



KIV/PC
JEDNODUCHÝ STEMMER

Jakub Vítek - A16B0165P

Prosinec 2018

Obsah

1	Zadání	3
2	Analýza úlohy	5
2.1	Dostupné datové struktury	6
2.1.1	Spojový seznam	6
2.1.2	Hashovací tabulka	6
2.1.3	Trie	6

1 Zadání

Naprogramujte v ANSI C přenositelnou **konzolovou aplikaci**, která bude pracovat jako tzv. *stemmer*. Stemmer je algoritmus, resp. program, který hledá kořeny slov. Stemmer pracuje ve dvou režimech: (i) v režimu **učení**, kdy je na vstupu velké množství textu (tzv. *korpus*) v jednom konkrétním etnickém jazyce (libovolném) a na výstupu pak slovník (seznam) kořenů slov; nebo (ii) v režimu zpracování slov, kdy je na vstupu slovo (nebo sekvence slov) a stemmer ke každému z nich určí jeho kořen. Tento proces, tzv. *stemming* je jedním ze základních stavebních kamenů nesmírně zajímavého odvětví umělé inteligence, které se označuje jako NLP (= Natural Language Processing, česky zpracování přirozeného jazyka).

Vaším úkolem je tedy implementace takového stemmeru, ovšem velice jednoduchého, podle dále uvedených instrukcí.

Stemmer se bude spouštět příkazem **sistem.exe** (corpus-file | ["]word-sequence["]) [-msl=<celé číslo>] [-msf=<celé číslo>].

Symbol (**corpus-file**) zastupuje jméno vstupního textového souboru s korpusem, tj. velkým množstvím textu, který se použije k "natrénování" stemmeru. Přípona souboru nemusí být uvedena; pokud uvedena není, předpokládejte, že má soubor příponu **.txt**. Symbol (**word-sequence**) zastupuje slovo nebo sekvenci slov, k nimž má stemmer určit kořeny. Režim činnosti programu je dán předaným parametrem. Je-li parametrem jméno (a případně cesta k) souboru, pak bude stemmer pracovat v režimu učení, tedy tvorby databáze kořenů a na základě analýzy dat z korpusu. Je-li parametrem slovo nebo sekvece slov (ta musí být uzavřena v uvozovkách), pak stemmer bude pracovat v režimu zpracování slov, tedy určování kořene každého slova ze sekvence.

Program může být spuštěn se dvěma nepovinnými parametry:

-msl - Nepovinný parametr **-msl**=(*celé číslo*) určuje minimální délku kořene slova (msl = Minimum Stem Length), který bude uložen do databáze kořenů. Není-li tento parametr předán, použije se implicitní minimální délka kořene 3 znaky. Tento parametr je tedy zřejmě použitelný jen v kombinaci s cestou ke korpusu, tedy v režimu učení stemmeru

-msf - Nepovinný parametr **-msf**=(*celé číslo*) určuje minimální počet výskytů příslušného kořene (msf = Minimum Stem Frequency). Pokud se tento kořen v korpusu nevyskytl aspoň tolikrát, kolik je určeno tímto parametrem, nepoužije se při zpracování slov, tj. stemmer nemůže u žádného zpracovávaného slova oznámit, že tento kořen je kořen předmětného slova. Není-li tento parametr předán, použije se implicitní minimální počet výskytů kořene 10x. Tento parametr je tedy zřejmě použitelný jen v kombinaci se slovem nebo sekvencí slov, tedy v režimu zpracování slov.

Program může být během testování spuštěn například takto (režim učení):

```
. . . \>sistem.exe e:\data\czech-corpus.txt -msl=4
```

Následně v režimu zpracování slov třeba takto:

```
. . . \>sistem.exe "šel pes do lesa" -msf=15
```

```
. . . \>sistem.exe bezdomovec -msf=5
```

Úkolem vašeho programu tedy je v režimu učení vytvořit databázi kořenů (textový soubor) a v režimu zpracování slov vypsát kořen každého slova ze vstupní sekvence.

V případě, že nebude programu předán parametr prvního nebo druhého uvedeného typu, vypíše krátké chybové hlášení (anglicky) a oznamte chybu operačnímu prostředí pomocí nenulového návratového kódu. Pokud bude stemmeru při prvním spuštění předáno slovo nebo sekvence slov, ukončete jej chybovým stavem (a krátkým vysvětlujícím hlášením), indikujícím, že nedošlo k předchozímu vytvoření databáze kořenů slov, a tudíž není možné u slov ze sekvence jejich kořeny určit.

Hotovou práci odevzdejte v jediném archivu typu **ZIP** prostřednictvím automatického odevzdávacího a validačního systému. Archiv nechtě obsahuje všechny zdrojové soubory potřebné k přeložení programu, **makefile** pro Windows i Linux (pro překlad v Linuxu připravte soubor pojmenovaný makefile a pro Windows makefile.win) a dokumentaci ve formátu PDF vytvořenou v typografickém systému $\text{T}_{\text{E}}\text{X}$ resp. $\text{L}_{\text{A}}\text{T}_{\text{E}}\text{X}$. Bude-li některá z částí chybět, kontrolní script Vaši práci odmítne.

