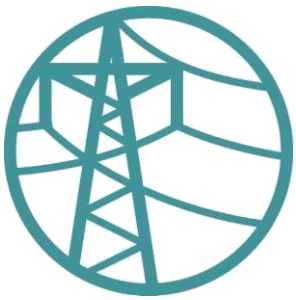

WYDZIAŁ
ELEKTROTECHNIKI
I INFORMATYKI
POLITECHNIKI RZESZOWSKIEJ

Autorzy pracy : Tomasz Cioch, Tomasz Jachna, Kamil Kłos

Temat : Aplikacja antyplagiatowa

Projekt z przedmiotu „Programowanie w języku Python”

Prowadzący
mgr inż. Patryk Organiściak

Rzeszów, 2023

Spis treści

1. Wstęp teoretyczny

1.1 Opis problemu

Celem projektu było stworzeniu programu, który potrafiłby wykryć czy dany tekst jest plagiatem. Wkraczamy obecnie w dobę AI, które potrafią wygenerować niepowtarzalny tekst, ale jest to dopiero początek tej technologii i wciąż bardzo wiele prac opiera się na fragmentach bezpośrednio skopiowanych z Internetu. I nie jest realne całkowite uniknięcie powtórzenia fraz, które ktoś już kiedyś wykorzystał i opublikował, ale narzędzie to pozwoli oszacować jaki procent pracy już pojawił się online.

1.2 Wykrywanie plagiatu

Profesjonalne programy służące do wykrywania plagiatu operują na bardziej złożonych algorytmach, takich jak szacowanie lokalnego lub globalnego podobieństwa, czy analiza powtórzeń określeń. Poniższa praca stosuje naiwny model wyszukiwania konkretnych zdań lub ciągów znaków, co pozwala sprawdzić czy tekst nie został skopiowany w większych kawałkach. Nie pozwoli wykryć zmian w szeregu zdania lub zastąpienia jego fragmentów synonimami. Jest to raczej prosty proof-of-concept prezentujący z jaką łatwością można stworzyć bardzo podstawową aplikację do kontroli tekstu.

1.3 Wykorzystane technologie

Program napisany został w IDE PyCharm, a wyszukiwanie tekstu online wykorzystuje wyszukiwarkę Bing. Początkowo używana była Google, ale blokuje ona IP po wykonaniu kilku wyszukiwań. Alternatywną opcją było wykorzystanie VPNa do ominięcia blokady, ale dla ułatwienia wykonania projektu zdecydowano się na zmianę wyszukiwarki. W projekcie użyto szeregu bibliotek:

1.3.1 Biblioteki

Os – biblioteka służąca do operacji na plikach

Sys – biblioteka służąca do obsługi argumentów wiersza poleceń

Re - biblioteka służąca do wykonywania operacji na wyrażeniach regularnych

Textract - biblioteka służąca do ekstrakcji tekstu z plików .docx i .pdf

Requests - biblioteka służąca do wysyłania zapytań HTTP

Nltk.tokenize - biblioteka służąca do dzielenia tekstu na zdania

Nltk - biblioteka służąca do obsługi tokenizacji tekstu

Typing - biblioteka służąca do adnotacji typów

Colorama - biblioteka służąca do formatowania tekstu w konsoli

2. Realizacja projektu

2.1 Założenia

Do obsługi programu potrzebny jest zestaw bibliotek opisanych w podpunkcie 1.3.1, połączenie z Internetem i plik z tekstem w jednym z obsługiwanych formatów (pdf, docx, txt). Aby wywołać program należy użyć polecenia:

python NAZWA.PROGRAMU.py test.txt

, gdzie „test.txt" to ścieżka do pliku który ma być sprawdzony.

