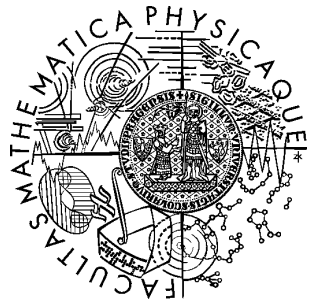
Univerzita Karlova v Praze
Matematicko-fyzikální fakulta

BAKALÁŘSKÁ PRÁCE



Martin Majliš

Sumarizace textu

Ústav formální a aplikované lingvistiky

Vedoucí ročníkového projektu: Mgr. Pavel Pecina
Studijní program: Informatika, programování

2008

Děkuji svému vedoucímu za odborné vedení má práce, za rady, materiály a čas, který mi během jejího vypracování věnoval, stejně jako všem, kteří mě nějak podpořili.

Prohlašuji, že jsem svou bakalářskou práci napsal samostatně a výhradně s použitím citovaných pramenů. Souhlasím se zapůjčováním práce a jejím zveřejňováním.

V Praze dne 8. 8. 2008 Martin Majliš

Obsah

Seznam příkladů

Seznam tabulek

Název práce: Sumarizace textu
Autor: Martin Majliš
Katedra (ústav): Ústav formální a aplikované lingvistiky
Vedoucí bakalářské práce: Pavel.Pecina@mff.cuni.cz
e-mail vedoucího: Pavel.Pecina@mff.cuni.cz

Abstrakt: V předložené práci jsou vysvětleny základní principy automatické sumarizace, evaluace a základními pojmy, které se v této oblasti používají. Dále obsahuje popis implementace systém pro automatickou sumarizace a evaluaci textů – CsummaK (Czech Summarization Kit). Součástí tohoto systému jsou základní algoritmy pro tvorbu extraktů a jejich evaluaci, jejichž popis je také součástí této práce. Tento systém byl použit pro tvorbu automatických extraktů z novinových článků. Pro získání referenčních extraktů byl vytvořen další systém, který umožňuje uživatelům on-line vytvářet extrakty novinových článků. V práci je také provedeno měření kvality jednotlivých algoritmů, jejich kombinací s různou hodnotou parametrů společně s diskuzí nad možnostmi praktického nasazení.

Klíčová slova: automatická sumarizace, extrakt, automatická evaluace

Title: Text summarization
Author: Martin Majlis
Department: Institute of Formal and Applied Linguistics
Supervisor: Mgr. Pavel Pecina
Supervisor's e-mail address: Pavel.Pecina@mff.cuni.cz

Abstract: The present work explains the basic principles of automatic summarization, evaluation and fundamental concepts, which are used in this field. It also includes a description of a system for automatic text summarization and evaluation - CSummaK (Czech Summarization Kit). As part of this system are basic algorithms for creating sentence extract summaries (Cenroid, Lead, Position, Random, Relevance Measure, etc.) their evaluation (Precision, Recall, F-Measure, etc.), whose description is also part of this work. This system was used for production of automatic extracts from news articles. Another system was developed for obtaining reference extracts, which allows users to create on-line extracts from news articles. In this work is also evaluated quality of single algorithms, their combination with of different parameters, together with discussion of the possibilities of practical application.

Keywords: automatic summarization, extract, automatic evaluation

Kapitola 1

Úvod do problematiky

V současné době přibývají nové informace tak rychle, jako nikdy v historii. Každý den přibývá několik milionů nových webových stránek [1], takže již není v lidských silách přečíst každou informaci, která by ho zajímala. Proto roste význam nástrojů, které nám umožní snadnější rozhodnutí, zda-li nás daná informace zajímá, nebo ne. Mezi tyto nástroje můžeme zařadit i automatickou sumarizaci.

Ve všednodenním životě se setkáváme s určitou formou sumarizace téměř všude, aniž bychom si to nějak výrazně uvědomovali. Novinové nadpisy jsou shrnutím obsahu samotného článku. První odstavec novinového článku může být také jeho shrnutím. A samotný novinový článek může obsahovat shrnutí mnohem obsáhlejší studie. Trailer na nový film je přehledem nejzajímavějších momentů, které nás mají přesvědčit, abych film shlédli celý. Abstrakty vědeckých článků jsou tradiční formou souhrnů, které píší buď samotní autoři nebo profesionálové, kteří se řídí doporučenými postupy. Tabulku s fotbalovými výsledky je také možné považovat za shrnutí průběhu fotbalové sezóny.

Cílem automatické sumarizace je vzít zdroj informace, vybrat z něho důležité informaci a prezentovat je takovým způsobem, aby uživateli přinesl maximální užitek. Proto se můžou souhrny z jednoho zdroje lišit v závislosti na tom, jaké na ně uživatelé kladli požadavky. Zda-li je zajímal celkový přehled, nebo spíše jedno konkrétní téma.

Smyslem této práce je zjistit, jak tvoří extrakty novinových článků lidé, protože nejsou dostupná data pro češtinu. Proto bude nutné implementovat systém, který usnadní získávání těchto dat. Dalším cílem je implementovat systém, který by umožňoval tvorbu automatických extraktů. Tento systém by měl být použitelný jak pro koncového uživatele, tak i pro výzkum jednotlivých algoritmů. Součástí práce je také implementace nejvýznamnějších algoritmů na tvorbu extraktů. Tyto automaticky vytvořené extrakty budou dále porovnány s extrakty vytvořenými lidmi, aby bylo možné porovnat jejich kvalitu, potažmo jednotlivých algoritmů a jejich kombinací. Proto budou také implementovány algoritmy pro automatické srovnávání extraktů. Všechny tyto dílčí části budou spojeny do jednoho komplexního celku s názvem *CSummaK* - Czech Summarization Kit.

Kapitola dvě se zabývá samotnou problematikou sumarizace a shrnuje jednot-

livé kroky celého procesu společně s přehledem nejdůležitějších pojmů, které se v této oblasti používají. Součástí této kapitoly je také přehled již existujících řešení. V závěru této kapitoly jsou prezentovány požadavky na systém pro automatickou sumarizaci společně s návrhem toho, jak jednotlivé požadavky vyřešit.

Třetí kapitola obsahuje přehled podporovaných formátů a jednotlivých komponent, které byly v rámci práce implementovány.

Čtvrtá kapitola se zaměřuje na uživatelskou část používání systému. Popisuje nezbytné softwarové vybavení pro jeho kompilaci. Dále obsahuje popis parametrů, které program podporuje. Převážná část je ale věnována popisu konfiguračního souboru společně s velkým množstvím příkladů pro snazší pochopení.

Pátá kapitola představuje další podpůrné programy, které byly v rámci práce realizovány.

V šesté kapitole jsou prezentovány výsledky hodnocení intelektuálních extraktů, automatických extraktů dílčích algoritmů a jejich kombinací společně s diskuzi nad dosaženými výsledky a důvody úspěchu, případně neúspěchu jednotlivých metod.

Osmá a závěrečná kapitola shrnuje výsledky dosažené v této práci společně se s rekapitulací toho, co nebylo implementováno, a co by implementováno být mohlo.

Kapitola 2

Sumarizace textů

V této kapitole se nachází seznámení se základními principy automatické sumarizace a evaluace, pojmy, které se v tomto oboru používají společně s přehledem obdobných systémů. Závěr kapitoly je věnován návrhu CSummaKu.

2.1 Co je sumarizace

Cílem sumarizace je zredukovat obsah dokumentu tak, aby zůstaly zachovány hlavní myšlenky. Podle Luhna [9] je možné vytvořit extrakt dokumentu, který je velký 20% vstupního dokumentu ale se srovnatelným množstvím informací jako obsahoval celý dokument. Výsledkem sumarizace může být buď *extract*, nebo *abstrakt*. Mani [10] ve své knize uvádí následující definice.

> Extrakt je typ souhrnu, který obsahuje výhradně materiál, který byl součástí vstupního dokumentu.

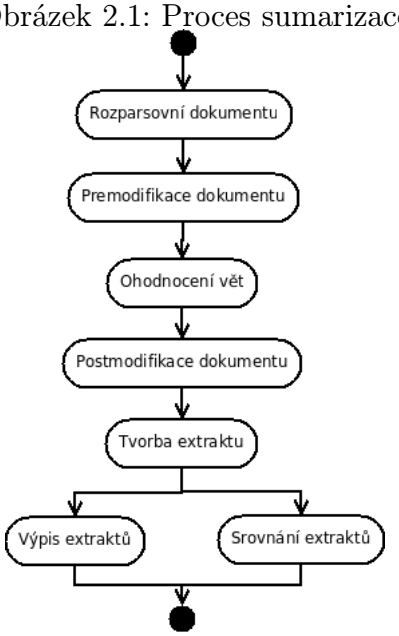
Takže extrakt s mírou komprese 20% obsahuje 20% materiálu z původního dokumentu. Může obsahovat 20% slov z původního dokumentu, nebo 20% vět případně 25% odstavců.

> Abstrakt je typ souhrnu, který obsahuje alespoň část textu, která není součástí vstupního dokumentu.

Typicky abstrakt parafrázuje věty, které jsou součástí původního dokumentu. Abstrakt může mít vyšší míru komprese a přesto může obsahovat více informací než mnohem delší extrakt.

Aby měl vůbec souhrn nějaký smysl, je nutné, aby přinášel uživateli nějaký užitek. Proto je důležité, aby souhrn byl *relevantní* (relevance, salience). Tady aby z původního textu obsahoval ty nejdůležitější informace a to jak z pohledu dokumentu, tak i účelu, ke kterému byl daný souhrn vytvořen. Jednotlivé věty by měly na sebe plynule navazovat, mít dostatečnou *soudružnost* (coherence, cohesion), aby byl výsledek dobře čitelný. V neposlední řadě je důležité, aby dokument neobsahoval *redundantní* (redundant) informace. Tyto informace by totiž nepřinášely uživateli žádný dodatečný užitek, jen by prodlužovali souhrn.

Obrázek 2.1: Proces sumarizace



Jak jsem se již zmínil v úvodu, tak sumarizaci mohou provádět buď lidé, potom se jedná o *intelektuální sumarizaci*, nebo může být prováděna strojově a jedná se tady o *automatickou sumarizaci* (FAS - Fully Automatic Summarization).

Počítače jsou lepší než lidé při vytváření souhrnů z větších objemů dat, zatímco lidé jsou mnohem lepší při odvozování založeném na kontextu případně na obecných znalostech. Takže stejně jako u automatického překladu můžeme uvažovat o spolupráci mezi počítačem a člověkem. U automatického překladu se používá MAHT (Machine Assisted Human Translation) a HAMT (Human Assisted Machine Translation) tak obdobně se dá u automatické sumarizace použít MAHS (Machine Assisted Human Summarization) a HAMS (Human Assisted Machine Summarization).

Sumarizátor je nástroj, nejčastěji program, který provádí automatickou sumarizaci. Součástí sumarizátorů můžou být také komponenty, které neprovádí samotnou sumarizaci, ale jsou nezbytné pro praktickou použitelnost sumarizátoru. Tvorba automatického sumarizátoru byla jedním z cílů této práce, a proto v následující kapitole popíšu podrobněji proces automatické sumarizace a jak se tento proces projevil v návrhu systému.

2.2 Proces automatické sumarizace

Proces sumarizace se skládá z několika na sobě nezávislých kroků. Každý z těchto kroků je možné vykonat několika různými způsoby.Jednotlivé kroky jsou zobrazeny na obrázku 2.1. Jejich bližší popis je zmíněn dále v dokumentu.

2.2.1 Parsování dokumentu

Nejprve je nutné vstupní data načíst. Jako zdroj dat může sloužit textový dokument, webová stránka případně databáze. Aby vůbec bylo možné data tímto systémem zpracovat, tak musí být v elektronické a textové podobě. V závislosti na tom, odkud byla data získána je potřeba provést dodatečné předzpracování. Po provedení těchto kroků by měl být dokument převeden na takovou interní reprezentaci, se kterou je schopen pracovat zbytek systému.

Převedení na společnou znakovou sadu

Vstupní data můžou mít rozdílnou znakovou sadu. Proto je nutné nejdřív všechny převést na jednu společnou. V tomto případě byla jako společná znaková sada zvolena ISO-8859-2, protože s ní byly schopny pracovat všechny externí nástroje.

Čištění vstupního dokumentu

Čištění vstupního dokumentu je nutné provést pokaždé, pokud vstupní dokument obsahuje také jiné informace než je čistý text. Příkladem dokumentu, na kterém je nutné toto čištění provést je webová stránka. Na webové stránce je nutné nejdřív nutné identifikovat část, která obsahuje text článku a z této části ještě navíc odstranit formátovací značky, reklamy, odkazy na další články atd. Data, pro tuto práci byla získávána z webových stránek, a proto je bylo nutné před použitím pročistit.

Segmentace

Cílem segmentace je ve vstupního textu rozpoznat hranice základních jednotky jako jsou odstavce, věty, slova a interpunkční znaménka. Tento problém není zdaleka triviální, protože rozpoznávání na základě mezer a interpunkčních znamének není dostatečné, protože věty může obsahovat zkratky, řadové číslovky, sousloví apod. V tomto případě byl k provedení lexikální analýzy použit externí nástroj *textseg* od Pavla Češky.

Lemmatizace

V dokumentu (obzvláště v češtině) se vyskytují slova v mnoha různých tvarech, a proto je výhodné převést je do základního tvaru - provést *lemmatizaci*. Pokud se v textu objevují slovní tvary: „Extrakty“, „extraktů“, „extrakt“, „extrakty“ nejedná se o čtyři různé pojmy, ale o 4 různé tvary od základního tvaru „extrakt“. Výsledný text poté obsahuje méně slov, což zmenšuje velikost prostoru ve vektorovém modelu [19].

