Extrakce N-gramů z rozsáhlých textů

Zdeněk Češka[1], Ivo Hanák[1], Roman Tesař[1]

[1]Katedra informatiky a výpočetní techniky, Fakulta aplikovaných věd,
Západočeská univerzita v Plzni, Univerzitní 22, 306 14 Plzeň, Česká republika
{zceska, hanak, romant}@kiv.zcu.cz

Abstrakt. V úlohách zpracování přirozeného jazyka jsou k reprezentaci textových dokumentů nejčastěji používána jednotlivá slova. Celkové výsledky lze však často vylepšit použitím dalších, sofistikovanějších položek. Mezi ně patří i N-gramy, pro jejichž extrakci byly publikovány algoritmy založené na různých principech. Existující techniky však nejsou primárně určeny pro zpracování velkého objemu dat, což je v současné době zásadní požadavek. V tomto článku prezentujeme algoritmus pro extrakci N-gramů z rozsáhlých textových korpusů. Srovnání s jinými přístupy naznačují, že naše řešení dosahuje výrazně lepších výsledků s ohledem na čas a množství zpracovaných dat.

Klíčová slova: rozsáhlé korpusy, dávkové zpracování, extrakce N-gramů.

1 Úvod

S využitím N-gramů, obecně definovaných jako sled N po sobě jdoucích položek z dané sekvence, se můžeme setkat v mnoha oblastech zpracování přirozeného jazyka. Slovní N-gramy jsou používány při klasifikaci textu k obohacení základního modelu dokumentu založeného na jednotlivých slovech [3,7]. N-gramy složené z písmen slouží k rozpoznání jazyka textových dokumentů [1]. A v neposlední řadě pak analýza biologických textů zahrnuje extrakci N-gramů pro nalezení četností proteinových sekvencí [4].

V [5] a později v [9] byl prezentován algoritmus pro extrakci N-gramů založený na sufixových polích, který je v současnosti často využíván s různými modifikacemi v různých programech. Další možností je použití sufixového stromu [8]. Dříve prováděné experimenty [6] však prokázaly, že algoritmus založený na sufixových polích překonává dvě různé implementace vycházející ze sufixových stromů nejen svou rychlostí, ale i paměťovými nároky. K extrakci N-gramů byl také využit invertovaný index [8] a modifikovaný Apriori algoritmus [3,7].

Žádný z výše zmíněných přístupů ovšem není primárně určen ke zpracování velkého objemu dat přesahující rámec dostupné operační paměti. Toto omezení může být zásadní pro dosažení lepších výsledků v různých úlohách zabývajících se zpracováním rozsáhlých textových dat.

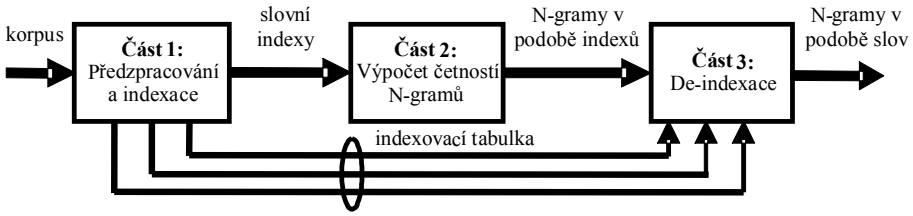
Možným řešením tohoto problému je využití více počítačů pracujících paralelně, což úspěšně využívá společnost Google společně s jejich programovým nástrojem MapReduce [2], který zajišťuje paralelní distribuci a zpracování dat. Nicméně ne každý disponuje sadou počítačů s prostředím pro paralelní výpočet. Pořizovací cena tohoto řešení může být dosti vysoká a hrát významnou roli při volbě vhodného algoritmu.

V tomto článku prezentujeme alternativní řešení dostupné komukoliv. Námi navržený algoritmus je určen pro extrakci N-gramů s využitím jediného počítače nezávisle na jeho výkonu, který dovoluje zpracovávat velmi rozsáhlá textová data bez potřeby speciálního hardwarového vybavení. Nad tímto algoritmem jsme vytvořili nástroj Teraman, který je popsán v sekci 2. Experimenty a testování následují v sekci 3 a shrnutí celého článku je uvedeno v sekci 4.

2 Jak algoritmus pracuje

Po počáteční analýze jsme se rozhodli vytvořit mnohem pokročilejší nástroj, který bude schopen extrahovat slovní N-gramy z velmi rozsáhlých textových dat, bez ohledu na výkon počítače. To vyústilo v nasazení techniky dávkového zpracovávání, která rozdělí data na menší části a každou zpracuje ve volné operační paměti.

Pro dosažení maximálního výkonu je vhodné nezpracovávat přímo textové řetězce různé délky. Z tohoto důvodu převádíme jednotlivá slova v textu na unikátní 32-bitová celá čísla, která zabírají méně operační paměti a dovolují rychlejší zpracování.



Obr. 1. Přehled hlavních částí algoritmu

Teraman se skládá ze tří základních částí, jak je zobrazeno na obrázku 1. Pro lepší škálovatelnost jsou všechny části navrženy jako nezávislé moduly, které mohou být samostatně využity v jiných aplikacích. V dalších podsekcích následuje detailní popis jednotlivých částí algoritmu.