2.2 Działanie programu

Program działa w dwóch trybach. Najpierw dzieli podany tekst na części, w pierwszym trybie dzieląc go na zdania, a w drugim na ciąg znaków. Domyślnie ciąg znaków ma długość 10 i jest to wartość którą można łatwo zmodyfikować w kodzie. Możliwość sparametryzowania tej wartości jest opcją rozwoju aplikacji w przyszłości. Pierwszym trybem jest więc podzielenie tekstu na zdania, drugim podzielenie go na ciągi znaków. Następnie każdy z tych kawałków jest wprowadzany do wyszukiwarki Bing. Jeśli jest on znaleziony, to traktuje się go jako plagiat. Po sprawdzeniu wszystkich fragmentów tekstu program podaje procentową wartość plagiatu w obu trybach – podzielenia tekstu na zdania i podzielenia tekstu na ciągi liter. Następnie wyświetla tekst z zaznaczonym na czerwono fragmentem, który uznany został za plagiat.

2.3 Funkcje programu

Aplikacja składa się z szeregu względnie prostych funkcji,. Których funkcjonalność opisano poniżej

2.3.1 **read_file** – funkcja służąca do odczytu treści pliku tekstowego w formacie PDF, DOCX lub TXT. W przypadku plików TXT wystarczy otworzyć plik i odczytać go, w przypadku pozostałych formatów użyta jest biblioteka „textract”

```
def read_file(file_path: str) -> str:
    _, file_extension = os.path.splitext(file_path)
    if file_extension == '.txt':
        with open(file_path, encoding='utf-8-sig') as file:
            return file.read()
    elif file_extension in ['.docx', '.pdf']:
        return textract.process(file_path).decode('utf-8')
    else:
        raise ValueError("Unsupported file format")
```

2.3.2 **split_int_chunks** – funkcja służaca do podzielenia tekstu na mniejsze części, domyślnie
o długości 10 znaków. Jest używana do drugiego trybu sprawdzania plagiatu, gdzie w wyszukiwarce wpisujemy ciągi znaków.

```
def split_into_chunks(text: str, chunk_size: int = 10) -> List[str]:
    words = text.split()
    chunks = [' '.join(words[i:i + chunk_size]) for i in range(0, len(words), chunk_size)]
    return chunks
```

2.3.3 **split_into_sentences** – funkcja służaca do podzielenia tekstu na zdania. Jest używana do pierwszego trybu sprawdzania plagiatu, gdzie w wyszukiwarce wpisujemy pojedyncze zdania.

```
def split_into_sentences(text: str) -> List[str]:
    sentences = sent_tokenize(text)
    result = []
    for sentence in sentences:
        parts = re.split(r'"', sentence)
        for i, part in enumerate(parts):
            if i % 2 == 0:
                result.append(part.strip())
            else:
                result.append(f'"{part}"')
    return result
```

2.3.4 **bing_search_single_fragment** – funkcja służąca do wyszukiwania pojedynczych fragmentów tekstu w wyszukiwarce Bing. Zwraca krotkę zawierającą fragment oraz liczbę wyników wyszukiwania.

```
def bing_search_fragments(fragments: List[str]) -> List[str]:
    plagiarised_fragments = []
    with concurrent.futures.ThreadPoolExecutor() as executor:
        future_to_fragment = {executor.submit(bing_search_single_fragment, fragment): fragment for fragment in fragments}
        for future in concurrent.futures.as_completed(future_to_fragment):
            try:
                result = future.result()
                if result[1] > 0:
                    plagiarised_fragments.append(result[0])
            except Exception as e:
                print
    return plagiarised_fragments
```

2.3.5 **bing_search_fragments** – funkcja służąca do wyszukiwania wielu fragmentów. Używa wielowątkowości do szybszego przetwarzania i zwraca listę fragmentów, które zostały znalezione w wynikach wyszukiwania

```
def bing_search_single_fragment(fragment: str) -> Tuple[str, int]:
    headers = {"Ocp-Apim-Subscription-Key": "5c4a9fc00ad742b898a72848b7857fdb"}
    params = {
        "q": '"' + fragment + '"',
        "count": 10,
    }
    response = requests.get("https://api.bing.microsoft.com/v7.0/search", headers=headers, params=params)
    response.raise_for_status()
    search_results = response.json()
    if 'webPages' not in search_results:
        return fragment, 0
    return fragment, len(search_results['webPages']['value'])
```

