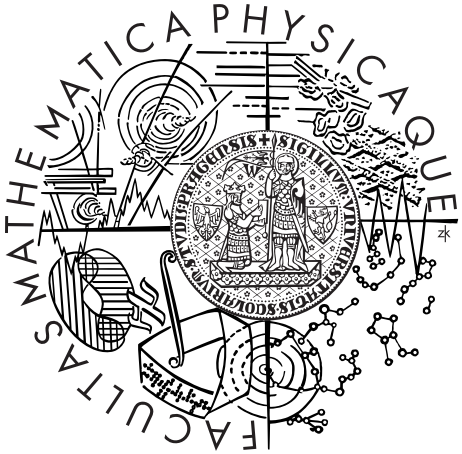
Univerzita Karlova v Praze
Matematicko-fyzikální fakulta

BAKALÁŘSKÁ PRÁCE



Tomáš Dvořák

Rekonstrukce pořadí slov ve větách

Ústav formální a aplikované lingvistiky

Vedoucí bakalářské práce: Mgr. Barbora Vidová Hladká, PhD.
Studijní program: Informatika
Studijní obor: Obecná informatika

Praha 2011

Tímto bych chtěl poděkovat mé vedoucí Mgr. Barboře Vidové Hladké, PhD. za vynikající přístup při psaní nejen této práce.

Prohlašuji, že jsem tuto bakalářskou práci vypracoval(a) samostatně a výhradně s použitím citovaných pramenů, literatury a dalších odborných zdrojů.

Beru na vědomí, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorského zákona v platném znění, zejména skutečnost, že Univerzita Karlova v Praze má právo na uzavření licenční smlouvy o užití této práce jako školního díla podle §60 odst. 1 autorského zákona.

V Praze dne 19. 4. 2011 Tomáš Dvořák

Název práce: Rekonstrukce pořadí slov ve větách

Autor: Tomáš Dvořák

Katedra: Ústav formální a aplikované lingvistiky

Vedoucí bakalářské práce: Mgr. Barbora Vidová Hladká, PhD., ÚFAL

Abstrakt: Rekonstrukcí původního pořadí slov věty je v této práci myšleno přerovnání slov věty takovým způsobem, aby vznikla gramaticky korektní věta v daném jazyce. Rekonstrukce původního pořadí slov ve větách je velice užitečná část automatického zpracování přirozeného jazyka počítačem, která může nalézt široké uplatnění. Především velice důležitou roli hraje při automatickém překladu přirozeného jazyka, rozpoznávání řeči nebo při konstrukci umělých komunikačních partnerů.

Řešení této úlohy je korpusově orientované a při řešení jsou použity dva přístupy: morfologický a syntaktický. Každý přístup využívá výstup externího modulu, který poskytuje požadované morfologické či syntaktické informace. Rekonstrukční procedura je navržena tak, aby byla jazykově nezávislá. Cílovými jazyky jsou čeština a angličtina.

Klíčová slova: korpus, morfologická analýza, syntaktická analýza, pořadí slov

Title: Word order reconstruction

Author: Tomáš Dvořák

Department: Institute of Formal and Applied Linguistics

Supervisor: Mgr. Barbora Vidová Hladká, PhD.

Abstract: A word order reconstruction is a re-arrangement of words to get a grammatically correct sentence. It is a very useful task for the applications of natural language processing, machine translation, speech recognition or construction artificial communication partners.

We present a corpus-based approach to the task of word order reconstruction. We use two methods: morfological and syntactical method. Both methods use output from the external module. This approach is designed independently on the application where the word order reconstruction can help improve overall performance. Czech and English will be used as the object language.

Keywords: corpora, morfological analysis, syntactical analysis, word order

Obsah

Úvod

Rekonstrukce původního pořadí slov ve větách je důležitou úlohou při rozpoznávání mluvené řeči, statistickém automatickém překladu, internetovém vyhledávání, automatické korektuře a v mnoha dalších aplikací, kde je potřeba automaticky zpracovávat přirozený jazyk.

Práce implementuje dvě procedury, které v reálném čase rekonstruují pořádek slov ve vstupní posloupnosti. První provádí rekonstrukci českých a anglických dat na základě morfologické analýzy a druhá rekonstruuje české a anglické věty na základě syntaktické analýzy. Druhá z procedur využívá současně i morfologické analýzy, avšak primárně je algoritmus postaven na syntaktické analýze. Pro svou práci využívají procedury jednak externí moduly, především tagger a parser českých a anglických vět, a dále korpusy českých a anglických textů, které slouží jako zdroj gramaticky korektních vět daného jazyka.

Jak již bylo jednou řečeno, práce využívá dvou přístupů k problému. První přístup je založen na morfologické analýze. Tento způsob pracuje pouze na úrovni jednotlivých slov, resp. morfologických značek. Vztahy mezi jednotlivými slovy vůbec neuvažuje. Naproti tomu druhý způsob, založený na syntaktické analýze, již ke své práci využívá informace o tom, že mezi slovy mohou existovat nějaké vazby. Každý přístup je implementován jako samostatná aplikace. CeskeVetyI je název aplikace implementující morfologický přístup a CeskeVetyII je název aplikace implementující syntaktický přístup.

V práci pracujeme s externími nástroji, které nepracují se 100% úspěšností, a proto to musíme brát v úvahu při posuzování úspěšnosti rekonstrukčních procedur.

První kapitola podává jakýsi úvod do problematiky a letmo představuje již existující rekonstrukční modely.

Druhá kapitola obsahuje popis využívaných externích nástrojů a externích zdrojů dat. Obsahuje dále také popis pomocných procedur. Především míchací procedury, která v práci figuruje jako simulace reálné aplikace.

Ve třetí kapitole je popsán algoritmus řešení založený na morfologické analýze včetně rozboru jeho časové a paměťové náročnosti. Ve čtvrté kapitole je popsán algoritmus založený na druhém přístupu k problému - syntaktické analýze.

Pátá kapitola je věnována vyhodnocení úspěšnosti obou přístupů. Úspěšnost je testována na vzorku 100 českých vět a 100 anglických vět. Zároveň obsahuje různé statistiky těchto testovacích dat a popis provedených experimentů.

Programátorská a uživatelská dokumentace k oběma přístupům se nachází v kapitole šesté.

Po šesté kapitole následuje závěr a shrnutí. V tomto oddílu jsou popsány výhody a nevýhody obou přístupů. Dále jsou zde popsána možná vylepšení rekonstrukčních procedur.

1. Rešerše

Řešení úlohy rekonstrukce původního pořadí slov ve větách v této práci není prvním pokusem o vyřešení tohoto problému. Již existuje několik různě úspěšných řešení, které požívají různé přístupy k tomuto problému.

Jedním z nich je veřejně dostupná demo verze (viz [2]). Podle stránek, prezentujících toto demo, je k rekonstrukci využíván *5-gram* viz [6]. Bohužel detailnější popis řešení není uveden. Nevíme tedy jestli *5-gram* využívá slova nebo morfologické značky.

Obecně je však jasné, že objem dat, na kterém se musí *5-gram* trénovat se udává v řádech *TB*. Naproti tomu náš přístup využívá 100 000 zdrojových vět, což odpovídá objemu dat v řádu stovek *MB*.

Součástí této práce je i srovnání tohoto přístupu s naším přístupem (viz sekce 5.2).

2. Pomocné a externí zdroje

2.1 Externí datové zdroje a nástroje

Tento oddíl je věnován popisu externích datových zdrojů a nástrojů využívaných při rekonstrukci jak v morfologickém přístupu, tak při syntaktickém přístupu. Jako datové zdroje práce používá korpusy textů daného jazyka. Větám z těchto textů budeme říkat *zdrojové věty*.

2.1.1 Tool_chain

Tool_chain (součást ČAK 2.0 viz [3]) je nástroj pro morfologické a syntaktické zpracování českých textů. V případě morfologické analýzy pracuje tak, že jednotlivé věty ze vstupního souboru rozdělí na slova a poté ke každému slovu přiřadí morfologickou značku (viz dále) k němu odpovídající. Při syntaktické analýze provádí se vstupní větou větný rozbor a vrátí závislostní strom dané věty. Svůj výstup ukládá nástroj tool_chain ve formátech CSTS nebo PML. V naší práci pracujeme jen s výstupem ve formátu CSTS [8]. Předností CSTS formátu je především snadná práce s uloženými daty.

Morfologická značka je řetězec znaků určité délky, který obsahuje tvaroslovné informace o slově, ke kterému je přidružen. Morfologické značky používané nástrojem tool_chain jsou délky 15. Více o morfologických značkách viz [7].

Níže je uvedený příklad ukazující obsah souboru v CSTS formátu. Tento soubor vznikl aplikací nástroje *tool_chain* s parametry *-tATP*, kde *-t* značí proces tokenizace, parametr *-A* proces morfologické analýzy, *-T* je proces tagování a *-P* je proces parsování, na větu "Tím hůř pro ně."

Na obrázku 2.1 je zobrazen závislostní strom pro větu "Obecná odpověď na tuto otázku je sotva možná.".

2.1.2 Stanford POS Tagger

Jedná se o tagger pro anglické texty. Je to nástroj pro morfologickou analýzu anglických textů. Na vstupu vyžaduje soubor s prostým textem. Výstupní soubor je speciálního formátu. Na každé řádce je jedna věta. Věta se skládá z řetězců, kde každý jeden řetězec věty je tvaru: *"slovo_morf.značka"*. Pro větu:

"How are you?"

je odpovídající řádek výstupního souboru taggeru tvaru:

"How_WRB are_VBP you_PRP ?_."

Více o tomto taggeru viz [10].

2.1.3 Stanford Parser

Nástroj pro parsování neboli syntaktickou analýzu anglických textů. Na vstupu očekává soubor s prostým textem. Výstup je ve formě souboru, kde pro každou

Ukázka CSTS souboru

```
<csts lang=cs>
<doc file="tt" id=1>
<a>
<mod>?
<txtype>?
<genre>?
<med>?
<temp>?
<authname>?
<opus>tt
<id>001
</a>
<c>
<p n=1>
<s id=adam-001-p1s1>
<f id=adam-001-p1s1W1-Tm>Tím<l>ten<t>PDZS7———-
 <r>1<g>2<A>Adv
<f id=adam-001-p1s1W2-Tm>hůř<l>špatně<t>Dg——-2A—-
 <r>2<g>0<A>ExD
<f id=adam-001-p1s1W3-Tm>pro<l>pro-1<t>RR–4———-
 <r>3<g>2<A>AuxP
<f id=adam-001-p1s1W4-Tm>ně<l>on-1<t>P5XP4–3——-
 <r>4<g>3<A>Adv
<D>
<d id=adam-001-p1s19W5-Tm>.<l>.<t>Z:————-
 <r>5<g>0<A>AuxK
</c>
</doc>
</csts>
```

větu jsou dvě části. První vyjadřuje složkový strom věty a druhá (oddělená mezerou od první) je seznam závislostí ve větě. Níže je uvedena ukázka výstupního souboru pro větu: *"How are you?"*

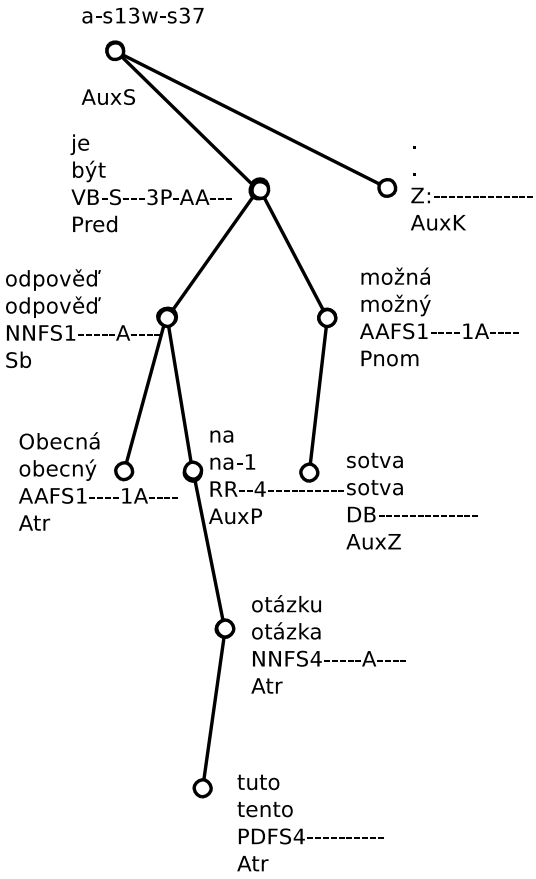
Obrázek 2.2 ukazuje závislostní strom pro anglickou větu "How are you?".

Více informací o tomto parseru viz [9].

2.1.4 Český národní korpus

ČNK neboli Český národní korpus je rozsáhlý soubor českých vět (100 mil. slov). Jedná se o různé články, výtahy z divadelních her či odborné texty, které nejsou vybírány náhodně, nýbrž jsou systematicky vybírány tak, aby obsáhly co největší pole různých žánrů.

Při řešení našeho problému je tento soubor textů využíván jako zdroj českých vět, podle kterých pracuje rekonstrukční algoritmus. Jinak řečeno, při rekonstrukci na základě morfologické analýzy je ČNK brán jako zdroj korektních posloupností morfologických značek, protože věty z ČNK chápeme jako grama-



Obrázek 2.1: Příklad závislostního stromu pro českou větu

ticky správné. Pomocí něj tedy zjistíme o konkrétních morfologických značkách, v jaké korektní konfiguraci se mohou vzhledem k sobě vyskytovat ve větě. V případě rekonstrukce na základě syntaktické analýzy poskytuje korpus soubor korektních závislostních stromů, pomocí kterých procedura pracuje. Využití ČNK při morfologickém přístupu je znázorněno na obrázku 2.3 a při syntaktickém přístupu na obrázku 2.4. Více informací o ČNK viz [4].