Lemmatizaci je možné provádět několika různými způsoby. Buď použitím slovníku, kde pro každé slovo máme uveden jeho základní tvar. Nevýhodou takovéhoto systému je velikost slovníku, ale při dostatečně velkém slovníku je možné

získat velmi dobré výsledky. Další možností je odstraňování afixů. *Afix* je společné označení pro prefixy (předpony) a sufixů (přípon). Toto odstranění může být opět prováděno pomocí slovníku jednotlivých prefixů a sufixů, nebo na základě pravidel, jak jsou tyto afixy tvořeny. Také je možné tuto lemmatizaci provádět statisticky na základě frekvence shluků jednotlivých písmen. Takto je možné rozpoznávat afixy a kořen. Výhodou této metody je, že je nezávislá na vstupním jazyce. V tomto případě byl k provedení lemmatizace použit externí nástroj od Jana Hajiče [7].

Převedení na jednotnou reprezentaci

Z předchozích kroků je zřejmé, že vstupní text mohl projít celou řadou externích nástrojů. Proto je nezbytné každý vstupní text převést na shodnou reprezentaci, která umožní v dalších krocích fungovat nezávisle na vstupním formátu.

V praxi můžou jednotlivé dílčí kroky vykonávat různé externí nástroje, které očekávají vstup a produkují výstup v nekompatibilních formátech. Proto i mezi jednotlivými kroku můžou být zařazeny transformátory mezi jednotlivými formáty.

V této práci je parsování rozděleno na několik úrovní a řeší ho několik na sobě nezávislých programů. Novinový článek z vybraných severů je možné převést na prostý text. Libovolný prostý text je možné převést do CSTS formátu (3.1.1) pomocí externích nástrojů zmíněných v kapitolách 2.2.1 a 2.2.1. CSummaK samotný očekává vstupní dokumenty buď ve formátu CSTS, nebo SentenceId ((3.1.2).

2.2.2 Premodifikace dokumentu

Před samotným hodnocením vět je dobré provést předzpracování, během kterého může být struktura dokumentu upravena tak, aby následné ohodnocení proběhlo rychleji a s lepšími výsledky. Mezi takovéto úpravy je možné zahrnout odstranění stop-slov (stop words) případně dalších nevýznamných slov, jako jsou předložky, spojky a částice.

2.2.3 Ohodnocení vět

Ohodnocování vět je proces, kdy jednotlivým větám z dokumentu je přiřazeno *dílčí skóre*, které reprezentuje významnost věty v rámci dokumentu z pohledu konkrétního algoritmu. Jednotlivé hodnotící algoritmy se liší v tom, které věty považují za důležité, ale obecně platí, že čím je věta důležitější, tím by měla obdržet vyšší skóre. Pro získání lepších výsledků může být výhodné kombinovat více metod současně. Potom je ale nutné přidat nástroje, které umožní změnit váhu, které jednotlivé algoritmy na výpočtu celkového skóre mají, stejně jako nástroje, které upraví rozsah, které jednotlivé algoritmy dávají jako skóre. Pokud by 1 algoritmus přiřazoval větám skóre v rozmezí 0 – 1, a další v rozmezí 0 – ∞, s tím, že průměrné skóre by bylo 100, tak by se hodnocení prvního algoritmu vůbec neuplatnilo.

2.2.4 Posmodifikace dokumentu

Po ohodnocení jednotlivých vět je ještě dobré provést další modifikace dokumentu. Po ohodnocení vět mohlo nastat, že dvě velmi podobné věty byly považovány za velmi důležité, ale ve skutečnosti jsou redundantní. Tak by jejich faktické vybrání uživateli nepřineslo žádný dodatečný užitek. Proto se může vyplatit jednu z nich odstranit.

2.2.5 Tvorba extraktu

V předchozích krocích byl zmodifikován dokument (mohlo dojít k odstranění slov z vět, nebo vět samotných) a jeho jednotlivým větám bylo přiřazeno skóre. V této fázi jsou z původního dokumentu vybráno požadované množství vět na základě *celkového skóre*, kterého je součtem dílčích skóre.

2.2.6 Prezentováni extraktu

Výsledný extrakt je nutné prezentovat uživateli v odpovídající podobě. Pokud uživatel vložit vstupní text prostřednictvím webového formuláře, tak jako webovou stránku. Případně ve formátu, který umožní nějaké další zpracování.

2.2.7 Srovnání extraktů

Tato část není nutná z uživatelského hlediska, ale může být užitečná, pokud plánujeme automaticky vytvořené extrakty mezi sebou porovnávat. Proto jsou součástí CSummaKu také komponenty pro automatickou evaluaci.

2.3 Důležité pojmy

V následující kapitole bych chtěl představit další pojmy, které souvisí s problematikou sumarizace, a které budu dále používat.

2.3.1 Evaluační metody

Pokud chceme vyvíjet systém pro automatickou sumarizaci, potřebujeme rozpoznávat, jak kvalitní souhrny vytváří. Lidská evaluace je vysoce kvalitní, ale je drahá a velmi pomalá, proto nemůže být použita v průběhu vývoje. Proto byly vyvinuty automatické evaluační metody, které umožňují provádět evaluaci mnohem častěji. Evaluačních metod existuje celá řada, blíže dále představím pouze ty, které je výhodné použít pro srovnávání automatických extraktů, jako jsou precision, recall a f-measure.

Precision

Precision (česky: přesnost) je míra, která udává, jak hodně získané věty odpovídají referenčnímu extraktu. Výpočet je:

$$\text{precision} = \frac{|\{\text{věty z ref. extraktu}\} \cap \{\text{věty vybrané systémem}\}|}{|\{\text{věty vybrané systémem}\}|}$$

Precision nabývá hodnot mezi 0 a 1. Čím víc se hodnota blíží 1, tím je shoda lepší.

Ze vzorce je zřejmá nevýhoda této míry. Pokud dokument obsahoval věty 1 - 10, a referenční extrakt obsahoval věty 1, 2 a 5. Tak pokud systém označil za důležitou pouze větu 1, tak dosáhl skóre 1, ale výsledek není nikterak kvalitní, proto je dobré nutné tuto míru kombinovat s metodou recall (2.3.1) nebo f-measure (2.3.1).

Recall

Recall (česky: úplnost) je míra, jak velký podíl z vybraných vět odpovídá referenčnímu extraktu. Výpočet je následující:

$$\text{recall} = \frac{|\{\text{věty z ref. extraktu}\} \cap \{\text{věty vybrané systémem}\}|}{|\{\text{věty z ref. extraktu}\}|}$$

Recall nabývá hodnot mezi 0 a 1. Čím víc se hodnota blíží 1, tím je shoda lepší.

Ze vzorce je zřejmá nevýhoda této míry. Pokud dokument obsahoval věty 1 - 10, a referenční extrakt obsahoval věty 1, 2 a 5. Tak pokud systém označil za důležité všechny věty (1 - 10), tak dosáhl skóre 1, ale výsledek není nikterak kvalitní, proto je dobré nutné tuto míru kombinovat s metodou precision (2.3.1) nebo f-measure (2.3.1).

F-Measure

F-Measure je míra, které kombinuje precision a recall tak, aby jediná hodnota vystihovala podobnost dvou dokumentů. Jedná se vlastně o jejich vážený harmonický průměr.

$$\text{F} - \text{measure} = 2 \cdot \frac{\text{precision} \cdot \text{recall}}{\text{precision} + \text{recall}}$$

2.3.2 Idf

IDF (inverse document frequency) je míra, která určuje, jak je daný term významný v dané kolekci termů (získá se vydělením celkového počtu dokumentů

počtem dokumentů, ve kterém se daný term vyskytoval a z této hodnoty navíc spočtením logaritmu).

$$\mathrm{idf_i} = \log \frac{|D|}{|\{d_j : t_i \in d_j\}|}$$

- $|D|$: celkový počet dokumentů v korpusu
- $|\{d_j : t_i \in d_j\}|$: počet dokumentů, ve kterých se daný term vyskytuje

2.4 Obdobné aplikace

Pro český jazyk není tolik dostupných nástrojů pro automatickou sumarizaci jako pro angličtinu. Pro angličtinu existují služby jako NewsInEssence [14], Newsblaster [12], NewsFeed Researcher [13], které plní obdobnou funkci, procházejí anglicky psané zpravodajské servery (CNN, Fox News, Reuters, atd.) shlukují podobné článku do skupin a provádějí multidokumentovou sumarizaci, případně hledají podobné články. Microsoft Word dokáže také vytvořit extrakt z editovaného dokumentu, ale tato funkcionalita není dostupná pro češtinu. Přestože ve srovnání produktů MS Word 2007 a OpenOffice.org Writer na oficiálních stránkách OpenOffice.org uvedeno [15], že Writer by měl také tuto funkcionalitu také obsahovat, nepovedlo se mi tuto funkci najít, ani v nápovědě, ani na oficiálním webu.

2.4.1 MEAD

MEAD [16] je je komplexní, veřejné, open source, multidokumentové vícejazyčné prostředí pro automatickou sumarizaci, který využívá více než 500 různých organizací. Implementuje celou řadu řadu algoritmů a metod pro automatickou evaluaci dokumentu. Jeho flexibilní architektura umožňuje přidávat vlastní algoritmy. Poskytuje také podporu pro strojové učení.

Při návrhu CSummaKu jsem vycházel z tohoto systému a implementoval jsem většinu základních algoritmů, které jsou jeho součástí.

2.4.2 Zprávy Google

Služba *Zprávy Google* [6] shlukuje podobné zprávy z různých zpravodajských serverů a jako náhled zobrazuje prvních asi 250 znaků. Tento přístup se jeví jako velmi výhodný, protože velké množství zpravodajských serverů má v prvním odstavci shrnutí celého článku, které napsal zřejmě samotný redaktor.

2.5 Příklady použití

Automatická extrakce textů má širokou škálu uplatnění. Při návrhu CSummaKu byl kladen důraz na použití v následujících aplikacích.

Přes webové rozhraní uživatel vloží text, kompresní poměr, případně z nabídky osvědčených konfigurací vybere takovou, která mu nejvíce vyhovuje a systém mu po zaslání požadavku zobrazí jim vytvořený extrakt.

Pro vědecké srovnání různých extraktorů, kdy je potřeba nejdříve najít optimální konfiguraci a následné je ověřit proti jiným systémům.

2.6 Analýza

2.6.1 Požadavky

Z popisu procesu sumarizace a funkčnosti MEADu je zřejmé, že na systém na automatickou sumarizaci je kladeno velké množství požadavků. Mezi základní bych zařadil požadavky na:

rozšiřitelnost Mělo by být možné systém rozšířit o další komponenty. Díky tomu bude možné systém používat i pro testování dalších algoritmů.

svoboda Systém by neměl omezovat uživatele v tom, jaký programovací jazyk použije pro implementaci nových komponent.

výkon Vzhledem k rostoucímu objemu informací a rostoucí komplexnosti algoritmů je nutné, aby byl systém dostatečně výkonný.

základ Systém by měl do začátku implementovat nejzákladnější komponenty pro jednotlivé kroku sumarizace.

integrovatelnost Systém samotný, případně jeho dílčí součástí by mělo být možné použít v dalších aplikací. Proto je nutné navrhnout přehledné API pro usnadnění pozdější integrace.

použitelnost Systém by měl umožnit své nasazení v celé šíři možných aplikací, od výzkumu jednotlivých sumarizačních metod až po tvorbu extraktů z textů zadaných přes webové rozhraní.

2.6.2 Návrh aplikace

Operační systém

Program bude primárně naprogramován pro operační systém GNU/Linux, případně další Unixové operační systémy. Důvodem pro tento typ operačního systému je dobrá podpora dávkového zpracování, dostupné nástroje pro testování kvality kódu - valgridn, rats, gcov, nástroje pro snadné spojování dílčích součástí - bash, sed, grep stejně jako snadná instalace potřebných knihoven.

Programovací jazyk

Jako programovací jazyk pro samotny systém bude C++, protože je řádově rychlejší než interpretované jazyky a je možné ho propojit s kódem napsaným v Perlu, Pythonu, případně dalších. Tím je zajištěno, že jádro aplikace bude dostatečně výkonné a uživatelé si budou moci vybrat, v čem napíší své komponenty.

Pro webovou část projektu budou použity programovací jazyk PHP a Javascript. Pro prezentaci bude použit značkovací XHTML společně s kaskádovými styly (CSS).

Pomocné skripty, které budou spojovat funkcionalitu různých nástrojů budou napsány ve vhodném jazycích - bash, PHP, R, sed.

Návrh architektury

Z kapitoly 2.2 je zřejmé, že je potřeba 6 rozdílných typů komponent, kde každý typ komponenty zajišťuje 1 krok samotné sumarizace. Původně jsem chtěl celý systém sumarizace založit na programech jako je Ant nebo make, protože proces sumarizace odpovídá způsobu jakým pracují. Aby bylo možné porovnat 2 extrakty, je nutné nejdřív tyto extrakty vytvořit. Aby bylo možné tyto extrakty vytvořit, je nutné načíst vstupní soubor, rozparsovat ho, ... a vytvořit extrakt. Abych tento koncept ověřil, vytvořil jsem systém modulů pro ant, které tuto funkčnost zajišťovali. Nevýhodou tohoto systému bylo, že konfigurační soubor byl velmi nepřehledný.

Předpokládaný způsob použití jsem proto konzultoval s vedoucím. Důležité bylo, aby fungoval obdobně jako další nástroje, které využívají. Proto byl návrh změněn na postupné zpracování jednotlivých fází. Vstupní dokumenty převést do CSTS, protože vstupem můžou již být soubory v CSTS formátu. Ze vstupních souborů vytvořit extrakty. Takto vytvořené extrakty porovnat s jinými extrakty. Díky tomu by mělo být snazší celý systém propojit s dalšími nástroji.

Komponenty

Při návrhu jednotlivých dílčích částí budu používat osvědčené návrhové vzory z *GoF* ([4]).

