PROJECT2 – DOM

```yaml
commands:
  - name: głośność
    type: noun
    code: vol
    configs:
      - subdevices/volume.yml
  - name: włączyć
    type: verb
    aliases:
      - name: załączyć
        type: verb
    code: 'on'
```

Każdy plik konfiguracyjny ma taką samą strukturę. Lista ‘commands’ zawierająca rekordy. Każdy rekord składa się z obowiązkowych pól ‘name’, ‘type’ oraz ‘code’, oraz z opcjonalnych list ‘configs’ i ‘aliases’. Lista ‘aliases’ zawiera rekordy składające się z pól ‘name’ oraz ‘type’. Lista ‘configs’ zawiera ścieżki do plików konfiguracyjnych dla podkomand.

Program działa w następujący sposób:

1. Przyjmuje polecenie.
2. Zaczyna od bazowej konfiguracji (z reguły z komendami dla pomieszczeń).
3. Dla każdej komendy porównuje wszystkie warianty nazw (warianty są generowane na podstawie nazwy głównej i nazw z aliasów na podstawie ich typów (przykładowe typy: noun, adjective-noun, noun-statics, etc., odmiany dla poszczególnych typów zdefiniowane są w inflection.py)) z podnapisami polecenia.
4. Znajduje najlepsze dopasowanie (o najmniejszej odległości edycyjnej w pierwszej kolejności, najdłuższe możliwe w drugiej kolejności).
5. Wycina dopasowanie z polecenia.
6. Jeśli istnieje sekcja ‘configs’ to rekurencyjnie przechodzi dla pozostałego napisu i podkomend.
7. Zwraca kod najbliższego dopasowania lub komunikat o błędzie zawierający pasujące dopasowania.

Dzięki liście ‘configs’ można zdefiniować bazowe zachowanie i dodatkowe zachowania, np. lampa (włączać, wyłączać), ciepłe światło (ciepło, zimno) i dodać oba z nich do configu, aby uzyskać lampę z podstawowymi cechami oraz zmianą ciepła światła.

Zaimplementowane typy odmiany:

- adjective-noun, np.: biały pies
- verb, np.: wyłączyć
- adjectives, np.: biały, dwudziesty pierwszy
- noun, np.: pies
- statics, np.: 1, TV
- adjective-noun-statics, np.: duży dom na drzewie
- noun-statics, np.: dom na drzewie

Przykładowe zagnieżdżenie komend:

- Sypialna
    - Lampa

        - Włączyć
        - Wyłączyć
    - Telewizor
        - Włączyć
        - Wyłączyć
        - Kanał
            - Pierwszy
            - Drugi
- Łazienka
    - Lampa
        - Włączyć
        - Wyłączyć
    - Radio
        - Włączyć
        - Wyłączyć
        - Kanał
            - Pierwszy
            - Drugi

Również dzięki liście ‘configs’ można zdefiniować opcjonalne komendy, np.:

- Telewizor
    - Włączyć
    - Wyłączyć
    - Ustawić
        - Kanał
            - Pierwszy
        - Głośność
    - Kanał
        - Pierwszy
    - Głośność

W tym przypadku Kanał oraz Głośność mogą być zdefiniowane w osobnym pliku i dołączone zarówno do jako drugi config do telewizora, oraz jako config dla opcji ustawić. Dzięki temu można przetworzyć polecenia:

- Ustaw kanał pierwszy w telewizorze -> tv set ch 1
- Telewizor kanał pierwszy -> tv ch 1

results.txt

on A1 | on A1 | Załącz oświetlenie górne na poddaszu w sekcji pierwszej

Error: Command not found: "na" in: "oświetlenie biurka" in: "poddasze" | toggle A2 | oświetlenie biurka na poddaszu

Plik zawiera odpowiedź programu, spodziewaną odpowiedź oraz zadane polecenie. Przez dodanie ‘toggle’ jako polecenia ‘zmienić’ nie radzi sobie z rozpoznaniem polecenia bez tej komendy. Dodanie rozwiązania w konfiguracji jest proste, w sposób analogiczny do kanału w telewizorze, jako dodatkowa opcja bez subkomendy specyfikującej, ale ja nie oczekiwałbym takiego zachowania, dlatego tego nie implementowałem. Podobnie jak komend z budzikiem wydanych bez informacji o sypialni.

