

POLITECHNIKA KRAKOWSKA

Jak napisać spell corrector?

Przetwarzanie języka naturalnego

Krystian Korbecki

Paweł Krzyściak

Spis treści

1. Wstęp	1
1.1 Cel i zakres projektu	1
1.2 Metodyka i narzędzia	1
3. Opis algorytmu	1
4. Model Językowy ByT5-text-correction	3
5. Implementacja	3
6. Testy programu	11
7. Podsumowanie	11
Bibliografia	13

Czy kiedykolwiek zdarzyło Ci się pisać ważny dokument, e-mail do szefa, czy nawet zwykłą wiadomość tekstową i zastanawiać się, czy aby na pewno wszystkie słowa są poprawnie napisane? Błędy ortograficzne mogą zdarzyć się każdemu, niezależnie od tego, jak dobrym jesteśmy pisarzem. Na szczęście technologia oferuje nam narzędzia, które mogą nas wspomóc w walce z literówkami i niepoprawną pisownią. Jednym z takich narzędzi jest "spell corrector", czyli korektor pisowni. Dzięki zaawansowanym algorytmom i analizie językowej, spell corrector nie tylko wykrywa błędy, ale także sugeruje poprawne formy wyrazów, czyniąc naszą komunikację bardziej profesjonalną i zrozumiałą. W niniejszej pracy postaramy się zaprezentować podstawy przetwarzania tekstu i naprawy błędów językowych.

1. Wstęp

W projekcie „Jak napisać spell corrector?” zajmiemy się podstawowymi operacjami wykonywanymi przy przetwarzaniu języka w oparciu o znane algorytmy. Pomoże to w zrozumieniu podstaw przetwarzania języka naturalnego i nakreśli dalszy kontekst dla bardziej zaawansowanych technik w tej dziedzinie

1.1 Cel i zakres projektu

Celem projektu jest stworzenie prostego korektora pisowni, który będzie w stanie poprawiać błędy ortograficzne i literówki w wprowadzonym tekście. Narzędzie będzie działać w trybie graficznym, z użyciem prostego GUI (ang. Graphical User Interface) zaimplementowanym w języku Java. Program będzie również dodawał znaki interpunkcyjne, oraz zaczynał zdanie z dużej litery.

1.2 Metodyka i narzędzia

Aplikacja została napisana w języku Java. Do utworzenia aplikacji okienkowej wykorzystaliśmy bibliotekę JavaFX. Program wykorzystuje algorytm stworzony przez Petera Norviga.

3. Opis algorytmu

Algorytm Petera Norviga do korekty pisowni jest prostym, ale efektywnym podejściem do automatycznego poprawiania błędów ortograficznych. Jego popularność wynika z użycia probabilistycznych modeli językowych oraz prostoty implementacji. Poniżej znajduje się szczegółowy opis algorytmu:

1. Zbieranie danych treningowych

Algorytm zaczyna od zebrania korpusu tekstu, który będzie służył jako baza danych dla modelu językowego. Korpus ten jest zazwyczaj dużym zbiorem tekstów w danym języku, który zawiera prawidłowo napisane słowa.

2. Budowanie słownika

Z korpusu tworzony jest słownik, który zawiera wszystkie słowa i ich częstotliwości występowania. Ten słownik jest używany do określania prawdopodobieństwa wystąpienia poszczególnych słów. Na przykład, w języku angielskim, słowa takie jak "the" czy "and" będą miały wysoką częstotliwość.

3. Generowanie kandydatów

Gdy użytkownik wprowadza słowo, które może zawierać błąd ortograficzny, algorytm generuje listę możliwych poprawek (kandydatów). Peter Norvig proponuje generowanie kandydatów poprzez:

- **Wstawienia:** Dodawanie każdej litery alfabetu w każdym możliwym miejscu słowa.
- **Usunięcia:** Usuwanie każdej litery ze słowa.
- **Zamiany:** Zmiana każdej litery na inną literę alfabetu.
- **Przestawienia:** Zamiana miejscami dwóch sąsiadujących liter.

4. Sprawdzanie poprawności

Dla każdego wygenerowanego kandydata algorytm sprawdza, czy jest on słowem w słowniku. Jeśli tak, to dodaje go do listy prawdopodobnych poprawek.

5. Wybór najlepszej poprawki

Z listy prawdopodobnych poprawek algorytm wybiera tę, która ma najwyższą częstotliwość występowania w słowniku. Jest to realizowane za pomocą funkcji maksymalizacji prawdopodobieństwa [1].