2.1 Předzpracování a indexace

Předzpracování a indexace je důležitý krok pro mnoho algoritmů zabývajících se zpracováním textů. V našem případě se jedná o transformaci vstupního korpusu na reprezentaci čísel, která je rozdělena do pěti fází, zobrazených na obrázku 2.

Fáze 1. Zpočátku se alokují dvě různá pole. Jedno z polí udržuje textové znaky načítané z korpusu. Obsah *pole znaků* se může od načítaného korpusu lišit v závislosti na aplikovaných filtrech, které budou popsány později. Do druhého *pole ukazatelů* jsou zaneseny počátky slov ukazující do *pole znaků*.

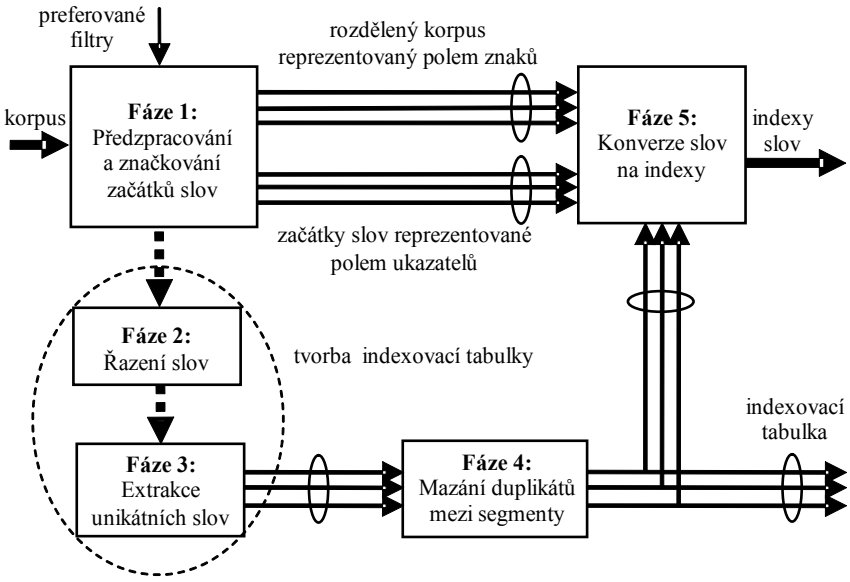
Z důvodu rozličné průměrné délky slov pro různé jazyky, je alokace obou polí provedena v jistém poměru s respektováním maximální dostupné paměti. Pro první alokaci je použit nejvhodnější poměr, který byl empiricky stanoven. U dalších bloků,

které vznikají dávkovým zpracováním, se vždy porovná naposledy použitý poměr s nově vypočteným, v závislosti na aktuálním využití obou polí. Pokud standardní odchylka obou poměrů přesáhne stanovený limit, provede se realokace s novým poměrem. Tento postup je zaměřen na efektivní využití dostupné paměti počítače.

Poté, co jsou obě pole připravena, začneme načítat znaky ze vstupního korpusu do *pole znaků*. Během načítání se zkoumají znaky a v případě, že je nalezen začátek nového slova, zapíše se jeho pozice do *pole ukazatelů*. Slovo je v tomto případě definováno jako posloupnost znaků, které je z obou stran ohraničeno bílými znaky. Pokud bylo během načítání dosaženo konce libovolného z obou polí, uloží se obě pole na disk a pro další pokračování se zaznamená poslední načtená pozice z korpusu.

V okamžiku načítání textových znaků z korpusu, lze text modifikovat až třemi různými filtry. Jedná se o filtr pro převod textu na malá písmena (MP), nahrazení interpunkce (NI) a nahrazení bílých znaků (NBZ). Filtry MP a NI jsou volitelné a záleží na volbě uživatele. Filtr NBZ je aplikován vždy a jeho úkolem je přepsat všechny nepodstatné bíle znaky (mezery, tabulátory a konce řádek) na nulový znak '\0', čímž se jednoznačně oddělí jednotlivá slova. Tento postup je velmi důležitý pro dosažení vysokého výkonu při zpracování, z důvodu nahlížení na slova ve *znakovém poli* jako nulou ukončené řetězce. Podobnou funkci má též NI filtr, který nahrazuje nulovým znakem veškerou interpunkci v textu. Jedinou odlišností je zápis nulového ukazatele do *pole ukazatelů*. N-gramy obsahující tuto značku nejsou ve druhé části algoritmu (sekce 2.2) extrahovány.

Časová složitost této fáze je $O(n)$, kde n je celkový počet slov v korpusu.



Obr. 2. Schéma předzpracování a indexace

Fáze 2. Druhá fáze je úzce svázána s první fází. Předtím než se zahodí obsah obou polí, aby se mohl zpracovávat další blok, vytvoří se z něho jeden ze segmentů in-

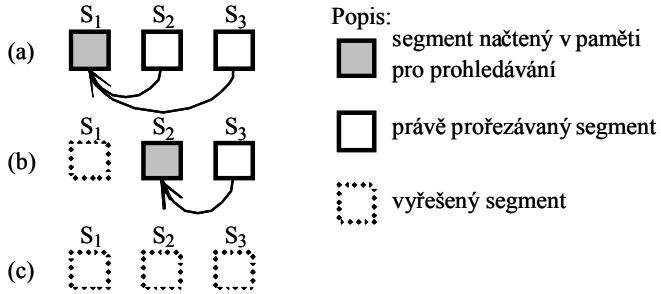
dexační tabulky. Pro získání tohoto segmentu nepřímo seřadíme *pole ukazatelů* dle slov, na která ukazují, ve vzestupném pořadí. Tento krok musí být proveden po uložení obou polí na disk, předtím než byla data přepsána následujícím blokem.