2.1.5 Americký národní korpus

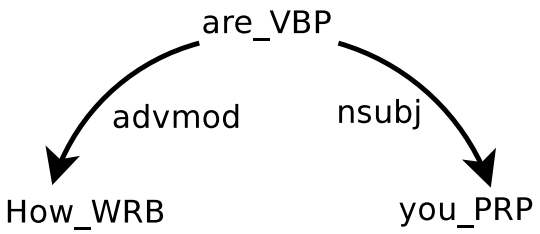
Americký národní korpus (ANK) je rozsáhlý soubor anglických textů (10 mil. slov). Je členěn do několika složek podle typu textu. Při řešení problému rekonstrukce je využíván jako zdroj gramaticky korektních anglických vět. Využití tohoto korpusu je obdobné jako u ČNK a je znázorněno na obrázcích 2.3 a 2.4.

Více informací o tomto korpusu naleznete v [1].

2.2 Pomocné procedury

Zde popíšeme dvě pomocné procedury. Jde o míchací proceduru a o proceduru počítající index podobnosti.

Ukázka věty zpracované parserem anglických textů
(ROOT (SBARQ (WHADVP (WRB How)) (SQ (VBP are) (NP (PRP you))) (. ?))) advmod(are-2, How-1) nsubj(are-2, you-3)



Obrázek 2.2: Příklad závislostního stromu pro anglickou větu

2.2.1 Míchací procedura

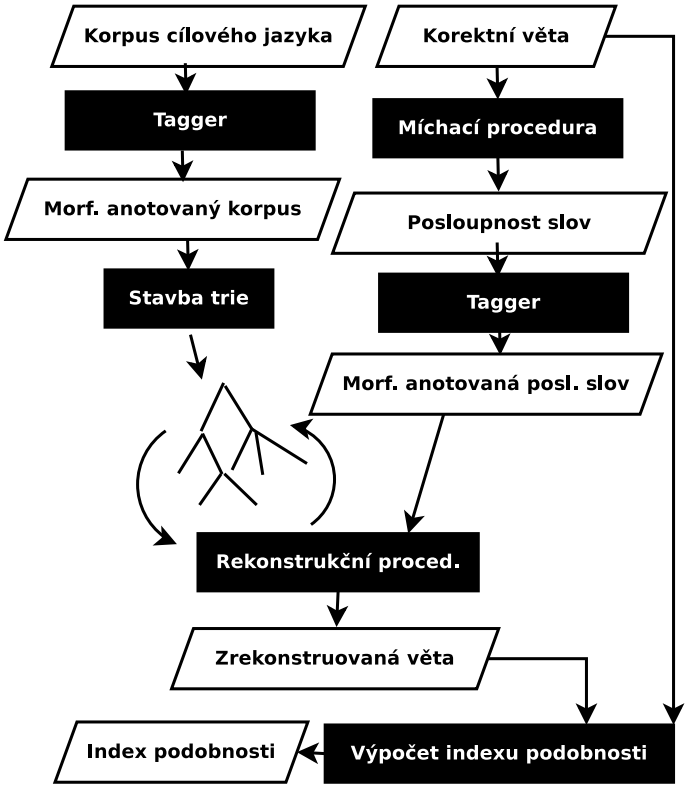
Míchací procedura v práci figuruje jako simulace reálné aplikace pracující s texty, například může suplovat proceduru strojového překladu.

Samotné míchaní funguje na principu generování náhodné permutace slov věty. Nejprve je vygenerováno náhodné číslo C z intervalu $R :=< 0, (l + 2))$, kde l je délka míchané věty. Délka věty znamená počet částí věty, které jsou od sebe odděleny mezerami. Rozsah je takto dán z toho důvodu, aby se zvýšila pravděpodobnost zamíchání extrémně krátkých vět. Nyní v cyklu od 0 do C probíhá vzájemné prohazování dvou náhodně vybraných slov. Nejprve se vygenerují dvě náhodná čísla C_1,C_2 z $R' :=< 0, (l - 1))$. Slova v původní větě na pozicích C_1 a C_2 se vzájemně prohodí. Může se samozřejmě stát, že zamíchaná posloupnost slov bude shodná s původní větou.

Interval R' volíme tak, aby koncové interpunkční znaménko zůstalo na svém místě. To musí zůstat na svém místě proto, aby na větu mohl být korektně aplikován tagger. Pokud by znaménko nebylo na konci, tak by tagger k této větě přistoupil jako ke dvěma různým větám a tudíž by ani rekonstrukce nepracovala správně. Zároveň tato podmínka není na škodu jelikož znaménko, které není na konci, lze jedním průchodem větou přemístit na konec.

Pseudokód míchací procedury vypadá takto (S je vstupní věta):

1. $l := length(S)$
2. $C := random(0, l + 2)$ //kolik proběhne výměn
3. $from\ 0\ to\ C\ do$
   - $C_1 := random(0, l - 1)$



Obrázek 2.3: Schéma využívání korpusů při morfologické rekonstrukci

- $C_2 := random(0, l-1)$
- $swap(S_{C_1}, S_{C_2})$

4. *return* S

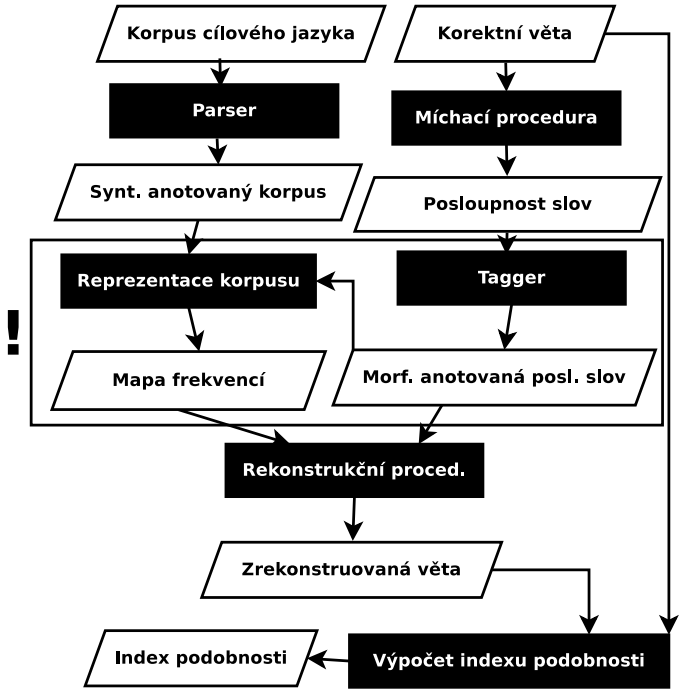
Míchací procedura je implementována jako samostatný program s názvem MichaciProcedura. Program je k nalezení na přiloženém cd (viz Příloha A). Pro více informací o tomto programu čtěte soubor "README.txt" na přiloženém cd v adresáři *MichaciProcedura*.

2.2.2 Index podobnosti

Pro vyhodnocení správnosti rekonstrukce vstupní věty slouží takzvaný index podobnosti. Procedura, která počítá index podobnosti, dostane na vstupu dvě věty a vrátí celé číslo. Toto číslo určuje míru podobnosti těchto dvou vět.

Původně byl index navrhnut jako počet dvojic slov, které musí být navzájem prohozeny, abychom z jedné věty dostali větu druhou. Tento index se však ukázal jako nedostatečně relevantní. Proto jsme navrhli index podobnosti jako počet extrémů v posloupnosti získané jako čísla pozic v zamíchané větě, na kterých se slovo vyskytuje v originální větě (věta před vstupem do míchací procedury). Extrémem je v tomto smyslu myšlena jakákoliv dvojice čísel, která nenásleduje bezprostředně za sebou (1,2 -není extrém; 1,3 -je extrém). Například pro věty:

1. "Pokud tam půjdu, tak přijdu pozdě."



Obrázek 2.4: Schéma využívání korpusů při syntaktické rekonstrukci

2. "tak přijdu pozdě. Pokud tam půjdu,"

Původní index pro tyto věty je roven číslu 4, přestože jsou si tyto dvě věty vcelku dosti podobné až na prohození hlavní a vedlejší věty. Při použití nově navrženého indexu bude situace následující. Posloupnost indexů v původní větě a situace po rekonstrukci:

1. $0, 1, 2, 3, 4, 5, 6, 7$
2. $4, 5, 6, 7, 0, 1, 2, 3$

Je vidět, že v posloupnosti indexů 2. je jen jeden extrém (7,0). Proto je nový index pro tyto věty roven číslu 1.

V obecném případě dává nový index o něco větší hodnoty. Pro demonstraci uvádím ještě tento příklad.

1. "Pokud tam půjdu, tak přijdu pozdě."
2. "tam Pokud půjdu. pozdě přijdu tak,"

Původní index pro tyto dvě věty je roven 3. Odpovídající posloupnosti indexů jsou v tomto případě:

1. $0, 1, 2, 3, 4, 5, 6, 7$
2. $1, 0, 2, 7, 6, 5, 4, 3$

Je tedy hned vidět, že posloupnost 2. obsahuje 7 extrémů a tedy hodnota nového indexu je rovna 7.

Procedura implementující index podobnosti je samostatný program s názvem IndexPodobnosti. Nachází se na přiloženém cd. Pro více informací čtěte soubor "README.txt" na přiloženém cd (viz Příloha A) v adresáři *IndexPodobnosti*.

3. Rekonstrukce na základě morfologické analýzy

V této sekci je detailně popsáno kompletní řešení problému rekonstrukce původního pořadí slov ve větách na základě morfologické analýzy. Řešení je implementováno jako program s názvem CeskeVetyI.

3.1 Popis řešení

3.1.1 Zpracování korpusů

Zpracování korpusů je prováděno automaticky, uživatel pouze definuje cílový jazyk a počet vět, které se mají k rekonstrukci použít. Nejprve je načten daný počet zdrojových vět z korpusu daného jazyka a jejich odpovídající posloupnosti morfologických značek. Toto je načítáno ze souborů, které vznikly aplikací taggeru daného jazyka na jednotlivé soubory textů korpusu. Jak ČNK, tak i ANK je taggován pouze jedenkrát a příslušné výstupní soubory jsou přiloženy k proceduře, která je nyní při každém spuštění pouze přečte.

3.1.2 Reprezentace korpusů

Pro reprezentaci korpusů je využíváno datové struktury - *trie*.

Trie je stromovitá datová struktura, vhodná pro uchovávání velkého množství řetězcových dat nad nějakou konečnou abecedou. Formálně jde o zakořeněný strom, kde každý uzel kromě kořene má jednoho předchůdce a každý uzel kromě listů má alespoň jednoho následníka. Trie se často využívá jako reprezentace slovníků, viz obrázek 3.1.

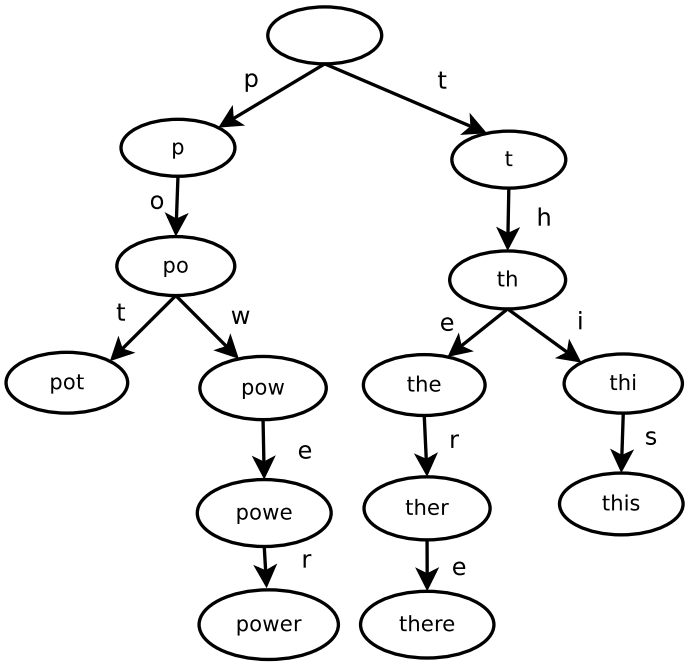
V této práci využíváme trii trochu jiným způsobem. V našem případě je abeceda dána množinou morfologických značek a jedno slovo odpovídá celé posloupnosti morfologických značek.

Použití této struktury v proceduře je dáno povahou řešeného problému. Trie se používá především pro ukládání řetězců, což věty bezpochyby jsou. Tato struktura je vhodná i z důvodu úspory paměti, jelikož při vkládání dvou shodných vět se vloží pouze jeden exemplář. Všechny zdrojové věty se při každém volaní procedury projdou pouze jednou, při stavbě trie. Naproti tomu kdybychom se rozhodli věty při rekonstrukci procházet sekvenčně, tak musíme skutečně projít všechny věty pro každou rekonstruovanou větu. Úspora, které se nám touto volbou dostává, je vidět v tabulkách 5.2 a 5.3.

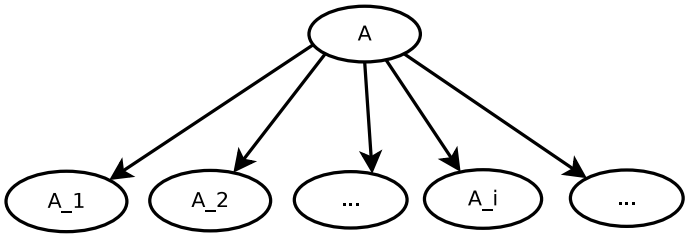
Samotná stavba trie probíhá tak, že začneme s prázdnou trií a postupně budeme vkládat do trie jednu větu za druhou.

