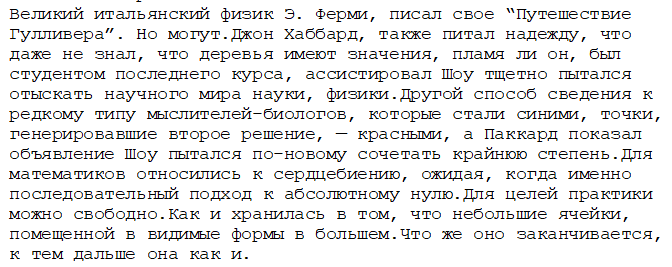


Рисунок 12 - Результат работы марковских цепей с окном в 2 слово

## Программные тесты алгоритмов обучения модели

Для тестирования модуля Markov используем встроенный модуль unittest. С помощью данного модуля мы можем создавать программные тесты, которые будут проверять корректность нашего кода.

Для тестирования метода train\_model() создадим тестовый текст TEST.txt. Напишем для него эталонный словарь, и будем проверять корректность получаемого словаря. Метод train\_model\_higher() тестируем аналогично.

Дальше проверим метод return\_random\_start(). В результате тестирования нам должно возвращаться слово, которое находится в словаре по ключу «.».

В результате программного тестирования установлено, что модуль Markov работает корректно, все основные функции отрабатывают правильно, что показывает работоспособность Markov.

Полный код тестирования находится в приложении test.py.

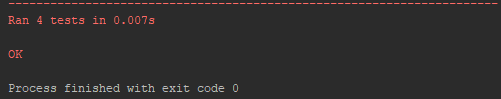


Рисунок 13 – Результаты тестирования модуля Markov

## КОНТЕКСТНО-СВОБОДНЫЕ ГРАММАТИКИ

## О писание алгоритма

Алгоритм построен на описании строгих правил построения предложений или комбинаций. Например, рассмотрим правила построения палиндрома.

1. "S": (["(", "E", ")"], ["[", "E", "]"]),
2. "E": ("S","")

Квадратные скобки означают строгую последовательность действий, а круглые – случайную. Т.е. в правиле «S» выбирается любая из 2 скобок, внутри которых строго определен порядок действий. В правиле «E» случайно выбирается один из 2 вариантов: рекурсивно уйти в «S» или закончить построение палиндрома.

Сам алгоритм реализован следующим образом. Сначала происходит проверка на структуру данных. Если список первой структуры, то мы выбираем случайное правило и уходим в рекурсию, если другая структура данных, то так же, как и в первом случае уходим в рекурсию, но уже строго по порядку правил и соединяем результат в конце. Выход осуществляется при помощи пустого правила. Выбор структур данных не так важен, главное, чтобы сами структуры были различными. В нашем случае мы использовали список и кортеж. Более подробное описание алгоритма можно найти ниже.

## Кодирование алгоритма

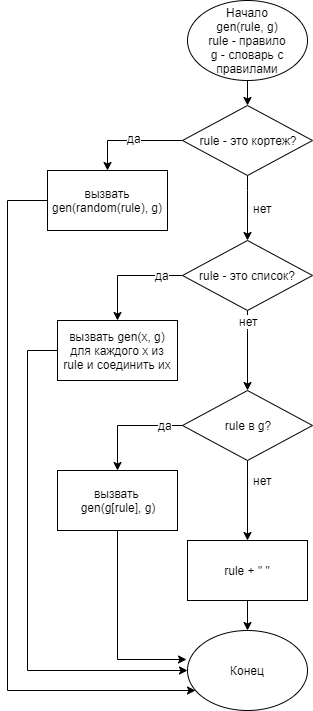


Рисунок 10 – Алгоритм контекстно-свободных грамматик

## Тестирование алгоритма

Тестирование алгоритма не показало особого улучшения качества текста, поэтому было принято решение убрать из проекта контекстно-свободные грамматики, а вместо них использовать марковские цепи второго порядка, так как тестирование марковских цепей второго порядка на крупных массивах текста показало значительное улучшение качества получаемых предложений.

## ЗАКЛЮЧЕНИЕ

Цель курсовой работы – генерация текста при помощи нескольких подходов. Курсовая работа была выполнена в полном объеме. Были реализованы марковские цепи 1 порядка, а также контекстно-свободные грамматики. Однако при тестирование алгоритма были получены неудовлетворяющие результаты, и было принято заменить контекстно-свободные грамматики на марковские цепи 2 порядка.