Složitost této fáze je $O(n \cdot log_2(n / k))$, kde n je celkový počet slov v korpusu a k je počet bloků vytvořených dávkovým zpracováním.

Fáze 3. V další fázi se jedním průchodem prozkoumá *pole ukazatelů* a rozdílná slova, jež jsou po seřazení umístěna vedle sebe, se uloží do souboru představující jeden segment indexační tabulky. Nulové ukazatele odpovídající interpunkci jsou v průběhu procesu ignorovány. Časová složitost pro průchod všech polí, vznikající dávkovým zpracováním, a uložení indexační tabulky je $O(n)$, kde n je celkový počet slov v korpusu.

Fáze 4. V důsledku dávkového zpracování korpusu a omezeného rozsahu na jeden segment, uvažovaného ve třetí fázi, mohou některé segmenty obsahovat duplikáty, znemožňující jednoznačnou indexaci slov. Proto je nutné porovnat všechna slova uvnitř k vytvořených segmentů. Porovnání je provedeno načtením segmentu i do paměti, kde $i \in <1, k>$, a pak postupným porovnáním všech slov z následujícího segmentu j, kde $j \in <i+1, k>$.

Pro urychlení využíváme faktu, že slova byla ve druhé fázi seřazena dle abecedy, takže můžeme využít algoritmus binárního vyhledávání. Postupně načítaná slova ze segmentu j jsou vyhledávána v načteném segmentu i. Pokud je v segmentu j nalezeno stejné slovo jako v paměti, je ze segmentu j smazáno. Příklad odstraňování duplikátů ze třech segmentů je naznačen na obrázku 3.



Obr. 3. Příklad mazání duplikátů

Po dokončeném procesu fáze 4 může být každé slovo jednoznačně identifikováno indexem, který lze vypočíst dle následující rovnice

$$index_w = p_{w,m} + \sum_{j=1}^{m-1} c_j \ , \qquad (1)$$

kde $p_{w,m}$ je pořadí slova w v segmentu m uvažované od začátku segmentu a c_j je celkový počet slov v segmentu j.

Časová složitost této fáze závisí na pravděpodobnostním rozložení slov uvnitř korpusu. Avšak podařilo se nám odvodit, že výsledná složitost se musí nacházet mezi následujícími dvěma limitními složitostmi $O(t \cdot log_2(t / k))$ a $O(k \cdot t \cdot log_2(t / k))$, kde t

je celkový počet slov obsažený ve všech extrahovaných segmentech indexační tabulky a *k* je počet všech segmentů.

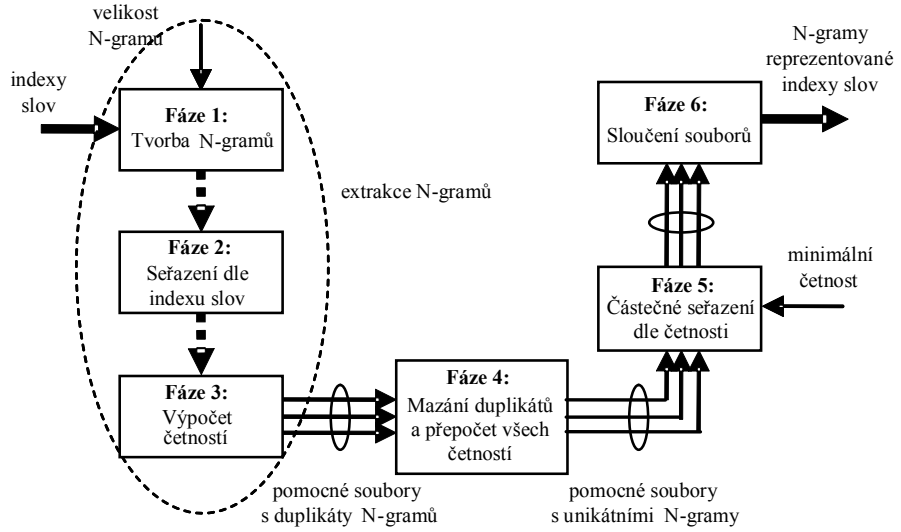
Fáze 5. V poslední fázi nahradíme slova čísly dle indexační tabulky. Do paměti vždy načteme *i*-té *pole ukazatelů* a *i*-té *pole znaků*, vytvořené a uložené fází 1, a s ohledem na zbývající operační paměť načteme několik segmentů indexační tabulky. Následně procházíme ukazatele na počátky slov a s využitím binárního vyhledávání prozkoumáváme načtené segmenty indexační tabulky. Pokud je nalezeno odpovídající slovo, spočte se jeho index dle rovnice (1) a zapíše v záporné hodnotě namísto původního ukazatele do *pole ukazatelů*.