Předpokládejme, že máme již postavenou trii T (na začátku je T prázdné) a chceme do ní vložit posloupnost morfologických značek P. Označme jako A aktuální vrchol trie T a A_i označuje i-tého následníka A. Znázorněno na obrázku 3.2.

Pokud je trie T prázdná, vytvoříme speciální vrchol (kořen trie) a do jeho identifikátoru vložíme řetězec "#". Následuje vložení P. Posloupnost vkládáme



Obrázek 3.1: Schéma trie jako slovníku nad konečnou abecedou

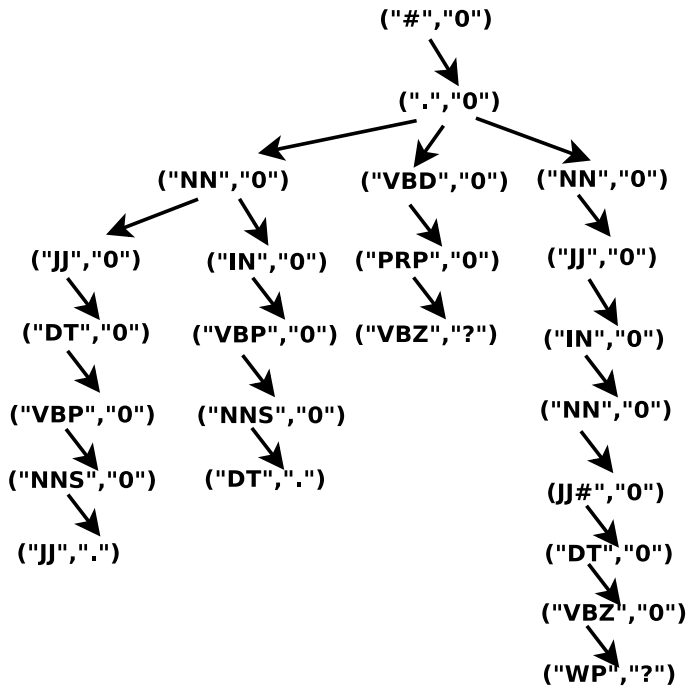


Obrázek 3.2: Schéma uzlu trie

od konce. Jako A je nyní považován kořen. Vytvoříme nový vrchol, do jeho identifikátoru vložíme poslední morf. značku z P a tento vrchol přidáme do následníků A. A nastavíme na tento nový vrchol a obdobně pokračujeme s předposlední morfologickou značkou P. V případě, že jsme vyčerpali všechny morfologické značky P, uložíme do A informaci o koncovém interpunkčním znaménku P. Touto informací je v případě ".","?","!" přímo toto znaménko, v každém jiném případě je to "" (prázdný řetězec). Tato informace nám zároveň říká, že v tomto vrcholu končí věta. Vrcholy, kde nekončí žádná věta, mají hodnotu této informace nastavenu na "0".

Pokud T není prázdná, A je nastaveno na kořen trie. Porovnáváme poslední morf. značku P postupně s identifikátory A_i. V případě shody s některým z identifikátorů A_i, nastavíme A na A_i a pokračujeme obdobně dále s předposlední morf. značkou P. V případě, že se morf. značka P neshoduje s žádným z identifikátorů A_i, vytvoříme nový vrchol jako následníka A, do jeho identifikátoru vložíme tuto morf. značku a pokračujeme obdobně dále. V případě, že jsme vyčerpali všechny morf. značky P a v A již nějaká věta končí, neděláme nic. V opačném případě postupujeme stejně jako při vkládání věty do prázdné trie.

Na obrázku 3.3 je znázorněna trie tak, jak ji využíváme v této proceduře.



Obrázek 3.3: Schéma trie po vložení tří vět

Obrázek znázorňuje vložení následujících vět do trie:

(i) *Unmarried*_JJ *partners*_NNS *receive*_VBP *no*_DT *such*_JJ *exemption*_NN ._.

(ii) *These*_DT *laws*_NNS *vary*_VBP *by*_IN *state*_NN ._.

(iii) *What*_WP *is*_VBZ *the*_DT *legal*_JJ *status*_NN *of*_IN *court-ordered*_JJ *desegregation*_NN *?*_.

(iv) *Has*_VBZ *it*_PRP *worked*_VBD *?*_.

Jelikož vkládáme do trie věty odzadu, tak pod kořenem budou uzly reprezentující koncová interpunkční znaménka věty resp. jejich morfologické značky. V našem příkladě je však jen jeden uzel pod kořenem, přestože jsou v trii vloženy věty s dvěma různými koncovými interpunkčními znaménky. Je to dáno tím, že nevkládáme interpunkční znaménko, ale jeho morfologickou značku. V případě anglických vět mají všechna koncová interpunkční znaménka morfologickou značku rovnou znaku '.'.

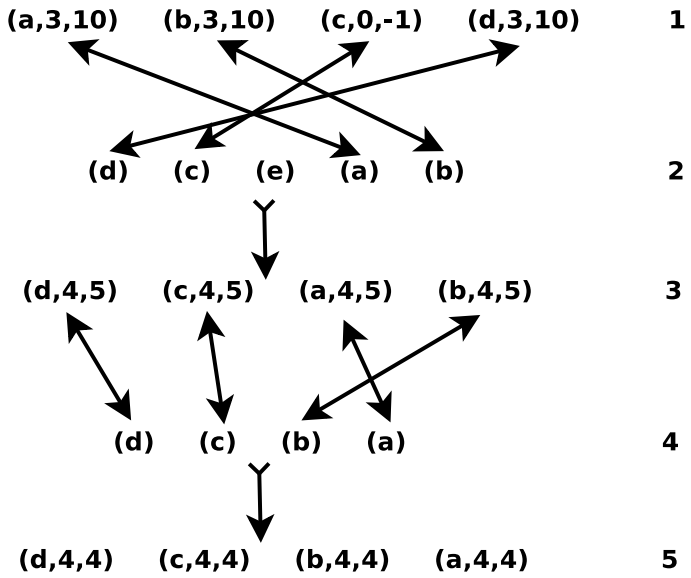
Na obrázku každý vrchol obsahuje dvojici (a,b), kde a reprezentuje morfologickou značku a b koncové interpunkční znaménko věty, která v tomto uzlu končí, pokud nějaká taková je.

3.1.3 Rekonstrukční algoritmus

Vstup rekonstrukční procedury je ve formě souboru s morfologicky anotovanými větami určenými k rekonstrukci. Ve skutečnosti to však nejsou věty, ale morfologicky anotované posloupnosti slov. Tato anotace vstupního souboru probíhá automaticky a obstarává ji uživatel (např. pomocí nástroje tool_chain).

Předpokládejme trii T, označme S náhodně uspořádanou posloupnost slov rekonstruované věty. MT nechť značí morfologické značky po řadě odpovídající posloupnosti S, I je celočíselná posloupnost inicializovaná na hodnotu 0 po řadě odpovídající slovům S. Kdykoliv v následujícím budeme mluvit o prohazování morf. značek v MT, budeme tím myslet, že současně došlo k prohození odpovídajících slov v S a odpovídajících hodnot v I.

Postavíme trii ze zadaného počtu zdrojových vět způsobem popsaným výše a začneme ji procházet do hloubky. Po cestě z kořene do listů si pamatujeme posloupnost identifikátorů (morfologických značek) navštívených vrcholů (označme ji A). Dojdeme-li do vrcholu, kde končí nějaká věta, zkontrolujeme, zda má odpovídající koncové interpunkční znaménko a pokud ano, spustíme porovnávací algoritmus na A (porovnávací algoritmus porovnává vždy MT s A). Nejprve srovnáme vzájemně A a MT a zjistíme, které morfologické značky se vyskytují v obou posloupnostech. Tyto společné morfologické značky budeme vzájemně srovnávat.



Obrázek 3.4: Schéma porovnávacího algoritmu

Nyní je třeba vysvětlit, k čemu slouží posloupnost I. Prvky posloupnosti I uchovávají informaci o počtu morf. značek, vzhledem ke kterým byla odpovídající morf. značka v MT již srovnána. Nikdy nedochází k prohození značek, které již byly srovnány vzhledem k většímu počtu značek než je aktuální počet srovnávaných morf. značek. Aktuálním počtem srovnávaných morf. značek je zde myšlen počet morf. značek vyskytujících se jak v MT, tak v A. V případě rovnosti těchto počtů algoritmus ještě porovná délku posloupnosti, podle které byla daná morf. značka již srovnána (L_1) a délku posloupnosti podle které by měla být srovnána nyní (L_2). Pokud tato nová délka je bližší délce rekonstruované posloupnosti, pak dojde k prohazování. Slovo "bližší" z předchozí věty lze definovat jako menší z hodnot $|L - L_1|$ a $|L - L_2|$, kde L je délka rekonstruované posloupnosti. Délkou posloupnosti resp. věty rozumíme počet částí posloupnosti resp. věty, které po aplikaci taggeru mají přiřazenu morfologickou značku.

Srovnání probíhá tak, že ze společných morfologických značek A a MT se

vyřadí ty, které již byly srovnány vzhledem k většímu počtu morfologických značek. Zbylé morfologické značky setřídíme tak, aby jejich vzájemné pořadí v MT bylo shodné s pořadím těchto v A. Značky z MT, které nejsou v A, musí při tomto srovnávání zůstat na svých místech. Po tomto kroku pokračujeme dále v procházení trie.

Vše je znázorněno na obrázku 3.4. Trojice (a,b,c), kde a je morfologická značka, b je počet morfologických značek, vzhledem k nimž již tato byla srovnána, a c je délka věty, podle které k tomuto srovnání došlo, vyjadřují stav naší rovnané posloupnosti morfologických značek MT. Posloupnost (na obrázku označená číslem 1) udává již částečně zrekonstruovanou posloupnost MT. Konkrétně je vidět, že v této posloupnosti byly vzájemně srovnány tři morfologické značky a k tomuto srovnání došlo podle zdrojové věty délky 10. Poté jsme v trii narazili na posloupnost (na obrázku označenou číslem 2) A. Zjistili jsme, že má s MT shodné čtyři morfologické značky. Všechny značky v MT byly prozatím srovnány nejvýše vzhledem ke třem ostatním a tudíž budeme srovnávat všechny čtyři společné značky. Podle srovnávacího algoritmu stabilně srovnáme MT tak, aby společně značky s A byly ve stejném vzájemném pořadí v MT jako v A. Dostaneme novou verzi posloupnosti MT (na obrázku označenou číslem 3).

Při průchodu trii dále narazíme na posloupnost (číslo 4 na obrázku) A. Najdeme shodné morfologické značky A a MT a zjistíme, že jsou čtyři. Protože všechny morfologické značky v aktuálním MT již byly srovnány vzhledem ke čtyřem ostatním značkám v posloupnosti, musíme se podívat na délku věty, podle které k tomuto srovnání došlo. Zjistíme, že tato délka je rovna pěti. Délka A je čtyři a tudíž můžeme srovnat MT s A. Stabilně setřídíme společné morf. značky (na obrázku číslo 5). Jak si lze všimnout, dostali jsme se do situace, kdy již nikdy nedojde k porovnání s žádnou zdrojovou větou, protože jsme srovnali všechny morfologické značky vzhledem ke všem v posloupnosti MT a délka věty, podle které k tomuto srovnání došlo, je rovna délce rekonstruované posloupnosti MT. Tudíž žádná lepší věta, s kterou by se dalo srovnávat, není. Maximálně může být v trii stejně dlouhá věta, která obsahuje všechny morfologické značky z MT. Pro tuto větu již ale nelze rozhodnout, jestli je lepší než ta, podle které jsme srovnávali naposledy.

Jelikož jsme vždy při srovnávání MT zároveň rovnali posloupnost S, tak máme nyní v posloupnosti S slova zrekonstruované věty.

3.2 Časová a prostorová složitost řešení

Při určování složitosti rekonstrukčního algoritmu si musíme určit, vzhledem k čemu budeme složitost počítat. V tomto případě jsme zvolili výpočet složitosti vzhledem k počtu zdrojových vět (věty načítané z ČNK). Dále předpokládáme, že délka vstupní věty (věta, kterou chceme rekonstruovat) je omezená nějakým celým číslem (například 100) neboli délka vstupní věty je konstantní.

Při rekonstrukci jedné věty je vždy procházena celá trie, což zabere $O(m)$, kde m je počet zdrojových vět. Každá zdrojová věta je použita nejvýše jedenkrát při rekonstrukci jedné vstupní věty. Protože je délka vstupní věty konstantní, je časová složitost celé rekonstrukční procedury $O(nm)$, kde n je počet vět určených k rekonstrukci.

Prostorová složitost je rovna $O(m)$ jelikož v nejhorším případě po vložení m zdrojových vět budeme mít v trii m různých cest z kořene do listů.

4. Rekonstrukce na základě syntaktické analýzy

V této sekci je popsán kompletní algoritmus řešení problému rekonstrukce původního pořadí slov za použití syntaktické analýzy včetně všech využívaných datových struktur.

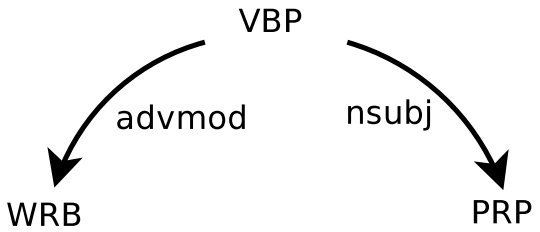
4.1 Popis řešení

4.1.1 Zpracování korpusů

Stejně jako v případě rekonstrukce na základě morfologické analýzy jsou ČNK a ANK zpracovány, tentokrát pomocí parseru (nástroj pro syntaktickou analýzu vět), pouze jedenkrát. Soubory vzniklé při parsování jsou přiloženy k rekonstrukční proceduře a při spuštění jsou přečteny. Tyto soubory mají pro každý cílový jazyk jiný formát. Přehled formátů, které v této práci používáme, a příklady souborů těchto formátů najdete v sekci 2.1.

