Wstęp do Algorytmów

Kierunek: Inżynieria Systemów

Semestr Letni – 2019/2020

Lista 6 – Podobieństwo Ciągów

Pisałem już, że polecam Cormena? – Naprawdę polecam, Grzegorz Popek.

Do tej listy nie wprowadzimy nowej struktury danych. Będziemy operować na ciągach znaków. Jednym z istotniejszych praktycznych problemów jest porównywanie ciągów znaków. Ma zastosowanie w ogólnej analizie tekstów, ale chyba najbliższy nam przykład związany jest ze słownikami, które sugerują poprawę błędów i literówek popełnionych podczas pisania.

Porównywanie tekstów może być czynione wprost, poprzez sprawdzanie rozmieszczenia znaków w porównywanych tekstach. Może też być czynione za pomocą różnych funkcji podsumowujących, np. poprzez sprawdzenie czy rozkład częstości poszczególnych liter w obu tekstach jest podobny (rozkłady te są bardzo różne od siebie dla różnych języków).

Warto zerknąć na bibliotekę (`https://pypi.org/project/textdistance/`), bo jej strona podaje garść użytecznych linków opisujących różne funkcje podobieństwa ciągów znaków. W rozwiązaniach korzystamy jednak z własnoręcznych implementacji funkcji odległości. Funkcje z biblioteki można wykorzystać do weryfikacji poprawności działania własnych implementacji.

1. Odległość Hamminga i jej modyfikacje

a) Zaimplementuj odległość Hamminga, zwracającą liczbę miejsc, w których dwa ciągi się różnią.

b) Zmodyfikuj odległość Hamminga tak, by sąsiadujące litery na klawiaturze miały odległość 1, a niesąsiadujące odległość 2. Np. odległość pomiędzy mama a nawa wynosi 3, ponieważ m i n sąsiadują na klawiaturze (+1 do odległości), a m i w nie sąsiadują (+2 do odległości). Spróbuj uwzględnić inne elementy (np. przypadkowe wciśniecie klawisza alt).

c) Stwórz słownik zawierający 100 słów (różnej długości). Napisz krótki program, który po wprowadzeniu słowa zwraca OK jeśli słowo jest w słowniku. Jeśli zaś słowa nie ma w słowniku, zwraca listę do trzech najbardziej podobnych słów. Zwróć uwagę, że odległość Hamminga służy do porównywania ciągów równej długości. Zupełnie nie wydaje się do wykrywania literówek typu płaszczka-płaszczkaa.

2. Częstość Znaków

a) Artykuł `https://en.wikipedia.org/wiki/Letter_frequency` zawiera tablicę częstości znaków dla kilku języków. Załóż, że ktoś pisze "bez ogonków" i stwórz tablicę częstości liter łacińskich dla języka polskiego, angielskiego i niemieckiego (np. częstość dla a w języku polskim będzie sumą częstości dla a i ą z tabeli).

b) Napisz funkcję zamieniającą kawałek wpisanego tekstu na tablicę częstości zawartych w nim liter (pamiętaj, by ignorować wielkość liter i znaki specjalne w tekście). Wymyśl i zaimplementuj sposób porównywania wynikowej tablicy częstości z tablicami dla poszczególnych języków. Wyświetl nazwę najbardziej podobnego języka. Zwróć uwagę, że wraz ze wzrostem długości tekstów program popełnia coraz mniej błędów. (mały bonus: W przypadku zmęczenia potrzebą wprowadzania tekstów stwórz kilka plików z tekstami w poszczególnych językach i pobieraj bardziej lub mniej kreatywnie teksty z nich.)

c) Stwórz skrócone tablice zawierające jedynie częstość samogłosek i spółgłosek dla powyższych trzech języków. Stwórz funkcję obliczającą częstość samogłosek i spółgłosek we wprowadzonym tekście. Powtórz eksperyment z przypisywaniem języka do tekstu bazując na tak skróconych tablicach.

3. Odległości Edycyjne

a) Napisz funkcję wyznaczającą najdłuższy wspólny podciąg (bez przerw) dla dwóch wprowadzonych tekstów. Np. dla kon**wali**a i za**wali**na wartość ta wynosi 4.

b) Napisz funkcję wyznaczającą najdłuższy wspólny podciąg (mogą być z przerwami) dla dwóch wprowadzonych tekstów. Np. dla **A**pq**BC**r**D**s**EF** i t**AB**u**C**v**DE**wx**F**yz wynosi ona 6.

c) Zwróć uwagę na złożoność obliczeniową algorytmów znajdowania najdłuższych wspólnych podciągów. Sprawia to, że rozważa się takie z pozoru banalne pomysły, jak tworzenie i przeglądanie bardzo długiej listy typowych literówek i poprawnych wersji dla nich. Zwłaszcza, jeśli lista taka jest efektywnie i kreatywnie zakodowana. Istnieje też wiele sposób kodowania samych słowników, np. drzewa trie `https://pl.wikipedia.org/wiki/Drzewo_trie`.

d*) Zapoznaj się z Odległością Levenshteina, zaimplementuj ją i zademonstruj jej działanie.