Úplné zadání je dostupné na adrese:

<https://www.kiv.zcu.cz/studies/predmety/pc/doc/work/sw2018-03.pdf>.

2 Analýza úlohy

Stematizace (anglicky *stemming*) je postup, během kterého se slova převádějí na jejich základ - tzv. *stem*. Základem rozumíme tu část slova, která se v různých tvarech téhož slova nemění. Obvykle bývá základ slova také jeho gramatickým kořenem, nemusí tomu však být pravidlem. Často jsou příbuzná slova převedena na stejný základ, ze kterého byla odvozena a to i přes to, že daný základ nemusí být jejich gramatickým kořenem. Tyto postupy se hojně využívají při zpracování přirozeného jazyka či například jako základ pro funkcionalitu internetových vyhledavačů.

Bohužel není stematizaci možné provádět se všemi jazyky - problematická je například čínština. Většina jazyků patřících do skupiny Indo-Evropské rodiny jazyků vytváří slova na základě jasně daných gramatických pravidel. Na slova těchto jazyků je možné použít některý ze stematizačních algoritmů. Pro tyto jazyky je pak typické, že jejich základem je kořen slova, ze kterého lze odvodit slova další přidáním předpon či přípon.

Stemmer vytvářený v rámci této práce nebude pravidlový, ale statistický. Kořeny slov budou v režimu učení odvozovány analýzou velkého množství textu. Výsledkem této analýzy bude databáze všech daných kořenů spolu s počtem jejich výskytů. V režimu zpracování pak bude databáze kořenů prohledávána a ke každému detekovanému slovu ve vstupním argumentu bude třeba nalézt nejdelší kořen vyhovující nepovinnému parametru.

V rámci práce bude nutné za pomoci standardní knihovny *stdio* přečíst obsah vstupního korpusu s textem. V tomto textu bude nutné nalézt slova a uložit je do některé z dostupných datových struktur. Čtení těchto dat můžeme provést načtením celé řádky a její následným rozdělením dle dělicích znaků či čtením jednotlivých znaků. Raději se přikláním k načítání obsahu souboru znak po znaku vzhledem k tomu, že to umožňuje snáze pracovat s načítanými daty. Se znaky v tomto případě můžeme manipulovat přímo v průběhu načítání před tím, než načtený znak uložíme do vyrovnávací proměnné. Danou manipulaci může být například převod velkého znaku na malý, což je triviálním problémem pro znaky ze sady ASCII, ale pro znakovou sadu CP1250 potřebujeme vytvořit vlastní převodní funkci. Znaková sada CP1250 má také své vlastní dělicí znaky. Manipulace při načítání nám ušetří nutnost vracet se k řetězci, jež byl načten jiným způsobem. V případě nalezení dělicího znaku pak obsah vyrovnávací paměti vložíme do zvolené datové struktury a obsah vyrovnávací paměti vynulujeme. Stejný způsob můžeme také použít při detekování slov ze vstupního argumentu.

2.1 Dostupné datové struktury

2.1.1 Spojový seznam

Spojový seznam (Linked list) je strukturou určenou k ukládání dat neznámého množství. Základem spojového seznamu je uzel, který vždy obsahuje ukládanou hodnotu a ukazatel na následující prvek. Implementace vkládání dat a vyhledávání v nich je velice jednoduchá, bohužel však není příliš efektivní vzhledem k tomu, že musíme daty proiterovat v případě vkládání až na poslední prvek. V nejhorším případě se pak dostaneme na složitost $O(n)$. Další nevýhodou je, že tato struktura ve svém základu nemá vkládané prvky abecedně či jinak řazené, prvky jsou strukturovány ve stejném pořadí, ve kterém jsou ukládány. Spojový seznam pak můžeme využít například pro implementaci hashovací tabulky.

!!! SMAZAT - VLOŽIT OBRÁZEK SPOJOVÉHO SEZNAMU !!!

2.1.2 Hashovací tabulka

Hashovací tabulka je datovou strukturou optimalizovanou pro vyhledávání. Struktura se využívá pro ukládání dvojic klíč-hodnota. Jejím základem je standardně pole s určenou velikostí a hashovací funkce, která převádí klíče na indexy. Díky této funkci jsme schopni přistupovat na daný index se složitostí $O(1)$, celková složitost však záleží na implementaci úložné struktury na daném indexu - velice často je jako úložná struktura využit spojový seznam se svou složitostí $O(n)$, alternativně je také možné pro ukládání využít binární vyhledávací strom. Ve svém základu hashovací tabulka není seřazena. Efektivita této struktury je závislá na vhodné hashovací funkci.

!!! SMAZAT - VLOŽIT OBRÁZEK HASHOVACÍ TABULKY !!!

2.1.3 Trie

!!! SMAZAT - NAPSAT TEXT K TRII !!!

!!! SMAZAT - VLOŽIT OBRÁZEK TRIE !!!