Parsery - Komponenty zajišťující parsování vstupu a převod na interní reprezentaci. (2.2.1)

Premodifikátory - Komponenty, které upraví strukturu dokumentu před ohodnocováním vět. (2.2.2)

Rankery - Komponenty, které ohodnotí věty v dokumentu, typicky implementace jednotlivých algoritmů. (2.2.3)

Postmodifikátory - Komponenty, které upraví strukturu dokumentu po ohodnocení vět. (2.2.4)

Writery - Komponenty, které budou zodpovědné za prezentaci dokumentů uživateli. (2.2.6)

Evaluatory - Komponenty, které budou sloužit pro automatickou evaluaci extraktů. (2.2.7)

Vzhledem k tomu, že má být možné vytvářet jednotlivé druhy komponent na základě konfigurace sumarizace, je nutné pro každý typ naprogramovat register. Pro tyto registry bude vhodné použít návrhové vzory Singleton, Registry a Abstract Factory. Tyto registry na základě názvu komponenty a její konfigurace vytvoří požadovanou komponentu.

Do těchto bude pro tyto komponenty vhodné použít návrhový vzor Singleton, Abstract Factory a Registry. Pro každý typ komponent bude existovat register, jediný pro celý systém, do kterého se budou registrovat továrny, které budou moci vytvářet jednotlivé typy komponent. Takto získaná továrna bude umět zpracovat konfiguraci komponenty, kterou vytváří a na základě toho ji vytvoří.

Ke každé komponentě bude navíc existovat továrna (Abstract Factory), která ji dokáže na základě konfigurace sestrojit. Ukázka registrace je v příkladu 2.6.1.

```
ParserFactory* pF = new ConcreteParserFactory();
Parser* p = ParserRegister::getInstance()\
    ->register(pF);
```

Příklad 2.6.1: Registrace továrny

Díky tomuto bude následné pohodlné vytvořit instanci konkrétní komponenty. Ukázka je na příkladu 2.6.2.

```
Parser* p = ParserRegister::getInstance()\
    ->create("concreteParser", configuration)
```

Příklad 2.6.2: Vytvoření komponenty

Komponenty samotné budou využívat návrhový vzor Template Method, protože práce komponent stejného typu bude velmi podobná, ale bude se lišit v použitém algoritmu.

Další třídy

Systém samotný bude muset obsahovat komponenty, které se budou muset v celém projektu vyskytovat právě jednou - jedná se o komponenty řídící extrakci, evaluaci, případně komponenty s nastavením - pro tento typ komponent bude použit návrhový vzor Singleton.

Další částí bude systém tříd, které budou reprezentovat strukturu dokumentu, tedy budou reprezentovat jednotlivé části dokumentu jako jsou odstavce, věty, slova, interpunkční znaménka.

```
// ohodnoti vety v dokumentu
void Ranker::rank(Document* d)
{
    doInit(d);
    doRank(d); // virtual, abstract
    doFinish(d);
}
```

Příklad 2.6.3: Základní funkčnost komponenty

Konfigurace

Aby bylo možné se systémem možné pohodlně pracovat, je nutné zvolit vhodný způsob konfigurace. Konfigurace má několik úrovní.

Defaultní hodnota - pokud bude některá komponenta použita, a nebude ji nastaven žádný parametr, tak se použije jeho výchozí hodnota.

Hodnota z konfiguračního souboru - pokud bude hodnota uvedena v konfiguračním souboru, tak se použije.

Hodnota z parametru programu - pokud je hodnota předána jako parametr programu, tak bude mít nejvyšší prioritu.

Konfigurační soubor

Aby byla konfigurace pohodlná, měl by být konfigurační soubor přehledný. Zvažoval jsem vlastní formát, který by byl stručný a výstižný. Nakonec jsem se rozhodl použít pro zápis konfigurace formát XML. Důvody jsem měl následující:

- není nutné psát žádný vlastní parser, stačí použít existující knihovnu
- existující editory umožní snadnou editaci - doplňování párových tagů, zvýraznění syntaxe
- uživatel se nemusí učit novou syntaxi, stačí se naučit pouze názvy

Nevýhodou formátu XML je velikost, která je způsobená velkým množstvím textu, který nenese žádnou užitečnou informaci, a tím pádem může být editovatelnost ztížena. Na druhou stranu se dá předpokládat, že konfigurační soubory se budou generovat.

Testování kvality

Pokud má být možné systém dále používat, tak by měla být otestována jeho funkčnost, a proto budou jednotlivé komponenty a jiné dílčí celky testovány *unit testy*, stejně jako nástroji pro statickou analýzu kódu.

Kapitola 3

Komponenty

V následující kapitole je uveden seznam komponent, které byly implementovány jako nezbytný základ, aby bylo možné začít CSummak používat. Nejdřív budou představeny podporované formáty, následně představím konkrétní implementace jednotlivých typů komponent, jak byly představeny v kapitole 2.6.2. Konkrétně se jedná o komponenty typu Parser (3.3), Premodifikátor (3.4), Ranker (3.5), Postmodifikátor (3.6), Writer (3.7) a Evaluator (3.8).

3.1 Formáty

Formát se považuje za *podporovaný*, pokud existuje Parser, který by ho byl schopen parsovat a Writer, který by ho umožňoval zapsat. V současné době jsou podporovány 2 formáty.

3.1.1 CSTS

CSTS [2] je značkovací jazyk, založený na SGML, který se používá pro značkování v Českého národního korpusu a dalších příbuzných korpusech, jako je Pražský závislostní korpus. Ukázka formátu je ukázána na příkladu 3.1.1.

3.1.2 SentenceId

SentenceId formát obsahuje pouze identifikátory vět z dokumentu. *Identifikátor věty* se skládá z čísla odstavce (odstavce jsou číslovány od 0), oddělující tečky a čísla věty v rámci odstavce (věty jsou číslovány od 0). Takže 2.3 znamená 3. odstavec a v něm 4. věta. Tento formát je výhodný pro ukládání výsledků sumarizace, pokud mají být následně vyhodnoceny. Ukázka formátu je ukázána na příkladu 3.1.2.

```
<csts lang=cs>
<doc>
<a>
</a>
<c>
<p n=1>
<s>
<f cap>Auto<MDl src="a">auto<MDt src="a">NNNS1-----A----
<f>jede<MDl src="a">jet-1<MDt src="a">VB-S---3P-AA---
<f>z<MDl src="a">z-1<MDt src="a">RR--2----------
<f>kopce<MDl src="a">kopec<MDt src="a">NNIS2-----A----
<D>
<d>.<MDl src="a">.<MDt src="a">Z:-------------
</s>
</p>
</c>
</doc>
</csts>
```

Příklad 3.1.1: CSTS formát

```
1.1
1.2
2.0
4.1
```

Příklad 3.1.2: SentenceId formát

3.2 Společné vlastnosti

Pro všechny komponenty platí několik společných pravidel. Tato pravidla by měli dodržovat i budoucí vývojáři dalších komponent.

- Název komponenty je jednoznačný v rámci jednoho typu komponent.
- Každý parametr má výchozí hodnotu, která se použije, pokud ji uživatel nespecifikuje.
- Neznámé parametry jsou ignorovány.
- Pokud je použitá hodnota mimo očekávaný rozsah hodnot, použije se nějaká náhradní.
- Pokud se parametry jmenují `min` a `max`, tak `max` se v případě nekorektní hodnoty nastaví na hodnotu `min`.

3.3 Parsery

Parsery rozparsují vstupní dokument a převedou ho do interní reprezentace, kterou používá celý zbytek systému. V současné době jsou v systému implementovány 2 parsery.

3.3.1 CSTSParser

CSTSParser parsuje soubory ve formátu CSTS (3.1.1). Vstupní soubor musí být navíc *vertikální*, to znamená, že na každém řádku vstupního souboru je informace pouze o 1 elementu. Zpracuje se pouze 1. dokument z tohoto souboru. Neznámé značky jsou ignorovány.

3.3.2 SentenceIdParser

SentenceIdParser parsuje soubory ve formátu SentenceId (3.1.2). Vstupní soubor musí být navíc vertikální.

3.4 Premodifikátory

Premodifikátory mění strukturu dokumentu ještě před ohodnocením jednotlivých vět. V současné době jsou v systému implementovány tyto 3 premodifikátory.

3.4.1 ShortSentenceRemover

ShortSentenceRemover odstraní před ohodnocováním z dokumenty věty, které mají méně než `minLength` slov. Pokud není tato hodnota specifikována, použije se hodnota *3*.

Parametry

minLength minimální počet slov ve větě, aby nebyla odstraněna

3.4.2 WordClassRemover

WordClassRemover odstraní před ohodnocováním z dokumentu slovní druhy, které obsahuje parametr `wordClasses`. Jednotlivé slovní druhy jsou od sebe odděleny čárkou. Pokud není nic uvedeno, použije se hodnota *pronoun, preposition, conjuction, particle, interjection, punctuation.*

Možné hodnoty jsou:

unknown neznámý slovní druh

noun podstatné jméno

adjective přídavné jméno

pronoun zájmeno

numeral číslovka

verb sloveso

adverb příslovce

preposition předložka

conjuction spojka

participle částice

interjection citoslovce

punctuation interpunkční znaménko

Parametry

wordClasses	seznam slovních druhů, které budou z dokumentu odstraněny

3.4.3 SubjectObjectFixer

SubjectObjectFixer přidá na konec každé věty, která nemá vyjádřený podmět všechna slova, o kterých se domnívá, že by mohly být podnětem. Doplní všechna podstatná jména z posledního `maxCount` vět, které mají shodný rod a číslo jako sloveso ve větě s nevyjádřeným podmětem. Pokud není tento parametr uveden, použije se hodnota *2*. Pokud se podstatné jméno použije ve větě, není tato věta započítávána do vzdálenosti.

Pokud je tedy použita defaultní hodnota, a vstupní dokument obsahuje věty: „Veverka si koupila auto. Bylo modré. Mělo kola. Potom si koupila motorku. Upadlo mu kolo.“, tak bude vypadat „Veverka si koupila auto. Bylo modré auto. Mělo kola auto. Potom si koupila motorku. Upadlo mu kolo auto.“

Parametry

maxCount	maximální počet vět, po které se bude podstatné jméno pamatovat

3.5 Rankery

Rankery hodnotí jednotlivé věty v dokumentu podle jejich významnosti. V současné době je implementováno 11 rankerů, z toho 8 skutečně ohodnocuje věty a zbylé mění přidělené skóre.

3.5.1 CentroidRanker

CentroidRanker hodnotí věty na základě jejich podobnosti k těžišti dokumentu. Tento ranker je naprogramován podle práce Erkana a Radeva [3]. Pro výpočet těžiště jsou použity pouze ty termy, jejichž tf-idf váha je vyšší než `threshold`. Pokud není tento parametr nastaven, použije se hodnota *0,01*.

Parametry

threshold	minimální tf-idf váha termu, aby byl použit pro výpočet těžiště

3.5.2 FirstSentenceOverlapRanker

FirstSentenceOverlapRanker přiřazuje skóre každé věty na základě jejího skalárního součinu s první větou v dokumentu. Tento ranker je implementován podle článku [17]. Tento ranker nevyžaduje žádné další parametry.

3.5.3 FirstParagraphRanker

FirstParagraphRanker přiřazuje skóre `value` každé věty v prvním odstavci. Pokud není tento parametr nastaven, použije se hodnota *1*.

Parametry

value	skóre, které se přiřadí vybraným větám

3.5.4 LinearReranker

LinearReranker upravuje skóre vět, které jim přiřadil jiný ranker a to tak, že projde skóre jednotlivá vět a upraví je do rozsahu `min` a `max`. Pokud není parametr `min` nastaven, použije se hodnota *0*. Pokud není parametr `max` nastaven, použije se hodnota *1*.

Pokud tedy vstupní dokument obsahuje věty se skóre 3, 5, 8, 9 a parametry `min` a `max` mají hodnoty *0* a *1*, potom nové skóre vět bude 0, 0,4, 0,8 a 1.

Parametry

min	minimální skóre, které bude větě přiřazeno
max	maximální skóre, které bude větě přiřazeno

3.5.5 LeadRanker

LeadRanker nastaví prvním několika větám z dokumentu skóre na `value` a zbylým větám nastaví 0. Význam a defaultní hodnoty parametrů `ratio`, `ratioType` a `tokenType` je stejný, jako u stejnojmenných parametrů v konfiguraci sumarizace (4.5.2).

Parametry

ratio	poměr/počet vybraných vět
ratioType	co hodnota ratio skutečně znamená
tokenType	typ tokenů, které se budou vybírat
value	skóre, které se přiřadí vybraným větám

3.5.6 ParagraphLeadRanker

ParagraphLeadRanker nastaví prvním několika větám z každého odstavce skóre na `value` a zbylým větám nastaví 0. Význam a defaultní hodnoty parametrů `ratio`, `ratioType` a `tokenType` je stejný, jako u stejnojmenných parametrů v konfiguraci sumarizace (4.5.2).

Pokud například jsou nastaveny parametry: `ratio`: *1*, `ratioType`: *absolute* a `tokenType`: *sentence*, potom bude hodnota `value` přiřazena pouze první větě každého odstavce.

Parametry

ratio	poměr/počet vybraných vět
ratioType	co hodnota ratio skutečně znamená
tokenType	typ tokenů, které se budou vybírat
value	skóre, které se přiřadí vybraným větám

3.5.7 PositionRanker

PositionRanker přiřazuje lineárně skóre mezi `max` (1. větě v dokumentu) a `min` (poslední větě v dokumentu). Pokud není parametr `min` nastaven, použije se hodnota *0*. Pokud není parametr `max` nastaven, použije se hodnota *1*.

Parametry

min	minimální skóre, které bude větě přiřazeno
max	maximální skóre, které bude větě přiřazeno

3.5.8 RandomRanker

RandomRanker přiřazuje každé větě náhodně skóre mezi `min` a `max`. Tento ranker se dá použít na vytváření srovnávacích extraktů, aby bylo možné říci, o kolik je daná metoda kvalitnější než náhodný výběr. Pokud není parametr `min` nastaven, použije se hodnota *0*. Pokud není parametr `max` nastaven, použije se hodnota *1*.

Parametry

min	minimální skóre, které bude větě přiřazeno
max	maximální skóre, které bude větě přiřazeno

3.5.9 RelevanceMeasureRanker

RelevanceMeasureRanker je implementován podle článku Gonga a Liua [5]. Tento ranker přiřazuje lineárně skóre mezi `max` (nejvyšší skóre podle této metody) a `min` (nejnižší skóre podle metody). Pokud není parametr `min` nastaven, použije se hodnota *0*. Pokud není parametr `max` nastaven, použije se hodnota *1*.

