FAKULTA
APLIKOVANÝCH VĚD
ZÁPADOČESKÉ
UNIVERZITY
V PLZNI

Semestrální práce z KIV/PC

Jednoduchý stemmer

Martin Dub (A16B0024P)

8. ledna 2019

Obsah

1 Zadání

Naprogramujte v ANSI C přenositelnou[1] konzolovou aplikaci, která bude pracovat jako tzv. stemmer. Stemmer je algorirmus, resp. progarm, který hledá kořeny slov. Stemmer pracuje ve dvou režimech: (i) v řežimu 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 řežimu zpracování slov, kdy je na vstupu slovo (nebo sekvence slov) a stemmer ke každému 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 v pom případě má soubor příponu
- Symbol **<word-sequence>** zastupuje slovo nebo sekvenci slov, pro které program najde kořeny
- -msl (nepovinný parametr) specifikuje minimální délku kořene slova. Defaultně je 3.
- -msf (nepovinný parametr) specifikuje minimalní frekvenci kořene. Defaultní je 10.

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

sistem.exe czech-corpus.txt -msl=4

A následně v režimu zpracování slov třeba takto

[1] Je třeba, aby bylo možné Váš program přeložit a spustit na PC s operačním prostředím Win32/64 (tj. operační systémy Microsoft Windows NT/2000/XP/Vista/7/8/10) a s běžnými distribucemi Linuxu (např. Ubuntu, Mint, OpenSUSE, Debian, atp.). Server, na který budete Vaši práci odevzdávat a který ji otestuje, má nainstalovaný operační systém Debian GNU/Linux 9.1 Stretch s jádrem verze 3.2.78-1 x86_64 a s překladačem gcc 6.3.0.

sistem.exe "šel pes do lesa a potkal dlažební kostku"-msf=15
sistem.exe bezdomovec -msf=5

Úkolem cašeho programu tedy je v režimu učení vytvořit databázi kořenů (textový soubor, viz. dále) a v režimu zpracování slov vypsat kořen každého slova ze vstupní sekvence. Jak kořeny slov najít bude naznačeno níže.

V případě, že nebude programu předán parametr prvního nebo druhého uvedeného typu, vypište kratké chybové hlášení (anglicky) a oznamte chybu operačnímu prostředí pomocí nenulového návratového kódu[2]. Pokud bude stemmeru při prvním spuštění předáno slovo nebo sekvence slov, ukončete jej chybovým stavem (a krátkým vysvětlujícím hlášením), indukujícím, že nedošlo k předchozímu vytvořenídatabáze kořenů slov, a tudíž není možné u slov ze sekvence určit jejich kořeny.

Hotovou práci odevzdejte v jediném archivu typu ZIP prostřednictvím automatického odevzdávacího a validačního systému. Postupujte podle instrukcí uvedených na webu předmětu. Archiv nechť 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 TEX, resp. LATEX. Bude-li některá z částí chybět, kontrolní skript Vaši práci odmítne

[2]Stejně tak čiňte i v případě jiných chyb. napr. nesprávném formátu vstuního souboru a podobně

2 Analýza úlohy

2.1 Učení stemmeru

2.1.1 Vytvoření frekvenčního slovníku

Nejprve je potřeba vytvořit frekvenční slovník slov ze vstupního souboru. Při vkládání nového slova je potřeba zjistit, zda slovník už slovo obsahuje, pokud ne tak se vloží do slovníku, jinak se pouze zvýší jeho počet ve slovníku. Jedna z možností vytvoření frekvenčního slovníku je spojový seznam. Algoritmická složitost hledání prvku je O(n) a prvku O(1), takže výsledná algoritmická složitost vkládání slova je O(n), kde n je počet slov ve slovníku. Další možnost je vytvoření frekvenčního slovníku pomocí rozptylové tabulky s řešením kolizí zřetězením. Algoritmické složitosti jsou stejné jako u spojového seznamu, takže výsledná algoritmická složitost je O(n), ovšem při správné volbě hašovací funkce je n mnohem menší než při implementaci spojovým seznamem. Frekvenční slovník lze vytvořit také pomocí trie. Trie má algoritmickou složitost hledání i vkládání prvku O(n), výsledná algoritmická složitost vkládání slova je tedy O(n), kde n je velikost hledaného slova. Z důvodu intenzivního vyhledávání frekvenčního slovníku má proritu rychlost hledání prvku, proto jsem se rozhodl pro vytvoření frekvenčního slovníku pomocí trie.

