Obsah

1 Zadání **2**
1.1 Zadání 2
1.2 Vstupy 2
1.3 Formáty vstupních a výstupních souborů a vstupu z konzole . 3

2 Analýza úlohy **4**
2.1 Režim učeni 4
2.1.1 Datové struktury 4
2.1.2 Algoritmus hledaní kořenu 5
2.2 Režim zpracovaní slov 5

3 Popis implementace **6**
3.1 Použité struktury 6
3.1.1 Trie 6
3.2 Kontrola vstupních parametrů 6
3.3 Načítání korpusu 7
3.4 Zpracovaní slov a vytváření databázi 7
3.5 Hledání kořenů v režimu zpracování slov 7

4 Uživatelská příručka **8**
4.1 Překlad v Linux a Windows 8
4.2 Spuštění programu 8

5 Závěr **9**

Kapitola 1

Zadání

1.1 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 reřimu 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).

1.2 Vstupy

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

Program může být spuštěn se dvěma nepovinnými parametry:

-msl — Nepovinný parametr -msl=<celé číslo> specifikuje 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 použžitelný 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 alespoň tolikrát, kolik je určeno tímto parametrem, nepoužije se při zpracování slov, tj. stemmer nemůůže u žádného

zpracovávaného slova oznámit, že tento kořen je kořenem 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.

1.3 Formáty vstupních a výstupních souborů a vstupu z konzole

Program bude pracovat výhradně s 1-bytovým (8mi-bitovým) kódováním znaků: Neuva- žujte žádné kódování znaků národních abeced, které používá více než jeden byte na znak nebo proměnný poèet bytů, tj. žádná kódování z rodiny Unicode (UTF-8, UTF-16, UCS-2, apod.).

Vzhledem k tomu, že ukázkový korpus na webu je v èetině, připadá v úvahu prakticky pouze kódování Windows/CP-1250 nebo ISO-8859-2 (ISO Latin 2). Stemmeru by to mělo být úplně jedno, pokud dodržíte pravidlo jeden byte kóduje jeden znak. Bude-li korpus užitý pro natré- nování složen např. z německých textů, tj. znaky budou kódovány podle Windows/CP-1252, pak i výstupní soubor s databází kořenů bude kódován CP-1252. V režimu zpracování slov pak bude stemmer urèovat kořeny jen u slov předaných v tomto kódování. Bude-li na vstupu slovo obsahující znaky národních abeced v jiném kódování, než ve kterém je databáze kořenů, pak pochopitelně kořen nebude (protože nemůže být) nalezen. Jednodue řeèeno, programátor do kódování znaků nemusí vůbec zasahovat...

Stejně tak při vstupu slova nebo sekvence slov z konzole můžete předpokládát, že uživatel má konzoli správně nastavenou na přísluné kódování znaků národních abeced tak, aby bylo shodné s kódem použitým v korpusu (a tedy i v databázi kořenů). Při testování programu bude použit pouze anglický korpus (tj. 7-bit ASCII) a èeský v kodování Win-1250.

Kapitola 2

Analýza úlohy

Prohrám bude pracovat v několika režimech. Proto úlohu jde rozdělit na dvě části. První část bude režim učeni, druha část bude režim zpracovaní slov. Se začátku v režimu učeni prohrám dostane na vstup soubor s textem a vypracuje z tohoto textu databázi kořenu slov. V režimu zpracovaní slov prohrám dostane na vstup slova a urči jim kořeni z databázi.

2.1 Režim učeni

Na ukládaní slov musím vybrat datovou strukturu a na hledaní kořenu slov vhodný algoritmus.

2.1.1 Datové struktury

Jsou hodně datových struktur, které jde vybrat pro řešeni této úlohy, takové jako BSD, stek, lineární spojový seznam, hašovací tabulka, trie atd. Pro řešeni úlohy budu vybírat mezi lineárním spojovým seznamem, hašovací tabulkou a trie.