Parametry

min	minimální skóre, které bude větě přiřazeno
max	maximální skóre, které bude větě přiřazeno

3.5.10 WeightedRanker

WeightedRanker upravuje skóre, které větě přiřadil jiný ranker a to tak, že ji vynásobí parametrem `weight`. Pokud není parametr `min` nastaven, použije se hodnota *0*. Pokud není parametr `weight` nastaven, použije se hodnota *1*.

Parametry

weight	relativní váha použitého rankeru
ranker	Ranker, který ohodnotí věty

3.5.11 WordUtilPowerRanker

WordUtilPowerRanker upravuje skóre, které větě přiřadil jiný ranker a to tak, že její skóre vydělí hodnotou $(pocet\ slov\ ve\ vete)^{\texttt{utilPower}}$. Pokud není parametr `utilPower` nastaven, použije se hodnota *0*.

Hodnota parametry `utilPower` a její vliv na skóre vět.

0	skóre není nijak modifikováno
0.5	delší věty jsou preferovány
1	skóre se dělí přímo počtem slov ve větě
1.5	kratší věty jsou preferovány

Parametry

wordUtil	hodnota modifikátoru
ranker	Ranker, který ohodnotí věty

3.6 Postmodifikátory

Postmodifikátory modifikují strukturu dokumentu po ohodnocení jednotlivých vět. V současné době jsou implementovány 2 postmodifikátory.

3.6.1 ShortSentenceRemover

PostShortSentenceRemover odstraní po ohodnocení vět z dokumenty ty věty, které mají méně než `minLength` slov. Pokud není tato hodnota specifikována, použije se hodnota *3*.

Parametry

minLength minimální počet slov ve větě, aby nebyla odstraněna

3.6.2 RedundantSentenceRemover

RedundantSentenceRemover odstraní z dokumenty věty, které jsou příliš podobné těm větám, které mají vyšší celkové skóre. Jako míra podobnosti se používá cosinova míra. Pokud není `maxSimilarity` specifikováno, použije se hodnota *0,8*.

Parametry

maxSimilarity maximální podobnost s již existujícími větami, aby zůstala v dokumentu

3.7 Writery

Writery umožňují vypsat zpracovaný dokument. V současné době je implementováno 7 writerů.

3.7.1 DetailWriter

DetailWriter vypíše dokument stejně jako SimpleWriter (3.7.5), ale na konec dokumentu přidá tabulku s přehledem použitých rankerů a dílčím skóre pro vybrané větě. Ukázka výstupu 3.7.1.

```
[    0.0]: Auto jede z kopce.
[    1.0]: Kopec je vysoký a rostou na něm stromy.
[    1.2]: Podél cesty rostou stromy.

SID  Total   CentroidRanker PositionRanker LeadRanker ...
0.0  63.72   50             3              5
1.0  115.58  110            1              3.5714
1.2  62.297  60             0              2.1429
```

Příklad 3.7.1: DetailWriter - výstup

3.7.2 DetailedSentenceIdWriter

DetailedSentenceIdWriter vypíše dokument ve formátu SentenceId (3.1.2). Za identifikátor věty je vypsáno nejdříve celkové skóre a následně skóre, které získala od jednotlivých rankerů v pořadí, v jakém jsou uvedeny v konfiguračním souboru. Výstup toho writeru ukazuje příklad 3.7.2.

```
0.0   13.5   8.3   5.2
1.0   12.1   5.8   6.3
1.2   11.8   9.2   2.6
```

Příklad 3.7.2: DetailedSentenceIdWriter - výstup

3.7.3 NormalWriter

NormalWriter vypíše výsledek jako běžný text. Jednotlivé odstavce jsou od sebe odděleny prázdným řádkem. Prázdný odstavec (odstavec bez vět) není vypsán. Ukázka je v příkladu 3.7.3.

```
Auto jede z kopce.

Kopec je vysoký a rostou na něm stromy. Podél cesty rostou stromy.
```

Příklad 3.7.3: NormalWriter - výstup

3.7.4 SentenceIdWriter

SentenceIdWriter vypíše dokument ve formátu SentenceId (3.1.2). Ignoruje požadavek na vypsání skóre. Ukázka 3.7.4.

```
0.0
1.0
1.2
```

Příklad 3.7.4: SentenceIdWriter - výstup

3.7.5 SimpleWriter

SimpleWriter vypíše věty v tom pořadí, v jakém jsou uvedeny v dokumentu. Každá věta je na samostatném řádku. Ukázka výpisu je v příkladu 3.7.5.

```
[    0.0]: Auto jede z kopce.
[    1.0]: Kopec je vysoký a rostou na něm stromy.
[    1.2]: Podél cesty rostou stromy.
```

Příklad 3.7.5: SimpleWriter - výstup

3.7.6 SortedWriter

SortedWriter vypíše věty v pořadí podle dosaženého skóre v sestupném pořadí (věta s nejvyšším skóre je první). Každá věta je na samostatném řádku. Ukázka výstupu je v příkladu 3.7.5.

```
[    1.0]: Kopec je vysoký a rostou na nìm stromy.
[    0.0]: Auto jede z kopce.
[    1.2]: Podél cesty rostou stromy.
```

Příklad 3.7.6: SortedWriter - výstup

3.8 Evaluatory

Evaluátory slouží k automatické evaluaci podobnosti 2 dokumentů. V současné době je implementováno 5 evaluátorů.

3.8.1 CosineMeasureEvaluator

CosineMeasureEvaluator spočítá podobnost 2 dokumentů na základě cosinové míry.

3.8.2 FMeasureEvaluator

FMeasureEvaluátor spočítá f-measure 2 dokumentů (2.3.1).

3.8.3 PrecisionEvaluator

PrecisionEvaluator spočítá precision 2 dokumentů (2.3.1).

3.8.4 RecallEvaluator

RecallEvaluator spočítá recall 2 dokumentů (2.3.1).

3.8.5 SentenceIdCosineMeasureEvaluator

CosineMeasureEvaluator spočítá podobnost 2 dokumentů na základě cosinové míry, ale pouze na základě vybraných vět - jejich identifikátorů.

Kapitola 4

Uživatelská část

V následující kapitole popíši v jaké formě se CsummaK distribuuje, jak ho zkompilovat, a co je k jeho kompilaci potřeba. Dále popíši parametry, které používá společně s podrobným popisem konfiguračního souboru.

Program je určen pro operační systém GNU/Linux. Jedná se o konzolový program, který se ovládá pomocí parametrů z příkazové řádky. Dílčí parametry sumarizátoru je možné nastavit pomocí přepínačů, tak i pomocí konfiguračního souboru. Hodnoty získané z přepínačů mají vyšší prioritu než hodnoty získané z konfiguračního souboru.

4.1 Distribuce

Program je možné získat ze stránek jako zkomprimovaný archív zdrojových kódů. Tento archív se generuje minimálně jedenkrát denně. Na CD, které je součástí této práce nachází poslední stabilní verze programu. Ukázka toho, jak získat zdrojové kódy je v příkladu 4.1.1.

```
wget 'http://csummak.m-core.net/csummak.tar.gz'
```

Příklad 4.1.1: Získání systému

4.2 Kompilace

Pro kompilaci ze zdrojových kódů je nutné mít kompilátor gcc ($\geq$ 4.2.3) a následující knihovny v posledních stabilních verzích.

- boost::regex
- boost::program_options
- libxml2

Samotná kompilace se provede standardním způsobem, pomocí standardních příkazů. Například tak, jak je ukázáno v 4.2.1.

```
tar -zvxf csummak.tar.gz
cd csummak-<verze>
./configure
make
```

Příklad 4.2.1: Kompilace

4.3 Spuštění

Hodnoty parametrů (4.4) předané programu prostřednictvím příkazové řádky mají vyšší prioritu než ty, které jsou specifikovány v konfiguračním souboru. Tento způsob umožňuje nastavit hlavní parametry sumarizace v konfiguračním souboru a dílčí ladění provádět prostřednictvím přepínačů, což je pro uživatele pohodlnější než přepisování souboru. Všechny cesty se berou od aktuálního adresáře.

Hlavním programem pro sumarizaci je soubor `summarizer`, který se nachází v adresáři `bin`. Ukázka, jak spustit sumarizátor na testovacích datech je v příkladu 4.3.1.

```
cd csummak-<verze>/bin
./summarizer -c ../test/data/summarizer.conf
```

Příklad 4.3.1: Spuštění testovací konfigurace

4.4 Parametry

Činnost sumarizátoru se ovlivňuje prostřednictvím parametrů z příkazové řádky.

`-v, --version` - Zobrazí aktuální verzi programu. Součástí je také informace o tom, ze které revize program pochází. Tato informace by měla být součástí každého hlášení o chybě.

`-h, --help` - Zobrazí stručnou nápovědu s přehledem parametrů.

`-c, --config` - Cesta ke konfiguračnímu souboru. Relativně zadaná cesta se počítá výchozím způsobem (od aktuálního adresáře). `./summarizer -c ../test/data/summarizer.conf -i ../test/data/01.csts`

`-f, --outputFormat` - Formát výstupního souboru. Možné hodnoty jsou stejné u stejnojmenného elementu v konfiguračním souboru (4.5.2).

`-r, --ratio` - Jak velký má extrakt být. Možné hodnoty jsou stejné u stejnojmenného elementu v konfiguračním souboru (4.5.2).

`-R, --ratioType` - Co znamená parametr ratio. Možné hodnoty jsou stejné u stejnojmenného elementu v konfiguračním souboru (4.5.2).

`-t, --tokenType` - jaké elementy se budou počítat. Možné hodnoty jsou stejné u stejnojmenného elementu v konfiguračním souboru (4.5.2).

`-F, --idfFile` - Cesta k souboru s vypočteným idf pro jednotlivá slova.

`-D, --idfDefault` - Výchozí hodnota idf pro slova, která nebyla nalezena ve slovníku.

Parametry `outputFormat`, `ratio`, `ratioType` a `tokenType` mají stejný význam jako parametry v konfiguračním souboru 4.5.2. Pokud je libovolný z těchto parametrů uveden, tak jeho hodnota se použije místo hodnoty z konfiguračního souboru.

Pokud je uveden název vstupního souboru, tak se neprovede obsah sekcí `texts` (4.5.9) a `evaluation` (4.5.13).

4.5 Konfigurační soubor

4.5.1 Struktura

Konfigurace se provádí pomocí konfiguračního XML souboru. Ukázkový konfigurační soubor je `test/data/summarizer.conf`. Konfigurační soubor můžeme rozdělit do několika sekcí, které popisují dílčí aspekty sumarizace. V konfiguračním souboru je možné specifikovat parametry sumarizace (4.5.2), způsoby modifikace dokumentu před (4.5.3) a po (4.5.4), které rankery mají být použity pro ohodnocení jednotlivých (4.5.6), stejně jako seznam dokumentů (4.5.9), které mají být zesumarizovány a požadavky na jejich následnou evaluaci (4.5.13).

Základní struktura konfiguračního souboru je znázorněna na 4.5.1.

```
<csummak>
  <config> ... </config>
  <preModifier> ... </preModifier>
  <postModifier> ... </postModifier>
  <rankers> ... </rankers>
  <texts> ... </texts>
  <evaluation> ... </evaluation>
</csummak>
```

Příklad 4.5.1: Základní struktura konfiguračního souboru

4.5.2 Element config

Element `config` slouží ke specifikaci parametrů sumarizace. Je zde možné nastavit formát vstupních a výstupních souborů, míru komprese atd. Možná konfigurace je v příkladu 4.5.2.

Jednotlivé elementy mají tento význam:

outputFormat - Definuje způsob výpisů výsledku. Znak @ určuje, kde se bude vypisovat celkové skóre věty. Pokud není tento element uveden, použije se *DetailWriter*. Hodnota musí být nějaký známý writer. Seznam předpřipravených writerů je v kapitole 3.7.

parser - Definuje defaultní parser pro parsování vstupu. Pokud není tento element uveden, použije se *CSTSParser*. Hodnota musí být nějaký známý parser. Seznam předpřipravených parserů je v kapitole 3.3.

evaluationParser - Definuje defaultní parser pro parsování dokumentů určených k evaluaci. Pokud není žádný uveden, použije se CSTSParser. Hodnota musí být nějaký známý parser. Seznam předpřipravených parserů - 3.3.

ratio - Definuje, jak velký má být výsledný extrakt. Význam této hodnoty je závislý na hodnotách elementů `ratioType` a `tokenType`. Pokud není tento element použit a `ratioType` má hodnotu *percentage*, použije se hodnota *0,2*. Pokud není tento element použit a `ratioType` má hodnotu *absolute*, použije se hodnota *5*.

ratioType - Definuje, co znamená hodnota uvedená v sekci `ratio`. Hodnota se počítá z prvků specifikovaných pomocí `tokenType`. Pokud není tento element uveden, předpokládá se, že měl hodnoty *percentage*. Možné hodnoty tohoto parametru jsou:

- **percentage** - Hodnota znamená poměrnou část z celkového počtu prvků daného typu. Takže hodnota 0,25 znamená alespoň 25% všech vět/znaků.
- **absolute** - Absolutní počet prvků daného typu. Takže hodnota 100 udává alespoň 100 znaků.

tokenType - Definuje, které typy prvky se budou počítat. Pokud není tento element použit a `ratioType` má hodnotu *percentage*, použije se hodnota *character*. Pokud není tento element použit a `ratioType` má hodnotu *absolute*, použije se hodnota *sentence*. Možné hodnoty tohoto parametru jsou:

- **character** - Základní jednotka jsou znaky.
- **sentence** - Základní jednotka jsou věty.

idfFile - Definuje cestu k souboru s vypočteným idf pro jednotlivá slova. Soubor musí být ve formátu z kapitoly 5.2. Pokud není tento parametr uveden, použije se `../data/idfScores.data`.

idfDefault - Definuje defaultní hodnotu pro slova, která nebyla nalezena v souboru s vypočtenými idf. Pokud není tento element použit, použije se hodnota *10*.

```
<csummak>
  ...
  <config>
    <outputFormat>@DetailWriter</outputFormat>
    <parser>CSTSParser</parser>
    <evaluationParser>SentenceIdParser</evaluationParser>
    <ratio>0.25</ratio>
    <ratioType>percentage</ratioType>
    <tokenType>char</tokenType>
    <idfFile>../data/idfScores.data</idfFile>
    <idfDefault>5</idfDefault>
  </config>
  ...
</csummak>
```

Příklad 4.5.2: Konfigurační soubor - konfigurace sumarizace

Takže příklad 4.5.2 říká, že pro parsování vstupních souborů má být použit CSTSParser, že se mají vybírat nejrelevantnější věty tak dlouho, dokud jejich délka nebude alespoň 25% znaků z celé délky dokumentu. Jako soubor s idf se má použít soubor ../data/idfScores.data, pro slova, která v tomto slovníku nejsou se má použít idf rovno 5. Extrakt dokumentu bude zapsán pomocí DetailWriteru. Pro parsování souborů pro evaluaci má být použit SentenceIdParser.

