

**Politechnika Warszawska**  
**Algorytmy i struktury danych**

**Laboratorium 5**  
**Projekt "Wyszukiwanie wzorca"**  
Informatyka – Inteligentne systemy

Paweł Sarnacki 305290

Piotr Niedziałek 304474

Prowadzący: dr inż. Łukasz Skonieczny

Warszawa 2023

## 1. Wstęp

Program został napisany w języku python z wykorzystaniem aplikacji Visual Studio Code. W programie zaimplementowane są trzy algorytmy wyszukiwania wzorca w tekście – algorytm N, KMP i KR. Poprawność działania algorytmów sprawdzono przy pomocy testów przypadkami brzegowymi. Pomiary wydajności zostały przeprowadzone poprzez wyszukiwanie 'n' pierwszych słów z tekstu „pan-tadeusz.txt”, gdzie n=100, 200, ..., 1000. Program był pisany w wersji python 3.9.13, a wykorzystane biblioteki to: time, random, string, matplotlib, unicode, re.

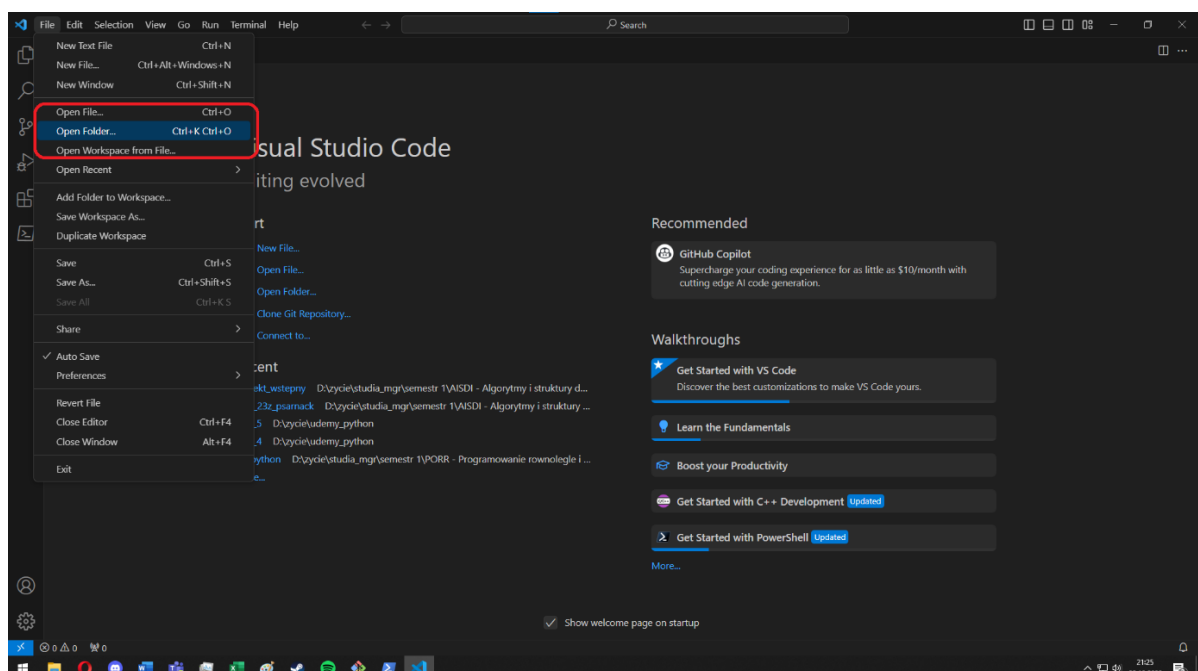
## 2. Struktura projektu

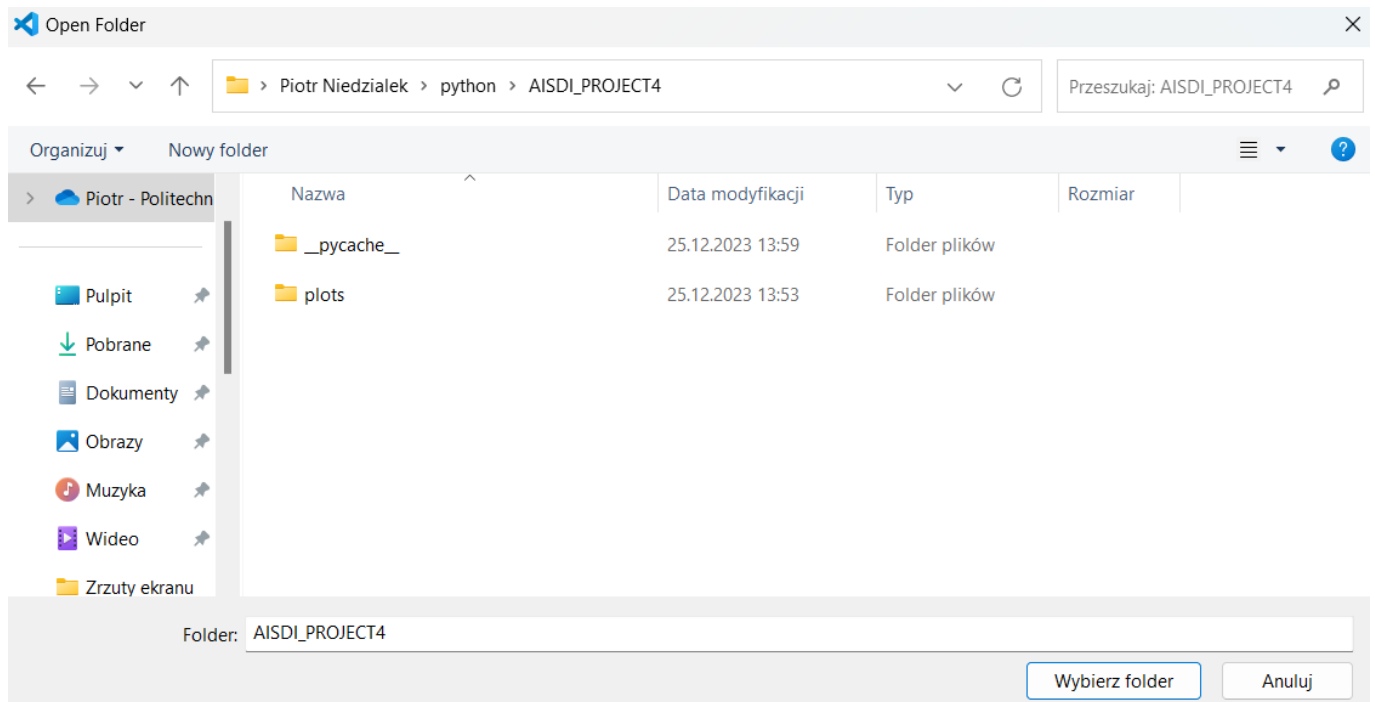
Projekt składa się z:

- Pliku main.py, który zawiera główny program wywołujący funkcje z plików i mierzący czas (Paweł Sarnacki, Piotr Niedziałek)
- Pliku file.py, w którym zaimplementowane jest wczytywanie słów z pliku tekstowego (Piotr Niedziałek)
- Pliku N.py, w którym zaimplementowany jest algorytm naiwnego wyszukiwania wzorca (Paweł Sarnacki)
- Pliku KMP.py, w którym znajduje się implementacja algorytmu Knutha-Morissa-Pratta (Piotr Niedziałek)
- Pliku KR.py, który posiada implementację algorytmu Karpa-Rabina (Paweł Sarnacki)
- Pliku test.py, w którym przeprowadzone są testy jednostkowe (Paweł Sarnacki, Piotr Niedziałek)
- Pliku plot.py, który zawiera implementację rysowania wykresów, są one zapisywane w katalogu plots (Paweł Sarnacki)

## 3. Uruchomienie projektu z środowiska Visual Studio Code

### 3.1. Otworzenie folderu, wybór ścieżki i kliknięcie wybierz folder





### 3.2. Wybranie terminalu bash, wpisanie „py main.py” lub „py test.py” do wykonania testów, lub kliknięcie F5, wyświetlony i zapisany zostanie wykres

## 4. Uruchomienie projektu bez środowiska Visual Studio Code

Przejdźcie do odpowiedniego folderu z plikami projektu i wpisanie w cmd „py main.py” lub „py test.py”.

```
Wiersz polecenia
Microsoft Windows [Version 10.0.22621.2861]
(c) Microsoft Corporation. Wszelkie prawa zastrzeżone.

C:\Users\piotr>cd python

C:\Users\piotr\python>cd AISDI_PROJECT4

C:\Users\piotr\python\AISDI_PROJECT4>py main.py
```