V dalším kroku, kdy se uvolní předchozí segmenty a načte se několik nových, se proces nahrazování opakuje, pouze jsou uvažovány ukazatele s *hodnotou* > *0*. Ukazatele s *hodnotou* < *0* představují již vyřešená a neindexovaná slova. Interpunkce reprezentovaná *hodnotou* = *0* je ignorována a ponechána beze změn. Poté co jsou všechny ukazatele nahrazeny, se veškeré hodnoty z *pole ukazatelů* převedou zpět na kladná čísla. Nakonec se toto pole připojí na konec souvislého souboru s indexy slov.

Výsledná časová složitost této konverze je $O(k \cdot n \cdot log_2(t / k))$, kde *n* představuje celkový počet slov v korpusu, *t* je celkový počet slov obsažený v indexační tabulce a *k* je počet segmentů indexační tabulky.

2.2 Výpočet četností N-gramů

Jednou z nejdůležitějších částí našeho algoritmu je výpočet četností N-gramů. Aby se předešlo vysokým požadavkům na operační paměť počítače, jsou jednotlivé velikosti N-gramů extrahovány postupně. Vlastní zpracování je rozděleno do šesti fází, jak je znázorněno na obrázku 4.



Obr. 4. Schéma výpočtu četností N-gramů

Fáze 1. Vstupem do první fáze je indexový soubor vytvořený v předchozí sekci 2.1. Jednotlivá slova jsou zakódována jako 32-bitová celá čísla v rozsahu $<1, 2^{31}-1>$, kde 0 označuje interpunkci a pomyslnou hranici pro nevytváření N-gramů. Nejvyšší bit je rezervován z důvodu jeho využití v předchozí části, kde se používal pro rozlišení ukazatele na počátek slova od již vyřešeného slovního indexu. Navzdory omezenému rozsahu na 2 147 483 647 unikátních slov, je velmi nereálné, že by některý z existujících jazyků dosáhl tohoto množství.

Pro rychlý výpočet četnosti N-gramů, který je založen na principu řazení, potřebujeme mít N-gramy vhodně uložené. K tomuto účelu jsme pro různé velikosti N-gramů připravili různé struktury s pevně definovanými pozicemi slovních indexů. Počet pozic v dané struktuře závisí na velikosti N-gramu. Například struktura pro trigram obsahuje tři vyhrazené pozice, schopné nést tři 32-bitová celá čísla slovních indexů.

K výpočtu četností se alokuje pole struktur, kde vhodná struktura je volena na základě počítaných N-gramů. Při nedostatku paměti se alokuje maximální dosažitelný rozsah pole s tím, že se rozdělí vstupní soubor a zpracuje v několika samostatných dávkách. Alokovaná paměť je ve skutečnosti větší než vstupní velikost souboru, proto hovoříme o rozšířených požadavcích na paměť, které závisí na velikosti počítaných N-gramů.

Požadavky na dostupnou paměť jsou $O(s \cdot n / k)$, kde s je velikost N-gramu, n je počet všech slovních indexů ve vstupním souboru a k je počet iterací nutných pro zpracování vstupního korpusu při nedostatku paměti. Toto je jediný případ, kdy rozšířená paměť nabývá jiné složitosti než $O(1)$, jak je později prezentováno v závěrečné tabulce 1 v sekci 4. V případě, že byl v první fázi předchozí sekce 2.1 aktivován NI filtr, jsou výsledné požadavky na paměť nižší, v důsledku respektování hranice mezi slovy a nevytváření N-gramů překračující hranici.

Pro sestavení N-gramů požadované velikosti a umístění do předpřipravených struktur se používá algoritmus klouzavého okénka. Pokud je ve vstupu nalezen slovní index s hodnotou 0, zásobník s naposledy načítaným N-gramem je vyprázdněn a pozice se přesune na následující nenulový index.

Časová složitost této fáze je $O(s \cdot n)$, kde s je velikost N-gramu a n je celkový počet všech slovních indexů.

Fáze 2. Pro výpočet četností jednotlivých N-gramů je nutné pole seřadit. Řazení je provedeno dle hodnot slovních indexů obsažených v jednotlivých strukturách N-gramů. Díky tomu, že pracujeme pouze s čísly, je vlastní řazení rychlé, vycházející z algoritmu Quick Sort.

Časová složitost řazení je $O(s \cdot n \cdot log_2(s \cdot n / k))$, kde s je velikost N-gramu, n je celkový počet slovních indexů a k počet iterací dávkového zpracování.

Fáze 3. Po seřazení N-gramů lze již vypočíst jejich četnosti. Pole seřazených N-gramů se jedenkrát projde a detekují se identické N-gramy, které po seřazení musí ležet vedle sebe. Při průchodu pole se uloží unikátní N-gramy do dočasného souboru a spolu s nimi i jejich detekované četnosti.

Ve výsledku získáme vždy jeden dočasný soubor s N-gramy a jejich četnostmi pro každou iteraci, která musí být vykonána při nedostatku paměti. Na začátku tohoto souboru se uloží informace o celkovém počtu N-gramů, zakódovaná jako celé číslo na 64-bitech, a velikost extrahovaných N-gramů, zakódovaná jako celé číslo na 32-bitech. Dále následuje popis N-gramů v podobě jejich slovních indexů a četnosti

zakódované jako celá 32-bitová čísla, viz obrázek 6(a). Pro uložení jednoho N-gramu je zapotřebí *(velikost N-gramu + 1) · 4 byty* místa na disku.

