Zadanie 1: Levenshtein

Łukasz Drożdż
127963

Opis problemu:

W ramach zadania należało zaimplementować algorytm poszukujący w zadanym pliku linii o najmniejszej odległości edycyjnej od podanego wzorca, zwanej odległością Levenshteina.

Opis zastosowanych struktur:

Levenshtein:

Zaimplementowano iteracyjny algorytm Levenshteina wykorzystujący dwa wektory liczb: Jeden do trzymania wyników poprzedniej iteracji „vecPrev" i drugi wypełniany przy aktualnej iteracji „vecCurrent". Konstruktor klasy Levenshtein przyjmuje poszukiwany wzorzec „pattenr". Metoda calculateDistance(…) oblicza odległość między podanym argumentem „input", a poszukiwanym wzorcem.

Na początku program sprawdza, czy wyliczenie odległości nie jest zadaniem trywialnym. Ma to miejsce w przypadkach:

- wzorzec jest równy sprawdzanemu argumentowi;
- sprawdzany ciąg znaków jest pusty;
- wzorzec jest pusty.

Następuje inicjalizacja wstępna wektora pamięci wartościami od „0" do wartości równej długości sprawdzanego ciągu i rozpoczyna się iteracja wg zasad opisanych pseudokodem:

```
for i from 1 to pattern.length
  for j from 1 to input.length
    if pattern[i] == input[j] then
      cost := 0
    else cost := 1
    vecCurrent[j + 1] := min(vecPrev    [j]   + 1,
                             vecCurrent [j-1] + 1,
                             vecPrev    [j-1] + cost)
```

Jako wynik obliczeń zwrócona zostaje liczba z ostatniej komórki wektora "vecCurrent".

Metoda *calculateDistance* przyjmuje jako argument opcjonalny maksymalny dystans, przy którym przeszukiwanie uznawane jest za nieudane. Zostało to wykorzystane w klasie AppClient, by zaniechać obliczeń w przypadku, gdy badana linia na pewno nie będzie charakteryzowała się krótszym dystansem edycyjnym, niż dotychczas znaleziona linia.

AppClient:

Główną logikę programu zaimplementowano w klasie AppClient. Po wstępnym sprawdzeniu formatowania wzorca „pattern" stworzone zostają potrzebne instancje klasy Levenshtein. Odległość

edycyjna wyliczana jest względem każdej możliwej kombinacji zadanego wzorca. Ze względu na następujące fakty :

- rekord opisany jest wyrażeniem regularnym „^ ([a-zA-Z]+[\s]+){1,2}[a-zA-Z] $”;
- niemożliwym jest odróżnienie imienia od nazwiska;
- w przypadku imion podwójnych, imiona muszą być koło siebie, nierozdzielone nazwiskiem;

wykorzystano następujące kombinajce:

- Dla rekordu z jednym imieniem ***str1 str2:***
  - ***str1 str2***
  - ***str2 str1***
- Dla rekordu z dwoma imionami ***str1 str2 str3:***
  - ***str1 str2 str3***
  - ***str2 str3 str1***
  - ***str3 str1 str2***

Dla porządku zastosowano prywatną klasę Line, wiążącą treść typu String z odpowiednim numerem linii.

Wyliczana jest odległość edycyjna każdej linii dla każdej z przyjętych kombinacji wzorca. Dotychczasowy najmniejszy dystans edycyjny podawany jest jako próg do obliczeń obiektu klasy Levenshtein, by zaniechać obliczenia, gdy jest pewność, że sprawdzana linia nie będzie linią o najkrótszym dystansie edycyjnym.

Program wypisuje znalezioną linię o najkrótszym dystansie Levenshteina względem zadanego wzorca.