**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-zśŚźŻżŻłŁąĄęĘóÓćĆńŃ]+,\s[A-Za-zśŚźŻżŻłŁąĄęĘóÓćĆńŃ]\.,?\s?)+)\((\d{4})\)"**

:? – grupa która nie przechwytuje, wykorzystamy to później

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

,\s – przecinek i spacja

[A-Za-zśŚźŻżŻłŁąĄęĘóÓćĆńŃ]\. – 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-zśŚźŻżŻłŁąĄęĘóÓćĆńŃ]+),\s([A-Za-zśŚźŻżŻłŁąĄęĘóÓćĆńŃ])\."**

# 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)**