4.5.3 Element preModifier

V elementu `preModifier` se deklarují modifikátory, které mají být použity před ohodnocením vět. Tento element může obsahovat libovolný počet elementů `modifier`. Modifikátory jsou spouštěny v tom pořadí, v jakém jsou uvedeny. Syntaxe modifikátorů je popsána v kapitole 4.5.5.

Začlenění do konfiguračního souboru je znázorněno na příkladu 4.5.3.

4.5.4 Element postModifier

Element `postModifier` obsahuje deklarace modifikátorů, které mají být použity po ohodnocení vět. Tento element může obsahovat libovolný počet elementů `modifier`. Modifikátory jsou spouštěny v tom pořadí, v jakém jsou uvedeny. Syntaxe modifikátorů je popsána v kapitole 4.5.5.

Začlenění do konfiguračního souboru je znázorněno na příkladu 4.5.4.

```
<csummak>
  ...
  <preModifier>
    <modifier>
      ...
    </modifier>
  </preModifier>
  ...
</csummak>
```

Příklad 4.5.3: Element preModifier

```
<csummak>
  ...
  <postModifier>
    <modifier>
      ...
    </modifier>
  </postModifier>
  ...
</csummak>
```

Příklad 4.5.4: Element postModifier

4.5.5 Element modifier

V elementu `modifier` se konfigurují parametry konkretního modifikátoru. Každý modifikátor musí mít sekci `name` se svým jménem a `params` se seznam parametrů. Jméno a parametry musí být součástí dokumentace modifikátoru. Parametry, které modifikátor nevyžaduje jsou ignorovány. CSummaK samotný obsahuje několik předpřipravených modifikátorů, které jsou uvedeny v kapitolách 3.4 a 3.6. Popis zápisu parametrů je shodný se zápisem parametrů u rankerů a je uveden v kapitole 4.5.8.

Začlenění do konfiguračního souboru je znázorněno na příkladu 4.5.5.

4.5.6 Element rankers

V elementu `rankers` se deklarují rankery, které mají být použity pro ohodnocení vět. Tento element může obsahovat libovolný počet elementů `ranker`. Jednotlivé rankery jsou spouštěny v tom pořadí, v jakém jsou uvedeny. Syntaxe rankerů je popsána v kapitole 4.5.7.

Začlenění do konfiguračního souboru je znázorněno na příkladu 4.5.7.

```
<csummak>
  ...
    <modifier>
      <name>ShortSentenceRemover</name>
      <params>
        <param name="minLength">2</param>
      </params>
    </modifier>
  ...
</csummak>
```

Příklad 4.5.5: Element modifier

```
<csummak>
  ...
  <rankers>
    <ranker>
      ...
    </ranker>
  </rankers>
  ...
</csummak>
```

Příklad 4.5.6: Element rankers

4.5.7 Element ranker

V elementu `ranker` se konfigurují parametry konkrétního rankeru. Každý ranker musí mít sekci `name` se svým jménem a `params` se seznam parametrů. Jméno a parametry musí být součástí dokumentace rankeru. Parametry, které ranker nevyžaduje jsou ignorovány. CSummaK samotný obsahuje několik předpřipravených rankerů, které jsou uvedeny v kapitole 3.5.

Začlenění do konfiguračního souboru je znázorněno na příkladu 4.5.7.

4.5.8 Element params

Element `params` obsahuje elementy potřebné pro dosažení potřebné funkcionality konfigurované komponenty. Tento element může obsahovat libovolný počet elementů `param` s povinným atributem `name` pro zadání jména parametru. Obsahem značky `param` může být libovolný. Každá komponenta je zodpovědná za zpracování svých parametrů.

Začlenění do konfiguračního souboru je znázorněno na příkladu 4.5.5 případně 4.5.7.

```
<csummak>
  ...
  <ranker>
    <name>LeadRanker</name>
    <params>
      <param name="ratio">20</param>
      <param name="ratioType">absolute</param>
      <param name="tokenType">char</param>
      <param name="value">5</param>
    </params>
  </ranker>
  ...
</csummak>
```

Příklad 4.5.7: Element ranker

4.5.9 Element texts

V elementu `texts` jsou specifikovány úkoly pro sumarizaci, které se mají zesumarizovat. Tento element může obsahovat libovolný počet elementů `text`, které jsou zpracovávány v tom pořadí, ve kterém jsou uvedeny.

Začlenění do konfiguračního souboru je znázorněno na příkladu 4.5.8.

```
<csummak>
  ...
  <texts>
    <text>...</text>
    <text>...</text>
  </texts>
  ...
</csummak>
```

Příklad 4.5.8: Element texts

4.5.10 Element text

V elementu `text` se specifikuje soubor, který má být zesumarizován. Musí obsahovat právě 1 vnořený element `in`, která specifikuje cestu ke vstupnímu souboru a libovolný počet elementů `out` pro určení cesty k souboru s výsledkem.

Začlenění do konfiguračního souboru je znázorněno na příkladu 4.5.9.

Podrobný popis elementů `in` a `out` je uveden dále v textu.

Příklad 4.5.9 - vstupní soubor je *../test/data/06.csts* a bude parsován defaultním parserem (uvedeným v sekci `config`). Po dokončení sumarizace bude výsledek uložen třemi různými způsoby. Poprvé bude uložen do souboru */tmp/csummak-sum-6-default.out* způsobem a mírou komprese specifikovanou v sekci `config`.

```
<csummak>
  ...
  <texts>
    ...
    <text>
      <in>../test/data/06.csts</in>
      <out>/tmp/csummak-sum-6-default.out</out>
      <out outputFormat="DetailWriter"
        ratio="0.2"
        ratioType="percentage"
        tokenType="char">
        /tmp/csummak-sum-6-DetailWriter.out
      </out>
      <out outputFormat="SentenceIdWriter"
        ratio="4"
        ratioType="absolute"
        tokenType="sentence">
        /tmp/csummak-sum-6-SentenceIdWriter.out
      </out>
    </text>
    ...
  </texts>
  ...
</csummak>
```

Příklad 4.5.9: Element text

Podruhé bude uložen do souboru */tmp/csummak-sum-6-DetailWriter.out*, ale pro výpis bude použit `DetailWriter`. Bude uloženo tolik vět, aby extrakt měl délku 20% vstupního dokumentu. A potřetí budou zapsány 4 nejdůležitější věty do souboru */tmp/csummak-sum-6-SentenceIdWriter.out* pomocí *SentenceIdWriter*.

4.5.11 Element in

V elementu `in` se uvádí vstupní soubor pro sumarizaci. Tento tag má jeden volitelný atribut `parser`, kterým je možné specifikovat jiný než výchozí parser vstupního souboru. Jedná se obdobu parametru `parser` z elementu `config` (4.5.2).

Začlenění do konfiguračního souboru je znázorněno na příkladu 4.5.10.

Vstupní soubor *../test/data/06.csts* bude parsován výchozím parserem, ale vstupní soubor *../test/data/f-file.sId* bude parsován pomocí parseru *SentenceIdParser*.

```
<csummak>
  ...
  <texts>
    ...
    <text>
      <in>../test/data/06.csts</in>
      <out>/tmp/csummak-sum-6-default.out</out>
    </text>
    <text>
      <in parser="SentenceIdParser">../test/data/f-file.sId/in>
      <out>/tmp/csummak-sum-x-file.out</out>
    </text>
    ...
  </texts>
  ...
</csummak>
```

Příklad 4.5.10: Element in

4.5.12 Element out

V elementu `out` se specifikuje název souboru, do kterého bude uložen výstup. Také je možné pomocí atributu `outputFormat` změnit výstupní formát. Atributy `ratio`, `ratioType` a `tokenType` umožňují změnit parametry sumarizace. Všechny atributy mají shodný význam jako stejnojmenné parametry v sekci `config` (4.5.2).

Začlenění do konfiguračního souboru je znázorněno na příkladu 4.5.9 s vysvětlením v kapitole 4.5.10.

4.5.13 Element evaluation

V elementu `evaluation` se specifikuje průběh evaluace zesumarizovaných dokumentů. Tento element obsahuje 2 podelementy - `methods` pro specifikaci evaluačních metod a `sets` pro specifikaci skupin dokumentů, které mají být mezi sebou porovnávány.

Začlenění do konfiguračního souboru je znázorněno na příkladu 4.5.11.

4.5.14 Element methods

V elementu `methods` se specifikují evaluační metody, které se mají použít pro vyhodnocení extraktů. Tento element může obsahovat libovolný počet elementů `method` (4.5.15). Jednotlivé metody jsou spouštěny v tom pořadí, v jakém jsou deklarovány.

Začlenění do konfiguračního souboru je znázorněno na příkladu 4.5.11.

```
<csummak>
  ...
  <evaluation>
    <methods>...</methods>
    <sets>...</sets>
  </evaluation>
  ...
</csummak>
```

Příklad 4.5.11: Element in

4.5.15 Element method

V elementu `method` se specifikuje použitá evaluační metoda. Její název musí být uveden v sekci `name` a musí jít o známou komponentu pro evaluaci. Seznam předpřipravených evaluačních metod je v kapitole 3.8. Dále v tomto elementu musí být element `file` pro určení výstupního souboru pro výsledky evaluace.

Výsledky jsou pro každý `set` na jednom řádku a to tak, že se jednotlivé sloupce odpovídají pozicím v horní trojúhelníkové matici bez diagonály vypsané po řádcích za sebou.

Pokud se mezi sebou porovnávají 3 dokumenty, tak vznikne matice 3×3 prvků.

$$\begin{pmatrix} d_1 \times d1 & d_1 \times d_2 & d_1 \times d_3 \\ d_2 \times d_1 & d_2 \times d_2 & d_2 \times d_3 \\ d_3 \times d_1 & d_3 \times d_2 & d_3 \times d_3 \end{pmatrix}$$

A v souboru s výsledky budou zapsány takto: $d_1 \times d_2 \quad d_1 \times d_3 \quad d_2 \times d_3$.

4.5.16 Element sets

V elementu `sets` se specifikují skupiny souborů, které se mají porovnávat mezi sebou navzájem. Každý skupina je popsána v elementu `set`. První `set` určuje, kolik souborů musí být ve zbylých skupinách. Skupiny s rozdílným počtem souborů jsou ignorovány.

Začlenění do konfiguračního souboru je znázorněno na příkladu 4.5.12.

4.5.17 Element set

V elementu `set` jsou uvedeny soubory, které se mají porovnávat mezi sebou. Tento element může obsahovat libovolný počet elementů `file` s volitelným atributem `parser`, který specifikuje, který parser se má použít pro rozparsování souboru.

Začlenění do konfiguračního souboru je znázorněno na příkladu 4.5.12.

V příkladu 4.5.12 obsahuje 1. `set` tři soubory, proto všechny skupiny musí také obsahovat právě tři soubory. Skupiny s jiným počtem souborů budou ignorovány. První soubor *../test/data/02.x* bude rozparsován parserem specifikovaným

```
<csummak>
  ...
  <evaluation>
    <sets>
      <set>
        <file>../test/data/02.x</file>
        <file parser="SentenceIdParser">../test/data/e2.b</file>
        <file parser="CSTSParser">../test/data/03.csts</file>
      </set>
    </sets>
    ...
  </evaluation>
  ...
</csummak>
```

Příklad 4.5.12: Element set

v sekci `config` (4.5.2), druhý soubor *../test/data/e2.b* bude rozparsován pomocí `SentenceIdParser` a třetí soubor bude rozparsován pomocí *CSTSParser*.

Kapitola 5

Co bylo implementováno

Následující kapitola obsahuje přehled dalších nástrojů, které byly v této práci vytvořeny. Mezi tyto nástroje patří systém pro usnadnění vytváření extraktů lidmi, dále nástroj pro výpočet idf z kolekce dokumentů a další pomocné skripty pro usnadnění práce.

5.1 Systém na získávání extraktů od lidí

Aby bylo možné měřit kvalitu jednotlivých algoritmů na extrakci textů, bylo nutné získat extrakty novinových článků, které by vytvořili lidé. Protože mi nebyly známa žádná dostupná data pro český jazyk, implementoval jsem systém, který by umožňoval snadno tyto extrakty lidem vytvářet.

5.1.1 Požadavky

Jako hlavní vlastnosti na systém pro tvorbu extraktů jsem si stanovil:

- Minimální nároky na technické vybavení uživatele.
- Snadná tvorba extraktů pro uživatele (uživatelská přívětivost).
- Snadné získání dat pro další zpracování.

Proto jsem systém navrhl jako webovou aplikaci, která tyto vlastnosti splňuje. Webová aplikace byla naprogramována v programovacích jazycích PHP a Javascript. Pro prezentaci byl použit značkovací jazyk XHTML a CSS. Pro ukládání dat byla použita MySQL databáze. Pomocné skripty, které zajišťovaly propojení externích nástrojů pro segmentaci a morfologickou analýzu, byly napsány v bashi.