2.1.2 Hledání a ukládání kořenů slov

Kořeny získáme nalezením nevětšího společného podřetězce každých dvou slov frekvenčního slovníku. Největší společný podřetězec dvou slov můžeme najít například tak, že porovnáme všechny možné kombinace podřetězců obou slov, ovšem algoritmicka složitost takového algoritmu je $O(m^2*n^2)$, kde m je velikost prvního slova a n je velikost druhého slova. Mnohem lepší možností je použití algoritmu lcs, který má algoritmickou složitost pouze O(m*n), kde m je velikost prvního slova a n je velikost druhého slova.

Kořeny budeme ukládat do databáze kořenů. Vzhledem k tomu, že databázi bude potřeba často prohledávat, tak jsem se rozhodl tuto databázi vytvořit pomocí trie. Důvody tohoto rozhodnutí jsou stejné jako v předchozí sekci. Databáze je poté uložena do datového souboru.

2.2 Zpracování slov stemmerem

2.2.1 Vytvoření databáze kořenů a vyhodnocení

Databázi kořenů z datového souboru jsem vytvořil pomocí trie, opět z důvodů popsaných v kapitole 2.1. Do databáze kořenů jsou ukládány pouze kořeny, které mají počet výskytu větší nebo roven parametru msf.

2.2.2 Vyhodnocení kořenů slov

Ze slova se vybereme nejdelší podřetězec, který je obsažen v databázi kořenů. Pokud je nalezeno několik nejdelších řetězců stejné délky, vybereme ten, který je podle ASCII hodnoty nejmenší.

3 Implementace

3.1 Datové struktury

3.1.1 Trie

Jedinou datovou stukturou v této práci je **trie**. Každý uzel trie v sobě uchovává počet výskytu slova, který uzel představuje, dále seznam uzlů, které představují následující písmeno a rodiče tohoto uzlu.

3.2 Algoritmy

3.2.1 LCS

Algoritmus lcs (Longest common subsequence) je algoritmus, který slouží k nalezení nejdelší společné sekvence.

3.3 Moduly

3.3.1 word_operations

Tento modul zajišťuje práci se slovy, jako alokování místa pro slovo nebo hledání nejdelšího společného řetězce dvou slov.

3.3.2 trie

Tento modul zajišťuje správu trie. Obsahuje funkcejako hledání slov a vkládání slov do trie, vytváření trie ze souboru.

3.3.3 main

Tento modul spojuje všechny části programu dohromady. Obsahuje také funkce pro ověření vstupních parametrů, výpis nápovědy a výpis výstupu.

4 Uživatelská příručka

4.1 Přeložení programu

4.1.1 Linux

Pro přeložení je potřeba mít nainstalovaný překladač gcc a program make.

Před přeložením se přesuneme do složky se souborem Makefile a příkazem make se program přeloží a vznikne soubor sistem.exe.

4.1.2 Windows

Pro přeložení je potřeba mít nainstalovaný překladač Visual C++ Build Tools

Před přeložením se přesuneme do složky se souborem Makefile.win a příkazem nmake /f Makefile.win se program přeloží a vznikne soubor sistem.exe.