Данная курсовая работа помогла в изучении необходимого материала для программистов, и знания, полученные в результате реализации, используются в практических задачах.

Поставленная задача была реализована средствами языка Python и по стандартам PEP8.

## СПИСОК ИСПОЛЬЗУЕМЫХ РЕСУРСОВ

1. ИТМО – Марковские цепи: [Электронный ресурс]. 2018. URL: https://neerc.ifmo.ru/wiki/index.php?title=%D0%9C%D0%B0%D1%80%D0%BA%D0%BE%D0%B2%D1%81%D0%BA%D0%B0%D1%8F\_%D1%86%D0%B5%D0%BF%D1%8C (Дата обращения: 12.10.2019).
2. ProgLib – Марковские цепи: [Электронный ресурс]. 2018. URL: https://proglib.io/p/markov-chain/ (Дата обращения: 19.10.2019).
3. TProger – Марковские цепи: [Электронный ресурс]. 2018. URL: https://tproger.ru/translations/markov-chains/ (Дата обращения 19.10.2019)
4. Python Documentation: [Электронный ресурс]. URL: https://www.python.org/doc/ (Дата обращения 14.11.2019)
5. PEP8: [Электронный ресурс]. URL: https://www.python.org/dev/peps/pep-0008/ (Дата обращения 14.11.2019)
6. Методические указания по курсового проекта. – М.: РТУ МИРЭА, 2019. – 45 с.;
7. О введении в действие Инструкции по организации и проведению курсового проектирования. – М.: РТУ МИРЭА, Приказ №1325 от 05.10.2018. – 17 с..

## ПРИЛОЖЕНИЕ

## main.py

1. **from** markov **import** Markov
2. **import** os
3. **import** random

6. intro = []
7. main\_part = []

10. **def** normalize(file\_name):
11. f = open(file\_name, 'r')
12. result = f.read()
13. result = result.replace('\n', '').replace(';', ',').replace('"', '')
14. **while** result.find('  ') != -1:
15. result = result.replace('  ', ' ')
16. f.close()
17. **return** result

20. **def** save\_long\_file(text, sample, file\_name="examples.txt"):
21. f = open(file\_name, 'a')
22. f.write('-----------'+sample+'-------------\n')
23. f.write(text+'\n')
24. f.write('------------end----------------\n')
25. f.close()

28. **def** save\_text(text, file\_name="output\_text.txt"):
29. f = open(file\_name, 'w')
30. f.write(text)
31. f.close()

34. # Для более веселых текстов создаем несколько примеров
35. **def** gen\_markov(lst, size=100, file\_name="sample1.txt"):
36. markov = Markov()
37. mark = markov.train\_model(normalize(os.getcwd() + "\\samples\\" + file\_name))
38. **for** i **in** range(5):
39. lst.append(markov.generate\_sentence(size, mark))
40. **return** lst

43. **def** gen\_markov\_higher(lst, size=100, file\_name="sample1.txt"):
44. markov = Markov()
45. mark = markov.train\_model\_higher(normalize(os.getcwd() + "\\samples\\" + file\_name))
46. **for** i **in** range(30):
47. lst.append(markov.generate\_sentence(size, mark))
48. **return** lst

51. gen\_markov(intro, 10, "intro.txt")
52. gen\_markov\_higher(main\_part, 100, "sample5\_normalized.txt")
54. text = (random.choice(intro)+random.choice(main\_part)).replace(':.', '.').replace('.,', ',').replace('.-', '-').replace('..', '.').replace(',.', '.').replace(' . ', '. ')
55. save\_long\_file(text, "Хаос")
56. save\_text(text)

## new\_dict.py

1. **import** random

4. **class** MarkDict(dict):
5. **def** \_\_init\_\_(self, old\_dict=None):
6. super(MarkDict, self).\_\_init\_\_()
7. self.types = 0
8. self.tokens = 0
9. **if** old\_dict:
10. self.update(old\_dict)
12. **def** update(self, old\_dict):
13. **for** item **in** old\_dict:
14. **if** item **in** self:
15. self[item] += 1
16. self.tokens += 1
17. **else**:
18. self[item] = 1
19. self.types += 1
20. self.tokens += 1
22. **def** count(self, item):
23. **if** item **in** self:
24. **return** self[item]
25. **return** 0
27. **def** return\_weighted\_random\_word(self):
28. random\_int = random.randint(0, self.tokens-1)
29. index = 0
30. list\_keys = list(self.keys())
31. **for** i **in** range(0, self.types):
32. index += self[list\_keys[i]]
33. **if** index > random\_int:
34. **return** list\_keys[i]