Časová složitost výpočtu četnosti N-gramů je $O(s \cdot n)$, kde s a n mají stejný význam jako v předchozí fázi.

Fáze 4. Algoritmus této fáze je téměř identický s fází 4 v sekci 2.1. Situace odpovídá obrázku 3 s tím rozdílem, že nyní mažeme duplikáty N-gramů namísto slov. Soubor i, kde $i \in \langle 1, k \rangle$, se načte do paměti a následně jsou N-gramy ze souboru j, kde $j \in \langle i+1, k \rangle$, hledány mezi N-gramy načtenými v paměti. Celkový počet pomocných souborů je označen k. V případě, že je při zpracování souboru j nalezen v paměti stejný N-gram, je ze souboru j smazán a jeho četnost se přičte k identickému N-gramu v paměti. Není-li žádný duplikát nalezen, je N-gram v souboru j ponechán beze změn.

Časová složitost je opět velmi podobná a pohybuje se mezi limitními složitostmi $O(n \cdot \log_2(n / k))$ a $O(k \cdot n \cdot \log_2(n / k))$, kde n je celkový počet N-gramů obsažený v pomocných souborech a k je počet pomocných souborů.

Fáze 5. Z důvodu mnoha statistických potřeb je užitečné seřadit výsledné N-gramy dle jejich četnosti. N-gramy z jednotlivých pomocných souborů jsou načteny do paměti a seřazeny v sestupném pořadí dle jejich četností. V okamžiku načítání se též aplikuje filtr odstraňující N-gramy, které nesplňují minimální zadanou četnost. Pouze N-gramy se stejnou nebo vyšší četností jsou ponechány pro další zpracování.

Časová složitost vychází z algoritmu pro řazení N-gramů, které dosáhly minimální četnosti. Vlastní složitost je $O(n \cdot \log_2(n / k))$, kde n je celkový počet N-gramů s minimální četností a k je počet pomocných souborů.

Fáze 6. Úkolem poslední fáze je sloučit všechny částečně seřazené soubory s N-gramy do jediného souvislého souboru. Během procesu slučování jsou uvažovány četnosti N-gramů pro zachování sestupného pořadí.

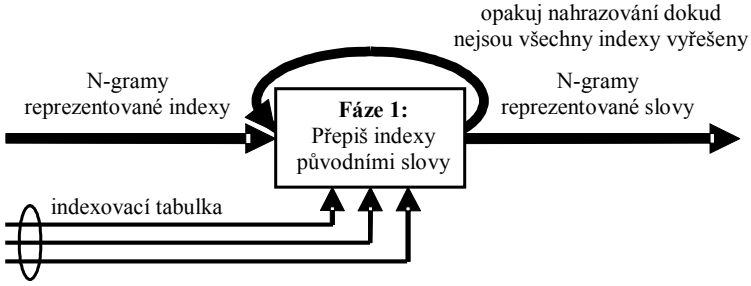
Protože celková velikost všech souborů přesahuje rámec operační paměti, nelze sloučení provést přímo v paměti. Z tohoto důvodu jsme aplikovali postupné hierarchické slučování dvou sousedních souborů. Jako příklad uvádíme sloučení čtyř souborů F_1, F_2, F_3 a F_4, kde nejprve dojde ke sloučení souborů F_1 a F_2, dále F_3 a F_4 a v posledním kroku $F_{1,2}$ spolu s $F_{3,4}$. Proceduru pro sloučení dvou sousedních souborů lze popsat následujícím algoritmem v pěti krocích:

1. Nejprve se načte první N-gram spolu s jeho četností do proměnné a ze souboru F_a a zároveň se načte N-gram s četností ze souboru F_b do proměnné b.
2. Porovnají se četnosti a a b, kde větší z obou četností je spolu s popisem N-gramu zapsána do souboru $F_{a,b}$.
3. V případě, že četnost a byla větší, načte se následující N-gram ze souboru F_a. Pokud byl dosažen konec souboru F_a, překopíruje se zbytek souboru F_b do souboru $F_{a,b}$ a dále se pokračuje krokem 5, jinak se algoritmus opakuje od kroku 2.
4. Pokud byla v předchozím případě větší četnost b, načte se N-gram ze souboru F_b. Při dosažení konce souboru F_b je překopírován zbytek souboru F_a do souboru $F_{a,b}$ a dále se pokračuje krokem 5, jinak se algoritmus opakuje od kroku 2.
5. Soubory F_a a F_b jsou sloučeny do souboru $F_{a,b}$ a vymazány z disku.

Časová složitost slučování všech dočasných souborů dosahuje $O(\log_2(k) \cdot n)$, kde k je počet slučovaných souborů a n je celkový počet všech N-gramů.

2.3 De-indexace

Poslední část našeho algoritmu se zabývá zpětným převodem slovních indexů, reprezentující N-gramy, na původní slova prostřednictvím indexovací tabulky vytvořené v sekci 2.1. Na obrázku 5 je znázorněno zjednodušené schéma de-indexace, zahrnující jedinou fázi.