5.1.2 Zdroje dat

Systém přebírá články ze zpravodajských serverů *Aktualne.cz*, *Idnes.cz*, *Ihned.cz* a *Novinky.cz*. Z těchto zpravodajských serverů nebylo přejímáno pouze hlavní

Tabulka 5.1: Počet kanálů z jednotlivých serverů

Server	Počet kanálů
Aktualne.cz	10
Idnes.cz	8
Ihned.cz	8
Novinky.cz	14

zpravodajství, ale všechny RSS kanály, které poskytují. Počet kanálů u jednotlivých serverů je znázorněn v tabulce 5.1.

5.1.3 Princip fungování

Uživatel si mohl ve vlastní integrované RSS čtečce nastavit, které kanály ho zajímají. Články z těchto rubrik mu byly nabízeny k sumarizaci. Uživatel měl také možnost zvolit *Náhodný článek*, což byl článek, který ohodnotil pouze 1 člověk. Na stránce s vybraným článkem mohl uživatel kliknutím na větu článku tuto větu označit, případně odebrat (Ctrl + klik). Uživatel byl navíc v průběhu informován o tom, kolik vět už takto označil. Postup tvorby extraktu z pohledu uživatele vypadal takto:

1. Přihlášení do systému prostřednictvím standardního formuláře.
2. Vybrání článku z preferovaných kanálů, případně zvolení náhodného článku.
3. V průběhu čtení článku označoval věty, které považoval za důležité. (U krátkého článku mohlo nastat, že takto vybral i 40% článku.)
4. Z takto označených vět vybral věty, které budou v extraktu. (Důvod těchto 2 fází byl ten, aby se uživatel nemusel zbytečně posouvat v textu. Pokud by neměl systém tyto dvě úrovně, poměrně často by nastávalo, že uživatel by limit využil na věty ze začátku dokumentu. Pokud by chtěl přidat některou vět z konce, tak by musel skrolovat nahoru, a hledat, kterou větu odebrat. Nebo naopak - uživatel by si šetřil prostor pro důležité věty na konci dokumentu, ale žádná taková by se neobjevila, takže by musel skrolovat nahoru, aby doplnil kvótu.
5. Odeslání vytvořeného extraktu. (Uživateli bylo poděkována za jeho čas, a byly mu nabídnuty možnosti z kroku 2.)

Proces tvorby extraktu vyžadoval asi 1 minutu navíc v porovnání s tím, kdyby byl článek čten bez této tvorby extraktu.

5.1.4 Autorská práva

Jako nejkomplikovanější část celé aplikace se ukázala otázka autorských práv. Po přečtení autorského zákona [20] jsem se domníval, že na základě § 31, odstavce 1, písmeno c (5.1.1) mám právo texty článků používat. A že tento systém může být veřejně přístupný. Požádal jsem své spolužáky, aby se ne tento paragraf podívali, a nezávisle na sobě jsme ho pochopili stejně - že mohu texty článků zobrazovat. Mimo samotného výkladu zákona jsem také vycházeli ze skutečnost, že internetové vyhledávače (google, jyxo) umožňují prohlížet stránky, které mají uložené ve svých databázích (zobrazují je ve svém design), stejně jako anonymizační servery (spysurfing.com, www.ibypass.com) zobrazují požadovanou stránku ve svém designu.

> § 31 Citace
> (1) Do práva autorského nezasahuje ten, kdo
> ...
> c) užije dílo při vyučování pro ilustrační účel nebo při vědeckém výzkumu, jejichž účelem není dosažení přímého nebo nepřímého hospodářského nebo obchodního prospěchu, a nepřesáhne rozsah odpovídající sledovanému účelu;

Příklad 5.1.1: Autorský zákon - § 31

Po předchozích zkušenostech jsem věděl, že můj vykládat zákonů se nemusí shodovat s oficiálním výkladem, proto jsem se rozhodl tuto otázku zkonzultovat s lidmi, kteří studují práva. Tito lidé se nezávisle došli ke stejnému výkladu, který byl rozdílný od mého. Proto jsem otázku autorských práv konzultoval také se svým vedoucím, který mi doporučil stejný postup, který navrhovali oslovení studenti práv. Zpřístupnit tuto aplikaci jen omezenému počtu lidí. Proto do systému přidána registrace a pouze registrovaní uživatelé mají právo vytvářet texty.

S narůstající kvalitou automatických sumarizace vyvstává také otázka autorských práv. Zda-li je, nebo není takto vytvořený dokument také autorským dílem.

5.1.5 Technické pozadí

Proces od požadavku na zobrazení určitého novinového článku až po jeho prezentaci se skládal z několika kroků, od kterých byl uživatel odstíněn. Skládal se z následujících kroků.

1. Byl stažen obsah stránky s požadovaným článkem.
2. Na základě toho, ze kterého serveru byl novinový článek, tak došlo ke specifickému zpracování - převedení na jednotnou znakovou sadu, odstranění menu, reklam apod.

3. Odstranění formátovacích značek.

4. Pomocí tokenizeru a taggeru došlo k převedení prostého textu do CSTS formátu.

5. Originální dokument spolu se svou rozparsovanou podobou byl uložen do databáze s dalšími informacemi, jako je datum vytvoření, název článku, URL článku.

6. CSTS formát byl rozparsován a jednotlivé věty byly obaleny značkami, aby je bylo možné pomocí Javascriptu snadné pracovat.

7. Výsledný text byl uživateli prezentován jako běžný novinový článek.

8. Extrakt vytvořený uživatelem byl uložen po jednotlivých větách do databáze.

5.1.6 Pohled na získaná data

Při psaní nástroje, který ze stažených stránek extrahuje hlavní text bez reklam, odkazů na související články a dalších částí, které nejsou součástí samotného textu mne překvapilo, že různé rubriky stejného serveru používají různé šablony (různě pojmenované bloky s reklamou). Například server aktualne.cz nemá zřejmě žádnou konvenci, jak uvozovat relevantní články a proto se na jeho stránkách objevuje minimálně 5 různých uvození - „Čtěte dále:“, „Dále čtěte:“, „Čtěte také:“, „Čtěte více:“ navíc různě obaly formátovacími značkami. Ze článků ze serveru novinky.cz byl odstraněn první odstavec, protože obsahoval autorem napsaný abstrakt k článku.

5.2 Výpočet idf

Pro výpočet idf (2.3.2) byla použita kolekce 134048 novinových článků,kterou jsem získal od svého vedoucího. Všechna slova z textu byla převedena tak, aby obsahovala pouze malá malá písmena. Idf bylo spočteno pro základní tvary jednotlivých slov. Celkem bylo získáno idf pro přibližně 382 tisíc slov.

Soubor s těmito daty a program, který ze zadané kolekce dokumentů tato idf spočítá je součástí práce. Soubor s daty se jmenuje `data/idfScores.data` a program na výpočet se jmenuje `idf` a nachází se ve složce `bin`.

Formát souboru

Formát souboru je navržen tak, aby byl snadno editovatelný v textovém editoru a aby bylo možné spojit výsledky z více korpusů, pokud je k dispozici pouze tento soubor (pro každý korpus jeden). První řádek obsahuje počet dokumentů, ze kterých byl výsledek spočítán. Každý další řádek obsahuje samostatné slovo

v základním tvaru, idf spočtené pro toto slovo, počet dokumentů, ve kterých se nacházelo a celkový počet výskytů tohoto slova.

```
134048
...
být 0.0174208 131733 2380559
...
lingvistika 9.32105 12 15
```

Příklad 5.2.1: Ukázka ze souboru idfScores.data

Na příkladu 5.2.1 je vidět, že korpus, ze kterého byly hodnoty spočteny obsahoval 134048 dokumentů. Slovo *být* se vyskytovalo v 131733 dokumentech a proto je jeho idf rovno *0,0174208*.

5.3 Pomocné skripty

Pomocné skripty slouží k usnadnění některých opakovaných činností.

5.3.1 Preprocessing

eval.sh

Tento skript slouží k převodu obyčejného textu do formátu CSTS. Jednotlivé externí nástroje jsou ve složce `preprocessing`.

Cesta:	`preprocess.sh`
Požadavky:	spustitelný textseg - 2.2.1 (Perl, ...) spustitelný tagger - 2.2.1 (tcsh, ...)
Spuštění:	`./preprocess.sh plain.txt res.csts`
Výsledek:	vstupní dokument musí být v UTF-8 výsledný soubor je v ISO-8859.2 `res.csts` je otagovaný vstupní soubor `plain.txt` do složky `/tmp` ukládá pomocná data

5.3.2 Evaluace

Pro usnadnění evaluace existují tyto skripty.

eval.sh

Cesta:	`evaluation/eval.sh`
Požadavky:	zkompilovaný CSummaK statistická nástroj R jako kostru konfiguračního souboru použít soubor `eval.conf`.
Spuštění:	`./eval.sh kiss`
Výsledek:	zkontroluje, zda-li se ve složce evaluation nachází soubor `eval-kiss.conf` vykoná sumarizaci podle tohoto skriptu z výsledků spočte precision, recall, f-measure výsledek vypíše na obrazovku do souboru `_result` uloží statistické údaje ze získaných dat do souboru `_res_kiss_[f-measure-score]-[aktualni-cas]` zkopíruje soubor `_result` do souboru `_res_kiss_[f-measure-score]-[aktualni-cas].bcf` zkopíruje soubor `eval-kiss.conf`

evalAll.sh

Cesta:	`evaluation/evalAll.sh`
Požadavky:	funkční skript `eval.sh`
Spuštění:	`./evalAll.sh`
Výsledek:	spustí `eval.sh` na všechny konfigurace ve složce

Kapitola 6

Programátorská dokumentace

V této kapitole je představena architektura a důležité komponenty, které systém obsahuje.

6.1 Architektura

Návrh architektury CsummaKu vychází z procesu sumarizace (2.1). Výsledná podoba systému vychází z architektury, kterou jsem, kterou jsem si naplánoval v průběhu analýzy (2.6.2).

Přehled hlavních komponent je na obrázku 6.1.

6.2 Komponenty

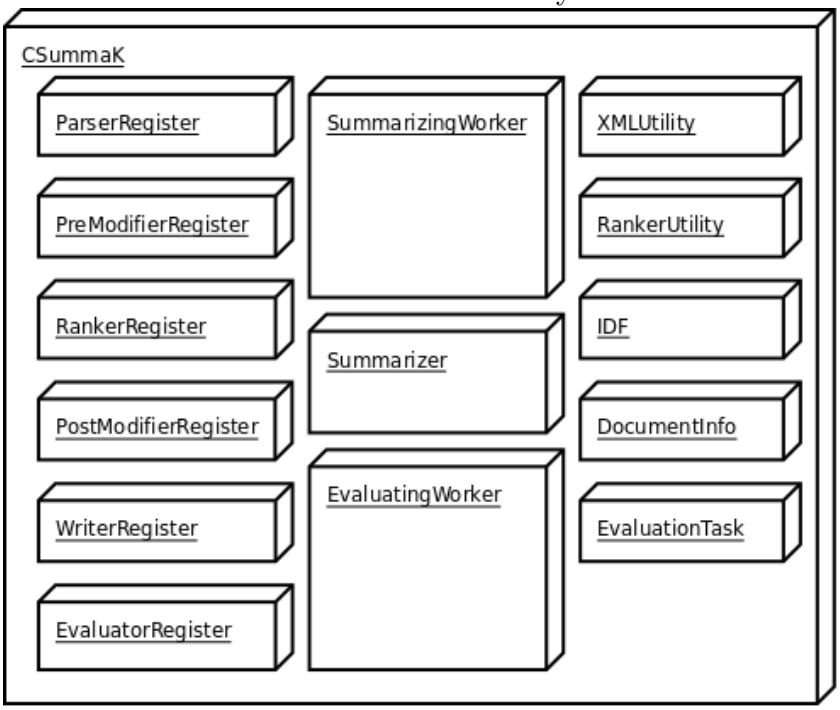
V následující kapitole popíši nejdůležitější komponenty, které se v systému nachází. Komponenta je považována za důležitou, pokud se jedná o abstraktní třídu, od které je možné dědit, nebo vykonává nějakou netriviální činnost. Celý program obsahuje přes 270 tříd, proto nemá ani smysl snažit se je všechny zmínit v této práci. Pro úplný přehled členských metod a proměnných doporučuji nahlédnout do automaticky vygenerované dokumentace, která je na CD.

Systém obsahuje 6 různých typů komponent - parsery, premodifikátory, rankery, postmodifikátory, writery a evaluatory. Základní práce se všemi typy je v podstatě stejná. Z tohoto důvodu, pokud některá popisovaná funkcionalita bude pro všechny typy shodná, budu místo konkrétního typu komponenty psát *X*. Takže *X*Register znamená ParserRegister, PreModifierRegister, Ranker, PostModifierRegister, WriterRegister a EvaluatorRegister.

6.2.1 *X*Register

Třídy typu *X*Register slouží k vytváření komponent typu *X*. Aby bylo možné tento typ vytvářet, je nutné do *X*Registeru nejdřív zaregistrovat továrnu, která dokáže objekt typu *X* vytvořit.

Obrázek 6.1: Hlavní části systému



Tento typ tříd používá několik návrhových vzorů - Singleton, takže v rámci celého systému existuje právě 1 instance tohoto objektu. Dále používá kombinaci návrhové vzoru Registry a Abstract Factory. Po zaregistrování určité konkrétní továrny je možné rovnou prostřednictvím funkce `create` vytvořit příslušný objekt.

Zaregistrování továrny názorně ukazuje příklad 2.6.1 a následné vytvoření konkrétní komponenty je zachyceno na příkladu 2.6.2.

K zaregistrování konkrétní továrny slouží funkce `registerFactory()`. Pro vytvoření konkrétních komponent je určena funkce `create()`.

6.2.2 *X* Factory

Tato třída je opět velmi podobná pro všech šest typ komponent, proto je popíší společně. Tato třída slouží k vytvoření konkrétní komponenty. Používá návrhový vzor Abstract Factory a Template Method.

