LABORATORIUM 4, Wyszukiwanie wzorca w tekście

Zespół: Zuzanna Filipkowska, Aleksandra Sypuła

Środowisko: Visual Studio Code

Link do repozytorium: https://gitlab-stud.elka.pw.edu.pl/zfilipko/aisdi_2021_104.git

Podział pracy:

Zuzanna Filipkowska:

- implementacja algorytmu KR (Karpa-Rabina)
- testy jednostkowe
- generowanie wykresów, przetwarzanie załączonego tekstu

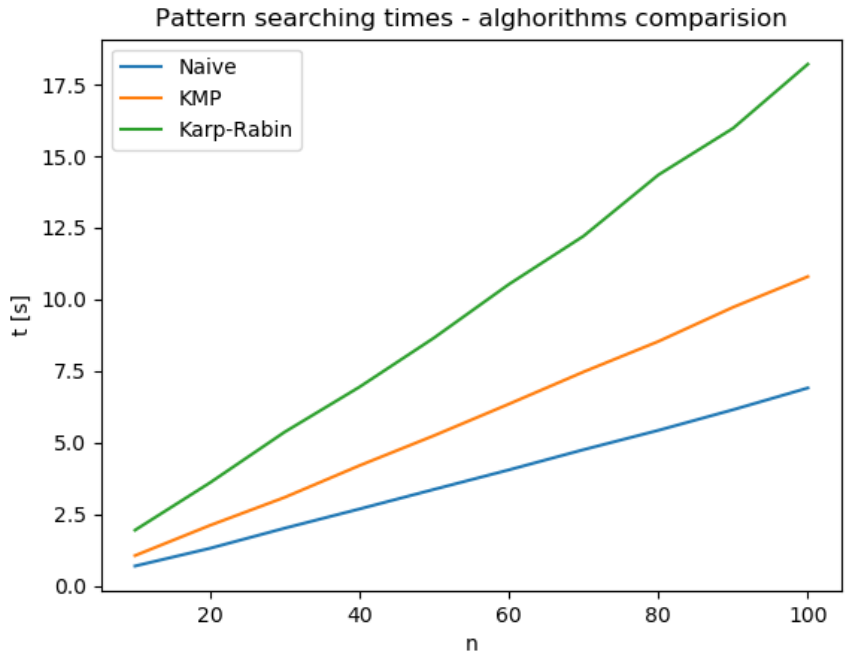
Aleksandra Sypuła:

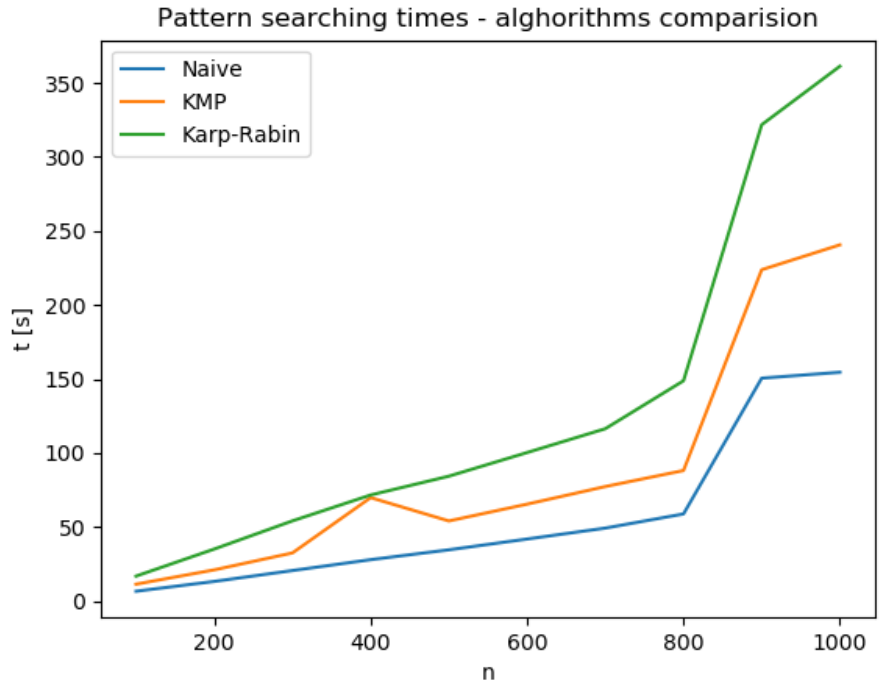
- implementacja algorytmu N (naiwny)
- implementacja algorytmu KMP (Knutha-Morrisa-Pratta)
- testy jednostkowe