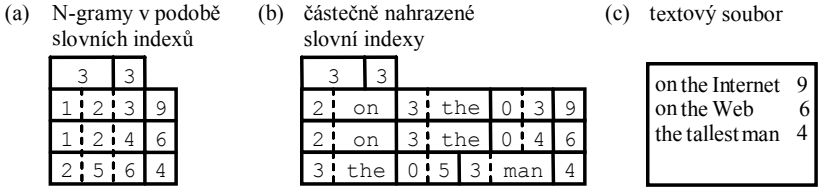


Obr. 5. Schéma de-indexace

Fáze 1. Před začátkem de-indexace se prozkoumají jednotlivé segmenty indexační tabulky a vypočte se hraniční hodnota mezi jednotlivými segmenty dle následující rovnice

$$hranice_m = \sum_{j=1}^{m-1} c_j \,, \tag{2}$$

kde $m \in \langle 1, k \rangle$ s k představujícím počet segmentů indexační tabulky. Parametr c_j označuje celkový počet slov v segmentu j. Na základě hodnot vypočtených z rovnice (2) jsme schopni porovnáním hranice a čísla indexu určit který z k segmentů obsahuje příslušné slovo.



Obr. 6. Příklad opakované de-indexace

Pokud se celá indexovací tabulka vejde do operační paměti, je de-indexace jednoduchá. Slovní indexy obsažené v N-gramech jsou přímo nahrazeny slovy nacházejících se na vypočtené pozici dle rovnice (3) v segmentu, jehož hraniční hodnota je ostře menší než číslo indexu.

$$pozice_m = index - hranice_m \tag{3}$$

V případě, že indexační tabulka přesahuje volnou operační paměť, je nutné de-indexaci opakovat. V prvním okamžiku se načte několik segmentů indexační tabulky v závislosti na volné operační paměti. Následně se vezmou N-gramy reprezen-

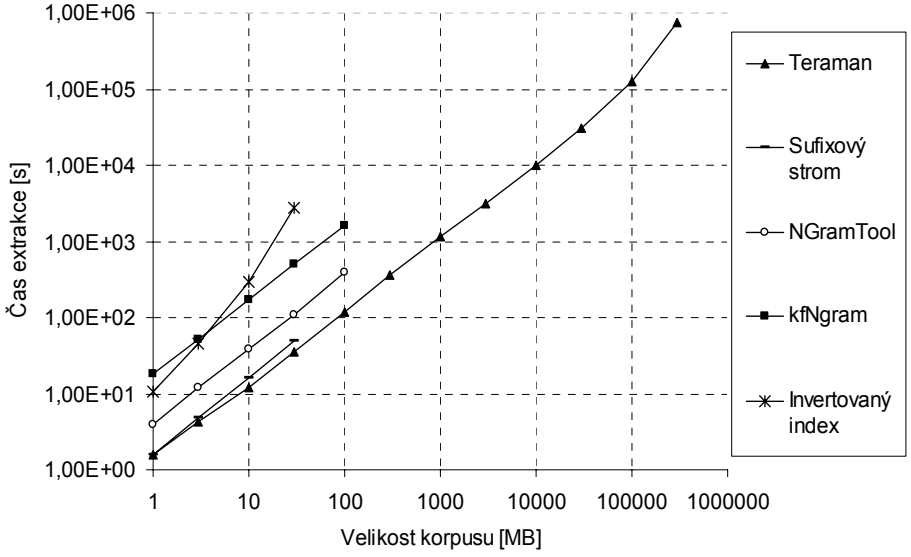
tované slovními indexy, viz obrázek 6(a), a postupně se nahradí indexy, jež se nacházejí v právě načtených segmentech. Obrázek 6(b) je ukázkou N-gramů s částečně nahrazenými slovy a nevyřešenými indexy zároveň. Dosud nevyřešené indexy jsou uvozeny *číslem = 0*, vyřešená slova pak *číslem > 0*, kde *číslo* označuje délku nahrazeného slova. Vlastní četnost N-gramu je vždy uložena až za popisem N-gramu. Další data v částečně nahrazeném souboru, uvedená na počátku souboru, na obrázku 6(b) informují o celkovém počtu N-gramů a velikosti N-gramů.

Procedura nahrazování může být opakována několikrát v závislosti na dostupné paměti a velikosti indexační tabulky. V každém kroku jsou vždy načteny nové segmenty a nahrazeny dosud nevyřešené indexy až do úplného nahrazení všech indexů. Výsledek je pak zapsán v podobě sekvence slov a jejich četností do čistě textového souboru, viz obrázek 6(c).

Časová složitost de-indexace je $O(k \cdot n)$, kde k je počet iterací potřebných pro nahrazení všech slovních indexů a n je celkový počet slovních indexů obsažený ve všech extrahovaných N-gramech.