Na obrázcích 4.1 a 4.2 jsou znázorněny věty "How are you?" a "Obecná odpověď na tuto otázku je sotva možná." jako závislostní stromy, kde slova jsou nahrazena příslušnými morfologickými značkami.

4.1.2 Reprezentace dat získaných z korpusů

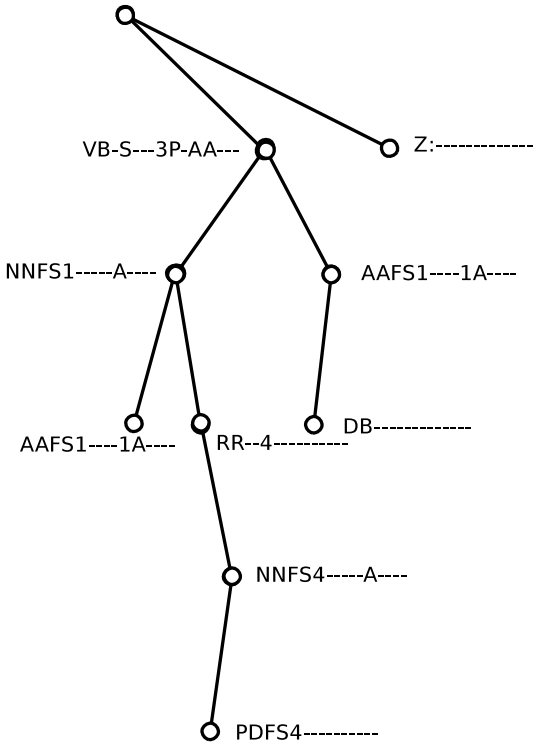


Obrázek 4.1: Závislostní strom anglické věty

Jeden z hlavních rozdílů mezi morfologickým a syntaktickým přístupem je v práci rekonstrukčního algoritmu. V případě morfologie je po spuštění procedury načten daný počet vět z korpusu daného jazyka do datové struktury - trie. Poté jsou postupně čteny jednotlivé věty určené k rekonstrukci a pomocí vytvořené datové struktury probíhá jejich rekonstrukce. V tomto případě se trie vytváří pouze jedenkrát na začátku procedury a je pro všechny rekonstruované věty shodná.

Naproti tomu rekonstrukční procedura založená na syntaxi načítá a zpracovává daný počet vět z korpusu cílového jazyka pro každou rekonstruovanou větu znovu. Informace získané z načítaných vět ukládá do datové struktury, v této práci nazvané - *mapa frekvencí*. Tento hlavní rozdíl je znázorněn na obrázcích 2.3 a 2.4.

Mapa je datová struktura s unikátním klíčem a hodnotou k tomuto klíči přidruženou. V podstatě jde o seznam, který lze indexovat řetězci. Procedura využívá



Obrázek 4.2: Závislostní strom české věty

jako klíč řetězce znaků a jako přidruženou hodnotu kladné celé číslo. Co vyjadřuje indexový řetězec a jemu přidružená hodnota je vysvětleno níže. Znázornění obecné mapy je vidět na obrázku 4.3.

Mapa frekvencí je tedy *mapa*, která využívá řetězcových indexů a celočíselných hodnot. Indexovým řetězcem je řetězec vytvořený spojením dvou morfologických značek, mezi které je vložen znak "|". Tento znak je vkládán z důvodu rozdílné délky morfologických značek u anglických textů. Příklad

$$"WRB|VBP"$$

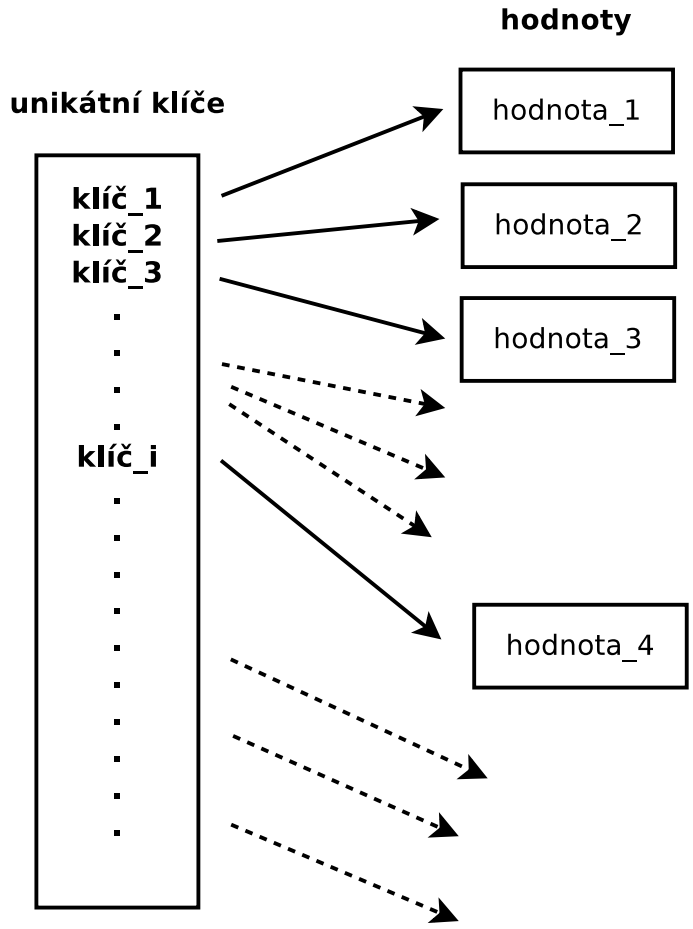
ukazuje vytvoření indexového řetězce z morfologických značek "WRB" a "VBP".

Při samotném vytváření *mapy frekvencí* je na počátku vytvořená prázdná *mapa*, kde všechny klíče mají hodnotu nastaveno na -1. *Mapa* je vytvářena ze znalosti morfologicky anotované věty určené k rekonstrukci. Klíče *mapy* jsou dány všemi možnými dvojicemi morfologických značek rekonstruované věty. Indexové řetězce jsou v podstatě variace druhé třídy ze všech morfologických značek rekonstruované věty. Počáteční prázdná *mapa* pro větu

"How_WRB are_VBP you_PRP ?_."

je znázorněna na obrázku 4.4.

Klíče (dvojice morfologických značek rekonstruované věty) vyjadřují všechny možné syntaktické závislosti všech slov rekonstruované věty, které by mohli potenciálně existovat. V závislostním stromě závisí vždy jedno slovo na jednom jiném. Slovo nemůže nikdy záviset na více než jednom jiném slovu. Každému slovu odpovídá jedna morfologická značka. Proto jsou klíče dvojicemi.



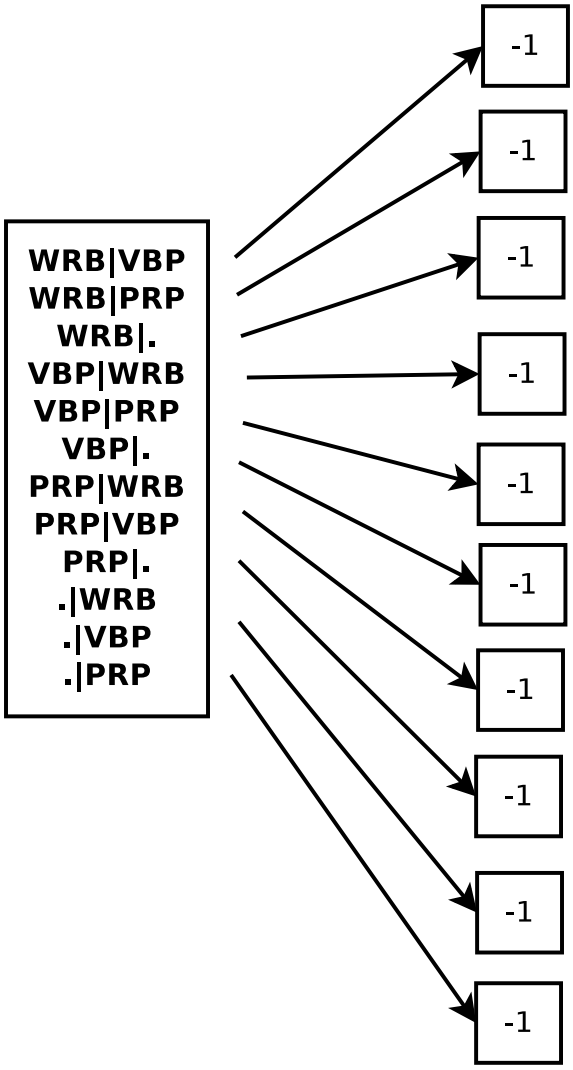
Obrázek 4.3: Schéma datové struktury - mapy

V dalším textu často používáme označení, že morf. značka závisí na jiné morf. značce. Ve skutečnosti je tímto označením myšleno, že na sobě závisí slova, těmto značkám odpovídající.

Nás však nezajímá zda morfologická značka A je závislá na morfologické značce B nebo naopak. Podstatný je fakt, že je mezi nimi závislost. Důvod, proč se v mapě frekvencí vyskytují oba klíče $A|B$ i $B|A$, je pro potřeby rozlišení, která značka z dané závislosti byla ve větě první a která druhá. Například klíč $A|B$ vyjadřuje, že ve větě existovala závislost mezi A a B a že A stálo v té dané větě před B.

Máme vytvořenou iniciální *mapu frekvencí* s hodnotami všech klíčů nastavenou na -1. Začneme načítat data z korpusu odpovídajícímu cílovému jazyku. Vždy po přečtení jedné zdrojové věty (věta načítaná z korpusu) a ověření shodnosti koncového interpunkčního znaménka s rekonstruovanou větou analyzujeme syntaktické závislosti této věty. Syntaktická závislost je vždy dána dvojicí morfologických značek v pevně daném pořadí. Oba parsovací nástroje poskytují u každé syntaktické závislosti navíc informaci o tom, která ze značek je před kterou.

Procházíme jednu závislost zdrojové věty za druhou. Označme aktuální analyzovanou závislost A, B kde A a B jsou morfologické značky v pořadí, v jakém se vyskytují ve zdrojové větě. Podíváme se do *mapy frekvencí*, zda existuje položka s klíčem $A|B$. Pokud ano, tak pokud je hodnota pod tímto klíčem rovna



Obrázek 4.4: Schéma iniciální mapy

-1, nastavíme novou hodnotu tomuto klíči na 1, jinak přičteme k hodnotě tohoto klíče 1. V případě, že položka s daným klíčem v *mapě frekvencí* není, neděláme nic a pokračujeme v analýze další syntaktické závislosti aktuální zdrojové věty.

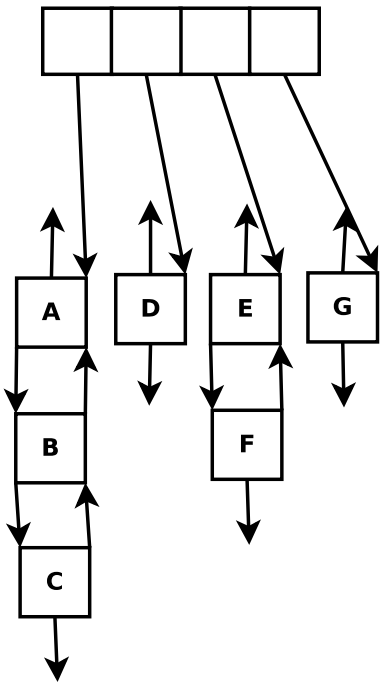
Tímto způsobem načteme a analyzujeme daný počet vět cílového jazyka. Získáme *mapu frekvencí* připravenou pro rekonstrukci rekonstruované věty. Hodnoty náležející jednotlivým klíčům této *mapy frekvencí* vyjadřují frekvence, v jakých se dané dvě morfologické značky vyskytují ve zdrojových větách ve vzájemné závislosti a daném pořadí. Je-li hodnota klíče $A|B = 132$ je jasné, že v daném počtu zdrojových vět se morfologická značka A závislá na morfologické značce B vyskytla celkem 132 krát v pořadí A před B.

Postup popsaný výše se provádí při rekonstrukci každé jedné věty znovu a *mapa frekvencí* je pro každou rekonstruovanou větu jinak velká a obsahuje jiné hodnoty za použití stejných zdrojových dat.

4.1.3 Rekonstrukční algoritmus

Rekonstrukční algoritmus pracuje na základě *mapy frekvencí* pro rekonstruovanou větu. Na vstupu tedy dostane morfologicky anotovanou větu určenou k rekonstrukci spolu s *mapou frekvencí* vztahující se k této větě.

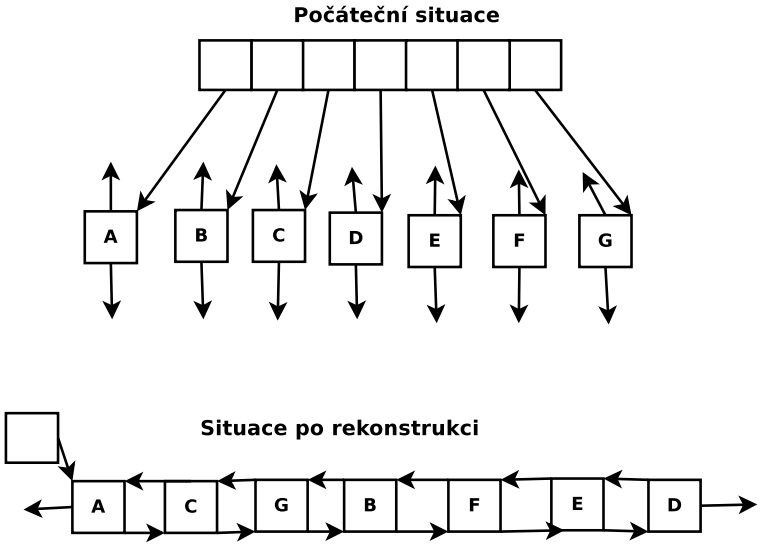
Na začátku je potřeba převést větu určenou k rekonstrukci do vhodné reprezentace. Procedura využívá reprezentace seznamu spojových seznamů. Prvek spojového seznamu odpovídá vždy morfologické značce spolu s odpovídajícím slovem a každý prvek má odkaz na svého předchůdce a svého následníka. Seznam spojových seznamů je udržován tak, aby v každé položce seznamu byl vždy počáteční uzel spojového seznamu. Reprezentace rekonstruované věty tvaru $ABCDEFG$ je znázorněna na obrázku 4.5. Šipka, která nespojuje dva uzly, symbolizuje, že prvek z kterého vychází, nemá žádného předchůdce nebo následníka.