4. Model Językowy ByT5-text-correction

Mały, wielojęzyczny model narzędziowy przeznaczony do prostych korekt tekstu. Jego celem jest poprawa jakości tekstów z internetu, które często nie zawierają interpunkcji lub mają nieprawidłową wielką literę. Model został przeszkolony do wykonywania trzech rodzajów poprawek:

1. Przywracanie interpunkcji w zdaniach.
2. Przywracanie wielkich liter na początku wyrazów.
3. Przywracanie znaków diakrytycznych w językach, które ich używają.

Model językowy ByT5, oparty na architekturze T5 (Text-to-Text Transfer Transformer) opracowanej przez Google Research, różni się od tradycyjnych modeli tym, że operuje na poziomie pojedynczych znaków (liter, cyfr, znaków interpunkcyjnych), a nie tokenów (czyli słów lub ich części). Dzięki temu jest bardziej elastyczny i lepiej radzi sobie z zadaniami wymagającymi analizy na poziomie znaków, takimi jak korekcja tekstu, detekcja błędów czy transliteracja. ByT5 jest modelem wielojęzycznym, zaprojektowanym do różnorodnych zadań związanych z przetwarzaniem języka naturalnego, w tym poprawy jakości tekstów, automatycznej korekty pisowni, tłumaczeń oraz analizy tekstów w różnych językach.

Model ten zostanie wykorzystany do jeszcze bardziej trafnego uzupełniania o prawidłowe treści rozwiniętych zdań oraz zestawienia danych między algorytmem a modelem. [6]

5. Implementacja

Przed rozpoczęciem implementacji zebraliśmy zbiór danych, zawierających słowa, które będzie wykorzystywał nasz program do korekty tekstu. Skorzystaliśmy z pliku tekstowego, zawierającego słownik polski, plik z książką Henryka Sienkiewicza pt. *Krzyżacy* oraz korpus języka polskiego. Tak duży zbiór danych pozwala na efektywną korektę tekstu [3] [4] [5].

Implementację należy rozpocząć od wczytania pliku z tymi zasobami, oraz załadowania wyrazów do słownika, wraz z ich ilością wystąpień. W naszym programie w klasie „Main” jest wczytywany plik z katalogu „resources”, następnie są one wczytywane do mapy, będącej słownikiem.

```
File file = new File(getClass().getResource( name: "/big.txt").getPath());
spellfixer = new Spelling(file);
```

Rys. 1 Wczytanie pliku źródłowego do programu

```
public Spelling(File dictionaryFile) throws Exception { 1 usage  🐞 korbeckik
    Stream.of(new String(Files.readAllBytes(dictionaryFile.toPath())).toLowerCase()
        .replaceAll(WORD_REGEX, EMPTY_CHARACTER)
        .split(EMPTY_CHARACTER))
        .forEach((word) → {
            dict.compute(word.trim(), (k, v) → v = null ? 1 : v + 1);
        });
}
```

Rys. 2 Parsowanie pliku źródłowego i wczytanie go do słownika

Kolejnym ważnym krokiem, zgodnie z punktem trzecim w rozdziale „Opis Algorytmu” należy zaimplementować metodę, w której słowa będą odpowiednio zmieniane, poprzez wstawianie, zamianę, usunięcie i przestawianie liter.

```
private Stream<String> rearrangeWords(final String word) { 3 usages  🐞 korbeckik *
    Stream<String> deletes = IntStream.range(0, word.length())
        .mapToObj((i) → word.substring(0, i) + word.substring( beginIndex: i + 1));
    Stream<String> replaces = IntStream.range(0, word.length()) IntStream
        .boxed() Stream<Integer>
        .flatMap((i) → ALPHABET.chars().mapToObj((c) → word.substring(0, i) + (char) c + word.substring( beginIndex: i + 1)));
    Stream<String> inserts = IntStream.range(0, word.length() + 1)
        .boxed().flatMap((i) → ALPHABET.chars().mapToObj((c) → word.substring(0, i) + (char) c + word.substring(i)));
    Stream<String> transposes = IntStream.range(0, word.length() - 1)
        .mapToObj((i) → word.substring(0, i) + word.charAt(i + 1) + word.charAt(i) + word.substring( beginIndex: i + 2));
    return Stream.of(deletes, replaces, inserts, transposes).flatMap((x) → x);
}
```