3 Experimenty a testování

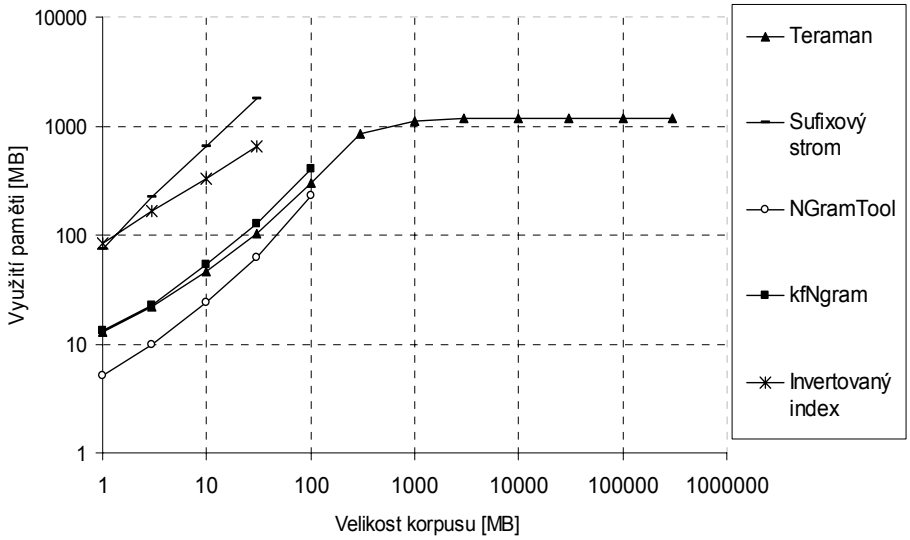
Pro ověření celkových časových a paměťových složitostí našeho algoritmu, jsme se rozhodli využít korpus „Web 1T 5-gram Version 1“ dostupný na [11], který byl vydán společností Google. Pro experimentální ověření jsme použili jednotlivé pentagramy ze souborů *5gm-0040*, *5gm-0041* a *5gm-0042*. Pentagramy jsme s respektováním jejich četností náhodně duplikovali pro získání 300GB textového souboru. Pro menší textové soubory byla použita stejná strategie, pouze se snížila četnost pentagramů.



Obr. 7. Čas nutný pro extrakci 1- až 4-gramů

Naše hlavní pozornost byla věnována porovnání algoritmu Teramana s ostatními dostupnými postupy. Do porovnání jsme zahrnuli algoritmy sufixového stromu a invertovaného indexu, detailně popsaných v [8]. Dále nástroj kfNgram [12], založený na sufixovém poli, původně navržený Kitem a Wilksem [5]. A nakonec algoritmus extrahující N-gramy, který je distribuovaný jako část balíčku NGramTool [13], publikovaný autory Nagao a Mori [10]. Ačkoli jsme zkoušeli hledat další implementace zaměřené na zpracování rozsáhlých dat, nebyli jsme úspěšní. Všechny výše zmíněné postupy jsou určeny především pro zpracování vstupních dat v operační paměti počítače.

Jak je patrné z obrázku 7, maximální množství dat zpracované nástrojem kfNgram a NGramTool bylo 100MB. Zbylé dva algoritmy (sufixový strom a invertovaný index) dokázaly zpracovat pouhých 30MB, z důvodu příliš vysokých paměťových požadavků. Při porovnání celkového času potřebného pro extrakci 1- až 4-gramů je patrné, že žádný z uvedených postupů nedokázal překonat Teramana. To je poměrně zajímavé zjištění, neboť jsme očekávali, že postupy pracující výhradně s operační pamětí by měly dosahovat lepších výsledků než naše aplikace, určená ke zpracování rozsáhlých dat. Jistě je možné, že důvodem jsou nepříliš optimalizované implementace ostatních aplikací, nicméně, algoritmus sufixového stromu byl námi pečlivě implementován v dřívější době, a přesto Teramana nepřekonal.



Obr. 8. Využití paměti během extrakce 1- až 4-gramů

V dalším kroku jsme porovnali využití paměti během extrakce 1- až 4-gramů výše uvedených postupů, které je zachyceno na obrázku 8. V případě menších korpusů (cca 300MB), které mohou být celé zpracovány v operační paměti počítače, alokuje algoritmus Teramana požadovanou paměť, aniž by využíval odkládání na disk. Pro větší korpusy pak automaticky alokuje veškerou volně dostupnou paměť a vhodně pracuje s diskem. Jak se dá očekávat, toto je v kontrastu s ostatními postupy, které vyžadují neúměrně více paměti se stale se zvyšujícím vstupním korpusem. Nástroje kfNgram a

NGramTool vcelku kopírují paměťové požadavky Teramana, nicméně další krok 300MB již pro ně není dosažitelný. Důvodem je prudký nárůst unikátních slov v korpusu 300MB a výrazně vyšší požadavky na operační paměť u obou nástrojů.

Všechny provedené experimenty byly provedeny na počítači Intel Core 2 Duo E6600, 2GB RAM, 1TB HDD (2x500GB v RAID-0) a operačním systému Windows XP Professional SP2.

4 Závěr

Ze sekce 3 je patrné, že Teraman překonává ostatní nástroje pro extrakci N-gramů nejen objemem zpracovaných dat, ale i rychlostí zpracování. Jeho hlavní výhodu představuje možnost zpracovat teoreticky neomezený objem dat s efektivním využitím aktuálně dostupné operační paměti. Jeho výkon výrazně neklesá ani v případě větších objemů dat, kdy je nucen pracovat s pevným diskem (viz obrázek 7).

Na základě našich experimentů s různými korpusy jsme určili, že průměrně je zapotřebí 2.2 násobek volného místa na disku k dočasnému uložení pomocných souborů vzhledem k velikosti vstupních dat, která chceme zpracovat. V našem případě jsme měli k dispozici pevný disk o velikosti 1TB, v sekci 3 jsme tedy byli schopni zpracovat maximálně 300GB korpus.

