**Author**: Krasnov Aleksandr

**Motivation:**

I decided to pick a project with Markov Chain as it was very briefly discussed in class and I wanted to know more about this topic and broaden my knowledge. I am expanding the code which was given in A07: Markov Chain.

**Purpose:**

I expanded the code for using a Markov Chain for different files (works in txt, html). The main idea is to see how the Markov Chain works.

**Audience:**

The class of CSC236 which will be studying Markov Chain and particularly do A07: Markov Chain assignment in the future. In files, there are comments that clearly state what can be improved.

**Instructions:**

Open the file new\_file\_test\_Markov.py and run it if you want to see how your text can be changed. You need to put either your own txt file in the same folder, or you can create a random file while running a program. If you are running on Linux, run:

$ git clone ….

$ python new\_file\_test\_Markov.py

**Design:**

// build data structures

w[1]...w[n] = No\_Word; // start of text

while( text\_exists ) {

w[n+1] = next\_word;

add w[n+1] to the group of suffixes of w[1]...w[n];

w[1]...w[n] = w[2]...w[n+1]; // add iterator incrementally

}

add No\_Word to the group of suffixes of w[1]...w[n]; // end of text

// output new text

w[1]...w[n] = No\_Word;

w[n+1] = the one suffix associated with a prefix that is all No\_Words;

while( w[n+1] != No\_Word ) {

output w[n+1];

w[1]...w[n] = w[2]...w[n+1];

w[n+1] = a randomly chosen suffix of w[1]...w[n];

/\* note: The random choice should be weighted based on the

multiplicity of each suffix. \*/

}

**Enhancement:** used a trigram instead of bigram to make the algorithm more effective. Expanded catching errors so the program will never crash with an error. If an error happens, the program will stop outputting the name of the error for debugging purposes. Moreover, it can generate a random number of words from the model “markov” and work with a randomly created txt file as well. Works with websites as well (need a lot of improvement as the code works, but a better design needed). Also, I created another way of Makrov chain in the folder diff\_implementation

**Functionality:**

1. Class Markov

A simple trigram Markov Model. The current state is a sequence of three words seen most recently. Initially, the state is (None, None, None)

1. \_\_init\_\_

Created an empty Markov model with initial state (None, None, None)

1. Add

Adds word as a possible following word for current state of the Markov model and sets state to incorporate word as most recently seen

1. randomNext

returns a random choice from among all possible choices of next words, given the current state, and updates the state to reflect the word produced

1. \_transition

Sets the state based on the next word

1. makeWordModel

creates a Markov model from the words in the file with filename

1. generateWordChain

generates chosen amount of words on output from the model “Markov”

1. redo

asks a user to redo the implementation of the text

**Files:**

Project \_ \_ .idea

| \_ \_ Krasnov\_csc236\_final\_project.iml

| \_ \_ misc.xml

| \_ \_ modules.xml

| \_ \_ workspace.xml

\_ \_pycache\_

| \_ \_ given\_file\_Markov.cpython-36.pyc  
 | \_ \_ new\_file\_Markov.cpython-36.pyc

\_ \_ diff\_implementation

| \_ \_ implement.py

\_ \_website

| \_ .idea

| \_ \_ misc.xml

| \_ \_ modules.xml

| \_ \_ website.iml

| \_ \_ workspace.xml

| \_ \_pycache\_

| \_ \_ Markov.cpython-36.pyc

| \_ \_ Markov.py

| \_ \_ website.html

| \_ \_ website.py

| \_ \_ website.txt

| \_ \_ website\_test.html

| \_ \_ website\_test.txt

* big.txt
* given\_file\_Markov.py
* given\_file\_test\_Markov.py
* krasnov\_csc236FP.docx
* new\_file\_Markov.py
* new\_file\_test\_Markov.py
* random\_file.txt
* test.txt

**Utilized Data Structures:**

Used dictionary for the model as it states the list of words and its frequency. Used list as a stack to create a chain model. Everything could have been implemented as lists but using stack makes more sense for the chain model as we really do not care what happens with words which have already been appended in the model, as they have been generated and initialized at the point of moving to the next possible word in the model.

**Big O Analysis:**

Lines 78 – 105 with comments around that part in new\_file\_test\_Markov.py. It has a linear complexity as we have one iteration.

**Resources:**

IDE – PyCharm – community edition

Language – Python 3.6

Libraries – RandomWords, Random, urlib, re

**Challenges:**

Make a random file with random text was a difficult task as I did not want to use API of any platform but rather a library. Obtain website’s html was a challenge as obtaining html itself isn’t as hard as it could be, but making sure that all css styles are downloaded as well was difficult.

**Testing:**

1. If a user inputs a txt file that exists there are two possible ways that happen:
2. Text in txt file in UTF-8.  
   If that is the case, it will execute successfully.
3. Text in txt file is not in UTF-8  
   If that is the case, the program will catch an error.
4. If a user inputs a txt file that does not exist, the program will ask a user to input something again. There is an alternative option to create a totally random file.
5. If a file is too big (more than 1GB of data) which is hard to find, but the program will still run. However, it will take close to 15 minutes of time. Also, if a user decides to generate over 1 billion random words, the program will take approximately 15 minutes to compile.

**Errors:**

1. A possible modification lies within a file for website implementation where a better algorithm should be developed.
2. A possible modification in designing a different implementation lies within making it more applicable to other documents rather than txt.
3. A user creates a list of random words which are only nouns. It will still work fine as you may take a look at random.txt file and output which would be two different things. However, a better and more convincing result will be if verbs, adverbs, adjectives would be added as well, so the text would look more human and readable.

**Summary:**

The design was to use trigram with three words instead of given two. Moreover, most of errors have been identified and caught by the program.

Implementation of error handling took few hours to identify what kind of errors are common. Implementation of new algorithm of Markov Chain in a folder diff\_implementation was complicated and took additional two hours. Developing the given code into a better one took no more than three hours.

**Comments:**

That would be more interesting to spend more time on that topic as it seems like a very good topic to explain algorithms.