Rys. 3 Przestawienie liter w słowie


Wygenerowane słowa przez te cztery operacje mogą nie istnieć w danym języku. Dlatego przed zaproponowaniem korekty, należy sprawdzić czy słowa istnieją w słowniku.

```
private Stream<String> existsInDict(Stream<String> words) { 2
    return words.filter(dict::containsKey);
}
```

Rys. 4 Sprawdzenie, czy słowo istnieje w słowniku

Użycie tych metod znajduje się w metodzie „correct”, która początkowo sprawdza, czy wyraz istnieje w słowniku, jeżeli tak, to słowo nie jest poprawiane. Kolejnym krokiem jest

wywołanie metody „rearrangeWords” wewnątrz metody „existsInDict”. Jeżeli słowo nie zostanie znalezione, odbywa się drugi etap polegający na jeszcze większym „mieszaniu” słów. W tym etapie, wcześniej wygenerowane słowa metodą „rearrangeWords” ponownie są przekazywane do tej metody. Zwiększa to prawdopodobieństwo znalezienia właściwej korekty.

```
public String correct(String word) { korbeckik *  
    if(dict.containsKey(word))  
        return word;  
    Optional<String> e1 = existsInDict(rearrangeWords(word))  
        .max((a, b) → dict.get(a) - dict.get(b));  
    if (e1.isPresent())  
        return e1.get();  
  
    Optional<String> e2 = existsInDict(rearrangeWords(word))  
        .flatMap(this::rearrangeWords)  
        .max((a, b) → dict.get(a) - dict.get(b));  
    return (e2.orElse(word));  
}
```

Rys. 5 Implementacja metody „correct”

Ostatnim etapem implementacji naszego programu, jest stworzenie graficznego interfejsu. Ze względu na prostotę aplikacji, nie wykorzystywaliśmy możliwości deklaracji widoków w plikach „fxml”. Implementację rozpoczęliśmy od deklaracji elementów prezentowanych na ekranie. Element „titleLabel” zawiera tytuł naszego programu, „inputField” to pole tekstowe, w którym będzie przekazywany tekst do korekty, „correctButton” służy do wywoływania metody „correct” oraz „outputArea” zawierająca zwracany tekst [2].

```

primaryStage.setTitle("Spelling Corrector");

Label titleLabel = new Label( s: "Spelling Corrector");
titleLabel.setStyle("-fx-font-size: 24px; -fx-font-weight: bold;");

TextField inputField = new TextField();
inputField.setPromptText("Enter a word to correct");
inputField.setStyle("-fx-font-size: 18px;");

Button correctButton = new Button( s: "Correct");
correctButton.setStyle("-fx-font-size: 18px;");

TextArea outputArea = new TextArea();
outputArea.setWrapText(true);
outputArea.setEditable(false);
outputArea.setStyle("-fx-font-size: 18px;");
ScrollPane scrollPane = new ScrollPane(outputArea);

```

Rys. 6 Deklaracja elementów prezentowanych na ekranie

Aby umożliwić wywołanie metody „correct” należy dodać dla przycisku „correctButton” akcję, która będzie wywoływana po kliknięciu [2].

```

correctButton.setOnAction(event → {
    String input = inputField.getText().trim();
    if (!input.isEmpty()) {
        String[] words = input.split( regex: "\\s+");
        StringBuilder output = new StringBuilder();
        for (String word : words) {
            String corrected = spellfixer.correct(word);
            output.append(" ").append(corrected);
        }

        outputArea.setText(output.toString());
    }
});

```

Rys. 7 Dodanie akcji dla przycisku „correctButton”

Wszystkie kontrolki są pogrupowane w kontenerach VBox, które układają elementy pod sobą. Następnie kontener „mainBox” jest umieszczany w klasie „BorderPane”, który jest jednym z

managerów układu dostępnych w JavaFX. Pozwala na rozmieszczenie elementów interfejsu w pięciu obszarach: na górze, na dole, z lewej strony, z prawej strony i w środku [2].