Obrázek 4.5: Ukázka seznamu spojových seznamů

Reprezentací věty pomocí seznamu spojových seznamů je spousta. Na počátku rekonstrukce je věta reprezentována jako seznam spojových seznamů délky 1. Neboli každá morfologická značka bude ve vlastním spojovém seznamu. Po rekonstrukci je věta reprezentována seznamem spojových seznamů, který má délku 1. Na konci tedy bude v seznamu pouze jeden spojový seznam odpovídající zrekonstruované větě. Na obrázku 4.6 je vidět reprezentace rekonstruované věty před a po rekonstrukci. Definujme pojem *část věty* jako jeden spojový seznam ze seznamu reprezentujícího rekonstruovanou větu. Část věty odpovídá vždy již zrekonstruované skupině morfologických značek, která je podmnožinou všech morfologických značek rekonstruované věty.

Vlastní rekonstrukce probíhá tak, že se z *mapy frekvencí* vybírá vždy nejfrekventovanější závislost. U této závislosti se nejprve ověří zda má *podporu* (viz níže) ve stávající reprezentaci rekonstruované věty, a pokud ano, tak se podle této závislosti za dodatečných podmínek popsaných níže spojí dvě části věty

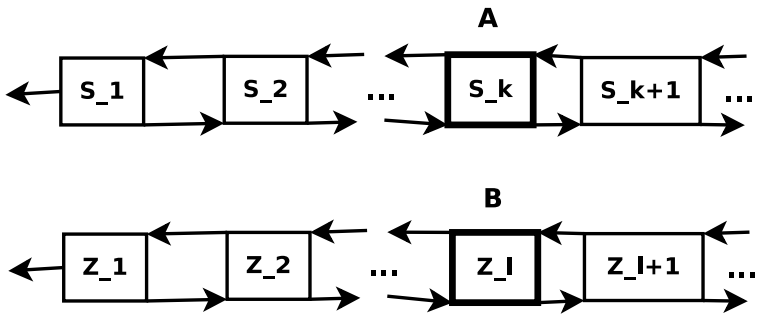


Obrázek 4.6: Ukázka reprezentace rekonstruované věty

do jedné nové části. V případě, že nejfrekventovanější závislost nemá *podporu* v aktuální reprezentaci rekonstruované věty, neděláme nic. Nakonec se aktuálně nejfrekventovanější závislost odstraní z mapy frekvencí a pokračuje se další iterací. Rekonstrukční algoritmus končí ve chvíli, kdy jsme buď vyčerpali všechny závislosti z *mapy frekvencí*, nebo pokud je aktuální reprezentace rekonstruované věty rovna reprezentaci po rekonstrukci, viz obrázek 4.6.

Pokud jsme vyčerpali všechny závislosti z mapy frekvencí a rekonstruovaná věta ještě není v reprezentaci po rekonstrukci (viz obrázek 4.6), provedeme sjednocení všech částí věty tak, jak jsou v seznamu za sebou.

Označme $A|B$ aktuálně nejfrekventovanější syntaktickou závislostí v *mapě frekvencí*. V aktuální reprezentaci rekonstruované věty se pokusíme nalézt morfologické značky A a B tak, aby A bylo v jiné části rekonstruované věty než B. Pokud se nám to podaří, tak říkáme, že syntaktická závislost $A|B$ má *podporu* ve stávající reprezentaci rekonstruované věty. Pokud se to nepodaří *podporu* nemá.

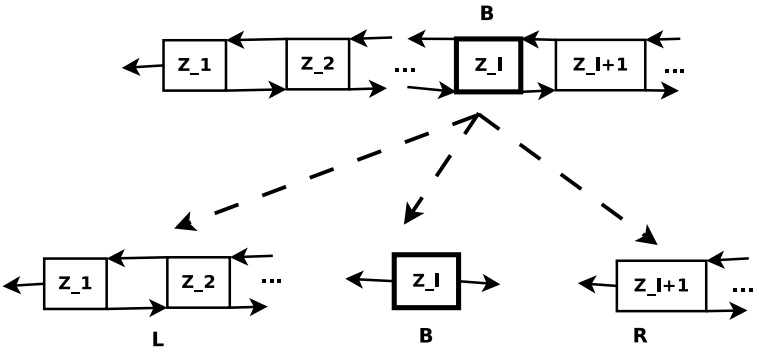


Obrázek 4.7: Situace na začátku spojování dvou částí věty

Zbývá vysvětlit, jak probíhá spojování dvou částí věty do jedné nové části. Označme $A|B$ syntaktickou závislost, podle které spojujeme. Označme S část věty, kde byla nalezena morfologická značka A a Z část věty, kde byla nalezena morfologická značka B. Nechť S_i značí i-tý prvek části S a Z_i značí i-tý prvek

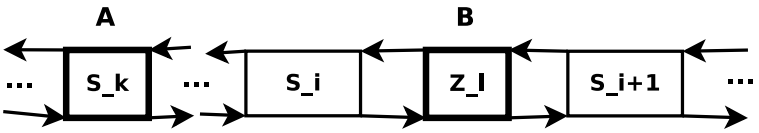
části Z. Index k značí pozici v části S, kde byla nalezena morfologická značka A a l je pozice v části Z, kde byla nalezena značka B. Tato počáteční situace je znázorněna na obrázku 4.7.

Nyní rozdělíme část Z na tři podčásti. První podčástí je část věty nacházející se nalevo od Z_l (označme ji L), druhá podčást je samotné Z_l a třetí část je část věty, která se nachází vpravo od Z_l (označme ji R). Viz obrázek 4.8.



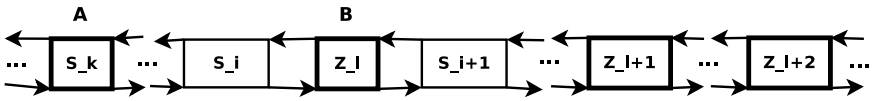
Obrázek 4.8: Situace po rozdělení na tři podčásti

Nyní vložíme podčást odpovídající Z_l do S dle *mapy frekvencí* (do S budeme postupně vkládat celou Z). Ze syntaktické závislosti $A|B$ víme, že S_k je před Z_l. Pokud S_k nemá žádného následníka, připojíme k S_k následníka Z_l. Pokud má S_k následníka podíváme se do *mapy frekvencí* na závislosti s klíči $Z_l|S_{k+1}$ a $S_{k+1}|Z_l$. Pokud první ze závislostí má větší hodnotu než druhá, tak vložíme Z_l mezi S_k a S_{k+1}. V opačném případě se posuneme z S_k na S_{k+1} a pokračujeme stejným způsobem. Nakonec získáme S', vyjadřující S po vložení Z_l. (obrázek 4.9)



Obrázek 4.9: Situace po vložení prostřední podčásti

Zbývá vložit podčásti L a R. Začneme s podčástí R. Podčást R rozdělíme na její první prvek Z_{l+1} a zbytek podčásti (Rs). O uzlu Z_{l+1} víme, že musí být za Z_l. Vložíme ho tedy za Z_l do S' stejným způsobem jako při vkládání Z_l za S_k do S. Uzly Rs vložíme obdobným způsobem jako uzel Z_{l+1}. Dostaneme se do situace znázorněné na obrázku 4.10. (S'')

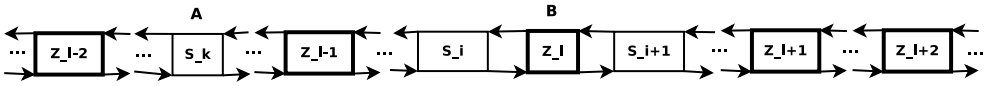


Obrázek 4.10: Situace po vložení pravé podčásti

Nakonec vložíme podčást L. Podčást L rozdělíme na její poslední prvek Z_{l-1} a zbytek podčásti (Ls). O Z_{l-1} víme, že musí být před Z_l v S''. Pokud Z_l nemá žádného předchůdce připojíme Z_{l-1} přímo před Z_l v S''. V opačném případě

(označme předchůdce Z_l S''_p) se podíváme do mapy frekvencí na hodnoty závislostí s klíči $S''_p|Z_{l-1}$ a $Z_{l-1}|S''_p$. Pokud je první hodnota větší než druhá, vložíme Z_{l-1} mezi S''_p a Z_l. V opačném případě se přesuneme z Z_l na S''_p a pokračujeme obdobným způsobem dokud se nepodaří vložit Z_{l-1} do S''. Po vložení Z_{l-1} pokračujeme stejným způsobem ve vkládání Z_{l-2} před Z_{l-1} v S'', dokud nevložíme celé Ls.

Výsledek spojení S a Z je vidět na obrázku 4.11. Prvky z Z vložené v S'' jsou vyznačeny tučným rámečkem.



Obrázek 4.11: Situace po spojení původních dvou částí do jedné nové

4.2 Časová a prostorová složitost řešení

Stejně jako v případě rekonstrukce na základě morfologické analýzy musíme nejprve určit vzhledem k čemu budeme složitost odhadovat. Opět to bude vzhledem k počtu zdrojových vět (vět načítaných z korpusu daného jazyka).

Každá zdrojová věta je načítána při rekonstrukci jedné věty právě jednou. Vzhledem k tomu, jak velké množství dat se používá k rekonstrukci věty, můžeme zanedbat vlastní délku ať rekonstruované věty tak i zdrojových vět. Z tohoto vyplývá, že časová složitost rekonstrukční procedury je lineární vzhledem k počtu využívaných zdrojových vět. Jelikož pro každou větu, která má být rekonstruována se musí znova přečíst všechny zdrojové věty ze souborů, je celková složitost tedy $O(nm)$. Kde m je počet zdrojových vět a n je počet vět určených k rekonstrukci.

Pro určení prostorové složitosti je důležité uvědomit si, že reprezentace všech zdrojových vět je zredukována na jednu *mapu frekvencí*, která má kvadratickou velikost vzhledem k délce rekonstruované věty. Je vidět, že prostorová složitost je tedy nezávislá na počtu načítaných zdrojových vět. Pokud tedy budeme brát délku rekonstruované věty za konstantu, je prostorová složitost procedury konstantní $O(1)$. Pokud bereme délku rekonstruované věty za nekonstantní (l), je prostorová složitost rekonstrukční procedury $O(l^2)$.

Oproti morfologickému přístupu jsme si tedy s časovou složitostí nepolepšili, nicméně při morfologickém přístupu jsme načetli zdrojová data pouze jednou a poté jsme je udržovali v paměti. V syntaktickém přístupu jsou zdrojová data načítána pokaždé, pro každou větu určenou k rekonstrukci, znovu ze zdrojových souborů. Ačkoliv tedy složitost časová zůstala, reálně bude syntaktický přístup pomalejší jelikož načítání ze souborů trvá déle. Na druhou stranu prostorová složitost se výrazně snížila. Z lineární složitosti na konstantní.

5. Vyhodnocení

V tomto oddílu uvádíme výsledky úspěšnosti rekonstrukčních procedur. Úspěšnost jsme testovali na vzorku 100 anglických vět a 100 českých vět různých délek. Vzorky vět byly vybírány náhodně, s ohledem na to, aby ve vzorku byly zastoupeny věty různých délek. Na tyto vzorky byla nejprve aplikována pomocná aplikace *MichaciProcedura*, která vzorek vět náhodně zamíchala. Zamíchání obou vzorků proběhlo pouze jednou. Na výstup aplikace *MichaciProcedura* byl poté aplikován tagger příslušného jazyka a jeho výstup byl dán na vstup postupně oběma rekonstrukčním procedurám s různým počtem zdrojových vět. Pro oba přístupy k rekonstrukci byla použita stejná testovací množina dat. Výsledky tohoto procesu jsou předloženy v této kapitole.

Při testování rekonstrukčích procedur bylo použito vždy až 100 000 zdrojových vět daného jazyka. Byly postupně brány vzorky 5, 10, 20, 30, 40, 50, 70, 100 tisíc zdrojových vět a pomocí nich byl zrekonstruován vzorek testovacích dat. Je důležité říci, že vzorky zdrojových vět nejsou disjunktní - vzorek 5000 zdrojových vět je podmnožinou vzorku 10000 zdrojových vět atd. V tabulce 5.1 uvádíme statistiku rozdělení souboru 100 testovacích vět obou jazyků podle délky jednotlivých vět.

Průměrnou délku vět ve vzorcích zdrojových vět si lze prohlédnou na obrázku 5.1

Definujme pojem *částečná shoda*, který znamená, že zrekonstruovaná věta má s původní větou před vložením do míchací procedury index podobnosti <= 2 a *úplná shoda* znamená, že věta po rekonstrukci má s původní větou před vložením do míchací procedury index podobnosti roven 0.

