TiDA 06[amb] – Parser outputu i wyrażenia regularne

Cel zajęć

- Tworzenie **złożonych wyrażeń regularnych** do rozpoznawania różnych formatów tekstu.
- Budowanie **modularnych parserów** do analizy wyników pomiarowych.
- Automatyzowanie zapisów danych do CSV z wykorzystaniem `csv.DictWriter` lub `pandas.DataFrame`.
- Tworzenie **wizualizacji porównawczych** różnych wariantów programu (np. indeksy vs wskaźniki).
- Stosowanie **testów jednostkowych** do walidacji poprawności parsera.

Kontekst zajęć

W poprzednich blokach rejestrowaliśmy wyniki działania programów w plikach tekstowych, np.:

```
Czas sortowania (indeksy): 0.123456 s
Czas sortowania (wskazniki) [N=100000]: 0.234 s
Czas sortowania (ptr) [Alg=Quick, N=10000]: 0.021 ms
```

Dzisiaj stworzymy parser, który odczyta wszystkie te linie, rozpozna typ testu, rozmiar danych, nazwę algorytmu i czas — niezależnie od jednostki.

Definicje

Parsowanie — analiza tekstu w celu wydobycia struktury i danych.
Parser — program, który wykonuje parsowanie.

W praktyce parser to często kilka linijek kodu Pythona z wyrażeniem regularnym, które rozpoznaje wzorzec w tekście i zamienia go na uporządkowane dane (słowniki, wiersze CSV, ramki danych Pandas).

Wprowadzenie do wyrażeń regularnych

Regex to **język wzorców tekstowych**. Najważniejsze elementy:

Symbol	Znaczenie
`\d`	cyfra
`\w`	znak słowa (litera, cyfra, _)
`\s`	spacja lub tab
`.`	dowolny znak
`*` / `+` / `?`	powtórzenia
`	`
`(?P<nazwa>...)`	grupa nazwana
`(?:...)`	grupa nienazwana
`^`, `$`	początek/koniec linii

W Pythonie używaj **surowych stringów**: `r"..."`, aby nie dublować backslashy.

👀 Parser uniwersalny (obsługa jednostek, algorytmu, rozmiaru)

```
import re
import csv
from pathlib import Path

pattern = re.compile(
    r"Czas\s+sortowania\s*"                          # nagłówek
    r"\((?P<tryb>indeksy|wskazniki|ptr|idx)\)"       # typ testu
    r"(?:\s*\[Alg=(?P<alg>\w+),\s*N=(?P<N>\d+)\])?"  # opcjonalne Alg i N
    r"\s*:\s*(?P<czas>\d+(?:\.\d+)?)\s*(?P<unit>s|ms|us)"  # czas + jednostka
)

def to_seconds(val: float, unit: str) -> float:
    if unit == 's': return val
    if unit == 'ms': return val / 1000
    if unit == 'us': return val / 1_000_000
    raise ValueError(f"Nieznana jednostka: {unit}")

rows = []
with open("wynik.txt", "r", encoding="utf-8") as f:
    for line in f:
        m = pattern.search(line)
        if m:
            d = m.groupdict()
            czas_s = to_seconds(float(d["czas"]), d["unit"])
            rows.append({
                "tryb": d.get("tryb"),
                "alg": d.get("alg") or "Unknown",
                "N": int(d["N"]) if d.get("N") else None,
                "czas_s": czas_s
            })

with open("wyniki.csv", "w", encoding="utf-8-sig", newline="") as f:
    w = csv.DictWriter(f, fieldnames=["tryb","alg","N","czas_s"], delimiter=';')
    w.writeheader()
    w.writerows(rows)

print(f"Zapisano {len(rows)} wierszy do wyniki.csv")
```

🧪✅❌⚠️ Walidacja i testy

Upewniamy się, że parser poprawnie konwertuje jednostki i nie łapie błędnych danych.

```
def test_to_seconds():
    assert to_seconds(1000, "ms") == 1.0
    assert to_seconds(1_000_000, "us") == 1.0
    assert to_seconds(1, "s") == 1.0
```

Dodatkowo można sprawdzić poprawność danych po sparsowaniu:

```
assert all(r["czas_s"] > 0 for r in rows)
assert all(r["N"] is None or r["N"] > 0 for r in rows)
```

📊🐼 Analiza wyników w Pandas

```
import pandas as pd

df = pd.read_csv("wyniki.csv", sep=';')
df["czas_s"] = df["czas_s"].astype(float)
print(df.head())

# Grupowanie po trybie i algorytmie
summary = df.groupby(["tryb", "alg"]).agg({"czas_s": "mean"})
print(summary)
```

Można też policzyć przyrost czasu dla rosnącego N:

```
dfN = df.dropna(subset=["N"]).copy()
dfN["N"] = dfN["N"].astype(int)
dfN = dfN.sort_values("N")
dfN["ratio"] = dfN["czas_s"] / dfN["czas_s"].min()
```

📈⚖️ Wizualizacja porównawcza

```
import matplotlib.pyplot as plt

for (tryb, alg), g in dfN.groupby(["tryb", "alg"]):
    plt.figure()
    plt.plot(g["N"], g["czas_s"], marker='o')
    plt.title(f"{alg} – {tryb}")
    plt.xlabel("Rozmiar tablicy N")
    plt.ylabel("Czas [s]")
    plt.grid(True)
    plt.show()
```

Dodatkowo można stworzyć wykres log-log, aby zobaczyć zależność złożoności obliczeniowej:

```
plt.figure()
plt.loglog(dfN["N"], dfN["czas_s"], 'o-')
plt.title("Wykres log-log (czas vs N)")
plt.xlabel("log(N)")
plt.ylabel("log(Czas [s])")
plt.grid(True, which="both")
plt.show()
```

🎓 Zadania (do oddania w formie sprawozdania)

1. **Rozbuduj parser** o możliwość rozpoznawania dodatkowych informacji, np. wersji kompilatora lub flag optymalizacji `/O2`, `/Ox`, `/Od`.
2. **Porównaj wykresy** dla różnych architektur CPU (jeśli output zawiera informację o `arch=x64` lub `arch=x86`).
3. **Dodaj test regresji**: uruchom parser na starych i nowych danych, sprawdź, czy wyniki są zgodne.
4. **Zintegruj z raportem Markdown** – wygeneruj z Pandas plik `.md` z tabelą wyników i automatycznie wstaw wykresy PNG.
5. **Zbuduj pipeline analizy** — skrypt `analyze_all.py`, który:
    - przetwarza wszystkie pliki w folderze `outputs/`,
    - łączy dane w jeden `all_results.csv`,
    - generuje zbiorczy raport Markdown.