Semestrální práce z KIV/PC

JEDNODUCHÝ STEMMER

Mukanova Zhanel
A16B0087P
mukanova@students.zcu.cz

21. 12. 2018

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. Celé zadání je na: *https://www.kiv.zcu.cz/studies/predmety/pc/doc/work/sw2018-03.pdf*

2 Analýza úlohy

Máme vytvořit program, který dokáže vygenerovat frekvenèní slovník ze vstupního souboru. Dalším krokem program ze frekvenčního slovníku vytvoří slovník kořenů, podle kterého bude potom nachazet koreny zadaných slov. Ze zadání je jasné, že potřebujeme naimplementovat v ANSI C vhodnou datovou strukturu pro reprezentaci slovníku. Do které by se ukládaly slova a jejich počet. Měla jsem na výběr následující datové struktury:

- **Hash-table**
  Pokud bych chtěla pouze uložit slova a pak zkontrolovat, zda je mezi nimi hledané slovo nebo ne, pak by standardní hash tabulka byla rozumná volba. Pokud by počet položek seznamu byl znám předem, použila bych ideální hash pro dosažení nejvyššího výkonu a optimální velikosti uložených dat.

- **Stromy**
  Prefix tree je vhodná datová struktura v pripade rychleho vyhledávání předpony slov, i když to může být trochu neefektivní z hlediska velikosti uložených souborů. Tato datová struktura také podporuje rychlé vkládání a odstraňování. Navíc Prefix tree má všechna slova seřazená podle abecedy, co v hash-tabulkách chybí.

- **Dalsi stromy**
  Kdybychom chtěli použít slovník pro operace, jako je kontrola pravopisu, kde je potřebné najít slova, která nejsou podobná ostatním, je BK-strom skvělou volbou.

Další problém je porovnání každého slova slovníku s ostatními. Kvůli tomu

ze iterační funkce v daných ADT jsou obvyklé rekurzivní, vhodnými datovými strukturami pro porovnání slovníku by byly:

- **Obecné pole retezcu**
  Je vhodné kvůli snádnému implementovani. Ale má pomalou iterační funkci.
- **Linked list**

Následujícím krokem programu je samotné porovnání slov.

- **Longest Common Subsequence (LCS)**
  LCS algoritmus je jedním ze způsobů jak posuzovat podobnost mezi dvěma řetězci. Ale neurčuje koreny, určuje jenom "společná pismena" slov.
- **Longest Common Substring (LCS)**
  Algoritmus zváži všechny podřetězce prvního slova a pro každý podřetězec zkontroluje, zda existuje podřetězec ve druhém slově. Vyhledá podřetězec maximálni délky.

3 Popis implementace

Režim učení
Začala jsem s tím, že jsem ošetřila vstupné parametry. Pomocí regular expression jsem určovala kam patří první parametr (je-li to cesta nebo sekvence slov). Po kontrole parametrů jsem začala implementovat datovou strukturu Trie. Vybrala jsem tuto strukturu kvůli rychlému zpracování a uložení slov podle ASCII hodnot, co jsem potřebovala kvůli zadání. Pomocí obecného parsování každé řádky vstupního souboru jsem vytvořila frekvenční slovník slov, který jsem uložila do souboru dictionary.txt. Dalším krokem bylo projít celý ten slovník a vytvořit slovník kořenů. Ale procházet skrz celý Trie dvakrát by byla velmi náročná akce. Takže jsem jedním procházením Trie vytvořila už seřazeny LinkedList pomocí kterého jsem porovnávala každá dva slova v slovníku. Na hledání kořenů jsem použila algoritmus LCS (longest common substring). Každý kořen jsem ukládala zase do nového Trie a potom už seřazeny slovník jsem ukládala do souboru stems.dat.

Režim zpracování slov
Na začátku jsem určovala zda li první parametr je to frekvence slov. Druhým krokem bylo vytvořit Trie ze stems.dat. Potom jsem brala každé slovo a

pomoci algoritmu LCS jsem procházela každé slovo v Trie. Každý nejdelší kořen jsem ukládala do bufferu. Kořenem slova je poslední nejdelší kořen z bufferu.

4 Uživatelská příručka

Semestrální práce je vytvořena programovacím jazykem ANSI C a lze spustit různými způsoby. Program byl vytvořen v distribuce Linuxu a proto následující uživatelská bude popisovat způsob spouštení pomoci Terminalu.

1. Příkazem *"cd path-to-program"* přijdeme do složky s programem.
2. Pro kompilace a generování spustitelného souboru použijeme příkaz *"make"*, který vytvoří spustitelný soubor sistem.exe
3. Pro spuštění programu v režimu učení použijeme příkaz *"./sistem.exe path/to/corpus/file -msl=[cislo]"*. Parametr *"-msl=[cislo]"* není povinný. Určuje minimální délku kořene. Implicitní minimální délka kořene 3 znaky. Program vytvoří soubor stems.dat, který bude obsahovat kořeny slov.
4. Pro spuštění programu v režimu zpracování slov použijeme příkaz *"./sistem.exe "sekvence slov" -msf=[cislo]"*. Parametr *"-msf=[cislo]"* není povinny. Určuje minimální počet výskytů příslušného kořene. Program vypíše do Terminalu zadane slovo a jeho koren.

5 Závěr

Dane zadani od zacatku me prislo jednoduchym. Ale behem implementovani jsem se dozvedela že pracování s řetězci v jazyce C je docela těžká akce. Prvním problémem se kterým jsem se setkala kódování. Programovací jazyk C umí pracovat jen s ASCII hodnoty od 0 do 127. Diky tomuto zadani jsem se seznámila s různými algoritmy a kódováním. Tento úkol přidal mi cenné zkušenosti v jazyce C. Zadani semestralni prace se mi moc libilo. Kvuli sve obtiznosti jsem se musela vsechno prostudovat a použít své znalosti ve svém programu. Myslím si že jsem tu praci zvládla a semestrální práci považuji za splněnou.