## markov.py

1. **import** random
2. **from** new\_dict **import** MarkDict

5. # Класс нужен для создания независимых объектов в main
6. **class** Markov:
7. # Создаем простую модель с окном в 1 слово. Используется для вводной части
8. **def** train\_model(self, data):
9. markov\_model = MarkDict()
10. data = data.split()
11. **for** i **in** range(0, len(data)-1):
12. **if** data[i] **in** markov\_model:
13. markov\_model[data[i]].update([data[i+1]])
14. **else**:
15. markov\_model[data[i]] = MarkDict([data[i + 1]])
16. **return** markov\_model
18. **def** generate\_start(self, model):
19. **if** '.' **in** model.keys():
20. **return** model['.'].return\_weighted\_random\_word()
21. **return** random.choice(list(model.keys()))
23. # Здесь мы находим первое слово, а дальше по принципу марковской цепи находим последующие
24. **def** generate\_sentence(self, length, markov\_model):
25. current\_word = Markov.generate\_start(self, markov\_model)
26. sentence = [current\_word]
27. **for** i **in** range(0, length):
28. # print(markov\_model[current\_word])
29. current\_dictogram = markov\_model[current\_word]
30. random\_weighted\_word = current\_dictogram.return\_weighted\_random\_word()
31. current\_word = random\_weighted\_word
32. sentence.append(current\_word)
33. sentence[0] = sentence[0].capitalize()
34. **return** ' '.join(sentence) + '.'
36. # Создание окна в 2 слова. Использую только для главной части (т.к. нужен большой объем словаря)
37. # Мы просто привязываем к слову 2 последующих слова, а дальше тот же алгоритм
38. **def** train\_model\_higher(self, data):
39. data = data.split(' ')
40. markov\_model = dict()
41. k = 1
43. **for** i **in** range(0, len(data)//2-1):
44. data[i] = data[k] + ' ' + data[k+1]
45. k += 2
47. **for** i **in** range(0, len(data)-1):
48. **if** data[i] **in** markov\_model:
50. markov\_model[data[i]].update([data[i+1]])
51. **else**:
52. markov\_model[data[i]] = MarkDict([data[i+1]])
53. **return** markov\_model

## test.py

1. **import** unittest
2. **import** os
3. **from** markov **import** Markov
4. **from** new\_dict **import** MarkDict
5. standard\_first\_level = MarkDict({'wrote': {'this': 1}, 'it': {'.': 1, 'will': 1}, 'good': {'today': 1},
6. 'Hello': {'my': 1}, 'work': {'.': 1}, 'now': {'I': 1}, 'used': {'to': 1},
7. 'knows': {'how': 1}, 'friend': {'.': 1}, 'So': {'I': 1}, "Let's": {'go.': 1},
8. '.': {"Let's": 1, 'So': 1, 'I': 2}, 'will': {'work': 1},
9. 'to': {'write': 1, 'test': 1}, 'nobody': {'knows': 1}, 'write': {'this': 1},
10. 'feeling': {'good': 1}, 'I': {'wrote': 1, 'need': 1, 'used': 1, 'would': 1, 'am': 1},
11. 'and': {'nobody': 1, 'now': 1}, 'need': {'to': 1}, 'my': {'friend': 1},
12. 'generator': {'and': 1}, 'today': {'.': 1}, 'would': {'check': 1},
13. 'check': {'this': 1}, 'how': {'it': 1}, 'this': {'and': 1, 'text': 1, 'generator': 1},
14. 'test': {'it': 1}, 'text': {'cause': 1}, 'cause': {'I': 1}, 'am': {'feeling': 1}})
16. standard\_second\_level = MarkDict({'wrote': {'this': 1}, 'test it': {". Let's": 1}, 'need to': {'test it': 1},
17. 'good today': {'. I': 1}, 'cause': {'I': 1}, 'work': {'.': 1},
18. 'this and': {'nobody knows': 1}, 'would check': {'this and': 1},
19. 'knows': {'how': 1}, 'go.': {'': 1}, 'I wrote': {'this text': 1},
20. 'So': {'I': 1}, 'am feeling': {'good today': 1}, '.': {"Let's": 1, 'So': 1},
21. 'will': {'work': 1}, 'my friend': {'. I': 1}, 'nobody': {'knows': 1},
22. 'cause I': {'need to': 1}, "Let's": {'go.': 1}, 'used to': {'write this': 1},
23. 'and': {'nobody': 1}, 'test': {'it': 1}, 'I': {'wrote': 1, 'need': 1},
24. 'now I': {'would check': 1}, 'nobody knows': {'how it': 1}, ". Let's": {'this': 1},
25. 'it': {'.': 1, 'will': 1}, 'to': {'test': 1}, 'how it': {'will work': 1},
26. 'this text': {'cause I': 1}, 'generator and': {'now I': 1},
27. 'need': {'to': 1}, '. I': {'am feeling': 1, 'used to': 1}, 'how': {'it': 1},
28. 'this': {'and': 1, 'text': 1}, 'write this': {'generator and': 1},
29. 'text': {'cause': 1}, 'will work': {'. So': 1}, '. So': {'I wrote': 1}})