Tabulka 1. Přehled časových a paměťových složitostí

		Nejlepší dosažitelná časová složitost	**Nejhorší dosažitelná časová složitost**	**Složitost rozšířené paměti**
Část 1	Fáze 1	$O(n)$		$O(1)$
	Fáze 2	$O(n \cdot \log_2(n / k))$		
	Fáze 3	$O(n)$		
	Fáze 4	$O(t \cdot \log_2(t / k))$	$O(k \cdot t \cdot \log_2(t / k))$	$O(1)$
	Fáze 5	$O(k \cdot n \cdot \log_2(t / k))$		$O(1)$
Část 2	Fáze 1	$O(s \cdot n)$		$O(s \cdot n / k)$
	Fáze 2	$O(s \cdot n \cdot \log_2(s \cdot n / k))$		
	Fáze 3	$O(s \cdot n)$		
	Fáze 4	$O(n \cdot \log_2(n / k))$	$O(k \cdot n \cdot \log_2(n / k))$	$O(1)$
	Fáze 5	$O(n \cdot \log_2(n / k))$		$O(1)$
	Fáze 6	$O(\log_2(k) \cdot n)$		$O(1)$
Část 3	Fáze 1	$O(k \cdot n)$		$O(1)$

Tabulka 1 shrnuje všechny časové a paměťové složitosti jednotlivých fází našeho algoritmu. Na základě těchto složitostí a experimentů provedených výše lze stanovit, že celková časová složitost Teramana je supralineární. Požadavky na dostupnou paměť počítače jsou vzhledem k velikosti vstupního korpusu lineární.

Tato práce byla částečně podporována z prostředků Národního Programu Výzkumu II, projekt 2C06009 (COT-SEWing), a z části projektem FRVŠ 1798/2008/G1.

Reference

1. W.B. Cavnar and J.M. Trenkle. NGram-Based Text Categorization. *In Proceedings of 3rd Annual Symposium on Dokument Analysis and Information Retrieval*, Las Vegas, USA pages 161–175, 1994.
2. J. Dean and S. Ghemawat. Mapreduce: Simplified data processing on large clusters. *In Proceedings of the 6th Symposium on Operating System Design and Implementation*, San Francisco, CA, pages 137–150, December 2004.
3. J. Fürnkranz. A Study Using *n*-gram Features for Text Categorization. Technical Report OEFAI-TR-98-30, Austrian Institute for Artificial Intelligence, 1998.
4. M. Ganapathiraju, V. Manoharan, and J. Klein-Seetharaman. BLMT: Statistical Sequence Analysis using N-Grams. *Applied Bioinformatics*, volume 3, issue 2, 2004.
5. C. Kit and Y. Wilks. The Virtual Corpus Approach to Deriving N-gram Statistics from Large Scale Corpora. *In Proceedings of 1998 International Conference in Chines Information Processing*, In C. N. Huang (ed.), Beijing, pages 223-229, November 1998.
6. S. Kozlowski. Ngram counting solution comparison. December 2003. Available at http://nlp.strefa.pl/ngrams/linux/index.htm
7. D. Mladenic and M. Grobelnik. Word sequences as features in text-learning. *In Proceedings 17th Electrotechnical and Computer Science Conference*, Slovenia, 1998.
8. R. Tesar, D. Fiala, F. Rousselot and K. Jezek. A comparison of two algorithms for discovering repeated word sequences. *In Proceedings of 6th International Conference on Data Mining, Text Mining and their Business Applications*, Skiathos, Greece, pages 121-131, May 2005. ISBN 1-84564-017-9.
9. M. Yamamoto and K. W. Church. Using Suffix Arrays to Compute Term Frequency and Document Frequency for All Substrings in a Corpus. In *Computational Linguistics, volume 27 (1)*, 2001.
10. M. Nagao and S. Mori. A New Method of N-gram Statistics for Large Number of n and Automatic Extraction of Words and Phrases from Large Text Data of Japanese. *In Proceedings of the 15th International Conference on Computational Linguistics (COLING 1994)*, Kyoto, Japan, 1994.
11. Linguistic Data Consortium. Web 1T 5-gram Version 1. *http://www.ldc.upenn.edu/*
12. W. Flecher. kfNgram. *http://www.kwicfinder.com/kfNgram/kfNgramHelp.html*
13. L. Zhang. Text2Ngram. *http://homepages.inf.ed.ac.uk/s0450736/ngram.htm*

Annotation:

N-gram Extraction from Large Datasets

In this paper, we present an algorithm for N-gram extraction from large datasets. To examine the overall time and memory complexities of our algorithm we employed the "Web 1T 5-gram Version 1" corpus released by Google. The experiments indicate that our approach reaches outstanding results among other available solutions in terms of speed and amount of processed data.

This paper is a shortened, Czech version of the following publication:

Z. Ceska, I. Hanak, R. Tesar: "Teraman: A Tool for N-gram Extraction from Large Datasets". In *Proceedings of the IEEE 3rd International Conference on Intelligent Computer Communication and Processing*, Cluj-Napoca, Romania, September 2007, pp. 209–216. ISBN 978-1-4244-1491-8.