Pliki:

- Pattern_searching.py – implementacje wszystkich trzech algorytmów wyszukiwania wzorca w tekście
- test_pattern_searching.py – testy do algorytmó wyszukiwania wzorca w tekście
- compare.py – plik umożliwiający generowanie wykresów z pomiarami czasów wyszukiwania wzorca w tekście
- Pattern_searching_times_01/02/03.png – trzy wykresy z czasami poszukiwania n kolejnych wzorców w całym tekście porównujące działanie algorytmów: naiwnego, KP, KMP

Wykresy przedstawiajace zależność czasu wyszukiwania wzorców w tekście
od liczby wyszukiwanych wzorców





Wnioski:

Bazując na sposobie implementacji oraz założeniach algorytmów wyszukiwania wzorca w tekście, najwolniejszym algorytmem powinien być algorytm naiwny, który nie korzysta z żadnych wcześniejszych założeń oraz nie korzysta również z informacji zdobytych w trakcie wyszukiwania i jego średnia złożoność czasowa wynosi O(n), a w najgorszym przypadku złożoność ta rośnie do O(nm) (złożoności kwadratowej O(n^2)), gdzie n – długość tekstu, m – długość wzorca,. Dwa pozostałe

algorytmy – KP oraz KMP powinny być bardziej efektywne i tak w przypadku algorytmu KMP, dzięki działaniom wstępnym – tworzeniu tablicy z długościami prefikso-sufiksów kolejnych prefiksów łańcucha (złożoność O(n)), ostateczna złożoność wyszukiwania wzorca skraca się do złożoności O(n+m). Dla algorytmu KP średnia złożoność również skraca się (w porównaniu z algorytmem naiwnym) do O(n+m), dzięki wykorzystaniu funkcji haszującej do porównywania tekstu ze wzorcem.

Dla tych algorytmów moglibyśmy oczekiwać dość podobnych wyników dla bardzo krótkich tekstów, jednak różnica między algorytmem naiwnym a pozostałaymi dwoma algorytmami – KP oraz KMP powinna się zwiększać wraz z rosnącą liczbą elementów do sprawdzenia (pesymistyczna złożoność $O(n^2)$ dla naiwnego oraz O(m+n) dla pozostałych algorytmów). Na wykresach możemy zaobserwować jednak nieco odmienne zachowanie tych algorytmów. Najszybszym okazuje się algorytm naiwny, a najwolniejszym algorytm KP. Różnice pomiędzy teorią a naszymi wynikami mogą być skutkiem prób optymalizacji działania naszego programu przez Pythona i tak jak dla algorytmu naiwnego optymalizacja ta przyniosła pozytywne skutki, tak np. dla algorytmu KP Python przy dużych wartościach liczbowych, przechodzi na tym zmiennych PyLongLong, dla których operacje zajmują znacznie więcej czasu, a dla algorytmu KMP różnica może wynikać z dodatkowego czasu wstępnego przetwarzania tablicy, którego Python nie był w stanie wystarczająco zoptymalizować, jak to się udało w przypadku algorytmu naiwnego.

Jak można się również było spodziewać czasy przetwarzania dość szybko rosną wraz z zwiększającą się liczbą elementów do przetworzenia, a piki widoczne na wykresach mogą być spowodowane dodatkowymi procesami, które zachodziły jednocześnie na komputerze i zaburzały wyniki czasowe.

Testy

Testy zostały napisane przy użyciu framework'u pytest.

Uruchomienie: pytest test_pattern_searching_py