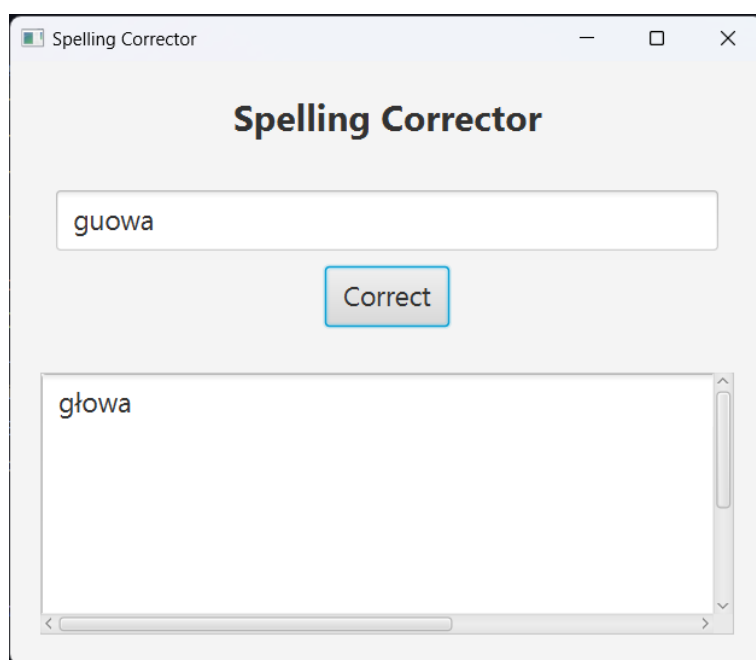
```
// Układ aplikacji
VBox inputBox = new VBox( v: 10, inputField, correctButton);
inputBox.setAlignment(Pos.CENTER);
inputBox.setPadding(new Insets( v: 10));

VBox mainBox = new VBox( v: 20, titleLabel, inputBox, scrollPane, aiLabel, aiScrollPane);
mainBox.setAlignment(Pos.CENTER);
mainBox.setPadding(new Insets( v: 20));

BorderPane root = new BorderPane(mainBox);
Scene scene = new Scene(root, v: 500, v1: 600);
```

Rys. 8 Grupowanie elementów interfejsu

Finalna wersja graficznego interfejsu prezentuje się następująco:



Rys. 9 Wygląd aplikacji

Aby natomiast rozszerzyć i zwiększyć skuteczność uzyskiwanego tekstu. Zaimplementowano model językowy `sdadas/byt5-text-correction` dostępny na platformie <https://huggingface.co> udostępniony przez Sławomir Dadas.

Po zalogowaniu się i uzyskaniu odpowiedniego tokenu uzyskano dostęp do funkcji.

```
1 usage  ▲ Paweł Krzyściak
def correct_text(text):
    model_name = "sdadas/byt5-text-correction"
    model = AutoModelForSeq2SeqLM.from_pretrained(model_name)
    tokenizer = AutoTokenizer.from_pretrained(model_name, model_max_length=512)

    # Tokenizuj tekst wejściowy
    inputs = tokenizer(text, return_tensors="pt", padding=True, truncation=True)

    # Wygeneruj poprawiony tekst
    with torch.no_grad():
        outputs = model.generate(**inputs, max_new_tokens=len(text) + 10)

    # Dekoduj wygenerowany tekst
    corrected_text = tokenizer.decode(outputs[0], skip_special_tokens=True)
    return corrected_text
```

Rys. 10 Kod odpowiedzialny za wygenerowanie poprawnego wyjścia.

W funkcji `correct_text` dzieje się następująco:

Nazwa modelu: Najpierw definiowana jest nazwa modelu, który będzie używany do korekcji tekstu. W tym przypadku jest to "sdadas/byt5-text-correction".

1. Ładowanie modelu i tokenizatora: Model oraz tokenizator są ładowane z biblioteki `transformers` na podstawie wcześniej zdefiniowanej nazwy modelu. Model odpowiada za generowanie poprawionego tekstu, a tokenizator za przekształcenie tekstu na format zrozumiały dla modelu.
2. Tokenizacja tekstu: Tekst wejściowy jest tokenizowany, czyli przekształcany w sekwencję tokenów (w tym przypadku na poziomie znaków). Tokenizator zwraca tensor (struktura danych używana w obliczeniach) i dodaje odpowiednie padding (wyrównanie długości) oraz truncation (przycinanie zbyt długich tekstów).
4. Generowanie poprawionego tekstu: Model generuje poprawioną wersję tekstu na podstawie tokenizowanych danych wejściowych. Używa się tutaj metody `generate`, która tworzy nowy, poprawiony tekst. Dodatkowo, wyłączone są gradienty (`torch.no_grad()`), co oznacza, że podczas generowania nie będą śledzone operacje w celu oszczędzenia pamięci i przyspieszenia obliczeń.
5. Dekodowanie wygenerowanego tekstu: Wygenerowane przez model tokeny są dekodowane z powrotem na ciąg znaków, tworząc poprawiony tekst. Proces dekodowania usuwa specjalne tokeny, które mogą być dodane przez model, aby struktura tekstu była bardziej zrozumiała.