Strom je struktura dat sestávající z uzlů. Má následující charakteristiky: každý strom má kořenový uzel (v horní části); kořenový uzel má nulu nebo více dětí; každý dětský uzel má nula nebo více podřízených uzlů atd. Binární vyhledávací strom má ještě dvě charakteristiky navíc: každý uzel má až dvě děti (potomci), pro každý uzel jsou jeho levé potomky menší než aktuální uzel, který je menší než prve potomci.

Hash tabulka je datová struktura, která implementuje mapové rozhraní, které umožňuje ukládat pár klíč / hodnota. Používá funkci hash pro výpočet indexu v poli, který lze použít k nalezení požadované hodnoty.

Každý uzel v trie obsahuje jedno písmeno slova. Když chceme vypsat

slovo, tak jsme větvemi trie a vypisujeme jedno písmeno najednou. Kroky se začínají rozcházet, když se pořadí písmen liší od jiných slov ve stromu nebo po skončení slova. Každý uzel obsahuje písmeno a logickou hodnotu, která označuje, zda je uzel posledním uzlem ve slově.

Já budu používat trie. Před BST má tu výhodu, že pokusy jsou více prostorově efektivní, když obsahují velký počet krátkých klíčů, protože se mezi sobou klíči s běžnými počátečními subsekvencimi shodují. Před Hash Table má výhodu, že má tendenci být v průměru rychlejší při vkládání než hash tabulky, protože tabulka musí znovu sestavit svůj index. Trie mnohem lépe vymezuji nejhorší časové náklady. Tím, že se vyhnete funkci hash, budete rychlejší než hash tabulky pro malé klíče, jako jsou celá čísla a ukazatele. A také trie může poskytnout abecední uspořádání položek pomocí klíče.

Datová struktura	Přístup	Vyhledávání	Vkládání	Mazání
Hašovací tabulka	O(n)	O(1)	O(1)	O(1)
BST	O(log(n))	O(log(n))	O(log(n))	O(log(n))
Trie	O(n)	O(n)	O(n)	O(n)

Tabulka 2.1: Porovnaní rychlosti datových struktur

2.1.2 Algoritmus hledaní kořenu

Na hledaní kořenu slov budu používat algoritmus LCS (Longest Common Substring). Nejdelší podřetězec, je to nejdelší posloupnost písmen přicházejících za sebou ve dvou slovech, která mají být porovnána. Nejlehčí řešeni této úlohy je brát vdechne podřetězce jednoho slova a hledat jich v druhem slově. Při tomto postupe musíme vždy kontrolovat délku podřetězce. Při programovaní dynamicky je možnost najit nejdelší podřetězec za čas O(mn). Můžeme naprogramovat dvě tabulky a umisťovat tam veškeré délky pro dva slova.

2.2 Režim zpracovaní slov

V režimu zpracovaní slov dostanu řetězec na vstup a budu muset ho rozdělit na slova. Pak budu určovat kořene pro této slova s načtené databázi. V téhle části semestrální prací nebudu potřebovat speciální datové struktury a algoritmy. Budu používat knihovny poskytnuté ANSI C.

Kapitola 3

Popis implementace

Program je rozdělen do několika hlavičkových a zdrojových souborů. Soubor main.c zkontroluje uživatelem předané parametry a na základě předaných parametrů nastaví proměnné v programu a rozhodne, jestli se stemmer spustí v režimu učení nebo v režimu zpracování slov. O načítání dat ze vstupního souboru a načítání databáze kořenů se stará knihovna *writereader*. Knihovna *arrworker* obsahuje funkce pro úpravua práci s poli. Knihovna *const* obsahuje konstanty které použiju ve vice než jedné knihovně. Knihovna *trie* obsahuje funkce pro praci s trie.

3.1 Použité struktury

3.1.1 Trie

Trie se používá k ukládání slov. Trie trik je implementován jako řada referencí. Každý prvek trie představuje písmeno abecedy (256 symbolů ascii). Toto je podřízený dopis pro každé písmeno.

