

## Dokumentacja wyszukiwarki boolowskiej

*Tomasz Jurdzinski**Aleksander Balicki, Tomasz Maciejewski*

## 1 Instalacja

Pliki projektu wgrywamy do jednego folderu.

Do podfolderu `data/` wgrywamy pliki źródłowe morfologika i wikipedii.

## 2 Użytkowanie

¡DOROBIC FAJNE UI!

## 3 Opis użytych algorytmów i struktur danych

### 3.1 Tworzenie indeksu

Proces tworzenia wykorzystuje ideę MapReduce.

#### 3.1.1 Faza map

Na początku przechodzimy przez plik wikipedii po linii i wyrażeniem regularnym wyznaczamy słowa. Dla każdej znormalizowanej formy słowa. tworzymy parę (*słowo*, *nr\_dokumentu*, *pozycja*) i dodajemy ją do pliku tymczasowego `WORDS` jako jedną linię.

#### 3.1.2 Faza reduce

Po przejściu przez cały plik wikipedii sortujemy plik `WORDS`, stabilnie, po pierwszym słowie, tym sposobem mamy zachowaną kolejność wystąpień dokumentów i pozycji w ramach artykułu. Przechodzimy teraz przez posortowany plik `WORDS.sorted`, i dla każdego trójliterowego prefiksu (lub krótszego, jeżeli całe słowo jest krótsze niż 3 litery) tworzymy tablice hashującą z listami postingo-  
wym (odpowiednio skompresowanymi lub nie). Zapisujemy tą tablice do pliku z użyciem biblioteki do serializacji `marshal`. W ten sam sposób najpierw serializujemy morfologika, aby potem móc szybko normalizować słowa.

### 3.2 Wyszukiwanie

Sposób wyszukiwania interaktywnego, to szczególny przypadek wyszukiwania w formie wsadowej. Wyszukiwarka w formie wsadowej, po wczytaniu wszystkich zapytań, gromadzi z nich słowa, gru-

pując po prefiksie (hash z prefixami jako klucze i pythonowymi setami słów). Dla każdego prefiksu w hashu otwieramy plik odpowiadający za słowa z tym prefiksem i wczytujemy do kolejnego hashu listy postingowe, które przydadzą się później. Tak samo obsługiwane jest pobieranie informacji z morfologika.

Po wczytaniu wszystkich potrzebnych postingów, parsujemy zapytania i robimy odpowiednie scala-  
lania list postingowych, zgodnie z rozwiązaniami przedstawionymi na ćwiczeniach. Po utworzeniu  
postingów dla wszystkich zapytań, wczytujemy plik z tytułami i wypisujemy tytuły dla zapytań.

### 3.3 Normalizacja

Dla słowa  $w$ , odczytujemy plik z informacjami z morfologika dla  $w$ . Sprawdzamy czy słowo jest w tym słowniku, jeśli tak zwracamy wszystkie jego formy bazowe, odpowiednio po operacji stemmingu lub nie.

### 3.4 Struktury danych

1. `dict()` - pythonowa wbudowana tablica hashująca
2. `set()` - pythonowa wbudowana implementacja zbioru, też bazowana na tablicy hashującej
3. `list()` - pythonowa wbudowana implementacja listy

## 4 Opis użytych bibliotek

### 4.1 marshal

Jest to biblioteka do serializacji obiektów pythonowych, według testów najszybsza z dostępnych. Zapisujemy obiekt poleceniem:

```
marshal.dump(obiekt, uchwyty_do_pliku)
```

Odczytujemy:

```
obiekt = marshal.load(uchwyty_do_pliku)
```

Używamy jej do serializacji słowników przechowujących postingi i dane z morfologika oraz listy tytułów.

### 4.2 gzip

Jest to biblioteka do zapisywania i odczytywania plików skompresowanych programem `gzip`.

Używamy specjalnej funkcji do otwarcia pliku, która zwraca nam uchwyt, do którego piszemy lub z niego czytamy:

```
handle = gzip.open(filename, 'wb')
handle.write("text")
```

### 4.3 cProfile i pstats

Są to biblioteki do profilowania programów w pythonie.

Uruchamiamy plik, który chcemy zprofilować tak:

```
python3.1 -m cProfile -o profile ./boolsearch.py
```

To trwa długo, więc można przerwać w każdej chwili i operować na danych częściowych. Dane są zapisane do pliku profile. Aby odczytać te dane korzystamy z programu w pythonie:

```
#!/usr/bin/env python3.1
import pstats
p = pstats.Stats('profile')
p.sort_stats('time').print_stats() #zamiast time może być 'cumulative'
```

który wyświetla nam czasy trwania funkcji.

### 4.4 unittest

PONTON UZUPEŁNIJ

## 5 Opis testów