4.2 Spuštění programu

Program spustí příkazem **./sistem.exe <corpus-file|["]word sequence["]> [-msl] [-msf]**,

kde **<corpus-file>** je cesta ke vstupnímu souboru pro režim učení stemmeru, pokud je zadán název souboru bez přípony, předpokládá se, že se jedná o textový soubor **<word sequence>** sekvence slov k rozpoznání kořenů nepovinný parametr **msl** určuje minimální délku vytvářených kořenů při vytváření kořenů ze slov, pokud není zadán, je tato hodnota nastavena na 3 nepovinný parametr **msf** určuje minimální počet výskytu kořene v databázi kořenů, pokud není zadán, je tato hodnota nastavena na 10

Pokud není zadán žádný parametr je vypsána krátká nápověda, jak program používat.

```
dubs@LAPTOP-L5ACMKKL:/mnt/c/Users/Dubs/CLionProjects/Stemmer$ ./sistem.exe
No parameter given!

Usage:
 sistem.exe <corpus-file|["]word-sequence["]> [-msl=<positive integer>][-msf=<positive integer>]

-msl = Minimum Stem Length - optional parameter, usable only with corpus-file parameter, default value = 3
-msf = Minimum Stem Frequency - optional parameter, usable only with word-sequence parameter, default value = 10

Example with corpus-file:
sistem.exe Corpus.txt -msl=5

Example with word-sequence:
sistem.exe "Hello world!" -msf=15
dubs@LAPTOP-L5ACMKKL:/mnt/c/Users/Dubs/CLionProjects/Stemmer$
```

Pokud je zadán 1 parametr, který je existujícím souborem s textem, tak se v aktuální složce vytvoří soubor stems.dat.

```
dubs@LAPTOP-L5ACMKKL:/mnt/c/Users/Dubs/CLionProjects/Stemmer$ ./sistem.exe "dasenka_cili_zivot_stenete"
dubs@LAPTOP-L5ACMKKL:/mnt/c/Users/Dubs/CLionProjects/Stemmer$
```

Pokud je zadán 2. parametr ve špatném formátu, je opět vypsána nápověda.

```
dubs@LAPTOP-L5ACMKKL:/mnt/c/Users/Dubs/CLionProjects/Stemmer$ ./sistem.exe "dasenka_cili_zivot_stenete" -ms=a
Usage:
 sistem.exe <corpus-file|["]word-sequence["]> [-msl=<positive integer>][-msf=<positive integer>]

-msl = Minimum Stem Length - optional parameter, usable only with corpus-file parameter, default value = 3
-msf = Minimum Stem Frequency - optional parameter, usable only with word-sequence parameter, default value = 10

Example with corpus-file:
sistem.exe Corpus.txt -msl=5

Example with word-sequence:
sistem.exe "Hello world!" -msf=15
dubs@LAPTOP-L5ACMKKL:/mnt/c/Users/Dubs/CLionProjects/Stemmer$
```

Pokud je zadána sekvence slov a ještě neproběhla fáze učení stemmeru, tak se vypíše chybové hlášení.

```
dubs@LAPTOP-L5ACMKKL:/mnt/c/Users/Dubs/CLionProjects/Stemmer$ ./sistem.exe "šel pes do lesa a potkal dlažební kostku"
stems.dat not found, it is required execute stemmer with corpus file for the first time

dubs@LAPTOP-L5ACMKKL:/mnt/c/Users/Dubs/CLionProjects/Stemmer$
```

Pokud je zadána sekvence slov a již existuje soubor stems.dat, tak se vypíší jednotlivá slova a k nim přiřazené kořeny, pokud se nenajde žádný kořen, přiřadí se slovu 0.

```
dubs@LAPTOP-L5ACMKKL:/mnt/c/Users/Dubs/CLionProjects/Stemmer$ ./sistem.exe "šel pes do lesa a potkal dlažební kostku"
šel -> 0
pes -> 0
do -> 0
lesa -> 0
a -> 0
potkal -> kal
dlažební -> dla
kostku -> kos
dubs@LAPTOP-L5ACMKKL:/mnt/c/Users/Dubs/CLionProjects/Stemmer$
```

5 Závěr

Zadání se povedlo splnit. Program správně přiřazuje kořeny ke slovům dle zadání. Jeden z větších problémů, které jsem řešil byl ten, že jsem si neuvědomil, že ze vstupního souboru načítám **char**, který má rozsah -127 až 128, ale já jsem počítal s tím, že dostávám **unsigned char**, který má rozsah 0 - 255.