

**Wyższa Szkoła Informatyki i Zarządzania z siedzibą w Rzeszowie**

**Kolegium: Informatyka Stosowana  
Kierunek: Informatyka**

Dokumentacja projektowa

Przedmiot Python: Internet Rzeczy - zaawansowany

«Generator tekstu na podstawie metody Łańcuchów Markowa»

*Prowadzący: Wykonawca:*

inż. Mateusz Smok Maksym Miroshnychenko, w60054

Spis treści

[Definicja i cel projektu 3](#_Toc73125791)

[Uzasadnienie biznesowe 3](#_Toc73125792)

[Założenia projektu 3](#_Toc73125793)

[Szczegółowy opis projektu 3](#_Toc73125794)

[Opis Algorytmu Łańcuchów Markowa 3](#_Toc73125795)

[Opis kodu źródłowego 4](#_Toc73125796)

[Sposób użycia 7](#_Toc73125797)

[Podsumowanie 7](#_Toc73125798)

[Literatura 7](#_Toc73125799)

# Definicja i cel projektu

Celem projektu jest stworzenie oprogramowania, które wykorzystując mechanizmy grafów będzie w stanie samodzielnie generować teksty, podobne do tekstu przykładowego. Generacja tekstu będzie polegała na zaimplementowanym algorytmie Łańcuchów Markowa. Pierwszym krokiem będzie wczytanie z pliku tekstu przykładowego lub folderu z umieszonymi w nim plikami tekstowymi (.txt), na podstawie których algorytm wygeneruje ciąg kombinacji kluczowych słów i najczęściej używanych słów, które występują po słowach kluczowych. Następnie, użytkownik będzie miał możliwość wprowadzenia ilości słów w tekście, który zostanie wygenerowany przez algorytm. Wynik zostanie wyświetlony w konsoli.

# Uzasadnienie biznesowe

Oprogramowanie może być wykorzystywane jako narzędzie do badań naukowych prowadzonych na uczelni lub w innych instytucjach, lub jako narzędzie go generacji tematycznych śmiesznych tekstów (np. nazwy filmów). Jako, że interfejs będzie bardzo intuicyjny dla użytkownika, program będzie mógł być wykorzystywany jako prezentacja działania algorytmu osobom, które mają niewielką wiedzę z zakresu grafów i sztucznej inteligencji.

# Założenia projektu

1. Oprogramowanie będzie działać na dowolnym systemie operacyjnym Windows/Linux/Mac OS/Android.
2. Algorytm ma być zrealizowany na podstawie Klas w języku Python
3. Program musi umieć wczytywać dowolną ilość plików tekstowych.

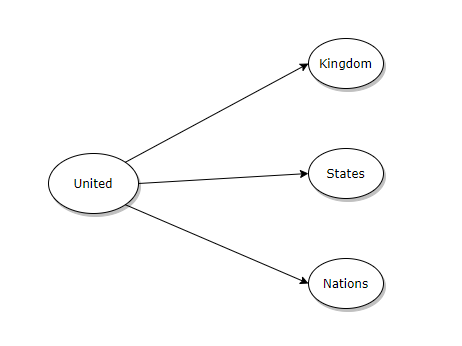
# Szczegółowy opis projektu Opis Algorytmu Łańcuchów Markowa

Łańcuch Markowa, to grafowy model opisujący przejście między stanami z uwzględnieniem różnego prawdopodobieństwa. Najczęściej mamy z nim do czynienia podczas obsługiwania inteligentnej klawiatury na urządzeniach mobilnych, bo to właśnie łańcuch odpowiada za podpowiadanie słów, które użytkownik może zechcieć użyć podczas pisania tekstu.

Działanie algorytmu:

1. Weź pierwsze słowo z listy i oznacz je jako poprzednie
2. Weź następne słowo z listy i dodaj je jako powiązane od poprzedniego
3. Jeśli poprzednie posiada już powiązanie do słowa, to zwiększ licznik połączeń o 1
4. W przeciwnym wypadku ustaw licznik połączeń na 1
5. Zaznacz aktualne słowo jako poprzednie
6. Idź do pkt. 2, jeśli na liście znajdują się jeszcze jakieś nieprzetworzone słowa

Powyższy algorytm utworzy nam graf wyglądający mniej więcej tak:



Rys. 1 Graf połączonych słów

Lub w postaci obiektu Dictionary, języka Python:

{”United” : [”Kingdom”, ”States”, ”Nations”]}

Ważnym jest zauważyć że w wygenerowanym grafie połączone słowa mają tak zwane wagi, czyli wartości częstotliwościowe, które reprezentują częstotliwość spotykania słowa drugiego (i+1). W przypadku generacji, te wagi mogą reprezentować wiarygodność procentową (jest niezbędne przetwarzanie wszystkich wag do obszaru 0..1 metodą SoftMax). Ale zdaniem autora, jest to zbędne, ponieważ ten algorytm jest bardzo prosty i biorąc pod uwagę sposób wylosowania słowa [i+1] wystarczy trzymać w pamięci operacyjnej strukturę grafu w postaci słowo\_kluczowe: [słowa\_połączone, …]. Takim sposobem słowa, które częściej się spotykają będzą mieli więcej swoich kopii w ciągu słów połączonych, co spowoduje większą wiarygodność ich wylosowania metodą np.random.choice(array[]), co i było zaimplementowane w kodzie.

