Zaawansowane programowanie obiektowe, 2019/20, studia niestacjonarne

Lab. 2

Termin nadesłania zadań: do 23.03, godz. 23:59.

1. (1.5 pkt) Zaimplementuj funkcję
   double LevQWERTY(String s1, String s2),
   która zwraca ważoną odległość Levenshteina między napisami s1 i s2, gdzie wagi zależne są od wzajemnego położenia pary znaków na klawiaturze.
   Konkretniej, odl. Levenshteina bazuje na 3 elementarnych operacjach: wstawienia znaku (ang. *insertion*), usunięcia znaku (ang. *deletion*) oraz zastąpienia znaku innym (ang. *substitution*). W naszym przypadku waga operacji insercji i delecji ma wynosić 1, natomiast waga substytucji wynosi:
   - 0.5, jeśli odnośna para znaków sąsiaduje w rzędzie na klawiaturze,
   - 1, w przeciwnym przypadku.

   Zakładamy, że s1 i s2 mogą zawierać tylko małe litery łacińskie.

   Przykłady:
   LevQWERTY("kot", "kita") == 1.5 (1 insercja (a) + 1 substytucja znaków sąsiadujących w rzędzie (o <--> i)).
   LevQWERTY("drab", "dal") == 2 (1 delecja (r) + 1 substytucja znaków niesąsiadujących w rzędzie (b <--> l)).

   Napisz testy z użyciem jUnit sprawdzające poprawność napisanej funkcji. Istotne jest dobranie własnych sensownych przykładów.

   Wskazówka: zastosuj tablicę asocjacyjną z małymi literami łacińskimi jako kluczami oraz zbiorami liter z nimi sąsiadujących jako wartościami.

   Formuła programowania dynamicznego dla obliczania odl. Levenshteina + przykład:
   https://pl.wikipedia.org/wiki/Odleg%C5%82o%C5%9B%C4%87_Levenshteina

2. (1 pkt)
   Napisz klasę zawierającą metody sortujące napisy z uwzględnieniem alfabetu polskiego (np. „Łukasz” ma być między „Lucyna” a „Marek”).
   Wskazówka: wykorzystaj klasę java.text.Collator.

   Konkretnie napisz 3 metody sortujące:

   public static void sortStrings(Collator collator, String[] words)
   – sortującą napisy ręcznie i naiwnie, z użyciem sortowania przez wstawianie (*insertion sort*),

   public static void fastSortStrings(Collator collator, String[] words)
   i
   public static void fastSortStrings2(Collator collator, String[] words)
   – sortujące napisy z użyciem Arrays.sort(…).
   Różnica między tymi dwiema metodami jest taka, że fastSortStrings ma używać

anonimowego obiektu komparatora, zaś fastSortStrings2 ma wykorzystać funkcję lambda.

W testach (z użyciem JUnit) porównaj zgodność wyników zwracanych przez wszystkie te 3 funkcje, a także wyświetl wyniki na konsoli dla następującej tablicy:

```
String[] names = {"Łukasz", "Ścibor", "Stefania", "Darek", "Agnieszka",
                  "Zyta", "Órszula", "Świętopełk"};
```

Wykonaj również test wydajnościowy tych 3 metod, sortując powyższą tablicę imion w pętli 100 tys. razy (oczywiście na starcie ma być za każdym razem nieposortowana). Tym razem nie wypisuj tablicy na ekranie. Wykorzystaj metodę System.nanoTime().

3. (1.5 pkt) Narysuj w konsoli, z wykorzystaniem rekurencji, pionową linijkę o zadanych parametrach: (długość w danych jednostkach, liczba poziomów zagnieżdżeń). Podziałka ma być „dwukierunkowa", co widać na obrazku poniżej. Udokumentuj swoją funkcję i obejrzyj wygenerowanego (-> javadoc) HTML-a. https://www.oracle.com/technetwork/java/javase/documentation/index-137868.html

   Przykład, (2, 5):

```
---------- 0
    --
   ----
    --
  ------
    --
   ----
    --
 --------
    --
   ----
    --
  ------
    --
   ----
    --
---------- 1
    --
   ----
    --
  ------
    --
   ----
    --
 --------
    --
   ----
    --
  ------
    --
   ----
    --
---------- 2
```