Maciej Wiśniewski Wtorek 13.15 A

Algorytmy Tekstowe 2025

Laboratorium 1

1. Ekstrakcja informacji z publikacji

Opis użytego kodu:

authors_year_pattern =r"(?:[A-Za-zśŚźŹżŻłŁąĄęĘóÓćĆńŃ]+,\s[A-Za-zśŚźŹżŻłŁąĄęĘóÓćĆńŃ]\.,?\s?)+)\((\d{4})\)"

:? – grupa która nie przechwytuje, wykorzystamy to później

(?:[A-Za-zśŚźŹżŻłŁąĄęĘóÓćĆńŃ]+ - dopasowanie nazwiska,

,\s – przecinek i spacja

[A-Za-zśŚźŹżŻłŁąĄęĘóÓćĆńŃ]\. – inicjał i kropka

,?\s? – przecinek i spacja, jeśli byłoby dwóch autorów

)+ - może się powtarzac wielokrotnie

((\d{4})\) – 4 cyfry na rok

title_journal_pattern = r"\.\s([^.]+)\.\s([^,]+),\s(\d+)"

\.\s([^.]+) – kropka, spacja, tytul do kropki

\.\s([^,]+) – kropka, spacja nazwa do przecinka

,\s(\d+) – przecinek, spacja, numer tomu

volume_issue_pages_pattern = r"(?:\((\d+)\))?,\s(\d+)-(\d+)"

?:\((\d+)\))? – numer wydania(nie musi występować)

,\s(\d+)-(\d+)" – przecinek, spacja, zakres numerów stron

full_pattern – połączenie

match = re.search(full_pattern, reference) # dopasowanie

authors_str = match.group(1) # wyciągnięcie autorów

author_pattern = r"([A-Za-zśŚźŹżŻłŁąĄęĘóÓćĆńŃ]+),\s([A-Za-zśŚźŹżŻłŁąĄęĘóÓćĆńŃ])\."

podobnie jak wyżej

authors_list = [{'last_name': author_match.group(1), 'initial': author_match.group(2)} for author_match in re.finditer(author_pattern, authors_str)] # rozciągamy autorów

year = int(match.group(2)); title = match.group(3); journal = match.group(4); volume = int(match.group(5)); issue_str = match.group(6); issue = int(issue_str) if issue_str else None; start_page = int(match.group(7)); end_page = int(match.group(8)) # wyciągamy resztę

3. Analiza pliku tekstowego

Opis użytego kodu:

words = re.findall(r'\b\w[\w-]*\b', content.lower())
\b – granica słowa
\w – dowolny znak
[\w-]* - dopasowanie słowa(uwzględniajac myślinki)

sentence_pattern = r'(?<![A-Z][a-z]\.)(?<=\.|\?|\!)\s'
?<! – negative lookbehind
?<= - positive lookbehind
\.|\?|\! – kropka/znak zapytania/wykrzynik

email_pattern = r'\b[A-Za-z0-9._%+-]+@[A-Za-z0-9.-]+\.[A-Z|a-z]{2,}\b'
\b – granica slowa
[A-Za-z0-9._%+-]+ – losowe znaki, jeden lub wiele
[A-Za-z0-9.-]+ - nazwa domeny
\. – korpka przed poddomeną
[A-Z|a-z]{2,} – minimum 2 znakowa poddomena

filtered_words = [word for word in words if word not in stop_words and len(word) > 2]
filtrujemy wedle założen
frequent_words = dict(Counter(filtered_words).most_common(20))
20 najczesciej wystepujacych slow do postaci slownika

```
date_patterns = [
    r'\b\d{4}-\d{2}-\d{2}\b',            # Year-Month-Day
    r'\b\d{2}\.\d{2}\.\d{4}\b',          # Day.Month.Year
    r'\b\d{2}/\d{2}/\d{4}\b',            # Month/Day/Year
    r'\b\d{2}-\d{2}-\d{4}\b',            # Month-Day-Year
    r'\b(?:Jan|Feb|Mar|Apr|May|Jun|Jul|Aug|Sep|Oct|Nov|Dec)[a-z]* \d{1,2}, \d{4}\b', # Month Day,Year
```
Wybór 2/4 znaków i łącznika

for pattern in date_patterns:
    dates.extend(re.findall(pattern, content))
filtrujemy po patternach I wrzucamy do listy jeśli jest datą

paragraphs = [p.strip() for p in re.split(r'\n\s*\n', content) if p.strip()]
#dzielimy przy >1 znakach nowej lini
paragraph_sizes = {i: len(re.findall(r'\b\w+\b', p)) for i, p in enumerate(paragraphs, 1)}
numeracja paragrafów od jedynki i zliczanie słów w każdym paragrafie

4. Implementacja uproszczonego parsera regexpów

initial_regex = simplify(regex) # uproszczenie początkowego wyrażenia

if str(initial_regex) not in regex_to_state: # jeśli regex jest nowy

statename = f"q{state_counter}" # nazywamy stan

state_counter += 1

states.add(statename)

state_to_regex[statename] = initial_regex # mapujemy stan na regex i odwrotnie

regex_to_state[str(initial_regex)] = statename

if initial_regex.nullable(): accept_states.add(statename)

jesli akceptuje puste dodajemy do stanów akceptujących puste

queue = deque([regex_to_state[str(initial_regex)]])

while queue:

current_state = queue.popleft() # pobieramy z początku

current_regex = state_to_regex[current_state]

for symbol in alphabet:

derivative = simplify(current_regex.derivative(symbol)) # obliczanie pochodnej brzozowskiego

derivative_str = str(derivative)

if derivative_str not in regex_to_state: # jeśli jest nowa to robi to samo co 10 lini wyżej

new_state = f"q{state_counter}"

state_counter += 1

states.add(new_state)

state_to_regex[new_state] = derivative

regex_to_state[derivative_str] = new_state

if derivative.nullable(): accept_states.add(new_state)

queue.append(new_state) # i dodaj do kolejki

next_state = regex_to_state[derivative_str]

budujemy przejście do teraźniejszego stanu przez pchodna

transitions[(current_state, symbol)] = next_state

start_state = regex_to_state[str(initial_regex)] # poczatkowy regex

return DFA(states, alphabet, transitions, start_state, accept_states)