5.1 Vyhodnocení morfologického a syntaktického přístupu

Hodnoty v tabulkách 5.4, 5.5, 5.6, 5.7, 5.8, 5.9, 5.10, 5.11 jsou procentuelním vyjádřením počtu úspěšně zrekonstruovaných vět dané délky zaokrouhlené na dvě platné cifry za použití daného počtu zdrojových vět. Položka *celkem* v tabulkách vyjadřuje v procentech počet úspěšně zrekonstruovaných vět ze vzorku 100 vět odpovídajícího jazyka za použití příslušného počtu zdrojových vět.

V tabulkách 5.2 a 5.3 jsou zobrazeny údaje týkající se zdrojových vět pro každý jazyk zvlášť. Především jde o počet různých posloupností morfologických značek, průměrný počet vět na posloupnost morfologických značek a konečně počet různých cest z kořene do listů v trii. Poslední sloupec těchto tabulek se týká pouze morfologického přístupu.

Soubory s daty, podle kterých byly vytvořeny tabulky 5.4, 5.5, 5.6, 5.7, 5.8, 5.9, 5.10, 5.11, jsou k nalezení ve složce $CeskeVetyI/test\_files$ resp. ve složce $CeskeVetyII/test\_files$ na přiloženém cd (viz Příloha A). Všechny tyto soubory jsou ve tvaru čtveřic řádků: originální věta, zrekonstruovaná věta, index podobnosti, oddělovač ve formě řetězce "——".

délka věty (d.v)	anglická data # vět	česká data # vět
2	0	4
3	7	12
4	17	17
5	37	21
6	8	15
7	1	11
8	2	4
9	1	6
≥ 10	27	10
celkem	100	100

Tabulka 5.1: Rozdělení vět podle délky v testovacích množinách

# zdrojových vět (K)	# různých posloupností značek	průměrný počet vět na posloupnost značek	# cest kořen-list
5	2130	2.35	2118
10	5326	1.88	5080
20	14836	1.35	14138
30	23406	1.28	22369
40	33121	1.21	31872
50	42786	1.17	41184
70	61465	1.14	59083
100	87510	1.14	84339

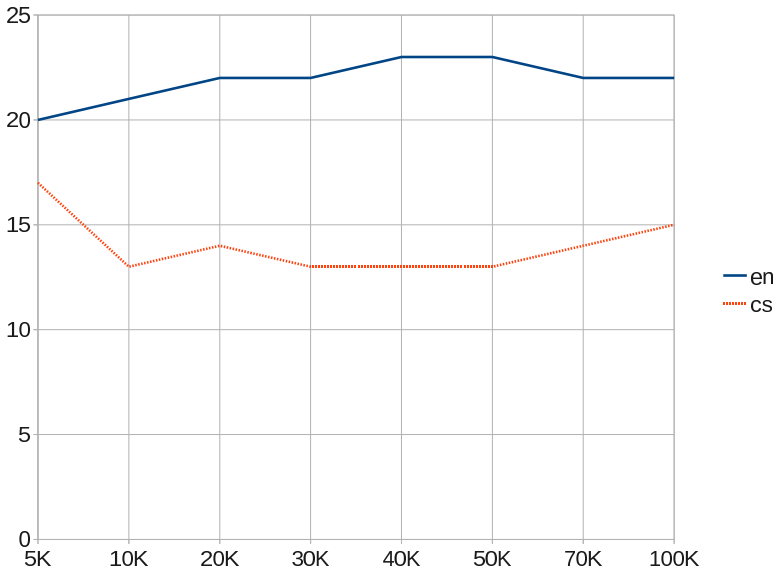
Tabulka 5.2: České vzorky

5.2 Porovnání s externím přístupem

Rekonstrukční procedury jsme porovnali s volně přístupnou rekonstrukční demo verzí na internetu (viz [2]) založenou na n-gramovém modelu. Porovnání proběhlo pomocí vzorku anglických dat, za použití 100 000 zdrojových vět. Jedná se o ten samý vzorek 100 anglických vět, který byl použit při vyhodnocení úspěšnosti našich rekonstrukčních procedur.

V tabulce 5.12 uvádíme pár příkladů tohoto porovnání. S je původní věta, S' je zamíchaná věta, S''_{morf} je rekonstrukce pomocí procedury založené na morfologii, S''_{synt} je rekonstrukce založená na syntaxi a S''_{demo} je rekonstrukce za použití porovnávaného dema.

Soubor s porovnáním všech 100 vět je k nalezení na přiloženém cd v adresáři *doc/*. Soubor má název *compare_two_methods* a je ve formátu pětice řádků: originální věta, zamíchaná věta, věta zrekonstruovaná pomocí morfologického přístupu, věta zrekonstruovaná pomocí syntaktického přístupu a věta zrekonstruovaná pomocí externí procedury. U zrekonstruovaných verzí věty je v závorce uveden



Obrázek 5.1: Graf průměrných délek vět v jednotlivých vzorcích

index podobnosti. Jednotlivé pětice jsou od sebe odděleny řetězcem ”———”.

5.3 Experiment s překladačem

Během práce byl proveden experiment s automatickým překladem. V rámci tohoto experimentu byl vzorek 100 anglických vět, používaných při vyhodnocování úspěšnosti našich procedur, přeložen pomocí aplikace Google Translator [5] a na výsledek překladu byly postupně spuštěny obě procedury s použitím 100 000 zdrojových vět. Pár příkladů je uvedeno v tabulce 5.13.

S je originální věta, S_{trans} je originální věta přeložená pomocí Google Translatoru, S_{morf} je zrekonstruovaná S_{trans} pomocí morfologického přístupu a S_{synt} je zrekonstruovaná S_{trans} pomocí syntaktického přístupu.

Soubor s výsledkem celého experimentu je k nalezení na přiloženém cd v adresáři *doc/*. Každá věta z testovací množiny vět je v tomto souboru ve tvaru čtveřice řádků: originální anglická věta, originální anglická věta přeložená do češtiny, přeložená věta zrekostruovaná pomocí morfologického přístupu a přeložená věta zrekonstruovaná pomocí syntaktického přístupu. Jednotlivé čtveřice jsou odděleny řetězcem ”———”.

5.4 Diskuse nad výsledky

Z výsledků je vidět, že obě rekonstrukční procedury jsou úspěšné především pro krátké věty. Je vidět, že evaluace na úplnou shodu má u obou přístupů a obou jazyků úspěšnost 20-30%. Vyjímkou je však syntaktický přístup u anglických dat. Zde úspěšnost dosahuje menších hodnot, zhruba 10-12%. Tento pokles je nejspíše způsoben velkou rozdílností vět v testovacím vzorku a zdrojových vět.

# zdrojových vět (K)	# různých posloupností značek	průměrný počet vět na posloupnost značek	# cest kořen-list
5	4909	1.02	4817
10	9474	1.06	9276
20	18123	1.10	17625
30	23312	1.29	22652
40	32737	1.22	31846
50	42261	1.18	41057
70	61234	1.14	59387
100	89821	1.11	86992

Tabulka 5.3: Anglické vzorky

Morfologický přístup								
d.v.	5K	10K	20K	30K	40K	50K	70K	100K
2	100	100	100	100	100	100	100	100
3	75	75	83	100	91	91	83	83
4	47	47	35	35	17	29	23	35
5	23	38	19	23	28	33	33	28
6	0	0	0	6	6	6	20	20
7	0	0	0	0	0	0	0	0
8	0	0	0	0	0	0	0	0
9	0	0	0	0	0	0	0	0
≥ 10	0	0	0	0	0	0	0	0
celkem	26	29	24	28	25	28	28	29

Tabulka 5.4: Česká data s úplnou shodou

Jinak řečeno ve zdrojových větách je nejspíše velmi malý výskyt syntaktických závislostí, které by se mohli potenciálně vyskytnout v testovacích větách.

Výsledky vyhodnocení na částečnou shodu se pohybují kolem 35-50% pro oba jazyky a oba přístupy k rekonstrukci.

Procedury nefungují pro delší věty. Jedním z důvodů je fakt, že čím je věta delší tím je větší pravděpodobnost, že neexistuje pouze jeden korektní pořádek slov této věty. Delší věty také obsahují více slov se shodnou morfologickou značkou, na něž je nahlíženo jako na shodná slova a tudíž procedury neví jak tyto slova vzájemně seřadit.

Z výsledků je vidět, že zvýšení počtu zdrojových vět ne vždy způsobí zvýšení úspěšnosti procedur. Je to způsobeno principem fungování rekonstrukčních algoritmů.

Morfologický přístup								
d.v.	5K	10K	20K	30K	40K	50K	70K	100K
3	71	28	28	28	14	71	28	42
4	52	64	52	58	52	41	64	64
5	29	24	37	37	32	40	54	43
6	12	12	12	12	12	12	12	12
7	0	0	0	0	0	0	0	0
8	0	0	0	0	0	0	0	0
9	0	0	0	0	0	0	0	0
≥ 10	0	0	0	0	0	0	0	0
celkem	26	23	26	27	23	28	34	31

Tabulka 5.5: Anglická data s úplnou shodou

Morfologický přístup								
d.v.	5K	10K	20K	30K	40K	50K	70K	100K
2	100	100	100	100	100	100	100	100
3	100	100	100	100	100	100	100	100
4	64	82	70	76	64	58	76	82
5	42	66	57	61	47	57	61	52
6	6	13	13	20	20	13	26	26
7	9	9	0	0	0	0	0	0
8	0	25	25	25	25	25	25	25
9	0	0	0	0	0	0	0	0
≥ 10	0	0	0	0	0	0	0	0
celkem	38	48	43	46	41	41	47	46

Tabulka 5.6: Česká data s částečnou shodou

Morfologický přístup								
d.v.	5K	10K	20K	30K	40K	50K	70K	100K
3	100	100	100	100	100	100	100	100
4	88	82	88	82	76	76	88	82
5	43	45	51	54	45	54	59	51
6	12	25	37	37	37	25	50	25
7	0	0	0	0	0	0	0	0
8	0	0	0	0	0	0	0	0
9	0	0	0	0	0	0	0	0
≥ 10	0	0	0	0	0	0	0	0
celkem	39	40	44	44	40	40	48	42

Tabulka 5.7: Anglická data s částečnou shodou

Syntaktický přístup								
d.v.	5K	10K	20K	30K	40K	50K	70K	100K
2	100	100	100	100	100	100	100	100
3	50	50	66	66	66	75	75	66
4	29	35	35	41	41	41	41	35
5	23	23	28	38	38	23	38	23
6	26	6	6	6	13	6	6	13
7	0	0	9	0	0	0	0	0
8	0	0	0	0	0	0	0	0
9	0	0	0	0	0	0	0	0
≥ 10	0	0	0	0	0	0	0	0
celkem	24	22	26	28	29	26	29	25

Tabulka 5.8: Česká data s úplnou shodou

Syntaktický přístup								
d.v.	5K	10K	20K	30K	40K	50K	70K	100K
3	42	42	42	42	42	42	42	33
4	23	11	17	17	23	23	23	26
5	13	8	8	10	18	18	18	21
6	0	0	0	0	0	0	0	0
7	0	0	0	0	0	0	0	0
8	0	0	0	0	0	0	0	0
9	0	0	0	0	0	0	0	0
≥ 10	0	0	0	0	0	0	0	0
celkem	12	8	9	10	14	14	14	14

Tabulka 5.9: Anglická data s úplnou shodou

Syntaktický přístup								
d.v.	5K	10K	20K	30K	40K	50K	70K	100K
2	100	100	100	100	100	100	100	100
3	100	100	100	100	100	100	100	100
4	82	82	82	82	82	82	82	76
5	66	57	66	66	66	52	66	57
6	33	26	13	6	20	20	26	26
7	9	9	9	9	9	9	9	9
8	0	0	0	0	0	0	0	0
9	0	0	0	0	0	0	0	0
≥ 10	0	0	0	0	0	0	0	0
celkem	50	47	47	46	48	45	49	46

Tabulka 5.10: Česká data s částečnou shodou

Syntaktický přístup								
d.v.	5K	10K	20K	30K	40K	50K	70K	100K
3	100	100	100	100	100	100	100	100
4	82	88	82	82	76	76	76	80
5	37	45	37	32	40	40	40	45
6	12	12	12	12	12	12	0	0
7	0	0	0	0	0	0	0	0
8	0	0	0	0	0	0	0	0
9	0	0	0	0	0	0	0	0
≥ 10	0	0	0	0	0	0	0	0
celkem	36	40	36	34	36	36	35	35

Tabulka 5.11: Anglická data s částečnou shodou

S	*My name is peter .*	
S'	*My name Peter is .*	
S''_{morf}	*Peter is My name .*	$SI(S, S''_{morf}) = 3$
S''_{synt}	*My Peter is name .*	$SI(S, S''_{synt}) = 4$
S''_{demo}	*My name is peter .*	$SI(S, S''_{demo}) = 0$
S	*How are you ?*	
S'	*are How you ?*	
S''_{morf}	*How are you ?*	$SI(S, S''_{morf}) = 0$
S''_{synt}	*How you are ?*	$SI(S, S''_{synt}) = 3$
S''_{demo}	*How are you ?*	$SI(S, S''_{demo}) = 0$
S	*Yes , I do .*	
S'	*Yes , do I .*	
S''_{morf}	*Yes , I do .*	$SI(S, S''_{morf}) = 0$
S''_{synt}	*Yes I do , .*	$SI(S, S''_{synt}) = 3$
S''_{demo}	*Yes , I do .*	$SI(S, S''_{demo}) = 0$
S	*Look for some people .*	
S'	*Look for people some .*	
S''_{morf}	*Look for some people .*	$SI(S, S''_{morf}) = 0$
S''_{synt}	*for some Look people .*	$SI(S, S''_{synt}) = 2$
S''_{demo}	*some people Look for .*	$SI(S, S''_{demo}) = 2$
S	*How long are they ?*	
S'	*How long they are ?*	
S''_{morf}	*How are they long ?*	$SI(S, S''_{morf}) = 3$
S''_{synt}	*How long they are ?*	$SI(S, S''_{synt}) = 3$
S''_{demo}	*How long are they ?*	$SI(S, S''_{demo}) = 0$