6. Zwrócenie poprawionego tekstu: Na koniec funkcja zwraca poprawiony tekst

Następnie wywołanie programu odbywa się przy uruchomieniu skryptu z odpowiednimi parametrami.

```
if __name__ == "__main__":  
    text = sys.argv[1]  
    corrected_text = correct_text(text)  
    print(f"{corrected_text}")
```

Rys. 111 Kod odpowiedzialny za wygenerowanie poprawnego wyjścia.

Dla przykładu dla danych wejściowych "<pl> ciekaw jestem na co licza bron stawiajace na sykulskiego w nadziei na zwrot ku rosji" Uzyskamy "Ciekaw jestem na co liczą broń stawiające na Sykulskiego w nadziei na zwrot ku Rosji."

By można było wykonać ten program z poziomu pliku Java. Wykonano kolejne kroki które stworzyły plik wykonywalny:

Install PyInstaller from PyPI:

```
pip install pyinstaller
```

Go to your program's directory and run:

```
pyinstaller yourprogram.py
```

This will generate the bundle in a subdirectory called `dist`.

```
pyinstaller -F yourprogram.py
```

Rys. 122 Instrukcja stworzenia pliku wykonywalnego python [8]

Z poziomu Javy natomiast program wykonuje się przy użyciu tworzenia procesu, który przechwytyje rezultat do wyświetlenia z kodowaniem polskich znaków przy użyciu windows-1250.

```
String exePath = "src/dist/spell.exe";

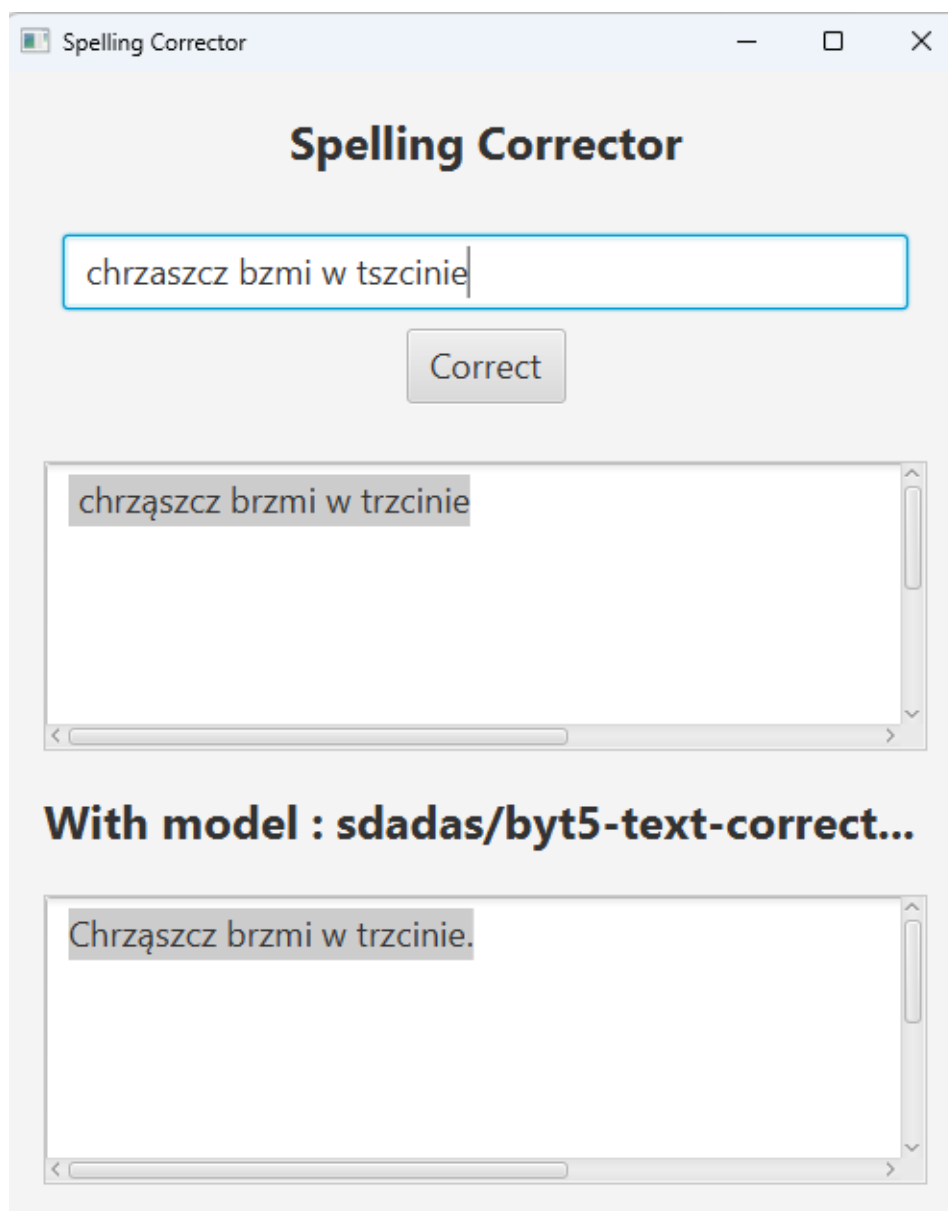
ProcessBuilder processBuilder = new ProcessBuilder(...command: exePath, "<pl>"+text);
processBuilder.redirectErrorStream(true);
processBuilder.redirectInput(ProcessBuilder.Redirect.PIPE);

Process process = processBuilder.start();

BufferedReader reader = new BufferedReader(new InputStreamReader(process.getInputStream(), charsetName: "windows-1250"));

String line;
StringBuilder output = new StringBuilder();
while ((line = reader.readLine()) != null) {
    output.append(line).append("\n");
}
```

