Jan Kaczmarski - Algorytmy Tekstowe Laboratorium 1

parse_publications.py

```python
import re
from typing import Optional


def parse_publication(reference: str) -> Optional[dict]:
    """
    Parse academic publication reference and extract structured information.

    Expected reference format:
    Lastname, I., Lastname2, I2. (Year). Title. Journal, Volume(Issue), StartPage-EndPage.

    Example:
    Kowalski, J., Nowak, A. (2023). Analiza algorytmów tekstowych. Journal of Computer Science, 45(2), 123-145.

    Args:
        reference (str): Publication reference string

    Returns:
        Optional[dict]: A dictionary containing parsed publication data or None if the reference doesn't match expected format
    """
    authors_year_pattern = r"(?P<authors>(?:[A-ZĄĆĘŁŃÓŚŹŻ][a-ząćęłńóśźż]+, [A-ZĄĆĘŁŃÓŚŹŻ]\.(?:, )?)+) \((?P<year>\d{4})\)"
    title_journal_pattern = (
        r"\. (?P<title>.+?)\. (?P<journal>[A-ZĄĆĘŁŃÓŚŹŻa-ząćęłńóśźż\s]+), "
    )
    volume_issue_pages_pattern = (
        r"(?P<volume>\d+)\(?(?P<issue>\d+)?\)?, (?P<start_page>\d+)-(?P<end_page>\d+)\."
    )

    full_pattern = (
        authors_year_pattern + title_journal_pattern + volume_issue_pages_pattern
    )

    match = re.match(full_pattern, reference)
    if not match:
        return None

    authors_raw = match.group("authors")
    authors_list = []
    author_pattern = r"([A-ZĄĆĘŁŃÓŚŹŻ][a-ząćęłńóśźż]+), ([A-ZĄĆĘŁŃÓŚŹŻ])\."
    for author_match in re.finditer(author_pattern, authors_raw):
        authors_list.append(
            {"last_name": author_match.group(1), "initial": author_match.group(2)}
        )

    return {
        "authors": authors_list,
        "year": int(match.group("year")),
        "title": match.group("title"),
        "journal": match.group("journal"),
        "volume": int(match.group("volume")),
        "issue": int(match.group("issue")) if match.group("issue") else None,
        "pages": {
            "start": int(match.group("start_page")),
            "end": int(match.group("end_page")),
        },
    }
```

Do testowania czy regex jest poprawny korzystałem ze strony https://regex101.com/ Moim pomysłem jest podzielenie jednego dużego regexa na 3 mniejsze części, tak jak zostało to zaproponowane w szkielecie zadania. Głównym zabiegiem które ułatwiło mi rozwiązanie tego zadania jest użycie `name capturing groups` które ułatwiają późniejszy dostęp do konkretnych pól w sparsowanym regex'ie. Dziękl zastosowaniu tego mechnizmu wykorzystuję `re.match.group` co można zobaczyć w kodzie źródłowym.

Wyniki testów

```
$ pytest tests/test_parse_publication.py
============================= test session starts ==============================
platform darwin -- Python 3.12.9, pytest-8.3.5, pluggy-1.5.0
rootdir: /Users/jkaczmarski/private/text-algorithms-course/python-labs/lab_1
plugins: anyio-4.9.0
collected 4 items

tests/test_parse_publication.py ....                                     [100%]

============================== 4 passed in 0.01s ===============================
```

extract_links.py

```python
import re


def extract_links(html: str) -> list[dict[str, str]]:
    """
    Extract all links from the given HTML string.

    Args:
        html (str): HTML content to analyze

    Returns:
        list[dict]: A list of dictionaries where each dictionary contains:
            - 'url': the href attribute value
            - 'title': the title attribute value (or None if not present)
            - 'text': the text between <a> and </a> tags
    """

    # TODO Implement a regular expression pattern to extract links from HTML.
    # The pattern should capture three groups:
    # 1. The URL (href attribute value)
    # 2. The title attribute (which might not exist)
    # 3. The link text (content between <a> and </a> tags)
    pattern = r'<a\s+[^>]*?href=["\'](?P<url>[^"\']+)["\'][^>]*?(?:\s+title=["\'](?P<title>[^"\']+)["\'])?[^>]*?>(?P<text>.*?)</a>'

    links = []

    # TODO Use re.finditer to find all matches of the pattern in the HTML
    # For each match, extract the necessary information and create a dictionary
    # Then append that dictionary to the 'links' list

    for match in re.finditer(pattern, html):
        links.append(
            {
                "url": match.group("url"),
                "title": match.group("title") if match.group("title") else None,
                "text": match.group("text").strip(),
            }
        )

    return links
```

Regex w tym przypadku jest stosunkowo prosty. Znowu bardzo pomaga tu użycie name capturing groups jak w przypadku zadania `parse_publications`. Jest to najważniejszy aspekt tego rozwiązania. Z rozwiązań zastosowanych w tym regexie wymienię 2 jako “ciekawsze”, są to:

- `[^>]` matchowanie jednego znaku który NIE należy do danego zbioru (w tym przypadku, wszystkie znaki oprócz `>`)
- `\s+` matchowanie jednego lub więcej białych znaków. Sprawia to, że regex jest “czystszy”

Wyniki testów

```
▶(venv) jkaczmarski@JAKACZMA-M-RT4J:python-labs/lab_1 (k8s: db-operations) ‹main*›$ pytest tests/test_extract_links.py
============================================== test session starts ==============================================
platform darwin -- Python 3.12.9, pytest-8.3.5, pluggy-1.5.0
rootdir: /Users/jkaczmarski/private/text-algorithms-course/python-labs/lab_1
plugins: anyio-4.9.0
collected 3 items

tests/test_extract_links.py ...                                                                          [100%]

=============================================== 3 passed in 0.01s ===============================================
```

analzye_test_files.py

```python

    # TODO Implement word extraction using regex
    # Find all words in the content (lowercase for consistency)
    words = re.findall(r"\b\w+\b", content)
    word_count = len(words)

    # TODO Implement sentence splitting using regex
    # A sentence typically ends with ., !, or ? followed by a space
    # Be careful about abbreviations (e.g., "Dr.", "U.S.A.")

    sentence_pattern = r"\s*([^.!?]*?(?:Dr|Mr|Mrs|Ms|Prof|Jr|Sr|U\.S\.A|etc|e\.g|i\.e)\.|\s*[^.!?]+)[.!?]"
    sentences = [m.group().strip() for m in re.finditer(sentence_pattern, content)]
    sentence_count = len([s for s in sentences if s.strip()])
73

    # TODO Implement email extraction using regex
    # Extract all valid email addresses from the content
    email_pattern = r"[A-Za-z0-9._%+-]+@[A-Za-z0-9.-]+\.[A-Za-z]{2,}"
    emails = re.findall(email_pattern, content)

    # TODO Calculate word frequencies
    # Count occurrences of each word, excluding stop words and short words
    # Use the Counter class from collections
    frequent_words = Counter([word for word in words if len(word) > 1])
    for word in stop_words:
        if word in frequent_words:
            del frequent_words[word]
    frequent_words = dict(frequent_words.most_common(10))

    # TODO Implement date extraction with multiple formats
    # Detect dates in various formats: YYYY-MM-DD, DD.MM.YYYY, MM/DD/YYYY, etc.
    # Create multiple regex patterns for different date formats
    date_patterns = [
        r"\b\d{4}-\d{2}-\d{2}\b",  # YYYY-MM-DD
        r"\b\d{2}\.\d{2}\.\d{4}\b",  # DD.MM.YYYY
        r"\b\d{2}/\d{2}/\d{4}\b",  # MM/DD/YYYY or DD/MM/YYYY
        r"\b\d{2}-\d{2}-\d{4}\b",  # MM-DD-YYYY or DD-MM-YYYY
        r"(?<!\d)\b\d{1,2} [A-Za-z]+ \d{4}\b",  # 12 March 2025 (Avoids matching "23")
        r"\b[A-Za-z]+ \d{1,2}, \d{4}\b",  # March 12, 2025
    ]
    date_regex = re.compile("|".join(date_patterns))
    dates = date_regex.findall(content)

    # TODO Analyze paragraphs
    # Split the content into paragraphs and count words in each
    # Paragraphs are typically separated by one or more blank lines
    paragraphs = re.split(r"\n\s*\n", content)
    paragraph_sizes = {
        i + 1: len(re.findall(r"\b\w+\b", para)) for i, para in enumerate(paragraphs)
    }

```

W moim rozwiązaniu wykorzystuję proste regexy. Ciekawe rozwiązanie zastosowałem w przypadku `sentence_pattern` w celu wykluczenia skrótów, potrzebuję bazę danych

skrótów, a przynajmniej nie jestem w stanie znaleźć poprawnego warunku, na odróżnienie zdania od skrótu. Też wykorzystałem nieoczywiste rozwiązanie w przypadku wyboru słów. Nie używam zwykłego splita z powodu specyfikacji zadania. Na przykład przy użyciu `.split` słowa “Analyze” i “Analyze,” co powodowało problemy przy kolejnych zdaniach. Dlatego zdecydowałem się na regex.

Wyniki testów