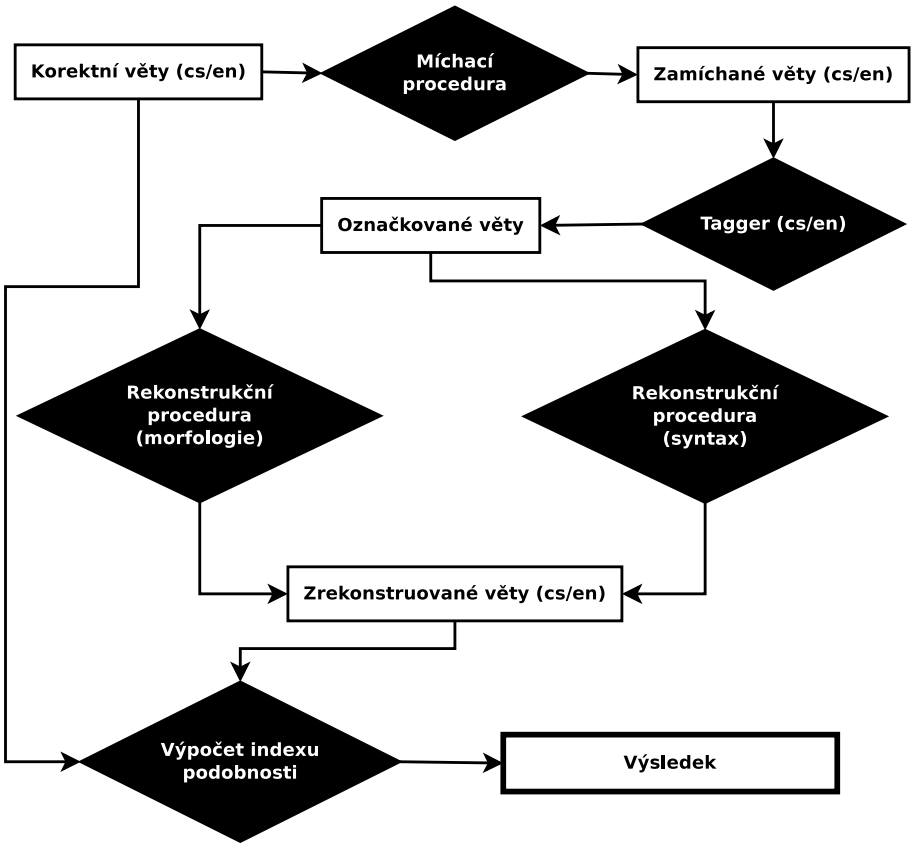
Tabulka 5.12: Porovnání s demo verzí viz [2]

S	*Will you be good ?*
S_{trans}	*Budete se dobře ?*
S_{morf}	*Budete dobře se ?*
S_{synt}	*Budete se dobře ?*
S	*How are you ?*
S_{trans}	*Jak se máš ?*
S_{morf}	*Jak se máš ?*
S_{synt}	*Jak se máš ?*
S	*I am fine , thanks .*
S_{trans}	*Mám se dobře, díky .*
S_{morf}	*Mám , dobře díky se .*
S_{synt}	*díky , dobře se Mám .*
S	*It is great .*
S_{trans}	*To je skvělé .*
S_{morf}	*je To skvělé .*
S_{synt}	*To je skvělé .*
S	*What is your name ?*
S_{trans}	*Jaké je vaše jméno ?*
S_{morf}	*Jaké vaše jméno je ?*
S_{synt}	*vaše jméno Jaké je ?*
S	*Look for some people .*
S_{trans}	*Podívejte se na některé lidi .*
S_{morf}	*Podívejte se některé na lidi .*
S_{synt}	*Podívejte na se některé lidi .*
S	*Do you like cooking ?*
S_{trans}	*Máte rádi vaření ?*
S_{morf}	*rádi Máte vaření ?*
S_{synt}	*rádi Máte vaření ?*
S	*What did they say ?*
S_{trans}	*Co říkali ?*
S_{morf}	*Co říkali ?*
S_{synt}	*Co říkali ?*
S	*My name is peter .*
S_{trans}	*Jmenuji se Petr .*
S_{morf}	*se Jmenuji Petr .*
S_{synt}	*Petr Jmenuji se .*
S	*Talk to my father .*
S_{trans}	*Promluvte si s mým otcem .*
S_{morf}	*s mým otcem Promluvte si .*
S_{synt}	*Promluvte si s mým otcem .*
S	*We had their dog .*
S_{trans}	*Měli jsme svého psa .*
S_{morf}	*svého psa jsme Měli .*
S_{synt}	*svého jsme Měli psa .*

Tabulka 5.13: Experiment s překladačem

6. Implementace - aplikace CeskeVetyI a II

6.1 Uživatelská dokumentace aplikací CeskeVetyI a II



Obrázek 6.1: Schéma použití této práce

Aplikace *CeskeVetyI* resp. *CeskeVetyII* je textově orientovaná aplikace běžící pod platformou UNIXových operačních systémů. Aplikace se ovládá výhradně pomocí příkazové řádky. Aplikace má několik volitelných parametrů, podle kterých pracuje. Po zpracování definovaných vstupů skončí. Přehled všech parametrů je uveden v tabulce 6.1. Na cd je vedle zdrojových souborů přiložen i binární soubor s názvem $ceskeVetyI$ resp. $ceskeVetyII$, který lze rovnou spustit. Lze ho také přímo zkompilovat, například pomocí kompilátoru g++ nebo pomocí přiloženého Makefile. Schéma použití aplikací je znázorněno na obrázku 6.1.

V případě, že je zadaný jiný než v tabulce 6.1 definovaný parametr, aplikace skončí odchycenou vyjímkou s popisem chyby. To samé se stane pokud například argument parametru $-c$ je mimo rozsah $0-100000$, cílový jazyk je jiný než český nebo anglický a pokud vstupní soubor neexistuje.

Parametr	Povinný	Význam
-i	ne	Definuje cestu ke vstupnímu souboru aplikace
-o	ne	Definuje cestu k výstupnímu souboru aplikace
-l	ne	Definuje cílový jazyk
-c	ne	Definuje počet zdrojových vět určeních k rekonstrukci.
-h	ne	Vypíše přehled všech parametrů aplikace.

Tabulka 6.1: Přehled všech parametrů aplikace CeskeVetyI a II

6.1.1 Vstup

Vstup aplikace tvoří morfologicky anotovaný soubor s větami určenými k rekonstrukci. Cesta k tomuto souboru se zadává jako argument parametru programu $-i$. Lze ji zadat buď absolutně nebo relativně vzhledem k aktuálnímu adresáři. V případě spuštění aplikace bez specifikace vstupního souboru se použije defaultní předpřipravený vstupní soubor nacházející se v $temp\_files/input$.

6.1.2 Výstup

Výstup aplikace tvoří textový soubor se zrekonstruovanými větami, kde každá je na novém řádku. Cesta k výstupnímu souboru se zadává jako argument nepovinného parametru $-o$. Opět ji lze zadat jak absolutně, tak relativně. V případě vynechání tohoto parametru při spouštění aplikace se použije defaultní výstupní soubor nacházející se v $temp\_files/output$.

Kromě výstupního souboru má aplikace ještě jeden výstup. Výpisy o průběhu rekonstrukce vstupních vět na standartním výstupu. Z těchto výpisů je snadné zjistit, které věty ze vstupního souboru již byly zrekonstruovány a které nikoliv. Na standartní výstup je postupně vypisována informace udávající, která vstupní věta je aktuálně rekonstruována.

6.1.3 Jazyk rekonstrukce

Aplikace rekonstruuje české a anglické věty. Deklaraci toho, které věty má rekonstruovat, zajišťuje parametr $-l$, za nímž následuje definice cílového jazyka. Na výběr je možnost cs/en. Pokud není tento parametr definován, jako defaultní jazyk je brán český jazyk.

6.1.4 Počet zdrojových vět

Dalším parametrem aplikace je parametr určující počet zdrojových vět, které se mají k rekonstrukci použít. Parametr má název $-c$ následovaný kladným celým číslem z rozsahu $0 - 100000$. Defaultní počet zdrojových vět je nastaven na 5000.

6.1.5 Nápověda

Aplikace má také parametr vypisující krátkou nápovědu k používání aplikace. Tímto parametrem je $-h$, který nemá žádný argument. Po spuštění aplikace s tímto parametrem se nespustí rekonstrukční procedura, jen se vypíše nápověda

a aplikace skončí. Tento parametr musí být zadáván samostatně a nesmí být zároveň s ním zadány žádné jiné parametry.

6.1.6 Příklad typ. použití aplikace CeskeVetyI resp. CeskeVetyII

Typický příklad příkazu spouštějící aplikaci CeskeVetyI resp. CeskeVetyII by mohl vypadat následovně:

$$ceskeVetyI\ -l\ cs\ -c\ 20000\ -i\ /home/user/input.txt\ -o\ /home/user/output.txt$$

Po spuštění takovéhoto příkazu bude aplikace:

- rekonstruovat české věty
- k rekonstrukci využije 20000 zdrojových vět z ČNK
- vstupní, morfologicky anotované české věty bude načítat z */home/user/input.txt*
- výsledek rekonstrukce uloží do */home/user/output.txt*

6.2 Programátorská dokumentace CeskeVetyI

Aplikace je psána v objektově orientovaném jazyce - C++.

6.2.1 Přehled souborů

Aplikace CeskeVetyI je rozdělená do několika souborů. Zvlášť jsou soubory s definicemi a deklaracemi tříd a metod. Až na pár vyjímek každý soubor obsahuje právě jednu třídu včetně definic jejích metod.

Následuje seznam souborů s definicemi hlavních tříd procedury.

- **backtrack.cpp** Obsahuje definice metod stejnojmenné třídy, která zajišťuje algoritmus rekonstrukční procedury popsaný v oddílu 3.1.3. Dále obsahuje definice pomocné třídy **WordInfo**, která uchovává informace o jednom slově rekonstruované věty. Věta je reprezentována jako seznam objektů této třídy.
- **loadData.cpp** Obsahuje definice metod třídy **DataLoader**, která zajišťuje veškeré načítání dat ze souborů, potřebných pro správnou práci aplikace (soubor s větami určenými k rekonstrukci, soubory korpusů obsahující zdrojové věty). Dále obsahuje definice pomocné datové třídy **WordMTag** sloužící pro načítání vět ze souborů.
- **mainClass.cpp** Obsahuje definice metod hlavní třídy aplikace, tato třída slouží v podstatě jako interface pro programátory a zjednodušuje používání ostatních tříd aplikace.
- **main.cpp** Obsahuje hlavní metodu **main()**, která voláním metod ostatních tříd provádí veškerou práci aplikace.

- **node.cpp** Obsahuje definice obsažené ve stejnojmenné datové třídě. Tato třída reprezentuje jeden uzel v datové struktuře - trie.
- **optionParser.cpp** Definice třídy zajišťující parsování a vhodné zpracování programových parametrů včetně ošetření chyb, které mohou nastat jak při zadávání parametrů, tak při jejich zpracování.
- **trie.cpp** Obsahuje definice metod stejnojmenné třídy, tato třída zajišťuje stavbu datové struktury - trie.
- **Makefile** Makefile aplikace obsahuje příkazy pro kompilaci ze zdrojových souborů.

Všechny výše uvedené soubory až na **main.cpp** a **Makefile** mají svůj hlavičkový soubor, kde se nacházejí deklarace metod a datových položek.

6.2.2 Zpracování programových parametrů

Parsování programových argumentů zajišťuje třída **OptionParser** v souboru **optionParser.cpp**. Obsahuje datové položky reprezentující parametry, které je možné aplikaci zadat. Zároveň obsahuje i ošetření nekorektně zadaných programových parametrů. K této třídě se přistupuje přes metodu **parseOptions()** třídy **Main**. Metoda **parseOptions()** nejprve pomocí třídy **OptionParser** naparsuje programové parametry do svých privátních proměnných a ověří, zda byly vůbec zadány, poté případně nastaví defaultní hodnoty.

6.2.3 Zpracování zdrojových dat

Po zpracování programových argumentů, začne procedura načítat a zpracovávat daný počet zdrojových vět daného jazyka. O načítání se stará třída **dataLoader** ze souboru **loadData.cpp**. Ke třídě se přistupuje, podle cílového jazyka, pomocí metod **loadSourceData()** a **loadSourceDataEn()** třídy **Main**.

Tyto metody postupně otevírají jeden zdrojový soubor za druhým a dokud není dosaženo požadovaného počtu zdrojových vět, tak v cyklu načítají jednu zdrojovou větu za druhou pomocí metod **getSentence()** a **getSentenceEn()** třídy **dataLoader**. Tyto metody vrací vždy jednu větu reprezentovanou seznamem ukazatelů na objekty pomocné třídy **WordMTag**. Každá načtená věta je vložena do budované datové struktury - trie, pomocí třídy **Trie** a její metody **pushInTrie()**, která má dva argumenty - vkládanou větu a její koncové interpunkční znaménko. Kořen budované trie je udržován v privátní datové položce **root** třídy **Main**.