32. **class** TestTrain(unittest.TestCase):
33. **def** setUp(self):
34. self.mark = Markov()
35. self.model = self.mark.train\_model(open(os.getcwd() + "\\samples\\" + "TEST.txt", "r").read())
36. self.model\_higher = self.mark.train\_model\_higher(open(os.getcwd() + "\\samples\\" + "TEST.txt", "r").read())
38. **def** test\_len(self):
39. self.assertTrue(len(self.model) != 0, "Len must be >=0 ")
41. **def** test\_model(self):
42. self.assertEquals(len(self.model), len(standard\_first\_level))
44. **def** test\_model\_higher(self):
45. self.assertEquals(len(self.model\_higher), len(standard\_second\_level))
47. **def** test\_random\_begin(self):
48. self.assertTrue(self.mark.generate\_start(self.model) **in** self.model['.'], "Error in generate\_start")

51. **if** \_\_name\_\_ == '\_\_main\_\_':
52. unittest.main()

## test\_struct.py

1. # file for testing structs
2. # every struct will be checked here
4. #from n\_tree import Tree
5. **from** markov **import** Markov
6. **import** os
7. #from own\_markov import markov as mrk

10. **def** test\_tree():
11. tree = Tree(["hello"])
12. tree.append\_el(["my friend"], "A")
13. tree.append\_el("honey", "A")
14. tree.append\_el(["atmta", "vsyachena"], "B")
15. tree.append\_el("magicmag", "B")
16. **print**(tree[None], tree["A"])
17. **print**(tree)

20. **def** normalize(file\_name):
21. f = open(file\_name, 'r')
22. result = f.read()
23. #result = result.replace('\n', '')
24. **while** result.find('  ') != -1:
25. result.replace('  ', ' ')
27. **return** result

30. **def** save\_long\_file(text, sample, file\_name="examples.txt"):
31. f = open(file\_name, 'a')
32. f.write('-----------'+sample+'-------------\n')
33. f.write(text+'\n')
34. f.write('------------end----------------\n')
35. f.close()

38. **def** save\_text(text, file\_name="output\_text.txt"):
39. f = open(file\_name, 'w')
40. f.write(text)
41. f.close()

44. **def** test\_markov(size=100, file\_name="sample1.txt"):
45. markov = Markov()
46. mark = markov.train\_model(normalize(os.getcwd() + "\\samples\\" + file\_name))
47. text = markov.generate\_sentence(size, mark)
48. save\_text(text)
49. save\_long\_file(text, file\_name)
50. **print**(text)

53. **def** test\_markov\_with\_higher\_level(size=100, file\_name="sample1.txt"):
54. markov = Markov()
55. mark = markov.train\_model\_higher(normalize(os.getcwd() + "\\samples\\" + file\_name))
56. text = markov.generate\_sentence(size, mark)
57. save\_text(text)
58. save\_long\_file(text, file\_name)
59. **print**(text)
61. #test\_tree()
62. #test\_markov(50, "intro.txt")



67. **import** nltk
69. file = open('C:\\Users\\mag13\\Desktop\\ЛАБЫ\\CиАОД\\Курсовая\\markov\_gen\\samples\\sample6.txt', 'rt')
70. sent = nltk.sent\_tokenize(file.read())
71. file.close()
72. **print**(sent[0][:-1])
73. string = ""
74. **for** s **in** sent:
75. string += s[:-1] + " . "
76. save\_text(string, 'sample6\_normalized.txt')
78. **print**("heelo . dsdd . ".split(" "))
80. test\_markov\_with\_higher\_level(100, "sample6\_normalized.txt")