Pro vytvoření komponenty stačí zavolat metodu `create()`. Tato metoda zavolá funkci `doParse()`, která zpracuje požadavky, kterou jsou na nově vytvořenou komponentu kladeny. Následně je zavolána metoda `doCreate()`, která komponentu skutečně vytvoří.

6.2.3 Parser

Abstraktní třída pro všechny parsery. Zajišťuje čtení vstupního proudu a volání metod, které musí být implementovány v konkrétních potomcích. Tato třída používá návrhový vzor Template Method.

Zavolání funkce `parse()` způsobí, že je zavolána abstraktní funkce `registerPatterns()`, která slouží k zaregistrování vzorů pro parsování vstupu. Tato funkce je volána pouze jednou. Poté začne samotné parsování vstupního proudu. Zavolá se funkce `doBeforeParsing()`, do které je potřeba napsat kód, který inicializuje proměnné. Následně jsou čteny jednotlivé řádky. Na každý takto přečtený řádek je zavolána funkci `doParseLine()`. V této funkci má být implementováno samotné parsování. Jakmile je přečten poslední řádek vstupu, je zavolána funkce `doAfterParsing()` a vrácen vytvořený dokument. Pokud v průběhu nastala nějaká chyba, funkce `parse()` ji zachytí a transformuje na výjimku `CsummakException`.

6.2.4 PreModifier, PostModifier a Modifier

Modifier je abstraktní třída pro všechny modifikátory. Kvůli typové kontrole jsou přidány abstraktní třídy PreModifier a PostModifier. Tyto třídy (a jejich potomci) slouží ke změně struktury dokumentu. Tyto komponenty používají návrhový vzor Template Method.

Dokument modifikuje funkce `modify()`, která zajišťuje volání dalších metod. Před samotnou modifikací je zavolána `doBeforeModifying()`, která slouží k inicializace proměnných. Následně je zavolána funkce `doModify()`, která provádí modifikaci dokumentu. Po této funkci je volána funkce `doAfterModifying()`, která slouží k úklidu. Pokud v průběhu dojde k nějaké chybě, funkce `modify()` ji transformuje na výjimku `CsummakException`.

6.2.5 Ranker

Ranker je abstraktní třída pro všechny rankery. Tato komponenta používá návrhový vzor Template Method.

Každý ranker má jedinečné identifikátor, který zajišťuje, že se dva různé rankery navzájem nepřepíší skóre. Výjimkou jsou naopak rankery, které modifikují skóre přidělené jiným rankerem.

Dokument ohodnocuje funkce `rank()`, která zajišťuje volání dalších metod. Před samotným ohodnocením je zavolána metoda `doInitDocumentInfo`, která inicializuje společné členské proměnné rankeru, jako je počet slov, vět v hodnoceném dokumentu a vážené vektory termů dokumentu a jednotlivých vět. Následně je volána funkce `doPre()`, která slouží k inicializace proměnných. Poté je zavolána funkce `doRank()`, která provádí ohodnocení dokumentu. Po této funkci je volána funkce `doPost()`, která slouží k úklidu. Pokud v průběhu dojde k nějaké chybě, funkce `rank()` ji transformuje na výjimku `CsummakException`.

6.2.6 Writer

Writer je abstraktní třída pro všechny writery. Tato komponenta používá návrhový vzor Template Method.

Dokument vypisuje funkce `write()`, která zajišťuje volání dalších metod. Pro vypsání je zavolána funkce `doWrite()`, která provádí výpis dokumentu. Pokud v průběhu dojde k nějaké chybě, funkce `write()` ji transformuje na výjimku `CsummakException`.

6.2.7 Evaluator

Writer je abstraktní třída pro všechny writery. Tato komponenta používá návrhový vzor Template Method.

Dokument vyhodnocuje podobnost 2 dokumentů pomocí funkce `eval()`, která zajišťuje volání dalších metod. Pro evaluaci je zavolána funkce `doEval()`, která provádí samotnou evaluaci. Pokud v průběhu dojde k nějaké chybě, funkce `eval()` ji transformuje na výjimku `CsummakException`.

6.2.8 Node

Node je abstraktní třída, která je předkem všech elementů, které se vyskytují v dokumentu.

Tato třída poskytuje základní funkcionalitu pro manipulaci se stromem, jako je přidávání a odebírání uzlů.

6.2.9 DocumentInfo

Třída DocumentInfo slouží k uchováni objekt třídy Document. Celá řada sofistikovanějších rankerů potřebuje ke své funkčnosti spočítat vážený vektor termů pro celý dokument, tak i pro jednotlivé věty. Pokud by v jedné konfiguraci bylo takovýchto rankerů více, tak by se tyto údaje počítaly několikrát. Aby se tomu zabránilo, třída DocumentInfo tyto informace o dokumentu uchovává. Pokud dojde ke změně dokumentu data jsou při dalším požadavku přepočítána.

6.2.10 Summarizer

Summarizer je třída, která provádí sumarizaci dokumentu. Jeho nejdůležitější funkcí je funkce `summarize`, která z dokumentu vytvoří jeho extrakt.

Před provedením sumarizace je potřeba zaregistrovat modifikátory a rankery, k tomuto má Summarizer metody `registerPreModifier()`, `registerPostModifier()` a `registerRanker()`. Dále je nutné nastavit, jak velký má vzniknout extrakt. Popis parametrů je u popisu konfiguračního souboru v kapitole 4.5.2. K tomuto účelu slouží třída `SummarySettings` a metoda `setSummarySettings()`.

Postup sumarizace je následující:

1. Na dokument se použijí premodifikátory v tom pořadí, ve kterém byly registrovány.
2. Na dokument se použijí rankery v tom pořadí, ve kterém byly registrovány.
3. Na dokument se použijí postmodifikátory v tom pořadí, ve kterém byly registrovány.
4. Velikost dokumentu se zredukuje na požadovanou velikost. Redukce probíhá následujícím způsobem:
   (a) Věty jsou seřazeny sestupně podle celkového skóre. V případě stejného skóre rozhoduje pořadí věty v dokumentu. Věta, která je blíže začátku má vyšší prioritu než věta, která je od začátku dál.
   (b) Věty jsou do extraktu vkládány tak dlouho, dokud není dosaženo požadované velikosti. Věty s celkovým skóre 0 jsou ignorovány.

6.2.11 EvaluatingWorker

EvaluatingWorker je třída, která zajišťuje hromadnou evaluaci extraktů. Tato třída také využívá návrhový vzor Template Method.

Před samotnou sumarizací je nutné nejdřív zaregistrovat jednotlivé evaluatory prostřednictvím funkce `registerEvaluator()`. Dále je nutné pomocí funkce `setParser()` zaregistrovat výchozí parser. Pro přidávání úkolů (`EvaluationTask` slouží metoda `appendTaks()`.

Nyní je možné spustit evaluaci zavoláním metody `evaluate()`, která postupně zpracovává jednotlivé úkolu a výsledky porovnání jednotlivých dokumentů zapisuje do proudu příslušného evaluátoru.

Bližší popis je možné najít v uživatelské části 4.5.13.

6.2.12 SummarizingWorker

SummarizingWorker je hlavní třídou celého systému, protože vykonává celou řadu činností. Z tohoto důvodu je u něho použit návrhový vzor Singleton.

- řídí parsování konfiguračního souboru
- řídí správu priorit (2.6.2)
- umožňuje registraci všech typů komponent (komponenty, které jsou součástí jsou již zaregistrovány)
- řídí tvorbu extraktů
- řídí evaluaci extraktů

Hlavní funkcí je `run`, která spustí kompletní zpracování konfiguračního souboru. Před samotným zpracováním může být užitečné registrovat parsery, modifikátory, writery a evaluátory, které nejsou jeho součástí. K tomuto účelu slouží funkce `register`*`X`*`Factory()`.

Funkce `run` funguje následovně.

1. Rozparsuje vstupní soubor - již před tímto krokem musely být známy všechny komponenty.
2. Vytvoří objekty tříd `Summarizer`, `EvaluatingWorker` a nastaví je podle konfiguračního souboru.
3. Na všechny vstupní soubory použije metodu `Summarizer::summarize`.
4. Provede evaluaci vytvořených extraktů.

6.3 Testování

Celý program byl po každém zkompilování otestován unit-testy. Pokrytí unit-testy je 86%. Automatické buildy, které se vytváří několikrát denně jsou testovány valgrindem a nástroji pro statickou analýzu kódu (rats, flawfinder). Bezchybné výsledky testů, žádné memory leaky byly podmínkou, aby bylo pokračováno ve vývoji.

Kapitola 7

Výsledky

V této kapitole budou představeny dosažené výsledky společně s jejich vyhodnocením. Celkem byly zkoumány 3 různé typy extraktů - intelektuální, automatické - s jediným algoritmem a automatické s kombinací algoritmů a modifikátorů. V závěru této kapitoly jsou navrženy vhodné kombinace pro praktické použití.

7.1 Intelektuální extrakty

Z novinového článku měli anotátoři za úkol vybrat 20% nejdůležitějších vět. V průběhu se ukázalo, že hranice 20% je pro většinu současných novinových článků příliš nízká. V současné době vychází nové zprávy velmi často (několik článků za hodinu) a obsahují popis aktuální události. Tyto články jsou velmi krátké, proto bylo přidáno několik dodatečných pravidel. Bylo možné vytvářet extrakty článků, které měly alespoň 5 vět. Pokud byl článek příliš krátký, tak z něho bylo možné vybrat 30%.

Pro sběr dat byl použit systém, který je blíže popsán v kapitole 5.1.

Celkem bylo anotátory vytvořeno 409 extraktů. V dalším zpracování se použili pouze ty extrakty, které nebyly jediným extraktem novinového článku, tzn. že k danému novinovému článku existovaly alespoň 2 různé extrakty. Tyto extrakty budu dále nazývat jako *validní*. Takových extraktů bylo 261.

Z validních extraktů bylo možné vytvořit 132 párů, které byly porovnány mezi sebou. Pro evaluaci byly použity tři metody - precision, recall a f-measure. Výsledky těchto metod jsou shrnuty v tabulce 7.1.

Tabulka 7.1: Evaluace extraktů vytvořených lidmi

Metoda	Medián	Průměr	Směrodatná odchylka
Precision	0.5000	0.4665	0.2559
Recall	0.5000	0.5116	0.2903
F-Measure	0.5000	0.4774	0.2565

Tabulka 7.2: Výskyt jednotlivých vět v extraktech

#	Věta	Výskyt	Věta	Výskyt
1	0.0	48%	0.0	48%
2	1.0	40%	1.0	43%
3	2.0	32%	2.0	31%
4	0.1	27%	0.1	26%
5	3.0	24%	3.0	26%
6	1.1	22%	1.1	22%
7	4.0	21%	4.0	21%
8	5.0	18%	5.0	16%
9	2.1	15%	6.0	15%
10	6.0	14%	3.1	15%

Tabulka (7.2) ukazuje, jak byly jednotlivé věty zastoupeny v extraktech. Údaj ve sloupci `Věta` je identifikátor věty a sloupec `Výskyt` udává, v kolika procentech extraktů se daná věta vyskytovala. První dva sloupce jsou údaje pro všechny extrakty a poslední dva pro validní.

Z tabulky 7.2 lze vyčíst, že uživatelé vybírají nejčastěji první větu z každého odstavce. Toto zjištění odpovídá tomu, jak je doporučováno psát texty - první věta v odstavci by měla obsahovat novou myšlenku a ve zbytku odstavce by měla být dále rozvedena.

Obdobného jevu si všimli i Hovy a Lin [8] na článcích ze Ziff-Davisova korpusu.

7.2 Automatické extrakty

Pro generování automatických extraktů bylo použito 131 článků, ke kterým existovaly validní extrakty. Nejdříve jsem provedl extrakty pouze jednotlivými algoritmy, abych zjistil, jak jsou kvalitní a mohl je následně vhodným způsobem zkombinovat.

7.2.1 Jednotlivé algoritmy

Výsledky automatických extraktů, které generují jednotlivé algoritmy jsou zobrazeny v tabulce 7.2.1. Všechny algoritmy byly použity bez dodatečných modifikátorů. Údaj v závorce u názvu blíže specifikuje konfiguraci použitého algoritmu. Hodnoty udávají průměr.

Z tabulky 7.2.1 je patrné, že nejlépe dopadly algoritmy Position, Lead a ParagraphLead, zatímco algoritmy RelevanceMeasure a Centroid dopadly nejhůř i algoritmus Random dosáhl lepších výsledků. Údaj v závorce u Lead a ParagraphLead udává, hodnotu parametru `ratio`. Údaj WUP znamená, že samotný algoritmus byl obalen WordUtilPowerRankerem s hodnotou parametru `utilPower`

Tabulka 7.3: Výsledky jednotlivých algoritmů

Algoritmus	Precision	Recall	F-Measure
Centroid	0.2581	0.2837	0.2622
Centroid(WUP)	0.5118	0.3952	0.4364
FirstParagraph	0.2586	0.4407	0.2920
FirstSentenceOverlap	0.3399	0.3062	0.3116
Lead(20%)	0.3977	0.4082	0.3917
Lead(30%)	0.5118	0.3952	0.4363
ParagraphLead(20%)	0.4651	0.3670	0.4013
ParagraphLead(30%)	0.4752	0.3761	0.4109
Position	0.5118	0.3952	0.4363
Random	0.3418	0.2599	0.2884
RelevanceMeasure	0.2490	0.2641	0.2480
RelevanceMeasure(WUP)	0.2939	0.2353	0.2543

rovnou *1*.

7.2.2 Složené konfigurace

Následně jsem vyzkoušel různé složitější konfigurace. U každé konfigurace je tabulka s výsledky dané konfigurace a jejich modifikací. Sloupce tabulek mají stejný význam jako u předchozí tabulky.

Jednotlivé konfigurace se nachází ve složce `evaluation` a platí, že daná konfigurace se nachází v souboru `eval-nazev.conf`. Pokud se v textu objeví výraz *(a; b; c)* znamená to hodnoty precison, recall, f-measure, kterých daná konfigurace dosáhla. Pokud je uvedeno NázevKomponenty(hodnota), tak pokud má komponenta 1 parametr, tak je to hodnota tohoto parametru. Tedy ShortSentenceRemover(3) znamená ShortSentenceRemover s hodnotou parametru `minLength` rovnou *3*.

