Algorytmy i Struktury Danych

Laboratorium nr 5- Wyszukiwanie wzorca

Autorzy:

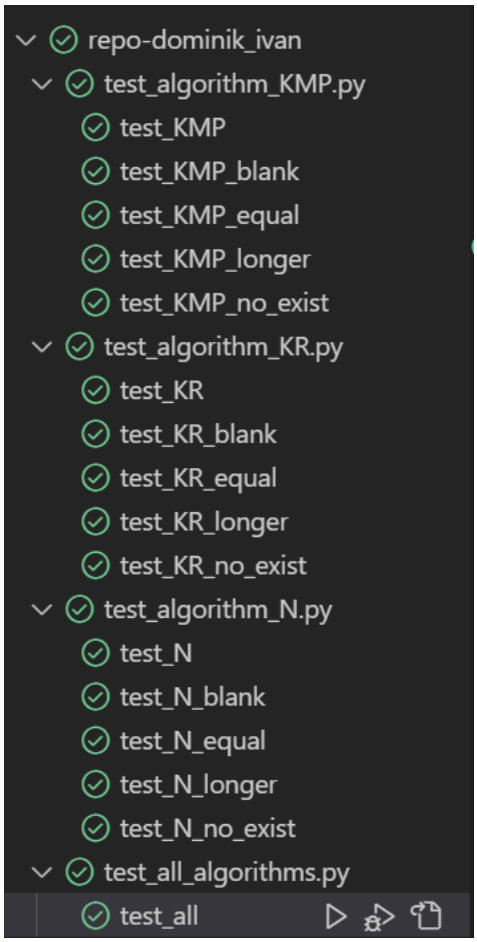
Dominik Wiącek (implementacja KR, wykonanie pomiarów czasu wyszukiwania i generacja wykresów, napisanie sprawozdania)

Ivan Ziubanav (implementacja algorytmu naiwnego oraz KMP, stworzenie testów jednostkowych)

1. Informacje wstępne

W gałęzi repozytorium znajdują się następujące pliki: pliki implementujące dany algorytm (wzór ‘’algorithm_X.py”), pliki testujące dany algorytm (wzór ‘’test_algorithm_X.py”), plik porównujący wyniki dla tych samych danych wejściowych pomiędzy wszystkimi algorytmami (‘’test_all_algorithms.py”), plik generujący wykres zbiorczy (‘’efficiency_tests_N_KMP_KR.py”), plik ‘’pan-tadeusz.txt” zapewniający daną wejściową ‘’text’’, plik ‘’wykres_wzorce.png” zawarty również w sprawozdaniu oraz plik pdf ze sprawozdaniem.

2. Sprawdzenie poprawności implementacji



Powyższy zrzut ekranu przedstawia testy implementacji algorytmów. Pliki ‘’test_algorithm_X.py” przeprowadzają komplet testów dla przypadków zwykłych jak i brzegowych, tj.: pusty jeden lub oba napisy wejściowe, napis string równy napisowi text, napis string dłuższy od napisu txt oraz napis string nie występujący w text. ‘’test_all_algorithms.py” z kolei generuje losowe zmienne string oraz text oraz porównuje zwracane listy pomiędzy wszystkimi algorytmami. Jak widać na wykresie wszystkie testy przeszły pomyślnie. Poniżej przedstawiony zostaje przykładowy wycinek testów:

```python
def test_KMP():
    text = "ALA MAMAŁEGO KOTA. MAŁEGO"
    string = "MAŁEGO"
    assert find(string, text) == [6, 19]

    text = "ALA MA MAŁEGO KOTA. MAŁEGO"
    string = "MAŁEGO"
    assert find(string, text) == [7, 20]

    text = "ALA MAMAMAGO KOTA. MAMAGO"
    string = "MAMAGO"
    assert find(string, text) == [6, 19]

    text = "BBABBBBBBBBB"
    string = "A"
    assert find(string, text) == [2]
```

Tutaj z kolei pokazane są (w większości) losowo wygenerowane wzorce string oraz napisy text oraz wyniki działania algorytmów dla tych danych:

Przypadek brzegowy: wzorzec dłuższy od napisu

```
String:  BAAAABABBAAABBBBBAABAABBBBBBBB  Text:  BABAB
Naiwny:  []
KMP:  []
KR:  []
```

Przypadek brzegowy: wzorzec nie znajduje się w napisie

```
String:  BABABAABBABBBB  Text:  BAAABBBBABABBABBBABBBABBBAAAABBABBBBAABAB
Naiwny:  []
KMP:  []
KR:  []
```

Przypadek brzegowy: wzorzec, napis lub oba puste:

```
String:    Text:  BBBABBAAAABBAABBBAAAAABABBAABBABAABABABABAA
Naiwny:  []
KMP:  []
KR:  []
```