Wszystkie takie niepełne komendy można zdefiniować w konfiguracji. Program działa, a przez wydzielenie wszystkich komend jako plików konfiguracyjnych o takiej samej strukturze jest bardzo uniwersalny. Początkowo zaprojektowałem 3 różne formaty (dla pokoi, urządzeń i komend) plików konfiguracyjnych i 3 funkcje je obsługujące, ale wyglądały bardzo podobnie, więc uprościłem i uelastyczniłem program.

PROJEKT1 – KATEGORIE

dict_creator.py:

A-DICT (C-DICT) zawiera różne formy rzeczowników (przymiotników) jako klucze, a wartością dla nich jest lista trójek: przypadek, liczba, bazowe słowo.

A-DICT:

{

'abakus': [(('M', 'lp_m_nzyw'), 'abakus'), (('B', 'lp_m_nzyw'), 'abakus')],

'abakusa': [(('D', 'lp_m_nzyw'), 'abakus')],

…

}

C-DICT:

{

'abdominalny': [(('M', 'lp_m_zyw_os'), 'abdominalny'), (('W', 'lp_m_zyw_os'), 'abdominalny'), (('M', 'lp_m_zyw_nos'), 'abdominalny'), (('W', 'lp_m_zyw_nos'), 'abdominalny'), (('M', 'lp_m_nzyw'), 'abdominalny'), (('B', 'lp_m_nzyw'), 'abdominalny'), (('W', 'lp_m_nzyw'), 'abdominalny')],

…
}

Następnie tworzymy A-C-DICT oraz C-A-DICT – zagnieżdżone słowniki zliczeń par rzeczownikowo przymiotnikowych. Korpusem tekstu jest zbiór artykułów z polskiej Wikipedii wstępnie przetworzony przez dr. A. Pohla na potrzeby konkursu PolEval 2019. Przeglądamy każde zdanie z korpusu i rozważamy okna słów długości 7. Jeśli wyraz centralny okna jest rzeczownikiem, a wyraz z otoczenia przymiotnikiem to sprawdzamy czy na ich liście form w A-DICT i C-DICT mamy wspólne pary przypadku i liczby, jeśli tak, to taka para traktowana jest jako występująca i inkrementowane są odpowiednie pola w A-C-DICT oraz C-A-DICT. Każda para jest zliczana jedynie raz, nawet jeśli spełnia więcej par słów (np. ‘czerwonych win’ zostanie potraktowane albo jako ‘czerwony’ + ‘wino’, albo ‘czerwony’ + ‘wina’, ale nie oba, a decyduje czynnik losowy.

vectors_creator.py

Dla rzeczowników pobieramy ich kategorie ze słowosieci za pomocą API w Pythonie (ma dostęp do starszej wersji słowosieci i zdarza się że błędnie kategoryzuje). Zwracana lista zawiera kategorie dla różnych znaczeń słów (np. woda jako jedzenie i jako substancja chemiczna). Spośród zwróconych kategorii wybieramy najbardziej prawdopodobną (występującą najwięcej razy, dla równolicznych występującą wcześniej).

Wybieramy też zestaw bazowy i testowy - odpowiednio pierwszych pięć (1-5) i drugich pięć (6-10) najbardziej licznie występujących rzeczowników w każdej kategorii. Obliczamy współczynniki N oraz NN w sposób podany na prezentacji (dla grupy wektorów bazowych).

tester.py

Wektor dla sprawdzanego słowa obliczamy tworząc z niego grupę jednowyrazową i obliczając współczynniki N dla wszystkich przymiotników, nie normalizujemy go, bo nie ma po czym, ma tylko jeden element.

Ten wektor porównujemy z wektorami bazowymi miarą cosinusową i wybieramy grupę o największej wartości cosinusa – najbardziej podobny wektor.

results.txt

word:rok count:392219 is_correct:True best:['noun.time', 'noun.event', 'noun.artifact', 'noun.phenomenon', 'noun.quantity'] correct:['noun.time', 'noun.time']

Plik zwiera podsumowanie dla wszystkich rzeczowników z korpusu (ponad 36 tys.), zawiera ten rzeczownik, liczbę jego wystąpień w korpusie, odpowiedź czy poprawnie zaklasyfikował (najbliższa kategoria taka sama jak najbardziej prawdopodobna kategoria ze słowosieci), następnie lista 5 najbliższych kategorii oraz wszystkich odpowiedzi ze słowosieci.