Rys. 133 Uruchomienie procesu w Java [8]



Rys. 144 Wygląd aplikacji z modelem

6. Testy programu

Ostatnim etapem tworzenia spell correctora, jest przetestowanie poprawności poprawianego tekstu. Wyniki przedstawione są w tabeli 1.

Wprowadzone słowo	Poprawione słowo	Poprawność (T/N)
guowa	głowa	T
kapusdta	kapusta	T
ksionżka	książka	T
ruża	duża	N
wiakro	wiadro	T
podruž	podróż	T
żeka	żuka	N

Tab. 1 Wyniki testów

Zastosowaniu modelu językowego poprawiło oczekiwany rezultat. Przyczyniło się do przedstawiania bardziej satysfakcjonujących wyników w przypadku dłuższych zdań. Kosztem czasu odpowiedzi.

Wprowadzone słowo	Poprawione słowo (sam algorytm)	Poprawione słowo (z modelem)
tszymaj sie	trzymaj sie	Trzymaj się!
chrzascz bzmi w tszczinie	chrząszcz brzmi w trzcinie	Chrząszcz brzmi w trzcinie.

7. Podsumowanie

Założenia programu zostały osiągnięte, a poprawności działania aplikacji na podstawie siedmiu prób wynosi około 72%. Na podstawie przeprowadzonych testów, doszliśmy do wniosku, że program można ulepszyć przez zamianę znaków ortograficznych przy wprowadzającym tekście i wczytywaniu wyrazów do słownika (np. rz → ż), a następnie, przed przekazaniem tekstu do użytkownika, zamieniać znaki zgodnie z poprawną wersją. Wtedy słownik musiałby przechowywać trzy wartości, czyli ilość wystąpień, poprawne słowo oraz słowo pozbawione ortografii. Ewentualnie, drugim sposobem na poprawę błędów

ortograficznych, można zastosować kolejną funkcję „mieszającą”, która zamieniała by wszystkie znaki ortograficzne (np. rurza → rórza, ruża, róža).

Bibliografia

- [1] Peter N., *How to Write a Spelling Corrector*, <https://www.norvig.com/spell-correct.html> [dostęp: 25.05.2024]
- [2] *Hello World, JavaFX Style*, https://docs.oracle.com/javafx/2/get_started/hello_world.htm [dostęp: 26.05.2024]
- [3] Henryk S., *Krzyżacy. Tom I*, <https://wolnelektury.pl/media/book/txt/krzyzacy-tom-pierwszy.txt> [dostęp: 24.05.2024]
- [4] *Narodowy Korpus Języka Polskiego*, <https://nkjp.pl> [dostęp: 24.05.2024]
- [5] *Słownik Języka Polskiego*, <https://sjp.pl/sl/growy/> [dostęp: 24.05.2024]
- [6] Sławomir Dadas <https://huggingface.co/sdadas/byt5-text-correction> [dostęp: 30.05.2024]
- [7] ByT5 https://huggingface.co/docs/transformers/model_doc/byt5 [dostęp: 30.05.2024]
- [8] <https://stackoverflow.com/questions/5458048/how-can-i-make-a-python-script-standalone-executable-to-run-without-any-dependen>