2.3.6 **calculate_plagiarism** – funkcja służąca do obliczenia procentu splagiatowania w obu trybach – wyszukiwania pojedynczych zdań i ciągów znaków.

```
def calculate_plagiarism(wholeList: List[str], results: List[str]) -> Tuple[float, float]:
    plagiarized_fragments_count = len(results)
    plagiarism_percentage = (plagiarized_fragments_count / len(wholeList)) * 100

    plagiarized_characters_count = sum([len(fragment) for fragment in results])
    total_characters_count = sum([len(fragment) for fragment in wholeList])
    plagiarized_characters_percentage = (plagiarized_characters_count / total_characters_count) * 100

    return plagiarism_percentage, plagiarized_characters_percentage
```

2.3.7 **display_result** – funkcja służaca do wyświetlania wyników splagiatowania obu trybów w konsoli

```
def display_results(plagiarism_result: float, fragments: List[str], plagiarised_framents: List[str]):
    print(f"Procent splagiatowania (fragment): {plagiarism_result[0]:.2f}%")
    print(f"Procent splagiatowania (znaki): {plagiarism_result[1]:.2f}%")
```

2.3.8 **display_colored_text** – funkcja służąca do wyświetlenia sprawdzanego tekstu w konsoli z tekstem splagiatowanym wyróżnionym na czerwono

```
def display_colored_text(fragments: List[str], plagiarised_framents: List[str]):
    for fragment in fragments:
        if fragment in plagiarised_framents:
            print(Fore.RED + fragment, end=' ')
        else:
            print(Fore.RESET + fragment, end=' ')
    print()
```

2.3.9 **main** – funkcja wywołująca działanie programu

```
def main(file_path: str):
    text = read_file(file_path)
    fragments = split_into_sentences(text)
    plagiarised_framents = bing_search_fragments(fragments)
    plagiarism_result = calculate_plagiarism(fragments, plagiarised_framents)
    display_results(plagiarism_result, fragments, plagiarised_framents)
    display_colored_text(fragments, plagiarised_framents)

if len(sys.argv) < 2:
    print("Usage: python simple_antiplagiarism.py <file_path>")
    sys.exit(1)
main(sys.argv[1])
```

2.4 Wynik działania programu

```
(venv) PS C:\Users\Tomasz\PycharmProjects\pythonProject2> python .\main.py .\test.txt
[nltk_data] Downloading package punkt to
[nltk_data]     C:\Users\Tomasz\AppData\Roaming\nltk_data...
[nltk_data]   Package punkt is already up-to-date!
Procent splagiatowania (fragment): 77.78%
Procent splagiatowania (znaki): 75.04%
Państwo hetyckie długo zmagało się z Egiptem o dominację w Syrii. Historia tych zmagań sięga czasów PRZERWANIE faraona Totmesa III. Wraz z rozpoczęciem reformy religijnej faraon
o pozwoliło Hetytom na aktywność w tym regionie. Po objęciu rządów w Egipcie przez przedstawicieli XIX dynastii, zainteresowanie tego państwa sprawami syryjskimi znów wzrosło.
I w tym regionie pozwoliły odbudować dawny prestiż Egiptu. Wkrótce też armia egipska zaatakowała Kadesz i Amurru[3]. Kraje te były wasalami hetyckimi. Taka agresja nie mogła poz
ydwoma krajami doszło
```

3. Podsumowanie

Program spełnia swoje założenia. Dzieli tekst na zadane fragmenty i wyszukuje je w wyszukiwarce, określając procent plagiatu w tekście. Jest to bardzo naiwny algorytm, który łatwo oszukać i tu należy szukać największych możliwości rozwoju projektu. Należałoby zaimplementować bardziej skomplikowane, trudniejsze do oszukania metody. Kolejnym krokiem mogłoby być dodanie kolejnego parametru do aplikacji, pozwalającego określić długość ciągu znaków używanego do wyszukiwania. Dobrym pomysłem byłoby też stworzenie GIU pozwalającego skopiować tekst do sprawdzenia zamiast przechowywać go w pliku, i wreszcie dodać możliwość wyszukiwania również obrazów.