Konfigurace Radev

Toto je konfigurace rankeru, kterou použil Radev et al. [17]. Článek ale nezmiňuje přesně, jaké hodnoty použili, proto jsem konfiguraci zmíněnou v článku použil jako základ a dalšími modifikátory jsem se snažil vylepšit její skóre. Výsledky jsou v tabulce 7.2.2.

radev - Základní konfigurace - rekonstruována podle článku.

radev-woutM - Základní konfigurace, ale odstranil jsem modifikátory.

radev-wUP - Základní konfigurace, ale ranker Centroid je obalen WordUtilPowerRankerem s hodnotou *utilPower* rovnou 1. Pro hodnotu 1 dosáhl tato konfigurace nejlepšího skóre.

Tabulka 7.4: Výsledky konfigurace 1

Konfigurace	Precision	Recall	F-Measure
radev	0.3415	0.3873	0.3494
radev-woutM	0.3684	0.3678	0.3568
radev-wUP	0.4491	0.4106	0.4170
radev-wUPwoutM	0.5040	0.3943	0.4321
radev-m0	0.5015	0.3971	0.4334
radev-m1	0.5065	0.4041	0.4401

Tabulka 7.5: Výsledky konfigurace KISS

Konfigurace	Precision	Recall	F-Measure
kiss-position	0.5118	0.3952	0.4363
kiss	0.5215	0.4016	0.4436
kiss-0	0.5220	0.4061	0.4462
kiss-fs-fp	0.5013	0.3900	0.4290

radev-wUPwoutM - Kombinace konfigurace wUP a woutM.

radev-m0 - Základní konfigurace s odstraněním všech modifikátorů zmíněných v článku. CentroidRanker byl obalen WordUtilPowerRankerem s hodnotou *utilPower* rovnou *1.4*. Navíc jsem přidal WordClassRemover s parametrem *preposition, interjection, participle, conjuction, pronoun*. Přidání libovolného jiného modifikátoru výsledné skóre jen snížilo.

radev-m1 - Konfigurace radev-m0, jen je změněn rozsah PositionRankeru(0, 4).

Konfigurace KISS

Smyslem této konfigurace bylo vytvořit s nejnižšími nároky - jak na výpočetní výkon, tak i na nároků míru předzpracování běžného textu. Zrušit předložky, spojky, částice není výpočetně náročné je odstranit, pokud už je informace součástí vstupu, ale je velmi obtížné tuto skutečnost rozpoznat. Výsledky jsou v tabulce 7.2.2.

kiss-position - Pouze PositionRanker.

kiss - Konfigurace, která kombinuje PositionRanker(0,4) ParagraphLeadRanker, který vybírá pouze první větu z odstavce.

kiss-0 - Konfigurace, která kombinuje PositionRanker(0,4) ParagraphLeadRanker, který vybírá pouze první větu z odstavce. Pokud se použil PositionRanker(0, 1), tak výsledek byl (0.4709; 0.3723; 0.4074), pokud PositionRanker(0, 3), tak (0.4988; 0.3909; 0.4281), pokud PositionRanker(0, 5), tak (0.5104; 0.3938; 0.4281). Konfigurace navíc obsahuje ShortSentenceRemover s minLength rovným 3.

Tabulka 7.6: Výsledky dalších konfigurací

Konfigurace	Precision	Recall	F-Measure
misc-0	0.5165	0.4032	0.4431
misc-1	0.5210	0.4066	0.4465

Tabulka 7.7: Výsledky konfigurace KISS

Konfigurace	Precision	Recall	F-Measure
misc-1	0.5210	0.4066	0.4465
kiss-0	0.5220	0.4061	0.4462
kiss	0.5215	0.4016	0.4436
misc-0	0.5165	0.4032	0.4431
radev-m1	0.5065	0.4041	0.4401

kiss-fs-fp - Konfigurace, kde FirstParagraphRanker přiřazuje hodnotu 1, a ParagraphLeadRanker přiřazuje 1. větě v dokumentu také 1.

Další konfigurace

Následně jsem zkusil vytvořit další konfigurace, které by byly dosahovaly lepšího skóre. Některé konfigurace zobrazuje tabulka 7.2.2.

misc-0 - Konfigurace, která kombinuje PositionRanker(0,4) s váhou 4, ParagraphLeadRanker, který vybírá pouze první větu z odstavce s váhou 1 a CentroidRanker, který je obalen WordUtilPowerRankenkerem(1.4) s váhou 3.

misc-1 - Jedná se o konfiguraci *kiss-0*, ale doplněnou o ShortSentenceRemover(3) a RedundantSentenceRemover(0.8).

Přehled nejlepších výsledků

V tabulce 7.2.2 je zobrazen 5 nejlepších konfigurací seřazených podle f-measure. Nejsou zobrazeny komponenty stejného typu, ale komponenty, které se liší o víc než pouze hodnotu parametru. Změnou parametru o 0,1 často vede ke změně skóre v řádech tisícin.

7.3 Diskuze

7.3.1 Relevance Measure a Centroid

Přestože v článku [5] dosahoval sumarizátor užívající jako sumarizační metodu Relevance Measure skóre f-measure okolo 0.6, tak v tomto experimentu dopadl nejhůř. Pokud se pro algoritmus Centroid nepoužije normalizace podle počtu slov ve větě, tak nedosahuje dobrých výsledků. V článku [18] dosahovali s Centroid algoritmem dobrých výsledků, ale neuvádějí, že by tuto normalizaci použili.

Tabulka 7.8: Průměrná délka vět v relevanceMeasure extraktech

Pořadí	Délka
1	216.441
2	179.788
3	155.288
4	142.025
5	127.153
6	121.797
7	106.305
8	98.314
9	92.89
10	83.492

Důvodem je rozdílný postup určování velikosti extraktu. Zatímco v odkazovaných článcích byly vybíráno n nejdůležitějších vět, tak v mé práci byla vybírána část dokumentu měřena ve znacích. Výše uvedené metody zvýhodňují delší věty, protože obsahují více slov, které hodnocené větě zvětšují skóre. Přehled průměrné délky vět v extraktech vytvořených pomocí algoritmu Relevance Measure je uvedena v tabulce 7.3.1. Z této tabulky je tato skutečnost jasně vidět, když pořadí vět podle skóre přesně odpovídá pořadí podle délky věty.

7.3.2 FirstParagraph, Lead, Position

Algoritmy FirstParagraph, Lead a Position shodně vybírají věty od začátku dokumentu. Přestože se jedná o primitivní algoritmy, tak dopadly velmi dobře.

FirstParagraph ztrácel na delších textech, kde délka prvního odstavce nestačila ani na naplnění požadované délky extraktu (Precision: 0,2586). Pokud by bylo omezení na délku extraktu zadáno ne procentuálně, ale fixně - např. 200 znaků mohl dopadnout srovnatelně s ostatními algoritmy z této skupiny.

Algoritmy Lead(20%) a Lead(30%) byly hodnoceny, protože některé extrakty byly vytvořené v kompresním poměru 20% a jiné 30%. Tyto rozdílné velikosti extraktů způsobily horší skóre u Lead(20%), protože většina článků byla krátká a byl tedy použit kompresní poměr 30%.

Algoritmus Position dopadl ze všech jednoduchých algoritmů nejlépe, přestože je nejsnadnější na implementaci.

7.3.3 ParagraphLead

Algoritmus ParagraphLead jsem implementoval, protože podle tabulky 7.2 anotátoři vybírali převážně první větu z odstavce. Tento algoritmus se samostatně neukázal jako příliš užitečný. Jeho implementace je komplikovanější než algoritmů Position nebo Lead, ale dosahuje horších výsledků.

7.3.4 Vliv modifikátorů

Přestože bylo implementováno několik různých modifikátorů (3.4 a 3.6) ukázalo se, že jejich význam je minimální. Při jejich použití dochází ke změnám v řádu tisícin. Takže mi pro praktické nasazení připadá výhodnější je nepoužívat. Navíc se dopředu nedá odhadnou, jak použití konkrétních modifikátorů ovlivní celkové skóre konfigurace. Stávalo se, že kombinace modifikátorů v jedné konfiguraci skóre zvýšila, ale pokud byla použita s jinými rankery, tak výsledek naopak zhoršovala.

7.4 Praktické nasazení

Pro praktické nasazení se jeví jako nejvýhodnější použití algoritmu Position, protože je implementačně nejjednodušší, nejméně náročný na výpočetní výkon a nejméně náročný na analýzu vstupního textu.

Na další místo bych umístil konfiguraci *kiss*, která obsahuje pouze algoritmy Position a ParagraphLead, tedy je srovnatelně náročná s předchozím algoritmem, ale dosahuje nepatrně lepších výsledků.

Je dobré zdůraznit, že toto doporučení se vztahuje pouze na tvorbu extraktů z novinových článků a nemůže být bez dalšího výzkumu zobecněno na jiný typ dokumentů.

Kapitola 8

Závěr

Cílem této bakalářské práce bylo vytvořit systém pro automatickou tvorbu extraktů z novinových článků a jejich následnou evaluaci. V průběhu práce na tomto systému byla vytvořeno několik dalších podpůrných nástrojů, které umožnily tento systém dokončit.

Mezi nejvýznamnější patří on-line systém pro usnadnění tvorbu extraktů uživateli. Díky tomuto systému byla dále získána kolekce více než 400 intelektuálních extraktů novinových článků, které mohou být použity pro srovnávání s automatickými extrakty.

Součástí CSummaKu je více než 30 různých komponent, které je možné spolu propojit, takže je snadné vytvořit konfiguraci, která ze vstupních dat vytvoří extrakty a následně vyhodnotí jejich kvalitu. Díky předpřipravenému rozhraní je možné systém rozšířit o vlastní komponenty.

Díky tomuto systému se ukázalo, že pro tvorbu extraktů z novinových článků stačí i jednoduché algoritmy, které dosahují srovnatelného skóre s mnohem sofistikovanějšími algoritmy. Není ale jasné, zda-li by měly i srovnatelný přísnost pro uživatele.

CSummaK v současné době neobsahuje tolik různých komponent jako srovnatelný systém MEAD, který je vyvíjen od roku 2000. Mezi nejdůležitější funkce, které by bylo dobré, kdyby je systém obsahoval patří bezesporu podpora strojového učení pro hledání správných kombinací komponent a jejich parametrů. Také není možné používat komponenty napsané v interpretovaných jazycích. Tvorbu konfigurací pro sumarizaci by usnadnila podpora XML entit možnost používat v názvech souborů regulární výrazy.

Literatura

[1] Jesse Alpert, Nissan Hajaj: *We knew the web was big...*, 25. 7. 2008, http://googleblog.blogspot.com/2008/07/we-knew-web-was-big.html

[2] *CSTS specifikace*, http://ufal.mff.cuni.cz/pdt2.0/doc/data-formats/csts/html/DTD-HOME.html

[3] Güneş Erkan, Dragomir R. Radev: *LexRank: Graph-based Lexical Centrality as Salience in Text Summarization*, Department of EECS, School of Information University of Michigan, Ann Arbor, MI 48109 USA

[4] Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides: *Design Patterns: Elements of Reusable Object-Oriented Software*, 1995, Published by Addison-Wesley

[5] Yihong Gong , Xin Liu: *Generic text summarization using relevance measure and latent semantic analysis*, Proceedings of the 24th annual international ACM SIGIR conference on Research and development in information retrieval, p.19–25, September 2001, New Orleans, Louisiana, United States

[6] Zprávy Google: http://news.google.com

[7] Jan Hajič: *Disambiguation of Rich Inflection (Computational Morphology of Czech)*, Charles University Press, Prague, 2004

[8] Eduard Hovy, Chin-Yew Lin: *Automated text summarization in SUMMAIRIST*, 1999, In Inderjeer Mani and Mark T. Maybury, editors, Advances in Automatic Text Summarization, chapter 8, p.81–94, MIT Press

[9] H. P. Luhn: *The automatic creation of literature abstracts*, J. Res. Develop., 2:159-165,1959

[10] Mani, I.: Automatic Summarization, John Benjamins Publishing Company, 2001

[11] McKeown, Kathleen and Dragonur Radev: *Generating summaries of multiple news articles*, 1995, In SIGIR 95 Proceeding

[12] Newsblaster: http://www1.cs.columbia.edu/nlp/projects.html

[13] NewsFeed Researcher: http://newsfeedresearcher.com/

[14] NewsInEssence: http://lada.si.umich.edu:8080/clair/nie1/nie.cgi

[15] *Comparison of OpenOffice.org 2 and MS-Office 2007*, March, 2007, http://www.openoffice.org/product/docs/ms2007vsooo2.pdf

[16] D. Radev, T. Allison, S. Blair-Goldensohn, J. Blitzer, A. Celebi, S. Dimitrov, E. Drabek, A. Hakim, W. Lam, D. Liu, J. Otterbacher, H. Qi, Saggion, S. Teufel, M. Topper, A. Winkel, Z. Zhang: *MEAD - a platform for multidocument multilingual text summarization*, 2004, In LREC, Lisbon, Portugal

[17] Dragomir Radev, Sasha Blair-Goldensohn, Zhu Zhang: *Experiments in single and multi-document summarization using MEAD*, In First Document Understanding Conference, New Orleans, LA, September 2001

[18] Dragomir R. Radev, Hongyan Jing, Malgorzata Budzikowska: *Centroid-based summarization of multiple documents: sentence extraction, utility-based evaluation, and user studies*, 2000, NAACL-ANLP 2000 Workshop on Automatic summarization, Seattle, Washington, p.21–30

[19] G. Salton, A. Wong, C. S. Yang: *A vector space model for automatic indexing*, Communications of the ACM, v.18 n.11, p.613-620, Nov. 1975

[20] *Zákon č. 121/2000 Sb., o právu autorském, o právech souvisejících s právem autorským a o změně některých zákonů*