3.2 Kontrola vstupních parametrů

Po spuštění programu je zkontrolován počet parametrů, pokud je parametrů málo nebo naopak moc, program vypíše chybovou hlášku a skončí. Pokud je zadán některý z nepovinných parametrů, program zkontroluje hodnotu. Poté program zkontroluje, v jakém režimu bude fungovat učení nebo zpracovaní slov. Když učeni, tak jestli první předaný parametr obsahuje příponu .txt, pokud ji neobsahuje, tak přidá ji k řetězci. V případě chyby vypíše chybovou hlášku a skončí.

3.3 Načítání korpusu

Program přečte celý soubor znak po znaku. Při čtení každého znaku program zjisti, zda je tento znak písmeno nebo ne. Pokud znak není písmeno, program se domnívá, že slovo skončilo, a pokud slovo není prázdné, pak ho do databáze vloží. Každé písmeno je zkontrolováno, zda je velké nebo ne. Pokud je písmeno napsáno velkými písmeny, je překládáno do malých písmen. Všechny znaky, které jsou písmeny, jsou v poli v knihovně *writereader*. Každé nalezené slovo se přidává do trie.

3.4 Zpracovaní slov a vytváření databázi

Pro hledání kořenů byl implementován algoritmus LCS, který je popsán v kapitole 2.1.2.

Všechna slova jsou z trie, přesunuta do pole kvůli s nadnějšímu procházení. Toto pole je procházeno dvojitým cyklem a pro každé dvě slova je použit algoritmus pro hledání kořenů. Všechny nalezené kořene požadované doložky jsou vkládány do nového trie. Potom trie uloženo do souboru *stems.dat*.

3.5 Hledání kořenů v režimu zpracování slov

Při spuštění stemmeru v režimu zpracování slov je předaná skvence slov upravena. Tuto sekvenci slov dělím na jednotlivá slova. Načítám databázi do dvou poli. Do jednoho poli načítám kořene, do druhého frekvence kořenů. Pak pro každé slovo procházím databázi kořenů a hledám nejdelší kořen, který se celý nachází ve zkoumaném slově jako jeho podřetězec a frekvence vyhovuje požadavkům uživatele. Tento kořen pak určuju jako kořen zkoumaného slova. Určené kořene zapisuju do pole, pak toto pole tisknu.

Kapitola 4

Uživatelská příručka

4.1 Překlad v Linux a Windows

Pro překlad aplikace v systému Windows je připraven soubor makefile.win. Pro přeloženi programu jde použit příkaz *"make -f makefile.win"* v adresáře se zdrojovými kódy.

Pro překlad aplikace v systému Linux je připraven soubor makefile. Pro přeloženi programu jde použit příkaz *"make"* v adresáře se zdrojovými kódy.

Ve Windows a Linux musí byt nainstalovaný překladač jazyka C.

4.2 Spuštění programu

Program může být spuštěn v režimu učeni a režimu zpracovaní slov.

V režimu učeni prohrám může byt spuštěn tímto příkazem:

sistem.exe <corpus-file> [-msl=<celé číslo>]

-msl — Nepovinný parametr -msl=<celé číslo> specifikuje minimální délku kořene slova.

V režimu zpracovaní slov prohrám může byt spuštěn tímto příkazem:

sistem.exe <c["]word-sequence["]> [-msf=<celé číslo>]

-msf — Nepovinný parametr -msf=<celé číslo> určuje minimální počet výskytů příslušného kořene.

Kapitola 5

Závěr

Vytvořený program správně vyhledá a třídí kořeny slov. Vytvoří potřebné soubory a uvolní obsazenou paměť.

Během analýzy práce a vytvoření programu bylo analyzováno mnoho datových struktur a zkoušeno několik vyhledávacích algoritmů pro nejdelší podřetězec. Nejvhodnější byly zvoleny pro psaní programu, tj. což nevyžaduje obrovské množství času na vývoj a umožňuje program pracovat poměrně rychle.

Beh programu na serveru trvá trohu déle než dvě minuty.