## 5. Otrzymane wyniki.

### Wyniki testów:

- dla każdego algorytmu przeprowadzono testy dla przypadków brzegowych typu wyszukiwanie pustego wzorca, wyszukiwanie w pustym tekście etc.
- następnie przeprowadzono 100 testów na losowych wyrazach z alfabetu {a,b} i porównano wyniki algorytmów KR i KMP z algorytmem naiwnym

```

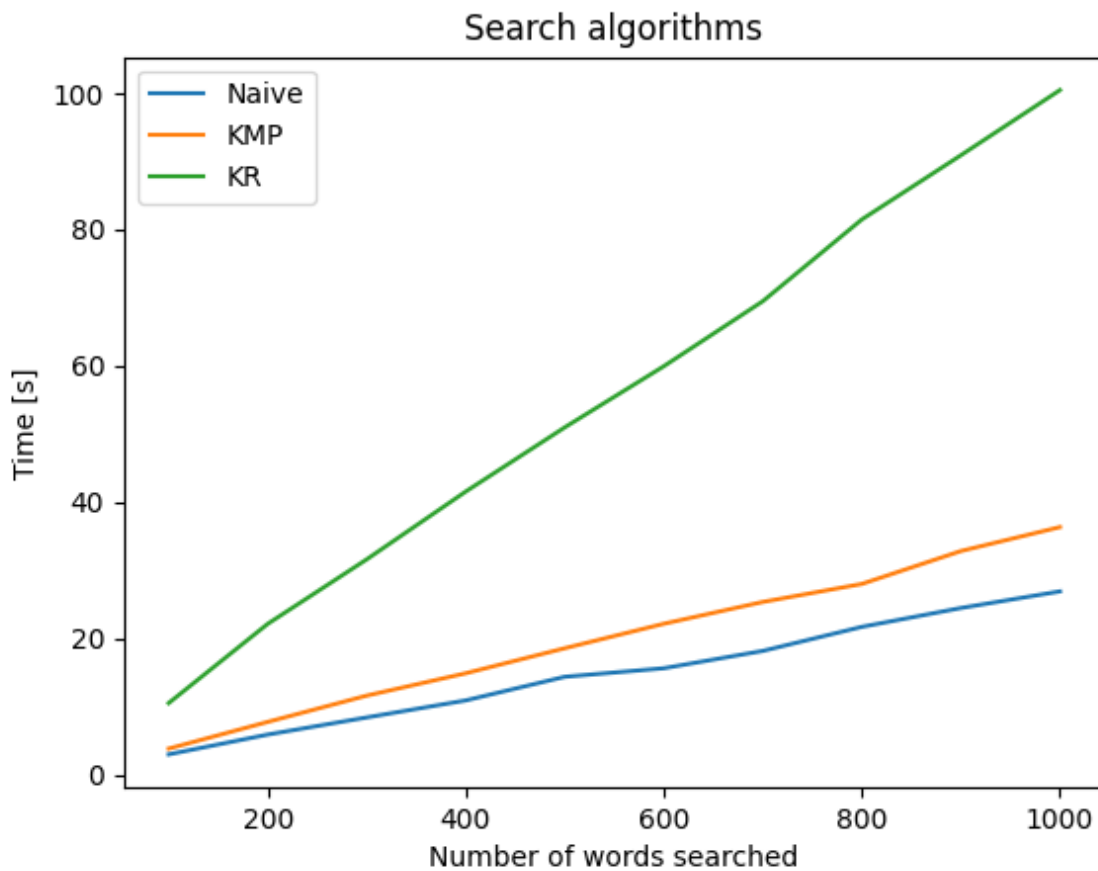
find_naive('', 'abcde') = [0, 1, 2, 3, 4, 5]
find_naive('abc', '') = []
find_naive('', '') = [0]
find_naive('abc', 'abc') = [0]
find_naive('abcdef', 'abc') = []
find_naive('xyz', 'abc') = []
find_naive('abc', 'abcdeabc') = [0, 5]
find_naive('abc', 'abcabcabc') = [0, 3, 6]
find_naive('abc', 'ababab') = []
find_naive('ab', 'ababab') = [0, 2, 4]
Naive passed tests
find_kmp('', 'abcde') = [0, 1, 2, 3, 4, 5]
find_kmp('abc', '') = []
find_kmp('', '') = [0]
find_kmp('abc', 'abc') = [0]
find_kmp('abcdef', 'abc') = []
find_kmp('xyz', 'abc') = []
find_kmp('abc', 'abcdeabc') = [0, 5]
find_kmp('abc', 'abcabcabc') = [0, 3, 6]
find_kmp('abc', 'ababab') = []
find_kmp('ab', 'ababab') = [0, 2, 4]
KMP passed tests
find_karp_rabin('', 'abcde') = [0, 1, 2, 3, 4, 5]
find_karp_rabin('abc', '') = []
find_karp_rabin('', '') = [0]
find_karp_rabin('abc', 'abc') = [0]
find_karp_rabin('abcdef', 'abc') = []
find_karp_rabin('xyz', 'abc') = []
find_karp_rabin('abc', 'abcdeabc') = [0, 5]
find_karp_rabin('abc', 'abcabcabc') = [0, 3, 6]
find_karp_rabin('abc', 'ababab') = []
find_karp_rabin('ab', 'ababab') = [0, 2, 4]
KR passed tests
all algorithms have same results for random text and pattern

```

Wszystkie algorytmy przeszły testy przypadków brzegowych.

Dla losowych wyrazów wszystkie algorytmy dały te same rezultaty.

Zbiórny wykres pokazujący zależność czasu wyszukiwania od liczby szukanych słów:



## 6. Wnioski

- dla tego przypadku algorytm naiwny okazał się najskuteczniejszy, dzieje się tak ponieważ algorytmy KMP i KR najskuteczniejsze są dla wyszukiwania długich wzorców, w innym przypadku koszt preprocesingu może przewyższać zysk czasu w wyszukiwaniu
- czas wyszukiwania KMP i algorytmu naiwnego jest podobny, zgadza się to z teorią – złożoność KMP to  $O(n+m)$ , gdzie  $m$  to długość wyrazu, dla algorytmu naiwnego najgorszy przypadek to  $O(mn)$  ale zazwyczaj w normalnych tekstach jest to również  $O(m+n)$
- w najgorszym przypadku algorytm KR ma złożoność  $O(mn+m)$ , stąd wynika jego najgorszy wynik