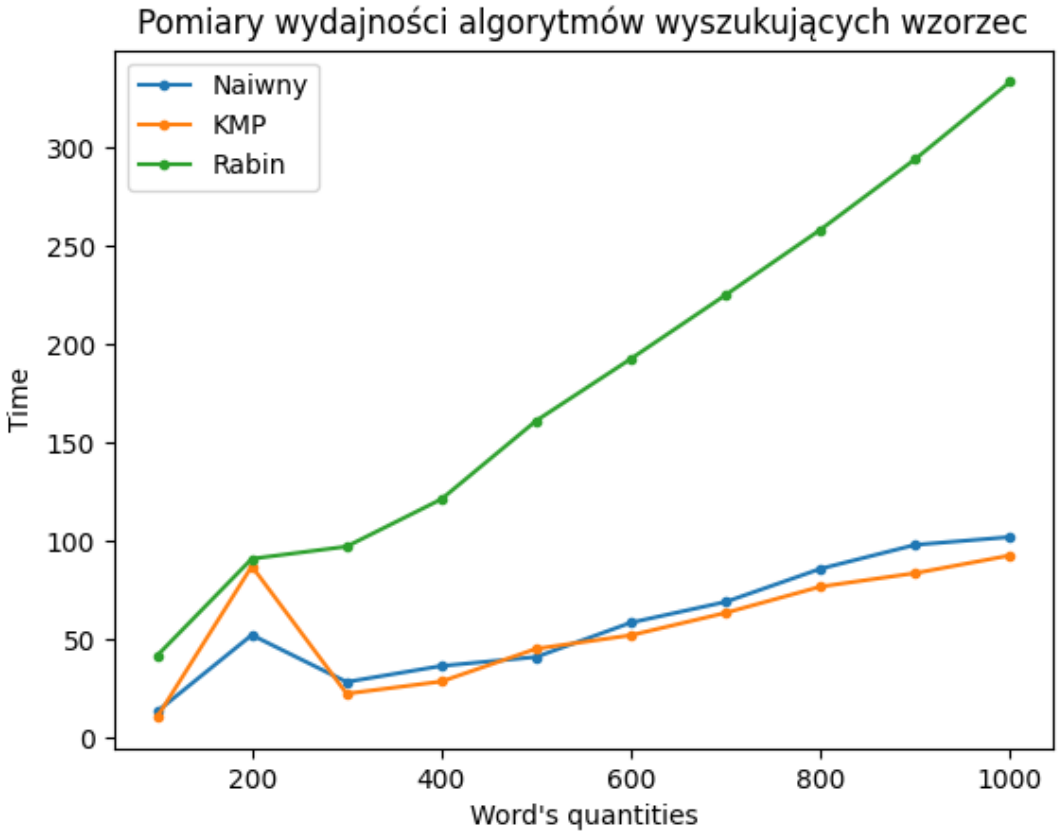
Przypadek brzegowy: wzorzec równy napisowi:

```
String:  ABAB  Text:  ABAB
Naiwny:  [0]
KMP:  [0]
KR:  [0]
```

Przypadek zwyczajny:

```
String:  BA  Text:  ABAABABABAAAABAABABAABBAAAA
Naiwny:  [1, 4, 6, 8, 13, 16, 18, 22]
KMP:  [1, 4, 6, 8, 13, 16, 18, 22]
KR:  [1, 4, 6, 8, 13, 16, 18, 22]
```

3. Pomiary wydajności algorytmów



Wykres 1-zależność czasu wyszukiwania kolejnych słów w pliku od ilości słów

Generalnie pomiary zgadzają się z założeniami w tym kontekście, że wszystkie złożoności są liniowe. O ile algorytm KMP rzeczywiście jest odrobinę szybszy od naiwnego (w szczególności kiedy należy znaleźć więcej wzorców), w końcu jest jego bardziej optymalną wersją, o tyle algorytm Rabina-Karpa jest relatywnie bardzo wolny. Możliwym powodem jest to, że przeszukiwany tekst składa się ze ‘’zwyczajnych” stringów, gdzie raczej nie zdarzają się przypadki, że zgadza się duża część okna przeszukiwanego tekstu ze wzorcem, a ostatecznie okno nie jest wzorcem, tylko raczej jak już pierwsze 3-4 znaki się zgadzają to jest to poszukiwany wzorzec. Sprawia to, że algorytmy naiwny oraz KMP bardzo szybko przechodzą przez tekst. Z kolei jeżeli chodzi o algorytm Rabina-Karpa, musi on i tak przesuwać okno o 1 pozycję w każdej iteracji i przeliczać hash.