Generacja tekstu w programie będzie polegała na:

1. Wylosowaniu pierwszego słowa z listy wczytanych słów
2. Wylosowaniu jednego słowa z listy powiązanych do poprzedniego słowa słów i przypisaniu do zmiennej tekstu wejściowego.
3. Idź do punktu 2, dopóki ilość słów w zmiennej tekstu wyjściowego jest mniejsza niż ustawiona liczba wartości pola danych klasy n\_words.

## Opis kodu źródłowego

Ogólnym zadaniem programu jest generacja tekstu na podstawie wprowadzonych danych (wczytanego pliku tekstowego). W naszym przypadku była stworzona klasa **Markov**, która będzie wczytywała tekst z plików, generowała tekst wyjściowy i przechowywała go w wyglądzie danych zapisanych do pola klasy.

Klasa Markov będzie zawierała następujące metody:

\_\_init\_\_(path) – wczytuje plik(pliki) tekstowy do listy słów i buduje ciąg grafów połączonych słów

Generate(n\_words) – Generuje tekst metodą Łańcuchów Markowa. Wielkość tekstu zależy od wprowadzonej liczby n\_words.

\_make\_pairs(words\_array) – Zwraca generator funkcyjny, który przetwarza wejściowy ciąg do postaci par (słowo[i], słowo[i+1]) i zwraca w postaci ciągu par słów. Jest to funkcja prywatna, klasy Markov.

class Markov():

    text = ''

    generated\_text = ''

    words\_dict = {}

    words\_array = []

    def \_\_init\_\_(self, path):

        if path.endswith('.txt'):

            with open(path, encoding='utf8') as f:

                self.text = f.read()

        elif path.endswith('/'):

            text\_files = [path + text\_file for text\_file in os.listdir(path) if text\_file.endswith('.txt')]

            for text\_file in text\_files:

                with open(text\_file, encoding='utf8') as f:

                    print(text\_file)

                    self.text += "\n " + f.read()

        else:

            raise Exception('Allowed only \*.txt files')

        self.words\_array = self.text.split()

        self.pairs = self.\_make\_pairs(self.words\_array)

        for word\_key, word\_value in self.pairs:

            if word\_key in self.words\_dict.keys():

                self.words\_dict[word\_key].append(word\_value)

            else:

                self.words\_dict[word\_key] = [word\_value]

    def Generate(self, n\_words = 100):

        first\_word = np.random.choice(self.words\_array)

        while first\_word.islower():

            first\_word = np.random.choice(self.words\_array)

        chain = [first\_word]

        for i in range(n\_words):

            chain.append(np.random.choice(self.words\_dict[chain[-1]]))

        self.generated\_text = ' '.join(chain)

        self.generated\_text = self.generated\_text.replace('.','.\n')

        return self.generated\_text

    def \_make\_pairs(self, words\_array):

        for i in range(len(words\_array)-1):

            yield (words\_array[i], words\_array[i+1])

## Sposób użycia

Oprogramowanie jest zrealizowane w postaci biblioteki, więc żeby wypróbować algorytm, trzeba utworzyć nowy obiekt klasy Markov() z argumentem path (to może być lub adres pliku tekstowego, lub foldera z kilkoma plikami tekstowymi). Po tym trzeba uruchomić funkcję Generate() z podaną wartością n\_words, jeżeli ta wartość nie będzie podana, zmienna automatycznie przypisze się do wartości 100. Po zakończeniu, ta funkcja zwróci wygenerowany tekst do zmiennej i wystarczy ją tylko wypisać poprzez funkcję print(text)

markov = Markov('texts/Sokrates.txt')

generated\_text = markov.Generate(n\_words = 200)

print(generated\_text)

Wynikiem będzie wygenerowany tekst:

python.exe /Markov\_python/v2.py

Stosował ją tylko wobec niektórych osób, w końcu do tezy absurdalnej lub wydobyć z twierdzeniem pierwotnym (aporii).

Była metodą negatywną, która wskazywała na błędność ich teoretycznie.

Metoda elenktyczna polegała na opinii Arystotelesa, który sam nic nie wie i dążył do określenia znaczenia pojęć, to nie można go uznać za błędny, pomimo faktu, że Sokrates stosował szereg metod nauczania, dyskusji i prawdziwej wiedzy prostaczka (eironeia, ironia, postawa „udawanej skromności”), który sam nic nie wie i argumentacji, łącznie nazywanych metodą pozytywną.

# Podsumowanie

Pod czas pracy nad projektem powstała biblioteka do generacji tekstów na podstawie Łańcuchów Markowa. Klasa jest niezawodna, zaprojektowana według cech programowania obiektowego i z wykorzystaniem funkcyjnych generatorów które oszczędzają pamięć operacyjna. Algorytm może wczytać jak i pojedynczy plik, tak i wszystkie pliki z folderu.

# Literatura

* <https://medium.com/@luc1ph3r/tworzenie-generatora-tekstu-za-pomoc%C4%85-%C5%82a%C5%84cucha-markowa-2bd2ace60b3a>
* <https://pl.wikipedia.org/wiki/%C5%81a%C5%84cuch_Markowa>
* <https://pl.wikipedia.org/wiki/W%C5%82asno%C5%9B%C4%87_Markowa>