6.2.4 Rekonstrukční algoritmus

Po načtení a reprezentaci zdrojových vět dojde k rekonstrukci vstupních vět. Samotnou rekonstrukci má na starosti třída **Backtrack**, která je obsluhována metodou **reconstructSentences()** třídy **Main**. Tato metoda postupně, podle cílového jazyka, načítá pomocí metody **getSentence()** resp. **getSentenceEn()** třídy **dataLoader** jednotlivé věty určené k rekonstrukci a voláním metody **trieBacktrack()** třídy **Backtrack** dojde k samotnému zrekonstruování načtené věty

podle algoritmu popsaného v 3.1.3. Výsledek je následně zapsán do výstupního souboru. Algoritmus provádí 3 kroky v cyklu dokud nejsme nakonci souboru s větami určenými k rekonstrukci:

1. Načti další větu určenou k rekonstrukci.
2. Zrekonstruuj tuto větu.
3. Zapiš výsledek do výstupního souboru.

6.3 Programátorská dokumentace CeskeVetyII

Stejně jako aplikace CeskeVetyI je i CeskeVetyII psána v jazyce C++.

6.3.1 Přehled souborů

- **dataLoader.cpp** Obsahuje definice metod stejnojmenné třídy, která zajišťuje veškeré načítání dat ze souborů, potřebných pro správnou práci aplikace (soubor s větami určenými k rekonstrukci, soubory korpusů obsahující zdrojové věty). Tento soubor obsahuje ještě definici a deklaraci třídy **Word**, pomocí které jsou reprezentovány rekonstruované věty. Instance této třídy odpovídá jednomu slovu rekonstruované věty. Spolu s vlastním slovem každá instance obsahuje i odpovídající morfologickou značku a odkaz na předchozí slovo a následující slovo. V podstatě se jedná o spojový seznam. Reprezentace věty jako seznam instancí této třídy je popsána v oddílu 4.1.3.
- **ceskeVetyII.cpp** Obsahuje definice metod hlavní třídy aplikace, tato třída slouží v podstatě jako interface pro programátory a zjednodušuje používání ostatních tříd aplikace stejně tak, jako tomu bylo u aplikace CeskeVetyI.
- **main.cpp** Obsahuje hlavní metodu **main()**, která voláním metod ostatních tříd provádí veškerou práci aplikace.
- **optionParser.cpp** Definice třídy zajišťující parsování a vhodné zpracování programových parametrů včetně ošetření chyb, které mohou nastat jak při zadávání parametrů, tak při jejich zpracování.
- **reorder.cpp** Obsahuje definice metod stejnojmenné třídy, tato třída zajišťuje provádění rekonstrukčního algoritmu.
- **Makefile** Makefile aplikace obsahuje příkazy pro kompilaci ze zdrojových souborů.

Soubory .cpp kromě **main.cpp** a **Makefile** mají odpovídající hlavičkové soubory s deklaracemi tříd a jejich metod.

6.3.2 Zpracování programových parametrů

Parsování programových argumentů probíhá naprosto shodně jako v 6.2.2.

6.3.3 Rekonstrukční algoritmus

Po naparsování programových argumentů neprobíhá krok zpracovávající zdrojová data. Tento krok je schovaný v rekonstrukčním algoritmu, jelikož pro každou větu určenou k rekonstrukci se zpracovávájí zdrojová data znovu.

Cyklus rekonstrukčního algoritmu má v tomto případě čtyři kroky:

1. Načti další větu určenou k rekonstrukci.
2. Zpracuj zdrojová data vzhledem k této větě.
3. Zrekonstruuj tuto větu.
4. Zapiš výsledek do výstupního souboru.

Celou rekonstrukci zajišťuje metoda **reorderSentences()** třídy **CeskeVety-II**. Tato metoda obsahuje již zmiňovaný hlavní cyklus procedury.

Nejprve se pomocí metod **readNextInputSentence()** a **readNextInput-SentenceEn()** třídy **DataLoader** čtou postupně věty určené k rekonstrukci. Věty určené k rekonstrukci jsou načítány do podoby seznamu spojových seznamů viz 4.1.3. Na každou načtenou větu je zavolána metoda **reorderSentence()** třídy **Reorder**, která provádí samotnou rekonstrukci dané věty.

Tato metoda nejprve voláním privátní metody **proccessResources()** provede načtení a zpracování zdrojových textů do reprezentace *mapy frekvencí* (viz 4.1.2) vzhledem k aktuálně rekonstruované větě, prostřednictvím metody **proccessData()** třídy **DataLoader**. Poté zavolá privátní metodu **algorithm()**, která na základě vytvořené *mapy frekvencí* zrekonstruuje danou větu podle algoritmu popsaného v oddílu 4.1.3.

Nakonec je zrekonstruovaná věta zapsána do výstupního souboru (každá věta na novou řádku) a pokračuje se další iterací, dokud nejsme nakonci souboru s větami určenými k rekonstrukci.

7. Závěr

Práce představuje dvě procedury pro rekonstrukci původního pořadí slov ve větách českých a anglických. Obě aplikace pracují v reálném čase. První rekonstruuje na základě morfologické analýzy a druhá na základě syntaktické analýzy.

V rámci této práce proběhlo:

1. seznámení s ČNK a ANK
2. seznámení a práce s českým a anglickým taggerem
3. seznámení a práce s českým a anglickým parserem
4. navržení, implementace a dokumentace rekonstrukční procedury založené na morfologii
5. navržení, implementace a dokumentace rekonstrukční procedury založené na syntaxi
6. navržení, implementace a popis míchací procedury
7. navržení, implementace a popis indexu podobnosti
8. vyhodnocení úspěšnosti rekonstrukčních procedur
9. porovnání úspěšnosti procedury s externími rekonstrukčními procedurami
10. aplikace rekonstrukčních procedur na vzorek 100 anglických vět přeložených do českého jazyka

V této sekci uvádíme některé dobré i špatné vlastnosti procedur. Nakonec předkládáme návrhy na zlepšení aplikace do budoucna.

7.1 Klady a zápory aplikace

Obě aplikace ke své práci využívají externí nástroje pro taggování a parsování textů. Z toho se odvíjí první zápor aplikace. Závislost úspěšnosti procedur na úspěšnosti externích nástrojů.

Nespornou výhodou procedur je využívání relativně malého množství zdrojových dat. Oproti rekonstrukčním algoritmům založených na n-gramovém principu [7], které se obvykle trénují na obrovských množstvích dat, je počet zdrojových vět v našich procedurách zanedbatelný.

Další výhodou je jistě také jazyková nezávislost procedur.

Je dobré si uvědomit, že námi nastavená metrika (index podobnosti) úspěšnosti rekonstrukce vět, není absolutně relevantní. Český jazyk je jazykem s velmi volným pořádkem slov ve větě, což náš index podobnosti nezohledňuje. Jedna věta může být vyjádřena stejnými slovy jen v jiném pořadí a stále to bude korektní česká věta. Nalezení indexu podobnosti, který by bral ohled na volnost slovosledu, je minimálně tak těžký problém, jako samotná rekonstrukce.

V práci se odkláníme od vlastních slov rekonstruované věty a místo toho se zaobíráme odpovídajícími morfologickými značkami. Tato volba skrývá další

zápor aplikace. V případě, že dvě slova věty mají shodnou morfologickou značku, není procedura schopna určit, které ze slov má být první a které druhé v pořadí věty, protože je na ně nahlíženo jako na shodná slova. Úspěšnost procedur tedy klesá s rostoucím množstvím slov v rekonstruované větě, která mají shodnou morfologickou značku.

Z evaluace je vidět, že procedury rekonstruují úspěšně především krátké věty. Podíváme-li se však do korpusů, zjistíme, že obsahují spíše rozsáhlá souvětí. Odtud pramení problém s interpunkcí na začátku zrekonstruovaných vět. Pokud se ve větě vyskytuje čárka nebo nějaká další interpunkce vyjma koncové, tak při rekonstrukci může dojít k umístění této interpunkce na začátek věty. Tento problém se týká především morfologického přístupu k problému. Krátká rekonstruovaná věta se může v korpusu vyskytnout jako vedlejší věta nějakého souvětí, která je od hlavní věty oddělena čárkou. Věta je tedy zrekonstruována podle tohoto souvětí s tím, že algoritmus neví, že rekonstruovaná věta již nemá žádnou hlavní větu a tudíž umístí interpunkci na začátek.

7.2 Návrhy na zlepšení

Přestože mají aplikace lineární časovou složitost, tak při větším množství zdrojových dat trvá rekonstrukce, především u rekonstrukce na základě syntaktické analýzy, nezanedbatelnou dobu. Většinu času spotřebovává načítání a zpracování daného počtu zdrojových vět. Dobrým nápadem na zlepšení by tedy do budoucna mohlo být nějaké vhodné dodatečné předzpracování zdrojových dat tak, aby načítání zabralo méně času.

Na druhou stranu každé další předzpracování zdrojových dat je přítěží pro uživatele. Pokud se nyní rozhodne uživatel otestovat algoritmus na větším množství zdrojových dat než současná aplikace povoluje, stačí pouze aplikovat parser resp. tagger na přidávaná zdrojová data a v aplikaci přenastavit jednu konstantu na požadovanou hodnotu a může začít testovat. Pokud by program vyžadoval nějak předzpracovaná data bude muset uživatel provádět ještě toto předzpracování.

Dalším možným zlepšením procedur je přidávání dalších podmínek, řídicích hlavní rekonstrukční algoritmus. Podmínky lze získat například rozborem struktury morfologických značek nebo využitím analytické funkce.

Přínosem by také určitě bylo navržení nějaké lepší metriky na určování korektnosti vět, která by byla nezávislá na originální větě a fungovala jako jakýsi regulátor rekonstrukčních algoritmů (V průběhu algoritmu by průběžně počítala svojí hodnotu a pokud by se dostala pod nějakou hranici, tak by algoritmus skončil). Důvod pro navržení této metriky tkví v tom, že v průběhu algoritmu je věta častokrát ve formě lépe zrekonstruované než nakonci algoritmu. Metrika by algoritmus zastavila ve chvíli, kdy je věta uspokojivě zrekonstruovaná.

Seznam použité literatury

[1] REPPEN, Randi; IDE, Nancy; SUDERMAN, Keith. *American National Corpus (ANC) First Release*. Linguistic Data Consortium, Philadelphia. 2003 [cit. 2011-05-22].

[2] XI, Ning. *Automatic Words Reordering* [online]. 2010 [cit. 2011-05-21]. Dostupné z WWW: `http://nlp.nju.edu.cn/~xin/Software/Reordering/default.aspx`

[3] Barbora Vidová Hladká, et al. *Czech Academic Corpus 2.0*. 2008 [cit. 2011-05-22]. Linguistic Data Consortium, Philadelphia.

[4] *Český národní korpus - SYN2005*. Ústav Českého národního korpusu FF UK, Praha 2005. Dostupný z WWW: `http://www.korpus.cz`.

[5] Google. *Google* [online]. 2010 [cit. 2011-05-21]. Google Překladač. Dostupné z WWW: `http://translate.google.cz/`

[6] N-gram. In *Wikipedia : the free encyclopedia* [online]. St. Petersburg (Florida) : Wikipedia Foundation, 2004-09-14, last modified on 2011-05-05 [cit. 2011-05-21]. Dostupné z WWW: `http://en.wikipedia.org/wiki/N-gram`

[7] RAAB, Jan, et al. *Průvodce českým akademickým korpusem 2.0* [online]. 2008 [cit. 2011-05-22]. Popis morfologických značek. Dostupné z WWW: `http://ufal.mff.cuni.cz/rest/CAC/doc-cac20/cac-guide/cz/html/ch13.html`.

[8] VIDOVÁ HLADKÁ, Barbora, et al. *Průvodce českým akademickým korpusem 2.0* [online]. 2008 [cit. 2011-05-22]. Průvodce Českým akademickým korpusem 2.0. Dostupné z WWW: `http://ufal.mff.cuni.cz/rest/CAC/doc-cac20/cac-guide/cz/html/index.html`.

[9] KLEIN, Dan; D. MANNING, Christopher. *The Stanford Natural Language Processing Group* [online]. 2003 [cit. 2011-05-21]. The Stanford Parser: A statistical parser. Dostupné z `http://nlp.stanford.edu/software/lex-parser.shtml`.

[10] TOUTANOVA, Kristina, et al. *The Stanford Natural Language Processing Group* [online]. 2003 [cit. 2011-05-21]. Stanford Log-linear Part-of-Speech Tagger. Dostupné z `http://nlp.stanford.edu/software/tagger.shtml`.

Seznam tabulek

Seznam obrázků

Příloha A - Obsah CD-ROM

Součástí práce je i CD-ROM s aplikacemi a elektronickou verzí této práce. V této příloze je popsána adresářová struktura přiloženého CD-ROMu.

doc/	obsahuje elektronickou verzi této práce, výsledek experimentu s překladačem a výsledek porovnání s externí rekonstrukční procedurou
MichaciProcedura/	obsahuje zdrojové a binární soubory aplikace a testovací soubor
README.txt	obsahuje návod k použití
IndexPodobnosti/	obsahuje zdrojové a binární soubory aplikace a testovací soubory
README.txt	obsahuje návod k použití
CeskeVetyI/	obsahuje zdrojové a binární soubory aplikace
doc/	
doxygen/	obsahuje programátorskou dokumentaci vygenerovanou programem Doxygen
README.txt	základní popis aplikace pro uživatele
resources/	obsahuje zdrojové soubory z ČNK a ANK
temp_files/	obsahuje defaultní vstupní a výstupní soubor
test_files/	obsahuje všechny soubory s výsledky testování podle kterých byly vytvořeny tabulky v sekci 5
CeskeVetyII/	obsahuje zdrojové a binární soubory aplikace
doc/	
doxygen/	programátorská dokumentace generovaná programem Doxygen
README.txt	obsahuje základní popis aplikace pro uživatele
resources/	obsahuje zdrojové soubory z ČNK a ANK
temp_files/	obsahuje defaultní vstupní a výstupní soubor
test_files/	obsahuje všechny soubory s výsledky testování viz sekce 5