

**Standardní zadání**

## KIV/PT – Semestrální práce

studenti: *Štěpán Martínek a Josef Kalivoda*

# Zadání

Vytvořte program, který načte text ze souboru a ze všech unikátních slov vyskytujících se v textu

vytvoří slovník. Slovník bude možné vyexportovat ve vhodně zvoleném formátu do souboru a znovu

použít při dalším spuštění programu. Program dále umožní napsat nebo načíst libovolný text a v něm

vyhledat zadané slovo. Pokud se slovo bude vyskytovat v prohledávaném textu, tak program vypíše

počet výskytů a uvede u všech výskytů počáteční a koncový index, kde se v textu slovo nachází. Pro

prohledávání textu využijte algoritmu komprimované trie. Pokud se zadané slovo v textu nenachází,

tak vypište maximálně 10 nejbližších slov ze slovníku. K porovnání vzdáleností mezi hledaným slovem

a slovy ze slovníku použijte Levensteinovu metriku probíranou na přednáškách. Uživatel by měl mít

také možnost přidat hledané slovo do slovníku, pokud se v něm nenachází.

Uživatelské rozhraní programu může být grafické i konzolové. Program bude mít ošetřeny všechny

vstupy a zdrojový kód projde validací nástrojem PMD. Kód programu bude okomentovaný javadoc

komentáři. Struktura dokumentace je uvedena níže v tomto dokumentu.

# Analýza

**Volba vhodné datové struktrury pro slovník:**

* Nejvhodnejší datovou strukturou bude kompaktní komprimovaná trie, která může mít i jediného následníka
* Kompaktní komprimovaná trie, jelikož pro případ nenalezení slova v textu potřebujeme seznam slov v trii, což nám kompaktní verze umožnuje, navíc zabírá menší datový prostor
* Jediný následník opět umožní menší zabraný prostor a zároveň nenaruší trii pokud poudmínkou jediného potomka je, že rodič je slovo samo o sobě

**Formát výstupu**

* Zvolili bychom klasický textový soubor, přestože binární soubor by byl menší, pro možnou jednoduchou rozšiřitelnost o různá formátování
* Syntax souboru navrhujeme jeden node trie na řádku kdy:
  + před nodem bude | tolikrát jako počet rodičů a - pro indikaci začátku slova
  + bude následovat prefix nodu
  + následně bude: jedná-li se o slovo, následovaná indexy oddělené čárkami
* Tato syntaxe bude dobře čitelná jak pro člověka, tak dobře parsovatelná zpět na trii

**Levenstein**

* Metodu pro Levensteinovu vzdálenost budeme implementovat iteračně, jelikož rekurzivní bude kvůli overheadu daleko pomalejší

Zvýraznění hledaných slov

* Pro zvýraznění hledaných slov použijeme místo TextArea třídu StyleClassedTextArea z knihovny RichTextFX, která umožnuje nastavit konkrétní styl na místo v textu podle indexu, a to je pro naše účely ideální.

Pro grafické rozhraní programu použijeme JavaFX, jelikož s ním máme zkušenosti z předmětu UUR a při průzkumu možností zvýrazňování v textu jsme narazili na zmíněnou knihovnu RichTextFX, která nám tento proces usnadní.

# Návrh programu

Návrh jsme popsali v uml diagramu:

