UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
INSTITUTO DE INFORMÁTICA
CURSO DE CIÊNCIA DA COMPUTAÇÃO

ALEXANDRE RAMOS COELHO

Stemming para a língua portuguesa: estudo, análise e melhoria do algoritmo RSLP

Trabalho de Graduação.

Profª. Drª. Viviane Moreira Orengo
Orientadora

Profª. Drª. Luciana Salete Buriol
Co-orientadora

Porto Alegre, junho de 2007.

AGRADECIMENTOS

Agradeço aos meus pais, pelo amor e educação dados, ao meu irmão, por sempre me divertir. Agradeço à minha namorada, Ana, por todo amor, paciência e por compreender a minha inevitável distância enquanto preparava este trabalho. Agradeço aos meus amigos e colegas, por tornarem essa árdua jornada que é a vida acadêmica ao menos divertida, mesmo naquelas péssimas semanas abarrotadas de provas e trabalhos para entregar.

E por último, mas não menos importante (antes pelo contrário), meu agradecimento às minhas orientadoras, Viviane e Luciana, por toda ajuda e suporte dados durante todo este trabalho. Muito obrigado, sem a valiosa ajuda de vocês obviamente este trabalho não existiria.

SUMÁRIO

LISTA DE ABREVIATURAS E SIGLAS

CLEF	Cross-Language Evaluation Forum
IR	Information Retrieval
MAP	Mean Average Precision
NIST	National Institute of Standards and Technology
RI	Recuperação de Informações
RSLP	Removedor de Sufixos da Língua Portuguesa
SRI	Sistema de Recuperação de Informações
TREC	Text Retrieval Conference
UFRGS	Universidade Federal do Rio Grande do Sul

LISTA DE FIGURAS

LISTA DE TABELAS

RESUMO

O presente trabalho apresenta uma ferramenta para a realização de *stemming* na língua portuguesa. O objetivo principal é proporcionar melhorias e disponibilizar as funcionalidades do algoritmo RSLP, previamente proposto na literatura. Além da análise, das melhorias de desempenho, da documentação do código e das novas funcionalidades adicionadas, o trabalho integrou o RSLP ao Zettair, um sistema de Recuperação de Informações. Os resultados de experimentos feitos com a ferramenta mostram que os ganhos de qualidade, em termos de precisão média, trazidos pelo algoritmo original são mantidos e comprovam os ganhos consideráveis de desempenho.

Palavras-Chave: recuperação de informações, radicalização, *stemming*, algoritmo RSLP, Zettair.

Stemming for the Portuguese language: study, analysis and improvement of the RSLP algorithm

ABSTRACT

This work presents a tool for performing stemming in the Portuguese language. The main goal is to improve and make the RSLP algorithm (previously proposed in the literature) available. In addition to the analysis, performance improvements, new features added and source code documentation, this work has integrated the RSLP into Zettair, which is an information retrieval system. Experimental results with the tool have shown that the quality gains, in terms of mean average precision, are maintained and attest the considerable gains in performance.

Keywords: information retrieval, stemming, RSLP algorithm, Zettair search engine.

1 INTRODUÇÃO

Sistemas de Recuperação de Informação utilizam-se de várias técnicas para melhorar seu desempenho em termos de qualidade de resultados. Uma destas técnicas é o *stemming*, que consiste em reduzir palavras relacionadas a uma forma mínima comum, de forma que possam ser combinadas sob uma única representação, chamada de *stem*. Com este processo, obtém-se índices mais enxutos e melhora-se a revocação dos resultados do sistema. Uma das formas mais utilizadas de *stemming,* a remoção de afixos da palavras, depende fortemente da língua para qual foi desenvolvida, sendo que, até pouco tempo, praticamente inexistiam algoritmos deste tipo para a língua portuguesa.

Neste trabalho, serão apresentadas as propostas mais conhecidas de *stemmers* para a língua portuguesa, e um deles, o "Removedor de Sufixos da Língua Portuguesa" (RSLP), proposto por Orengo e Huyck (2001), será analisado de forma mais profunda, sendo utilizado como base para a implementação aqui apresentada. Após a análise deste algoritmo, serão apresentadas propostas de melhorias e novas funcionalidades, visando aumentar seu desempenho e a qualidade de seus resultados. Com estas propostas devidamente implementadas e integradas a um Sistema de Recuperação de Informação, será possível a realização de experimentos, para verificar se as funcionalidades e melhorias propostas atingiram seus objetivos.

Apesar deste trabalho considerar o processo de *stemming* em um contexto de RI, existem outras áreas que podem se beneficiar com esta técnica: Processamento de Linguagem Natural, Mineração de Textos (*Text Mining*), Descoberta de Conhecimento em Bases de dados (*Knowledge Discovery in Databases*) e Recuperação de Informações Multilingües (*Cross-Language Information Retrieval*).

1.1 Motivação

Os motivos que incentivaram a escolha do tema foram:

a) a melhora na recuperação de documentos e do processo de indexação de uma coleção textual proporcionada pelo *stemming*. O ganho dá-se no tamanho do índice gerado, pois cada *stem* no índice irá corresponder a diversas formas variantes no texto;

b) existência de outros estudos que avaliam a validade da utilização de *stemming* em RI;

c) possibilidade de continuidade do trabalho proposto por Orengo e Huyck (2001), criando soluções para as limitações de *stemming* em português e integrando o *stemmer* em um Sistema de Informação;

d) atender a crescente demanda pela implementação do RSLP, observada principalmente no último ano, o que indica um interesse maior pela área e necessidade de novas soluções.

1.2 Objetivos

Os objetivos deste trabalho são:

a) estudar o processo de *stemming*;

b) analisar o algoritmo RSLP;

c) propor e implementar melhorias neste algoritmo, atualizando-o e melhorando seu desempenho geral, tanto em termos de performance quanto em qualidade de resultado;

d) reescrever seu código original, visando facilitar seu uso e integração com outras ferramentas e sistemas de RI, bem como permitir sua portabilidade para outras plataformas e sistemas operacionais;

e) documentar o seu código;

f) realizar experimentos com o algoritmo atualizado e analisar seus resultados, comparando-os com os obtidos pelo algoritmo original.

1.3 Organização da Monografia

Esta monografia encontra-se organizada da seguinte maneira:

a) Capítulo 2: neste capítulo é realizado o levantamento bibliográfico, abordando os conceitos básicos para o melhor entendimento do estudo. Além disso, são apresentados algoritmos propostos para *stemming* na língua portuguesa, entre eles o RSLP, objeto deste trabalho.

b) Capítulo 3: neste capítulo, são apresentadas as melhorias realizadas no algoritmo RSLP. Por fim, este é analisado, e sua complexidade é discutida.

c) Capítulo 4: a seguir, é apresentada a integração do RSLP com um sistema de RI. O sistema escolhido, Zettair, é apresentado e descrito.

d) Capítulo 5: os experimentos e resultados são apresentados neste capítulo, através de metodologia de testes, avaliação de desempenho do processamento e avaliação de precisão do resultado. Para tanto, são utilizados métodos e ferramentas conhecidos na área de RI.

e) Capítulo 6: conclusões do trabalho.

f) Referências, anexos e apêndices.

2 REVISÃO BIBLIOGRÁFICA

Nos subitens a seguir serão abordados referenciais teóricos indispensáveis para o entendimento deste trabalho, sendo apresentados os conceitos de Recuperação de Avaliação, *stemming*. Também serão apresentados os principais algoritmos de *stemming* para a língua portuguesa.

2.1 Conceitos de Recuperação de Informação

A área de Recuperação de Informação está presente no cotidiano da maioria das pessoas, quer quando buscam uma página da *internet* através de um mecanismo de busca, como o Google ou o Yahoo, ou quando buscam um livro ou artigo em um catálogo de obras de uma biblioteca, por exemplo. Segundo Vanrijsbergen (1979), desde a metade do século passado esta área ganhou importância, pois com o aumento da quantidade de informações ficou cada vez mais difícil o acesso preciso e rápido à informação.

A definição de "Recuperação de Informação" (abreviado para RI ou *IR*, do inglês *Information Retrieval*) varia de autor para autor. De acordo com Baeza-Yates (1999), RI trata da representação, armazenamento, organização e acesso a elementos de informação. Na mesma linha de pensamento, Jackson e Moulinier (2002)[1] *apud* Dias (2004) afirmam que recuperação de informação poderia ser definida como a aplicação de tecnologia computacional para tratamento de informação (sua aquisição, organização, recuperação e distribuição). Em suma, e ratificando a conclusão de Dias (2004), pode-se afirmar que um sistema de recuperação de informação é responsável por indexar, buscar e classificar uma coleção de documentos.

O objetivo principal de um Sistema de Recuperação de Informação (SRI) é, basicamente, organizar uma coleção de documentos, receber do usuário consultas (que exprimem textualmente sua necessidade de informação), processá-las e retornar ao usuário o máximo de documentos relevantes (que contêm a informação desejada pelo mesmo) e o mínimo de documentos não relevantes (aqueles que, por ineficiência do sistema, são incorretamente considerados como relevantes, porém não interessam ao usuário). Para auxiliar o SRI a melhorar seu desempenho são utilizadas várias técnicas, entre as quais destaca-se a de radicalização, ou *stemming*, aplicada por algoritmos chamados de *stemmers*. Alguns estudos usam os termos "radicalização" e "radicalizador", para indicar o processo de extração de radicais das palavras e seus algoritmos ou programas. Neste trabalho, contudo, optou-se pelos termos em inglês (*stemming*, *stemmer* e suas variações), por serem mais amplamente utilizados.

[1] JACKSON, P.; MOULINIER, I. **Natural Language Processing for Online Applications:** text retrieval, extraction and categorization. Amsterdan: John Benjamins Publishing Company, 2002, v. 5, 225 p.

2.1.1 Avaliação da Qualidade

Como já citado, nem todos os documentos recuperados por um Sistema de Recuperação de Informação a partir de uma consulta são relevantes, bem como nem todos documentos relevantes são recuperados. Isso se deve a fatores como ineficiência do próprio sistema, consultas dúbias ou mal formuladas, entre outros. Por isso, é necessário avaliar os resultados obtidos com determinado SRI, medindo seu desempenho. Para tanto, pode-se utilizar duas importantes métricas, dentre outras:

a) **precisão (*precision*):** indica a proporção entre o número de documentos relevantes recuperados em uma consulta sobre o total de documentos recuperados. Deseja-se que esse índice se aproxime ao máximo de 1, ou seja, que todo documento recuperado na consulta seja relevante.

b) **revocação (*recall*):** indica a proporção entre o número de documentos relevantes recuperados em uma consulta sobre o número total de documentos relevantes na coleção. Assim como na precisão, deseja-se que este índice se aproxime ao máximo de 1, indicando que todos os documentos relevantes da coleção foram recuperados.

De acordo com Orengo (2006), o aumento nos índices de precisão normalmente é acompanhado de baixos índices de revocação e vice-versa, ou seja, se o SRI for projetado para possuir grande precisão, possivelmente não teremos todos os dados relevantes recuperados e, se ao contrário, for projetado para recuperar o máximo de informações relevantes (revocação alta), a precisão provavelmente será baixa.

Como medida para avaliação da precisão, utiliza-se a "precisão média", ou *average precision*, que trata-se da média das precisões obtidas após cada documento relevante recuperado pelo sistema para uma consulta (WITTEN; MOFFAT; BELL, 1999). Com a média das consultas, obtém-se a média das precisões médias de cada consulta, medida também conhecida como *mean average precision* (MAP), que é muito utilizada para avaliar a precisão obtida por Sistemas de Recuperação de Informação. Devido ao seu extenso uso em trabalhos sobre avaliação destes sistemas, a sigla "MAP" será utilizada daqui em diante neste trabalho.

2.1.2 Campanhas de Avaliação de Sistemas

Com o objetivo de encorajar pesquisas na área de Recuperação de Informação, o governo norte-americano, através do *National Institute of Standards and Technology* (NIST) e do seu departamento de defesa, patrocina a *Text Retrieval Conference* (TREC)[2]. Esta conferência tem por objetivo, entre outros, fornecer coleções de documentos de testes, bem como consultas referentes a estes, para que os participantes possam utilizá-las em seus sistemas de RI e avaliar os resultados obtidos por estes. Os participantes utilizam seus sistemas para obter um conjunto de documentos considerados relevantes para cada consulta e o encaminham ao NIST, que avalia as respostas quanto à precisão do resultado (há um conjunto definido de documentos relevantes para cada consulta). Assim, é possível utilizar os dados fornecidos no TREC (coleção de documentos, consultas e resultados corretos), para avaliar a precisão do resultado obtido por determinado SRI, o que se mostrará importante para este trabalho, pois será necessário avaliar se as modificações efetuadas no algoritmo original do RSLP acarretaram melhoria nos resultados do sistema.

[2] Disponível em <http://trec.nist.gov/overview.html>. Acesso em 21 maio 2007.

Além do TREC, é importante citar também o *Cross-Language Evaluation Forum* (CLEF)[3], que trata-se de um projeto de avaliação e teste de Sistemas de Recuperação de Informação financiado pela União Européia, com objetivos similares ao TREC, porém focado nas línguas européias, como o português, espanhol e italiano, e não somente no inglês. Ambas campanhas utilizam a MAP como medida padrão em seus experimentos.

2.2 Definição de *stemmer*

Stemmers são programas ou algoritmos que têm por objetivo reduzir palavras a uma forma comum de representação, chamada de *stem* (ou "radical"), através do processo de *conflação*, que significa fundir ou combinar as formas morfologicamente variantes de um termo (FRAKES; BAEZA-YATES, 1992). A técnica de *stemming* consiste em reduzir as palavras aos seus radicais (*stems*), através da remoção de seus afixos (sufixos e prefixos), de forma que palavras morfologicamente relacionadas sejam representadas por uma única forma comum, permitindo assim serem combinadas. Por exemplo, as palavras "aprendo", "aprendeu" e "aprendendo" (todas formas derivadas do verbo "aprender"), poderiam, através da remoção de seus sufixos, ser reduzidas ao *stem* "aprend". Este radical não faz sentido como palavra na língua portuguesa, pois não é necessariamente o mesmo radical lingüístico da palavra (PALLES; FUEZI, 2005). Apesar disso, consegue manter informação suficiente sobre as palavras que o originaram, não perdendo precisão e permitindo recuperá-las em uma pesquisa.

A técnica de *stemming* é utilizada em RI com a intenção de melhorar a qualidade do resultado produzido pelo sistema. Para tanto, parte-se do pressuposto que o usuário, ao inserir determinado termo em uma consulta (por exemplo, "aprendeu"), está também interessado em documentos que possuem não somente este termo exato, mas também as suas variantes ("aprendo", "aprendido", "aprender", etc.). Desta forma, ao reduzirmos os termos de pesquisa aos seus respectivos radicais, bem como os termos dos documentos indexados, melhoramos as chances de um documento com a informação desejada pelo usuário ser recuperado, ou seja, melhoramos sua revocação. Além disso, a redução de várias palavras a um único termo raiz traz também a vantagem de reduzir o tamanho dos índices gerados pelo SRI, que pode chegar a um fator de compressão de mais de 50% em alguns casos (FRAKES; BAEZA-YATES, 1992).

Apesar das vantagens citadas, o uso de *stemming* traz também algumas desvantagens, dentre as quais destacam-se a perda de precisão na recuperação da informação (uma vez que não temos mais os termos exatos para a consulta, e sim seus radicais), e a perda do contexto da informação, que induz a produção de *stems* iguais para termos com sentidos diferentes mas de mesma escrita (homônimos), ocasionando a conflação de termos não relacionados. Como exemplo deste último caso, temos as palavra "verão", que pode ser o verbo "ver" conjugado na 3ª pessoa do plural do futuro do indicativo, ou um substantivo (uma das estações do ano).

Além disso, o uso de *stemming* pode causar erros que são divididos em duas categorias (ORENGO; HUYCK, 2001):

a) ***overstemming*, ou *excesso de radicalização*:** ocorre quando parte do *stem* também é removido na extração do sufixo, podendo resultar na conflação de palavras não relacionadas. Com isso, o número de documentos não relevantes recuperados tende a aumentar. Um exemplo é

[3] Disponível em <http://www.clef-campaign.org/>. Acesso em 21 maio 2007.

a remoção do sufixo “inho” (indicando diminutivo), que é correta na palavra “filhinho”, mas incorreta na palavra “linho”.

b) ***understemming*:** estes erros ocorrem quando o sufixo não é completamente removido da palavra, resultando em um *stem* incorreto. Tem como efeito prejudicar a conflação de palavras relacionadas, causando a não recuperação de documentos que seriam relevantes. Por exemplo, podemos citar a palavra “referência”, cujo *stem* obtido é “referênc”, em vez de “refer”, que seria o considerado correto (CHAVES, 2003).

Esses erros reduzem a qualidade dos resultados obtidos nos sistemas de RI, uma vez que interferem na conflação dos termos processados.

2.2.1 Métodos de *stemming*

São citados na literatura 4 métodos de *stemming* de palavras. Estes métodos são brevemente descritos abaixo:

a) **Consulta a dicionários:** a idéia deste método é possuir um dicionário com todas as palavras de determinada língua e seus respectivos radicais. A palavra a ser processada é procurada no dicionário, e, quando encontrada, consulta-se seu *stem*. Uma grande falha neste método é que obviamente não é viável identificar absolutamente todas as palavras (e suas variantes) de uma língua, além do ônus que representa armazenar tal dicionário, devido as suas dimensões. Se fosse possível criar este dicionário, este método de *stemming* certamente apresentaria os melhores resultados, pois seria exato. Pode ser aplicado em situações onde o domínio das palavras é reduzido (como o de palavras reservadas de uma linguagem de programação, por exemplo).

b) **Variedade de Sucessores:** necessita de um *corpus*[4] definido antes do processamento das palavras. Este método, proposto por Hafer e Weiss (1964)[5] *apud* Frakes e Baeza-Yates (1992), utiliza *substrings*[6] de uma palavra, contabilizando o número de caracteres que a sucedem nas palavras do *corpus*. Este número é a chamada “variedade de sucessores”, que será cada vez menor quanto mais longa for a *substring* utilizada. Este método é divido em 3 etapas distintas: determinação da variedade de sucessores, segmentação da palavra e, finalmente, a seleção do *stem* (seleção da *substring* que melhor represente o *stem* final).

c) **n-Gramas:** método que baseia-se na divisão das palavras em cadeias de *n* caracteres consecutivos (por exemplo, para n=2 teríamos digramas). Após essa divisão da palavra, é calculado um coeficiente sobre o número de n-gramas gerados (únicos e também o total de uma palavra) e então gerada uma matriz de similaridade entre todos os pares possíveis de todas as palavras processadas. Essa matriz indica a similaridade entre as palavras de um par. Como apontado por Palles e Fuezi (2005), não trata-

[4] Coletânea ou coleção de documentos.

[5] HAFER, M.; WEISS, S. Word Segmentation by Letter Succession Varieties. **Information Storage and Retrieval**, Oxford, v. 10, p. 371-385, 1974.

[6] Seqüência de caracteres consecutivos de uma palavra, representando somente uma subparte dela.

se exatamente de um método de *stemming* de palavras, pois as palavras não são reduzidas ao seu *stem*, mas sim um método de identificação de palavras similares.

d) **Remoção de afixos (sufixos/prefixos):** este método baseia-se na remoção de afixos das palavras, que resultariam em seus respectivos radicais. Estes afixos são definidos em um conjunto de regras pré-definido, que é verificado a cada palavra processada. Caso a palavra possua o prefixo ou sufixo definido na regra, ele é removido (ou substituído por outro, para remoção em uma regra posterior). Ao contrário dos métodos de Variedade de Sucessores e de n-Gramas, este método é dependente do vocabulário da língua (PALLES; FUEZI, 2005), pois os afixos mais conhecidos devem ser definidos nas regras citadas. Além destas, podem existir também listas de exceções, que contém palavras que não devem ter seus afixos removidos, evitando-se assim erros de *overstemming*. A maioria dos *stemmers* em uso baseiam neste método.

2.3 *Stemmers* para a língua portuguesa

Um problema dos algoritmos de *stemming* que utilizam remoção de afixos é que são extremamente dependentes das línguas para as quais foram escritos, pois baseiam-se diretamente nas regras de formação de palavras de cada língua para detectá-los e removê-los (HONRADO, 2000). A maioria destes algoritmos, por terem sido desenvolvidos para a língua inglesa, não apresentam um desempenho tão bom quanto o obtido sobre a língua original, quando aplicados a línguas derivadas do latim como o português, pois estas são mais complexas gramaticalmente. Além disso, a língua portuguesa apresenta inflexões que geralmente causam alterações profundas no radical das palavras, impedindo seu processo de conflação, no qual o *stemming* é baseado. Por este motivo, há a necessidade de se utilizar um *stemmer* especialmente projetado para o processamento de palavras escritas em português.

Dentre os algoritmos propostos para a língua portuguesa, três se destacam: a versão para português do algoritmo de Porter (2005), o "Removedor de Sufixos da Língua Portuguesa" (RSLP), proposto por Orengo e Huyck (2001) e o algoritmo STEMBR, proposto por Alvares, Garcia e Ferraz (2005). Há também o algoritmo "Pegastemming", de autoria de Marco Antonio Insaurriaga Gonzalez, porém não foi encontrada publicação descrevendo este algoritmo, exceto um estudo de Chaves (2003), onde o mesmo é comparado com o RSLP.

2.3.1 O algoritmo de Porter para o português

O algoritmo original de Porter (1980), projetado originalmente para a língua inglesa, é baseado em uma série de regras e condições aplicadas seqüencialmente. Uma regra é definida na forma

(condição) sufixo1 -> sufixo2

onde:

a) *(condição)* é uma condição a ser testada (por exemplo, tamanho da palavra restante após a remoção do Sufixo1).

b) *sufixo1* é o sufixo a ser procurado na palavra sendo processada.

c) *sufixo2* é o sufixo a ser adicionado à palavra resultante da remoção de *sufixo1*.

As regras são aplicadas caso o item "b" (*sufixo1*) seja encontrado na palavra analisada e as condições definidas no item em *(condição)* sejam atendidas, sendo então o *sufixo1* removido da palavra e substituído por *sufixo2*, caso este esteja definido. Assim, aplicando-se estas regras em uma seqüência pré-definida, a palavra resultante será o *stem* da palavra original. São definidas ao total 5 regras (algumas com várias sub-regras), responsáveis pela remoção de sufixos de plural, sufixos de verbo e sufixos comuns.

A implementação do algoritmo de Porter para português, apresentada posteriormente (PORTER, 2005), segue o mesmo modelo do algoritmo original, possuindo também cinco regras básicas:

a) regra de remoção de sufixos comuns.

b) regra de remoção de sufixos verbais, executado caso a regra em "a" não tenha modificado a palavra.

c) regra de remoção da vogal "i", caso precedida da consoante "c" no final da palavra.

d) regra de remoção de sufixos residuais ("os", "a", "i", "o", "á", "í", "ó"), executada caso ambas as regras "a" e "b" não tenham alterado a palavra.

e) regra de remoção dos sufixos "e", "é","ê", tratamento da cedilha e tratamento das sílabas "gue", "gué" e "guê".

Antes da aplicações das regras, as vogais nasalizadas "ã" e "õ" são substituídas no termo por "a~" e "o~". Após a aplicação das regras, essas vogais nasalizadas são retornadas à sua forma original.

2.3.2 O algoritmo STEMBR

Este algoritmo foi proposto baseado em um estudo estatístico do *corpus* LexWeb (MONTEIRO JR, 2004 *apud* ALVARES;GARCIA;FERRAZ, 2005), de onde se obteve uma lista mostrando a freqüência da última letra das palavras processadas. Ele se diferencia dos outros dois algoritmos analisados, pois realiza também a redução de prefixos. O algoritmo possui basicamente 3 módulos aos quais a palavra a ser processada é submetida (ALVARES; GARCIA; FERRAZ, 2005):

a) tratamento de casos específicos, onde os módulos "b" e "c" não devem ser aplicados, como verbos em suas formas irregulares de conjugação e palavras muito comuns (conhecidas como *stopwords*).

b) redução de prefixos, a partir de uma lista de prefixos conhecidos e suas exceções.

c) redução de sufixos, considerado o processo mais importante a ser executado. Consiste em um conjunto de regras (394 no total), onde cada regra indica o sufixo a ser removido, o tamanho mínimo do *stem* resultante e uma lista de exceções, de forma similar ao algoritmo RSLP .

Como será visto, este algoritmo guarda várias semelhanças com o RSLP, apresentado a seguir.

2.3.3 O Removedor de Sufixos da Língua Portuguesa (RSLP)

Foi proposto por Orengo e Huyck (2001) um algoritmo especialmente projetado para o português, chamado de "Removedor de Sufixos da Língua Portuguesa", ou RSLP. Este algoritmo será analisado em maior profundidade em relação aos demais apresentados, uma vez que trata-se do objeto deste trabalho. Sua arquitetura é similar a do algoritmo de Porter, no sentido de aplicar sucessivos passos de remoção de sufixos sobre uma palavra, definidos através de regras. A diferença é que, além de utilizar regras específicas para o português, conta também com um dicionário de exceções, evitando assim remover sufixos de palavras cuja terminação é somente similar a um sufixo (por exemplo, o sufixo de diminutivo "inho" pode ser removido de "passarinho", mas não da palavra "linho", apesar desta terminar com uma seqüência de caracteres idêntica ao sufixo em questão).

O processo de *stemming* do RSLP foi definido originalmente com uma seqüencia de 8 passos de redução executados em uma determinada ordem. Um passo define uma série de regras que contam cada uma com os seguintes itens:

a) um sufixo.

b) um tamanho mínimo para o *stem* resultante após a remoção do sufixo. Desta forma, evita-se assim remover sufixos de palavras muito pequenas, que resultariam em *stems* incorretos.

c) um sufixo de substituição (opcional).

d) uma lista de exceções (também opcional).

Cada regra é declarada na forma mostrada na Figura 2.1:

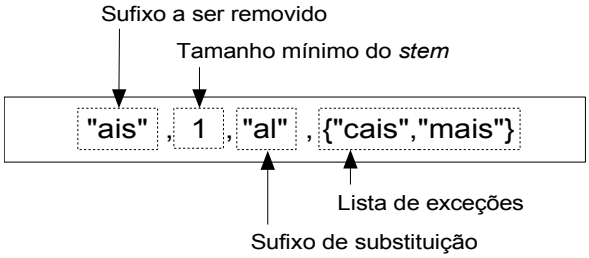

Figura 2.1: Exemplo de declaração de regra do algoritmo RSLP.

As regras de um passo são testadas sucessivamente, verificando-se a presença do sufixo na palavra processada. Se o sufixo da regra for encontrado, verifica-se se a palavra não consta na lista de exceções daquela regra. Caso não conste, verifica-se se o stem resultante da remoção do sufixo possui o tamanho mínimo requerido pela regra. Se possuir, remove-se o sufixo, adiciona-se o sufixo de substituição, se definido, e encerra-se a execução deste passo, passando-se para o próximo. Caso a palavra conste na regra de exceções, passa-se diretamente para a próxima regra. Caso a lista de regras termine sem que ocorra alguma modificação na palavra, continua-se no próximo passo. É importante salientar que as regras dentro de um passo seguem uma ordem determinada, de forma que o maior sufixo possível é testado antes de um sufixo menor. Isso garante

que, por exemplo, o sufixo “s”, pertencente a redução de plural, não seja incorretamente removido de uma palavra antes do sufixo “es” ser testado.

Os passos de redução são os seguintes:

a) **Redução do Plural:** remove os sufixos de plural das palavras (usualmente “s”). Cabe lembrar que nem todas as palavras terminadas em “s” estão no plural (*lápis*, por exemplo), por isso devem constar na lista de exceções deste passo. Possui 11 regras definidas.

b) **Redução do Feminino:** transforma o gênero da palavra de feminino para masculino, como, por exemplo, “casada” para “casado”. Tem 15 regras.

c) **Redução Adverbial:** este passo possui somente um sufixo (“mente”), pois é o único que denota advérbios em português.

d) **Redução do Aumentativo:** apesar do nome limitar a aumentativo, este passo reduz o grau das palavras de aumentativo, superlativo ou diminutivo para o normal. Possui 23 regras definidas.

e) **Redução Nominal:** este passo originalmente definia 61 sufixos para substantivos e adjetivos (ORENGO; HUYCK, 2001), porém os autores, em sua implementação do algoritmo, expandiram o número de sufixos para 84. Caso a palavra seja reduzida neste passo, os passos indicados por “f” e “g” não são executados.

f) **Redução Verbal:** este passo reflete a complexidade da língua portuguesa em relação à inglesa. São definidas 101 regras diferentes, pois os verbos regulares em português têm mais de 50 formas, cada uma com um sufixo específico. Após a execução deste passo, a palavra (verbo) é reduzida a sua raiz, indo direto para a execução do passo descrito no item “h”.

g) **Remoção de Vogais:** este passo tem por objetivo remover a última vogal (“a”, “e” ou “o”) de palavras que não foram reduzidas pelos passos de “e” e “f”, garantindo sua conflação com outras formas variantes que foram processas por passos anteriores. Por exemplo, a palavra “menino” que não foi reduzida por nenhum passo anterior, teria sua última letra “o” removida neste passo. Desta forma, o *stem* resultante (menin) seria igual ao de formas variantes da palavra, como “menininho”, “meninão”, “menina” (ORENGO; HUYCK, 2001).

h) **Remoção de Acentos:** este passo substitui todas as letras acentuadas por seus equivalentes sem acentos, garantindo assim que formas variantes com e sem acentos de uma palavra sejam reduzidas ao mesmo *stem*. Por exemplo, “psicologia” e “psicólogo” são reduzidos aos radicais “psicolog” e “psicológ”, respectivamente. Após a execução deste passo, os radicais se igualariam, devido a substituição de “ó” por “o”, e as duas formas seriam então combinadas. Como apontado por Orengo e Huyck (2001), este deve ser o último passo a ser executado, pois várias regras contam com sufixos e exceções acentuadas, que não seriam devidamente aplicadas caso a palavra original tivesse seus acentos removidos.

A Figura 2.2 mostra um fluxograma onde a ordem em que estes passos devem ser executados pode ser observada:

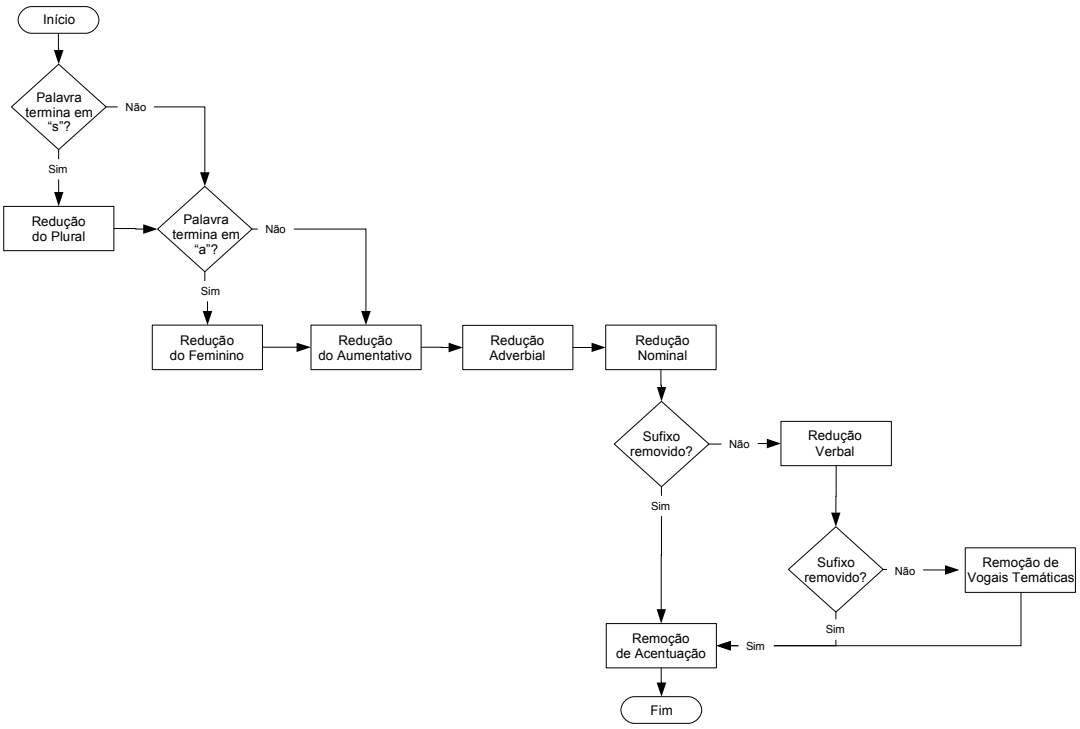

Figura 2.2: Seqüência de passos do algoritmo RSLP (ORENGO; HUYCK, 2001).

A Figura 2.3 mostra um fluxograma ilustrando o processamento interno dos passos de Redução do Plural, Redução do Feminino e Redução de Aumentativo (itens "a", "b" e "c"):

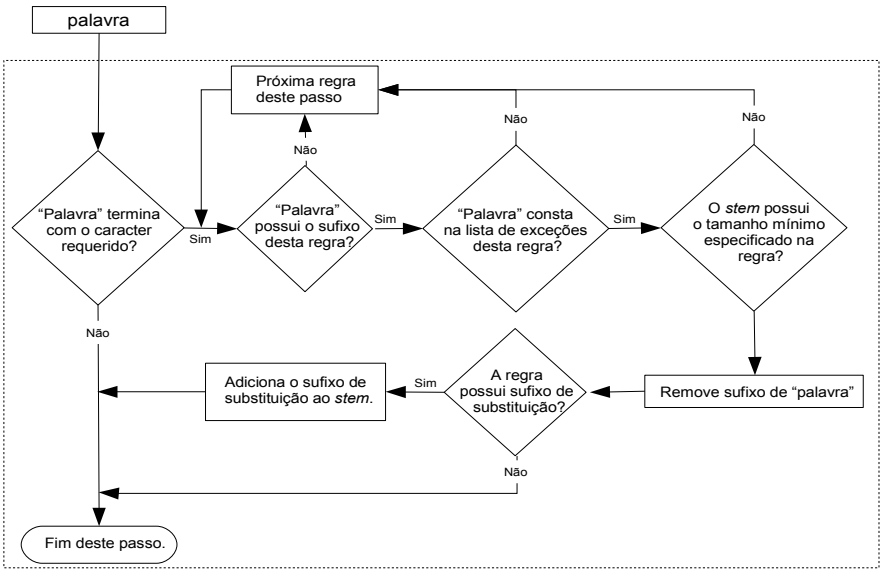

Figura 2.3: Fluxograma das reduções de plural, de feminino e de aumentativo.

Os demais passos de redução comportam-se de forma um pouco diferente da mostrada na figura anterior. Não há teste do último caractere da palavra, como ocorre para os passos descritos nos itens "a" e "b". Além disso, a verificação de exceções de

uma regra não é feita através comparação de palavras completas (comparação da *string*[7] da palavra com toda a *string* de cada exceção, caracter a caracter) e sim através de uma nova busca de sufixos na palavra a ser processada, sendo que os sufixos a serem buscados são cada uma das exceções da regra. Esta diferença é ilustrada na Figura 2.4.

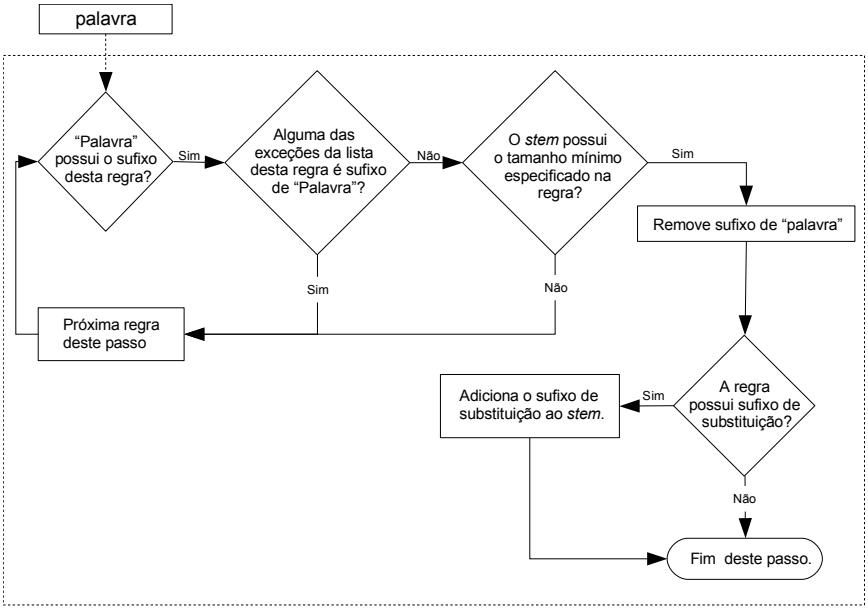

Figura 2.4: Fluxograma de execução dos demais passos de redução.

2.3.4 Comparação entre os algoritmos

O desempenho dos três algoritmos apresentados foram medidos e comparados em vários artigos. O algoritmo RSLP, ao ser comparado com a versão para português do algoritmo de Porter, é considerado mais eficiente, por cometer um número menor de erros de *overstemming* e de *understemming* (ORENGO; HUYCK, 2001), indo de encontro à conclusão do estudo de Palles e Fuezi (2005), que afirmam "[...] isso não significa que um algoritmo é melhor do que o outro, pois os métodos utilizados não oferecem uma estrutura para se fazer essa afirmação [...]". Ao ser comparado com o algoritmo Pegastemming, também apresentou um desempenho melhor, acertando o *stem* de 78% das palavras submetidas para processamento dos *stemmers* (CHAVES, 2003), e, ao ter seu desempenho comparado com o do algoritmo STEMBR, os algoritmos RSLP mostraram equivalentes, apresentando resultados praticamente idênticos na maioria das avaliações (ALVARES; GARCIA. FERRAZ, 2005).

2.4 Resumo do capítulo

Neste capítulo foram apresentados os conceitos básicos necessários para melhor entedimento deste trabalho. Apresentou-se a definição de Recuperação de Informação, os conceitos de precisão e revocação, utilizados para medir o desempenho de um Sistema de Recuperação de Informação. Também foi introduzido o funcionamento dos *stemmers*, seus principais métodos de funcionamento e as versões propostas para a língua portuguesa, bem como uma breve comparação entre elas. No próximo capítulo

[7] Seqüência de caracteres que representam uma palavra.

será feita uma análise mais detalhada do algoritmo RSLP (foco deste trabalho), e serão vistas as modificações efetuadas neste algoritmo, com o objetivo de melhorá-lo.

3 MELHORIA DO ALGORITMO RSLP

Neste capítulo, o algoritmo RSLP em sua forma original, proposto por Orengo e Huyck (2001), será analisado. Serão também descritas as melhorias nele inseridas, com o objetivo de se obter melhorias em termos de desempenho de processamento e qualidade de resultados.

3.1 Motivação

Notou-se, por parte da orientadora deste trabalho, uma crescente demanda pela implementação do RSLP, contabilizando mais de 30 solicitações durante a realização deste trabalho, vindas de diversas partes do país. A implementação original foi feita em 2001, como parte integrante de um projeto maior, ou seja, o RSLP tratava-se de uma ferramenta acessória para realização de outro trabalho. Desta forma, o código-fonte não foi otimizado e nem escrito de forma a ser usado por outras pessoas além dos autores, não havendo documentação explicativa da codificação, pois não havia esta necessidade no momento. Também não foi inicialmente prevista a integração do *stemmer* em um sistema maior (um Sistema de Recuperação de Informação, por exemplo), o que seria interessante para que o mesmo servisse para outras aplicações além do trabalho original dos autores.

Além dos motivos expostos anteriormente, ainda existia a possibilidade de dar continuidade ao proposto em Orengo e Huyck (2001), no sentido de criar soluções para as dificuldades de *stemming* em português apontadas[8], bem como integrar o *stemmer* em um Sistema de Recuperação de Informação para verificar seu impacto em resultados de *recall* e *precision*[9].

3.2 Análise da Implementação Original

A versão original do RSLP foi implementada em linguagem C, e diferenciava-se em alguns itens do algoritmo apresentado em Orengo e Huyck (2001):

a) a ordem de execução dos passos foi alterada, pois ao implementar o algoritmo, os autores perceberam que fazia mais sentido executar o passo de redução adverbial logo após a redução de plural e antes da redução de feminino, conforme ilustrado na Figura 3.1 (a área com hachurada destaca a mudança). Com esta mudança, quando o sufixo de advérbio é removido de uma palavra do gênero feminino, ela ainda é processada

[8] Capítulo 3, "Difficulties in Stemming Portuguese" (ORENGO; HUYCK, 2001).

[9] Capítulo 5, "Conclusion" (ORENGO; HUYCK, 2001).

pelo passo de redução de feminino (que provavelmente removerá mais um sufixo da palavra restante), o que não ocorria no algoritmo original. Desta forma, espera-se diminuir os erros de *understemming*.

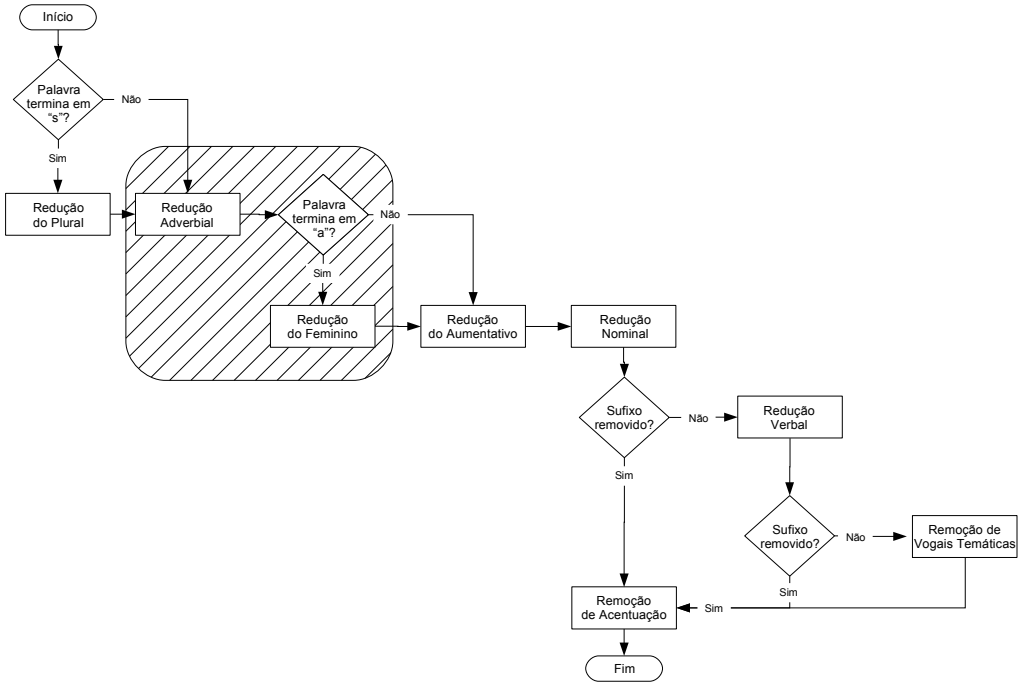

Figura 3.1: Fluxograma de execução dos passos da implementação proposta.

b) o número de regras de vários passos foi expandido, cobrindo assim um maior número de sufixos. Os passos e suas regras são mostrados no Apêndice A.

Cada um dos passos de redução é executado em sua própria função, que por sua vez define todas as suas regras através de uma estrutura que segue o exemplo apresentado em Orengo e Huyck (2001). Um exemplo de código para uma destas funções pode ser visto na Figura 3.2.

A declaração de vetores multi-dimensionais de grande tamanho em cada função de redução, como mostrado na Figura 3.2, não é a abordagem mais eficiente, pois a cada chamada de uma destas funções o programa aloca espaço na pilha da função, copia os valores do vetor para ela (no caso em questão, a estrutura de regras), e, no encerramento da função, libera a memória da pilha. Este procedimento é executado muitas vezes (uma para cada chamada a função, que pode ocorrer centenas de milhares de vezes, dependendo do número de palavras processadas) e, com isso, têm-se um desperdício de processamento, já que a mesma tarefa é executada repetidamente de forma desnecessária. Isto poderia ser evitado caso o vetor de regras de cada passo de redução fosse alocado uma única vez e reaproveitado pelas suas execuções subseqüentes.

```
01: /*****************************************************************************/
02: void plural(char w[])
03: {
04:
05:    struct stem_struct plural_suffix[PLURAL]= {
06:    {"ns",1,"m"},
07:    {"ões",3,"ão"},
08:    {"ães",3,"ão",{"mãe"}},
09:    {"ais",1,"al",{"cais","mais"}},
10:    {"éis",2,"el"},
11:    {"eis",2,"el"},
12:    {"óis",2,"ol"},
13:    {"is",2,"il",{"lápis","cais","mais","crúcis","biquínis","pois",
14:                                        "depois","dois","leis"}},
15:    {"les",3,"l"},
16:    {"res",3,"r"},
17:    {"s",2,"",{"aliás","pires","lápis","cais","mais","mas","menos",
18:                "férias","fezes","pêsames","crúcis","gás",
19:                "atrás","moisés","através","convés","ês",
20:                "país","após","ambas","ambos","messias"}}};
21:
22:    int i=0;
23:    int done = false;
24:
25:    while ((i<PLURAL) && (!done))
26:    {
27:        if (strip_suffix2(w, plural_suffix[i])) done = true;
28:        i++;
29:    }
30: }
```

Figura 3.2: Exemplo de função original do RSLP (ORENGO, 2001).

Deve-se considerar, também, que passos que possuem um número elevado de regras e exceções podem ocupar mais memória do que a disponibilizada na pilha de procedimentos pelo compilador utilizado. Este é o caso em questão, pois o código original do RSLP é normalmente compilado no *Borland C Compiler* (BCC), porém é recusado por compiladores mais modernos como o *GNU C Compiler* (GCC), por causar erros justamente pelo tamanho do vetor de regras no carregamento das funções[10], necessitando a reescrita do código para uso em outros compiladores. Com isso, têm-se um código não-portável, já que limita-se o número de compiladores que conseguem compilá-lo.

A implementação original do RSLP engloba também funções de leitura de arquivos com os documentos e extração de palavras dos mesmos. Estas funções não são relacionadas ao processo de *stemming* em si, fugindo do foco principal do algoritmo, e deveriam ser executadas por um programa externo (um Sistema de Recuperação de Informação, por exemplo), cabendo ao *stemmer* apenas as tarefas de receber uma palavra, processá-la e retornar o *stem* obtido.

[10] Erro retornado pelo compilador: "*error: initializer-string for array of chars is too long*".

3.3 Melhorias implementadas

Aqui serão descritas as melhorias introduzidas no RSLP com o objetivo de solucionar os problemas apontados na motivação.

A nova implementação do RSLP, aqui proposta, foi escrita em linguagem C (como a versão original), por ser uma linguagem muito difundida, portável (permitindo facilmente sua adaptação para diversos sistemas operacionais e plataformas) e por gerar código com excelente desempenho de execução, quando comparada com outras linguagens, como Java.

3.3.1 Documentação do código-fonte

Para documentação do código-fonte da nova implementação do RSLP, foi utilizado o padrão utilizado pela ferramenta Doxygen[11], que é um gerador de documentação a partir de comentários inseridos no código-fonte. Toda função, macro ou estrutura declarada no código possui uma breve descrição de seu funcionamento, descrição de todos seus parâmetros e seus valores de retorno.

Optou-se também por gerar a documentação do RSLP no formato HTML, permitindo assim publicá-la diretamente na *web*, ganhando-se em termos de facilidade de consulta, pois neste formato é possível navegar entre os vários arquivos fonte, suas variáveis, declarações de estruturas, etc. Um exemplo de página de documentação do RSLP neste formato é mostrado na Figura 3.3.

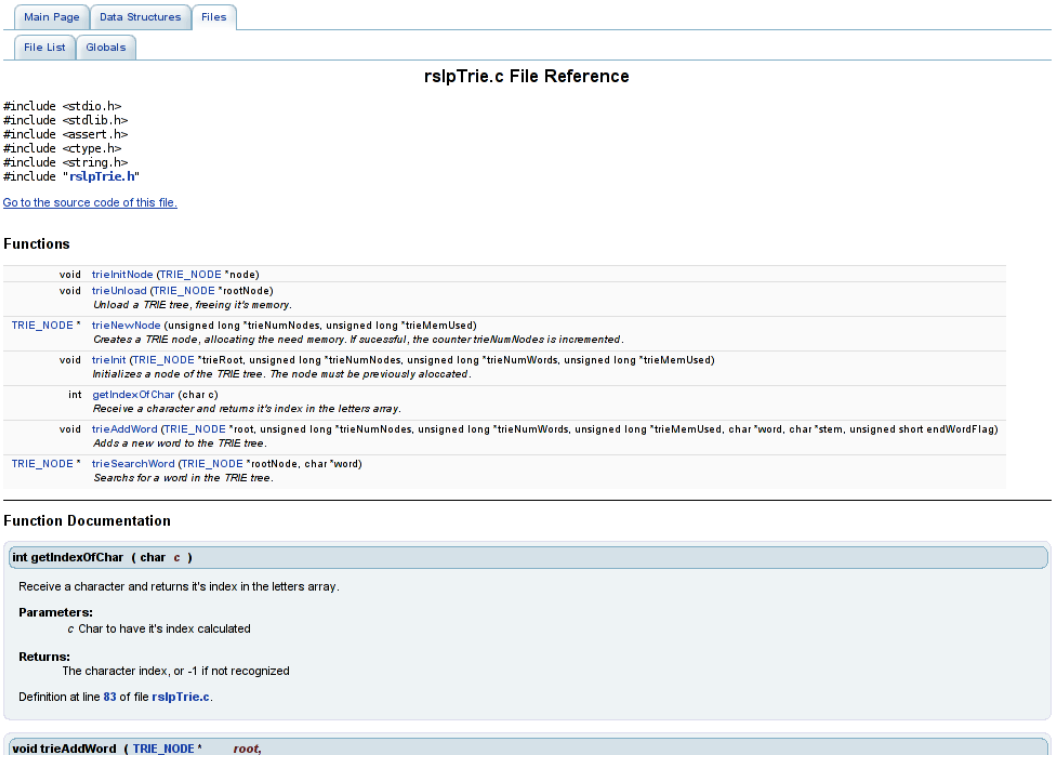

Figura 3.3: Exemplo de documentação do RSLP.

[11] Disponível em http://www.stack.nl/~dimitri/doxygen/. Acesso em 22 out. 2006.

3.3.2 Reorganização da implementação

Elegeu-se, como premissa inicial desta nova versão do RSLP, seu uso somente como *stemmer*, retirando-se o código das tarefas de extrair palavras dos documentos processados, como era feito na versão original. Para simplificar seu uso, foram criadas 3 funções principais:

a) **rslpLoadStemmer**: função que trata da configuração de todas as opções do *stemmer*, carga das definições dos passos de redução e suas regras, bem como demais procedimentos necessários para o seu funcionamento. É obrigatória sua chamada antes de qualquer chamada às outras funções do *stemmer*.

b) **rslpProcessWord:** função principal da nova implementação do RSLP, tem a tarefa de receber uma palavra e retornar o seu *stem*. Esta função tem essencialmente o mesmo intuito da implementação original do RSLP, aplicando os passos de redução sobre uma palavra.

c) **rslpUnloadStemmer**: função responsável por liberar toda a memória alocada para uso do RSLP (dicionários, passos, regras e exceções). Deve ser executada no encerramento do programa, quando as funções de *stemming* não serão mais utilizadas.

Somente estas funções são utilizadas pelo usuário do RSLP, todas as demais contidas no código são utilizadas internamente pelo *stemmer*. Além disso, para facilitar a integração da nova implementação do RSLP com outros programas e sistemas, implementou-se o código de forma similar a uma "biblioteca", bastando ao utilizador incluir o código-fonte do RSLP no seu próprio código (através da diretiva *#include* da linguagem C) e compilá-los juntos para que o uso do *stemmer* esteja disponível.

3.3.3 Carregamento de regras e exceções a partir de arquivo

Como já visto na descrição da implementação original do RSLP, o fato das regras e exceções serem declaradas na forma de um vetor dentro de cada função de redução causa grande impacto no desempenho do sistema. Tendo isto em vista, decidiu-se por manter toda a estrutura de regras e suas exceções carregadas em memória até o final da execução do código do programa.

Com o intuito de adicionar flexibilidade ao programa, evitando que modificações nas regras exigissem nova compilação do código-fonte do *stemmer*, implementou-se o carregamento destas e de suas definições a partir de um arquivo externo. Assim, na chamada da função de carregamento do *stemmer* (*rslpLoadStemmer*), todo o conjunto de regras e passos é lido de um arquivo indicado na configuração do RSLP e carregado para a memória, estando então disponível para acesso por parte das funções que aplicam as reduções já vistas.

O formato do arquivo com a declaração de passos e regras é mostrado na Figura 3.4:

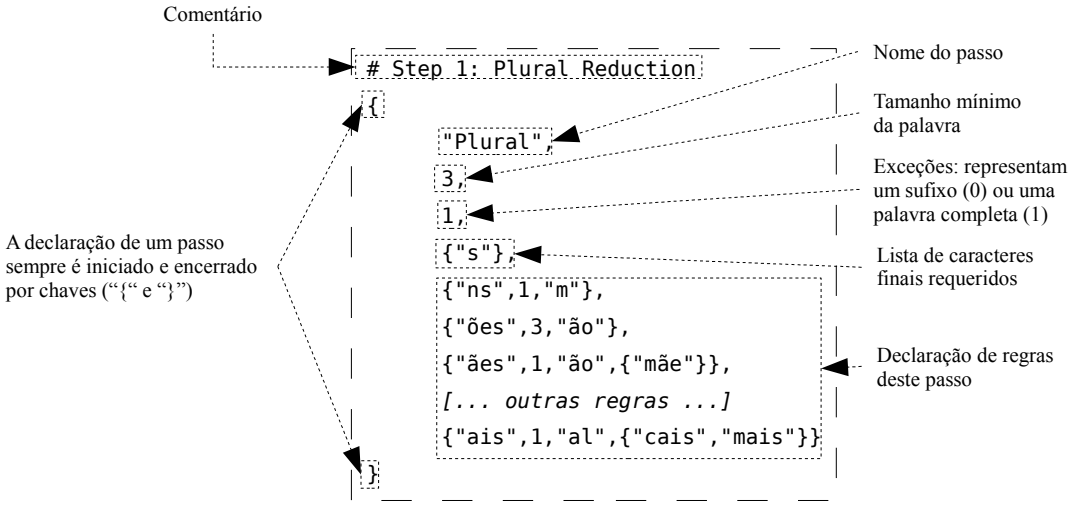

Figura 3.4: Formato do arquivo de passos e regras.

A declaração das regras permanece como mostrado na Figura 2.1, com a diferença que cada regra deve estar dentro de chaves, e as regras dentro de um passo separam-se por vírgulas. Todos os passos de redução devem estar declarados neste arquivo, com exceção do passo de Remoção de Acentos, que não possui regras ou exceções.

3.3.4 Dicionário de *stems*

Um dos objetivos deste trabalho é melhorar também o desempenho de processamento do RSLP. Constatou-se que um dos pontos críticos em termos de tempo de processamento era a repetição de palavras que já haviam sido previamente submetidas ao algoritmo e devidamente processadas. Deve-se lembrar que, dentro de um mesmo parágrafo, ou até dentro da uma mesma frase, há repetição de palavras. Projetando essa idéia em uma coleção completa de textos, claramente nota-se que uma mesma palavra seria encontrada várias vezes neste coleção, o que implicaria várias repetições de processamento dela ao longo da execução do algoritmo. Uma vez que um termo é reduzido e seu *stem* é obtido, não há necessidade de se executar todo o procedimento de redução novamente neste termo, pois obviamente o *stem* obtido a partir de uma nova redução será igual ao primeiro.

Partindo desta constatação, decidiu-se adicionar um dicionário de palavras ao RSLP, com a finalidade de armazenar palavras previamente processadas e os respectivos *stems* obtidos deste processamento. Desta forma, quando o algoritmo recebe uma palavra para ser processada, antes de submetê-la ao passos de redução, é verificada sua presença neste dicionário. Se a palavra for encontrada, obtém-se seu *stem* (também armazenado no dicionário) e encerra-se o seu processamento. Caso não seja encontrada, a palavra segue o fluxo normal do algoritmo, sendo submetida normalmente aos passos de redução. Quando as reduções são encerradas, a palavra é então adicionada ao dicionário e o *stem* obtido também é armazenado, associado a ela. Assim, caso esta palavra seja novamente encontrada no texto, ela já existirá no dicionário, evitando seu reprocessamento.

O uso do dicionário de *stems* é opcional, podendo ser ativado ou desativado no arquivo de configuração do *stemmer*. Além disso, pode ser configurado também seu tamanho máximo, em termos de memória ocupada, para evitar o uso exagerado de recursos de memória no sistema. Neste caso, quando o tamanho máximo do dicionário é

excedido, novas palavras não são mais inseridas nele. Cabe lembrar também que este dicionário não é persistente, sendo criado à cada nova execução do *stemmer,* e descarregado em seu encerramento.

3.3.4.1 Árvores TRIE

O dicionário de *stems* foi implementado através de uma árvore TRIE (o nome vem de *reTRIEval* em inglês, ou "recuperação" em português). As árvores TRIE, também conhecidas como "árvores de prefixos", são estruturas excelentes para armazenar *strings* (TRIES, 1999), por isso foram escolhidas para implementação dos dicionários. Seu funcionamento é simples:

a) primeiro, é definido um alfabeto de símbolos, necessário para armazenar os dados desejados. Para um dicionário, utilizamos como alfabeto as letras "a" até "z", já que desejamos armazenar palavras (HAYDEN, 1997).

b) define-se então a estrutura de um nodo desta árvore. Temos um vetor do tamanho do alfabeto utilizado, onde cada posição do vetor é uma ponteiro para um novo nodo, e corresponde a um símbolo do alfabeto. Desta forma, em um dicionário normal, cada nodo pode ter até 26 filhos (número de letras do alfabeto).

c) há também um indicador de fim de palavra, que pode ser definido como um valor booleano. Este indicador tem a função de sinalizar que o nodo analisado corresponde ao último símbolo de uma palavra, ou seja, todo o caminho percorrido da raiz da árvore até este nodo representa uma palavra completa.

A Figura 3.5 mostra detalhadamente a configuração de uma destas árvores, quando da inserção das palavras "ar", "arma", "iam" e "iate". São ilustrados os vetores com o alfabeto em cada nodo, os ponteiros entro nodos e o indicador de fim de palavra (mostrado na forma de um círculo com um "x" dentro).

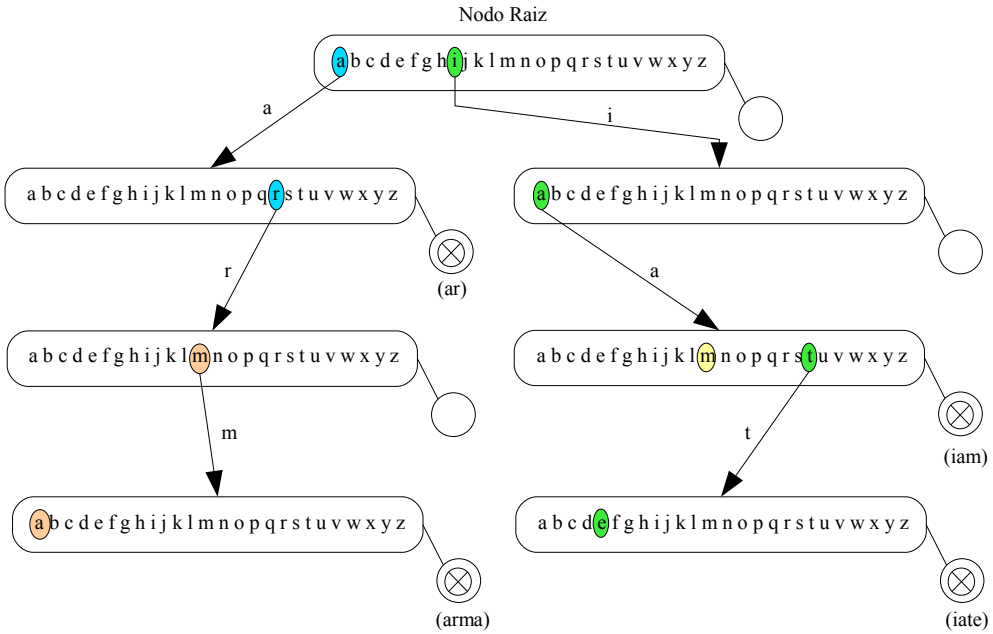

Figura 3.5: Exemplo de estrutura de uma árvore TRIE.

O nodo raiz conterá a primeira letra de todas as palavras. Uma letra, que corresponde a uma posição no vetor do alfabeto, aponta para um novo nodo na árvore, que conterá a letra seguinte na palavra, e assim por diante, até a última letra da palavra. O final da palavra é indicado no nodo, ativando-se o indicador de fim de palavra. Nota-se que um nodo marcado como final de palavra ainda aponta para outros nodos, dando seqüência a outra palavra. Os prefixos comuns (como "*ar*" em "*ar*" e "*ar*ma") são compartilhados entre as palavras, o que facilita e agiliza a busca de palavras. De fato, o tempo de busca em uma árvore TRIE para uma palavra de tamanho *m*, no pior caso, é O(*m*).

3.3.4.2 Implementação TRIE no RSLP

Devido a algumas peculiaridades do RSLP, a implementação TRIE utilizada é um pouco diferente da mostrada anteriormente. Para uso no RSLP, modificou-se a estrutura básica de uma árvore TRIE, como mostrado no trecho de código apresentado na Figura 3.6.

```
01: /**
02:  * \brief    Typedef for a structure with a node for a TRIE tree
03:  */
04: typedef struct trieNode TRIE_NODE;
05:
06: /**
07:  * \brief    Structure with a node for a TRIE tree
08:  */
09: struct trieNode {
10:     TRIE_NODE* nodes[RSLP_TRIE_NUM_LETTERS];
11:     unsigned short endWord;
12:     char *stem;
13: };
```

Figura 3.6: Estrutura de uma árvore TRIE no RSLP.

a) o vetor com o alfabeto ("*nodes"* na estrutura "*trieNode",* Figura 3.6) foi estendido, pois originalmente somente eram previstas as 26 letras do alfabeto. Nesta implementação, foram adicionados caracteres com acentuação, totalizando um total de 57 posições no novo vetor (por conveniência, foram adicionados também símbolos de outras línguas, conforme tabela de símbolos ISO-8859-1[12]). Assim, as posições 0 a 26 do vetor correspondem aos caracteres "a" até "z" da tabela ASCII, e as posições 27 em diante correspondem aos caracteres a partir de "à" na tabela ISO-8859-1.

b) o nodo prevê também um indicador de final de palavra (*endWord* na estrutura *trieNode*), que, além de ser usado para indicar se o nodo em questão representa o último de uma palavra, contém informações extras sobre esta palavra a serem utilizadas internamente pelo *stemmer*.

c) foi adicionado um ponteiro para *strings* (campo *stem* da estrutura *trieNode*), para armazenar no nodo o *stem* correspondente a esta palavra (caso o nodo esteja marcado como final de palavra).

Além da estrutura de uma árvore, também foram implementadas no RSLP suas funções básicas, como inserção e busca de uma palavra na árvore. a partir destas funções, foram implementadas as funções de tratamento do dicionário. Estas são genéricas o suficiente para serem utilizadas para outros fins além da recuperação de *stems*, como será visto mais adiante.

3.3.5 Dicionário de Nomes Próprios

Com o intuito de evitar o *stemming* de nomes próprios, que foi uma das dificuldades apontadas por Orengo e Huyck (2001), adicionou-se um segundo dicionário à nova implementação do RSLP. Neste dicionário são armazenados nomes próprios, carregados a partir de uma lista de nomes em um arquivo externo, criado pelo usuário. Para implementação deste dicionário foram utilizadas as mesmas estruturas e funções desenvolvidas para tratamento do dicionário de *stems*. O arquivo com a lista de nomes próprio é apenas um arquivo texto comum, com os nomes que se deseja utilizar no dicionário separados por espaços ou linhas.

Sempre que uma palavra a ser processada pelo *stemmer* inicia por letras maiúsculas, é verificada sua presença no dicionário de nomes próprios. Caso a palavra seja encontrada neste dicionário, o algoritmo passa diretamente à função de remoção de acentos, ou seja, não é aplicado qualquer passo de redução sobre esta palavra. Caso contrário, a palavra é processada normalmente. Com isto, pretende-se melhorar a precisão de respostas dos sistemas, uma vez que nomes próprios são armazenados no índice de busca em sua forma original (e não seus supostos *stems*), evitando-se sua conflação com termos não relacionados.

Em tempo, cabe salientar que o uso deste dicionário, assim como o dicionário de *stems*, é opcional, devendo o utilizador indicar no arquivo de configuração do *stemmer* que deseja usá-lo e indicar o arquivo com os nomes próprios. Da mesma forma, é possível configurar o tamanho máximo de memória que o dicionário pode utilizar.

[12] Tabela completa disponível em <http://htmlhelp.com/reference/charset/>. Acesso em: 10 jun. 2007.

3.3.6 Configuração do fluxo de execução

O algoritmo original de Orengo e Huyck (2001) propunha um fluxo dos passos de redução diferente do efetivamente implementado pelos autores, como já apresentado neste trabalho (Figura 2.2 e Figura 3.1). A mudança entre o fluxo proposto e o implementado decorreu do fato dos autores perceberem que poderia haver uma melhoria no processamento das palavras, caso a ordem as reduções fosse alterada.

Com base nesta idéia, e visando dar ao usuário a flexibilidade de decidir quais reduções aplicar e em que ordem aplicá-las, bem como poder testar qual seqüência traz melhores resultados, optou-se por implementar o fluxo de execução das reduções de forma modificável. Este fluxo pode então ser definido em um arquivo externo, seguindo uma estrutura simples, que é composta por linhas de 3 colunas, separadas por vírgulas. Cada linha corresponde a descrição do comportamento de um dos passos de redução. A descrição das colunas segue abaixo:

a) **coluna 1:** o nome do passo redução a ser executado. Este nome deve ser o mesmo utilizado no arquivo de definição de passos e regras (Figura 3.4).

b) **coluna 2:** o nome do passo a ser executado caso o passo da coluna 1 seja executado, removendo o sufixo da palavra. Equivale à opção "verdadeiro" de um teste "sufixo removido?", ou seja, lê-se como: "se o passo definido na coluna 1 foi executado, então executar o passo da coluna 2". Desta forma, o programa passa a executar a linha de fluxo que define o passo da coluna 2.

c) **coluna 3:** o passo a ser executado caso o passo da coluna 1 não seja executado. Equivale à resposta "falso" ao teste "sufixo removido?" proposto acima. Assim, o fluxo do programa segue na linha que define o passo da coluna 3.

Caso o fluxo de execução deva encerrar-se quando terminado o passo de redução atual, deve-se indicar, nas colunas "verdadeiro" ou "falso" (conforme o caso), com a palavra "NULL".

Por exemplo, o fluxograma mostrado na Figura 3.1 é representado no formato proposto da seguinte forma:

```
01: Plural       , Adverb       , Adverb
02: Adverb       , Feminine     , Feminine
03: Feminine     , Augmentative , Augmentative
04: Augmentative , Noun         , Noun
05: Noun         , NULL         , Verb
06: Verb         , NULL         , Vowel
07: Vowel        , NULL         , NULL
```

Figura 3.7: Arquivo de fluxo dos passos de redução do RSLP.

Neste exemplo, caso o passo de redução do plural seja executado com sucesso (remova um sufixo da palavra), o programa segue o fluxo definido pela linha que define o passo "Adverb", executando então esta redução. Em tempo, caso a redução de plural não remova o sufixo da palavra, o fluxo também passa para a linha de "Adverb", pois este passo foi declarado em ambas as colunas. Um comportamento diferente observa-se na execução do passo "Noun" (redução nominal): se este remove o sufixo da palavra, o

fluxo de execução se encerra, pois está definido “NULL” na coluna 2. Porém, caso não remova sufixo da palavra, o fluxo prossegue para a linha que define o passo “Verb” (redução verbal).

3.3.7 Configuração das funcionalidades do *stemmer*

Optou-se por controlar o comportamento das novas funcionalidades adicionadas ao RSLP através de um arquivo de configuração, permitindo assim que a forma de processar as palavras seja facilmente alterada, o que é desejável para a realização de testes. Desta forma, pode-se indexar uma coleção de várias formas diferentes (com ou sem remoção de acentos, usando ou não dicionários, etc.), sem a necessidade de recompilar ou alterar o código do *stemmer*.

O arquivo de configuração pode ser utilizado para controlar os seguintes comportamentos do RSLP:

a) uso de dicionários: através do arquivo de configuração, é possível ativar ou desativar o uso dos dicionários de *stems* e de nomes próprios (separadamente), bem como limitar a quantidade máxima de memória que cada um deles pode ocupar. Para o caso do dicionário de nomes próprios, é possível também informar o nome do arquivo que contém a lista de nomes que irão compor o dicionário.

b) remoção de acentos: é possível ativar ou desativar a remoção de acentos. Esta funcionalidade foi adicionada basicamente para testes.

c) indicar o arquivo de passos e regras, para seu carregamento na inicialização do *stemmer*.

d) indicar o arquivo de fluxo de execução de passos, caso o usuário deseje usar um fluxo diferente do padrão.

A maioria das funcionalidades mostradas são de uso opcional, exceto a indicação do arquivo de passos e regras, que é necessária para o funcionamento do algoritmo. Por este motivo, o uso do arquivo de configuração é obrigatório, sendo que o RSLP está programado para sempre buscar o arquivo chamado “*rslpconfig.txt*” na pasta em que está sendo executado. Caso este arquivo não seja encontrado, o *stemmer* não poderá ser utilizado.

A configuração propriamente dita das funcionalidades se dá declarando uma variável de controle e seu valor, como por exemplo:

```
USE_STEM_DICTIONARY=YES
```

A variável mostrada controla no exemplo o uso do dicionário de *stems*, e está sendo configurada para ativar o uso deste dicionário. Consta no Apêndice C uma listagem de todas as variáveis de configuração e seus valores possíveis.

3.4 Complexidade do Algoritmo

Na implementação original do RSLP, basicamente são realizados os seguintes procedimentos:

a) uma palavra é extraída do texto (pelo próprio RSLP);

b) verifica-se a última letra da palavra, conforme o passo: letra “s” para o passo de redução do plural, letras “a” e “ã” para o passo de redução o feminino. Os demais passos não verificam a letra final.

c) se a palavra termina com a letra especificada no item “b”, é executada a função de redução do passo em questão. Nesta função, para cada regra do passo é verificado simultaneamente:

- se a quantidade de caracteres restantes na palavra após a remoção do sufixo é igual ou maior a informada na regra;
- se a palavra original termina com o sufixo atual;
- se a palavra original não consta na lista de exceções desta regra, caso essa exista;

d) se a palavra atendeu a todas as verificações do item “c”, o sufixo é removido da palavra original. Se a regra possuir um sufixo de reposição, este é adicionado à palavra processada. Porém, se a palavra original não atendeu a qualquer uma das verificações do item “c”, o sufixo em questão é rejeitado, passando-se para o próximo da lista, se existir. Se não houver sufixo restante, este passo se encerra e programa segue para o próximo passo.

Analisando-se esta seqüência de eventos, pode-se concluir que a complexidade do algoritmo depende diretamente:

a) do número de palavras do *corpus*;

b) do número de passos (que são 8, portanto constantes);

c) da quantidade de regras definidas;

d) da quantidade de exceções definidas na lista de cada regra;

e) do tamanho original de cada palavra;

Assim, para cada palavra, considerando um determinado passo *i*, para a análise da regra *j* temos que comparar o sufixo e verificar a possibilidade desta palavra constar em uma lista de exceções, que varia conforme a regra analisada.

Desta forma, a complexidade para o processamento de *n* palavras, é obtida pela equação

$$n \,.\, |W| \,.\, \sum_{i=1}^{p} \left(\sum_{j=1}^{|R_i|} \left(|S_{ij}| \,.\, |E_{ij}| \right) \right) \tag{3.1}$$

onde:

n representa o número de palavras processadas;

$|W|$ é o tamanho, em caracteres, da maior palavra do texto de entrada (pior caso);
p representa o número total de passos.
R_i o número de regras para o passo *i*;
$|S_{ij}|$ é o tamanho, em caracteres, do sufixo da regra *j* do passo *i;*
$|E_{ij}|$ é o tamanho da lista de exceções da regra *j* do passo *i* (pior caso);

Porém, se considerarmos que uma palavra pode, no pior caso, ser verificada por todas as regras, independente do passo, podemos então considerar a lista de regras como um todo. Neste caso podemos simplificar a equação 3.1, pois o somatório mais externo da expressão torna-se desnecessário. Teríamos então a equação

$$n.|W|\sum_{i=1}^{|R|}(|S_i|.|E_i|) \tag{3.2}$$

onde:

$|R|$ indica o número total de regras do algoritmo;
$|S_i|$ é o tamanho, em caracteres, do sufixo da regra *i;*
$|E_i|$ é o tamanho da lista de exceções da regra *i.*

A complexidade do algoritmo pode ser simplificada para

$$O(nW) \tag{3.3}$$

pois o número de regras é fixo (242), assim como o tamanho em caracteres do maior sufixo (10 caracteres) e o número de exceções (43 na maior lista).

Como todos os valores desta expressão são constantes, não há alterações significativas a serem efetuadas no algoritmo que possam reduzir sua complexidade, já que não podemos controlar o tamanho das palavras de entrada, bem como não podemos alterar o tamanho dos sufixos e das exceções sem prejudicar o resultado do *stemmer*. As poucas alterações que podem ser realizadas restringem-se em otimizar a forma em que a comparação da palavra com os sufixos e com as exceções é feita, além de ordenar alfabeticamente as exceções dentro de cada regra, o que diminuiria o tempo de busca de uma exceção dentro da lista de exceções.

Na nova implementação do RSLP, a complexidade pessimista aumentou, pois o uso de dicionários trouxe novas variáveis a serem consideradas. Lembrando que, sempre que uma palavra é passada ao *stemmer* para ser processada, há ao menos uma consulta ao dicionário de *stems* para verificar se esta palavra consta no dicionário. De forma similar, quando utilizamos o dicionário de nomes próprios, há também uma consulta a este dicionário antes do processamento da palavra. Recordando que uma consulta ao dicionário tem o custo equivalente ao tamanho da palavra (representado por $|W|$), a Equação 3.2 pode ser reescrita como

$$n.|W|^3\sum_{i=1}^{|R|}(|S_i|.|E_i|) \tag{3.4}$$

pois temos duas novas varreduras da palavra completa. Da mesma forma, a complexidade pessimista para o novo algoritmo pode ser reescrita como

$$O(nW^3) \tag{3.5}$$

Apesar da complexidade pessimista ter piorado, espera-se que as funcionalidades que causaram esse aumento tenham, na verdade, trazido uma melhoria no resultado final em termos de desempenho e precisão. Isto será verificado em um próximo capítulo, onde serão realizados experimentos com o *stemme*r.

3.5 Resumo do capítulo

Neste capítulo foram apresentadas as modificações realizadas sobre o algoritmo original do RSLP. Com estas modificações, espera-se melhorar o desempenho de processamento do *stemmer* (com a adição da possibilidade de uso de dicionários, por exemplo), facilitar a integração com o códigos de outros programas (com a documentação do código detalhada e reescrita das funções utilizadas).

Para verificar se houve efetivamente melhorias, esta nova implementação do RSLP será integrada a um Sistema de Recuperação de Informação, como será mostrado no próximo capítulo, para que então se realizem experimentos e medições do desempenho do algoritmo. Estes experimentos e seus resultados serão mostrados no 5º capítulo.

4 INTEGRAÇÃO DO RSLP COM UM SISTEMA DE RI

Com o código do RSLP reescrito, partiu-se então para sua utilização com uma aplicação real. O objetivo é integrá-lo a um Sistema de Recuperação de Informação, como proposto em Orengo e Huyck (2001), permitindo assim a realização de experimentos com o *stemmer,* para avaliação dos resultados obtidos.

Para uso neste trabalho, foi escolhido o SRI Zettair, desenvolvido pelo *Search Engine Group* da RMIT University, na Austrália (SEARCH ENGINE GROUP, 2006).

4.1 Motivação da escolha do sistema

Vários motivos levaram a escolha do Zettair como Sistema de Recuperação de Informação a ter o RSLP integrado, entre os quais se destacam:

a) código-fonte aberto e bem documentado.

b) implementação em linguagem C, a mesma utilizada para escrever o RSLP, facilitando assim a integração dos dois programas.

c) organização do sistema: as várias partes do sistemas foram escritas de forma modular (cada parte principal do sistema em um arquivo fonte separado), de forma a facilitar a modificação por parte do usuário e a integração de novos módulos.

d) suporte nativo a experimentos TREC, que são de extrema valia para avaliar os resultados obtidos com o RSLP.

e) já possui 3 *stemmers* implementados internamente, todos específicos para a língua inglesa, já existindo um módulo escrito com o código de todos os algoritmos de *stemming*, bastando então adicionar o código do RSLP neste módulo. A utilização de *stemmin*g é opcional, tanto na indexação de textos quanto na busca, sendo configurável pelo usuário através de uma opção específica na chamada ao programa. Os *stemmers* internos são os seguintes:

   - uma implementação do algoritmo de Porter.
   - algoritmo chamado pelo sistema de “eds”, que somente remove os sufixos “e”, “ed” e “s” das palavras.
   - algoritmo chamado de “*light*”, que remove alguns sufixos bastante comuns na língua inglesa.

f) excelente desempenho em termos de tempo necessário para indexação ou recuperação de informações, bem como em termos de resultados de buscas, como verificado em duas disciplinas ministradas pela orientadora deste trabalho.

4.2 Descrição do Sistema Zettair

Para indexação de documentos no Zettair, deve-se informar os documentos a serem indexados, o *stemmer* a ser utilizado (se for o caso), e o nome do arquivo-índice a ser gerado. Após gerado o índice, é possível então realizar consultas sobre o mesmo, através do modo "interativo" do sistema.

Neste caso, executa-se o sistema de forma similar a de indexação, com a diferença de se indicar qual índice sera usado. O sistema então apresenta um *prompt*, no qual o usuário pode entrar sua consulta, submetê-la e receber o resultado do sistema imediatamente. A consulta submetida retorna uma lista de resultados, cada um informando onde ocorrem os termos da consulta na coleção de documentos, bem como a sua pontuação. Esta pontuação, que varia de 0 a 1, indica o quão relevante aquele resultado é para a consulta.

Este procedimento de consulta ao sistema pode ser automatizado, utilizando-se o modo "não interativo" do Zettair, onde informa-se o nome de um arquivo de consultas na execução do programa e demais parâmetros necessários, e o sistema retorna os resultados[13].

4.3 Integrando código externo ao Zettair

Para integrar o código-fonte do RSLP ao Zettair, foram necessárias algumas modificações no sistema, basicamente:

a) inserção de uma nova função de *stemming* no código original, e sua declaração no arquivos adequados no Zettair.

b) adicionar os arquivos com o código-fonte do RSLP na estrutura de arquivos do Zettair.

c) informar, no *Makefile* do Zettair, a presença de novos arquivos com código-fonte, para que fossem compilados com o sistema.

As alterações citadas são detalhadas no Apêndice B.

4.3.1 Modificações extras no Zettair

Como o Zettair é originalmente desenvolvido para a língua inglesa, durante os testes realizados verificaram-se alguns problemas no tratamento das palavras em português, que não ocorrem na língua original para a qual o sistema foi projetado.

O primeiro problema encontrado foi na forma que o sistema retira os termos dos documentos, pois todos os caracteres acentuados do texto são ignorados, uma vez que a língua inglesa não possui acentos. Assim, palavras acentuadas, como "açúcar" ou "café", são passadas ao *stemmer* como "acar" e "caf", respectivamente. A ausência de

[13] Informações detalhadas do uso do Zettair, tanto para indexação quanto para consultas, podem ser consultadas em <http://www.seg.rmit.edu.au/zettair/zettair/doc/Readme.html>. Acesso em 10 out. 2006.

caracteres acentuados nas palavras passadas ao RSLP para processamento é inadmissível, pois as descaracteriza completamente e prejudica todo o processo de *stemming*.

Neste caso, vários sufixos acentuados da lista de passos não seriam encontrados nas palavras descaracterizadas. Foi necessário modificar a função responsável pela leitura de palavras dos documentos, reescrevendo-a para que aceitasse caracteres acentuados, adicionando-os à palavra lida e não ignorando-os como era feito no código original. Para tanto, foi definido manualmente um conjunto de caracteres acentuados, em conformidade com a tabela ISO-8859-1 de caracteres internacionais.

Um segundo problema foi verificado, onde o hífen ("-") era ignorado em palavras compostas, juntando-as como se fossem uma só. Assim, por exemplo, a expressão "lembra-se" era passada ao *stemmer* como uma palavra só, resultante da composição das duas palavras da expressão ("lembrase"). Esta "palavra" resultante, além de não existir na língua portuguesa, geraria um *stem* completamente diferente do que seria gerado caso as palavras da expressão ("lembra" e "se") fossem processadas pelo *stemmer* separadamente. Este comportamento foi corrigido, fazendo com as palavras componentes de expressões com hífen fossem consideradas separadamente.

Devido a implementação do dicionário de nomes próprios no *stemmer*, o Zettair foi modificado para não alterar automaticamente a capitalização[14] das palavras lidas dos documentos, passando-as na sua forma original para o RSLP. Assim é possível verificar internamente no *stemmer* se a palavra a ser processada inicia por letras maiúsculas, indicando a possibilidade de tratar-se de um nome próprio e dando-lhe o devido tratamento.

4.4 Resumo do Capítulo

Neste capítulo foi brevemente introduzido o Sistema de Recuperação de Informação Zettair e a motivação que o levou a ser escolhido para ter o RSLP integrado. Com esta integração, será possível, através da realização de experimentos, verificar os resultados obtidos com o *stemmer*, como será mostrado no próximo capítulo deste trabalho.

[14] Alterar a escrita de letras maiúsculas para minúsculas, e vice-versa.

5 EXPERIMENTOS E RESULTADOS

Neste capítulo serão apresentados os resultados de testes de desempenho e precisão, obtidos sobre o algoritmo RSLP modificado apresentado neste trabalho. Com o novo código do RSLP integrado ao Zettair, e com os problemas surgidos da sua adaptação à língua portuguesa devidamente resolvidos, utilizou-se da combinação Zettair/RSLP a realização destes testes, indexando-se uma coleção de exemplo e submetendo uma série de consultas a mesma.

5.1 Metodologia de Testes

Para todos os testes aqui mostrados utilizou-se um *notebook* Dell D520, equipado com processador Celeron M420 de *clock* de 1,60Ghz e 512 *megabytes* de memória RAM.

As avaliações de desempenho apresentadas no item 5.2 foram realizadas em ambiente Windows XP, com os códigos-fonte de ambas as versões do RSLP (original e modificada) compilados através do *Borland C Compiler*.

As avaliações utilizando o Zettair foram todas realizadas com sistema operacional Ubuntu (Linux) 6.10, com os códigos-fonte compilados com o *GCC* versão 4.1.2. O ambiente foi configurado para utilizar como idioma o português brasileiro (código *PT_BR*) e caracteres da tabela ISO-8859-1. O uso desta tabela de caracteres é imprescindível para estes experimentos, uma vez que quando usado o padrão do sistema (UTF-8), os caracteres acentuados não são devidamente reconhecidos pelo RSLP, inviabilizando completamente o experimento.

5.1.1 Coleções de Testes

Como *corpus* para a realização dos experimentos, foram utilizadas as mesmas coleções utilizadas nas campanhas do CLEF para a língua portuguesa (PT), contendo documentos com notícias publicadas nos jornais “O Público” e “Folha de São Paulo”, nos anos de 1994 e 1995. Estas coleção consistem em vários arquivos, cada um com vários documentos previamente formatados para utilização em experimentos TREC. Exemplos de documentos neste formato podem ser vistos no Anexo A.

Uma observação: os documentos da coleção “Público” são escritos em português de Portugal (“português europeu”), que possui algumas diferenças na grafia das palavras em relação ao português brasileiro, dificultando o funcionamento do RSLP, uma vez que os sufixos e suas exceções foram escritas na forma do português brasileiro. Um resumo de ambas as coleções consta na Tabela 5.1.

Tabela 5.1: Dados do *corpus* utilizado nos experimentos

Coleções		Público	Folha de São Paulo
Anos		1994-1995	1994-1995
Edições		726	730
Documentos		106821	103913
Tamanho		348.078 kB	226.690 kB
Unidades	Total	64222797	42109286
	Diferentes	500197	426469
Palavras	Total	54947072	35699765
	Diferentes	472817	393885

5.1.2 Indexação das Coleções

A indexação das coleções, realizada através do Zettair, foi feita separadamente para cada variante do algoritmo, trabalhando-se assim com vários arquivos-índice diferentes, um para cada variante. Na criação dos índices, deve-se informar ao Zettair que os documentos a serem indexados já estão no formato TREC, adicionando-se a opção "**-t TREC**" na chamada ao programa, garantindo assim que todos os documentos condensados em um único arquivo sejam corretamente extraídos e processados.

Com o intuito de simular as campanhas CLEF de 2005 e 2006, as duas coleções de documentos foram utilizados para indexação. Para a simulação da campanha de 2004, porém, usou-se somente a coleção referente ao jornal "O Público", que era a única utilizada neste ano (documentos da Folha de São Paulo somente seriam incorporados a coleção a partir da campanha de 2005). Esta é uma observação importante, pois se fossem indexados todos os documentos do *corpus* para os testes da campanha de 2004, os resultados observados seriam mais baixos do que o esperado, devido ao número de documentos não-relevantes introduzidos. Deve-se lembrar que o arquivo de respostas para o ano de 2004 somente previa documentos do Público como relevantes, portanto quaisquer outros documentos que não pertencessem a esta coleção seriam considerados não-relevantes, mesmo que não o fossem.

5.1.3 Avaliação de Resultados

Nas avaliações mostradas neste capítulo foram utilizadas as consultas utilizadas no CLEF de 2004, 2005 e 2006. Estas consultas, também chamadas de "tópicos", estão agregadas em arquivos, separados por ano da campanha, já no formato utilizado por experimentos TREC. Um exemplo de um arquivo de tópicos pode ser visto no Anexo B. Nestes experimentos, as consultas são aplicadas aos índices utilizando-se o programa **zet_trec**, integrante do Zettair, que recebe como entrada um arquivo de tópicos no formato TREC, e gera uma saída pronta para ser processada pelo utilitário **trec_eval**[15], utilizado para avaliação dos resultados. Nos testes realizados neste trabalho, utilizou-se somente o título e a descrição do tópico na consulta (*tags* <*title*> e <*desc*>, respectivamente). Além disso, o utilitário foi configurado para utilizar a métrica Okapi BM25[16].

[15] Disponível em <http://trec.nist.gov/trec_eval/>.

Para a avaliação propriamente dita dos resultados das consultas, utilizou-se o utilitário **trec_eval** e os arquivos de resposta para cada ano da campanha (também conhecidos como "qrels"). Estes arquivos consistem em uma listagem com o número do tópico, uma série de códigos de documentos para cada tópico e a relevância de cada um destes documentos para esta consulta. Esta listagem é gerada pela comissão julgadora da campanha, que avalia se cada documento é ou não relevante como resposta àquela consulta. O Anexo C mostra um trecho de um destes arquivos de resposta.

5.1.4 Base de Nomes Próprios

Os nomes próprios utilizados como base para o dicionário de nomes próprios foram retirados da própria coleção. Com o auxílio de um *script*, todas as palavras que iniciam por letras maiúsculas foram colocadas em um arquivo, e depois refinadas manualmente (retirada de palavras que não eram nomes próprios, etc.). O arquivo com os nomes está disponível *online*[17].

5.2 Avaliação de Desempenho de Processamento

Este experimento tem o objetivo de verificar o quanto o desempenho de processamento do *stemmer* melhorou. Para tanto, utilizou-se o algoritmo original do RSLP, e outra versão especialmente do mesmo algoritmo desenvolvida para este teste, onde as funções de *stemming* originais foram substituídas pelas do algoritmo aprimorado. Nesta versão de teste, nada além das funções de *stemming* foi alterado, mantendo-se todas as funções de leitura de arquivos, extração de palavras dos textos e escrita dos *stems* em arquivos idênticas entre as duas implementações do teste.

O experimento consistiu em medir o tempo necessário para processar toda o *corpus* de teste, utilizando-se as seguintes variantes do RSLP:

a) a implementação original do algoritmo RSLP (indicada por **RSLP-O**).

b) a versão aprimorada (objeto deste trabalho), sem a utilização de dicionário de *stems* e dicionário de nomes próprios, identificada por **RSLP-M1**.

c) a versão aprimorada, com o uso do dicionário de *stems* e do dicionário de nomes próprios, representada por **RSLP-M2**.

d) a versão aprimorada, somente com o uso do dicionário de *stems* ativado (o dicionário de nomes próprios permaneceu desativado), identificada por **RSLP-M3**.

Os resultados obtidos podem ser observados na Figura 5.1:

[16] Função de similaridade muito utilizada na avaliação de Sistemas de Recuperação de Informações. Definição disponível em <http://en.wikipedia.org/wiki/Probabilistic_relevance_model_(BM25)>. Acessado em 03 Jun. 2007.

[17] Disponível no endereço <http://www.inf.ufrgs.br/~arcoelho/rslp/nomes_proprios.txt>.

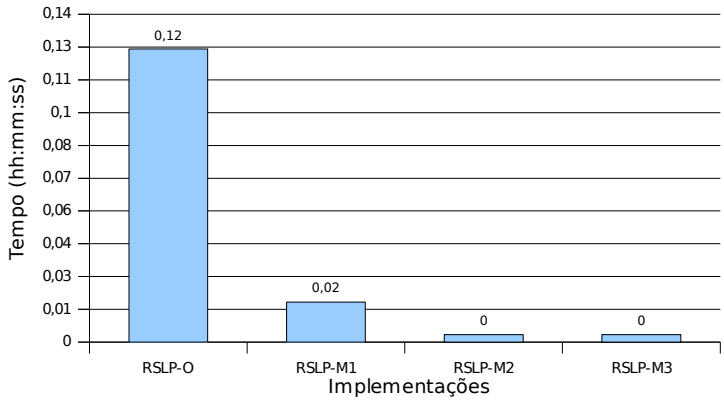

Figura 5.1: Tempo de processamento da coleção.

A partir da análise da figura, nota-se que o tempo necessário para processamento de toda coleção foi reduzido em aproximadamente 7 vezes com a nova implementação do RSLP, se comparada à original. Se o uso do dicionário de *stems* é ativado, este tempo se reduz em praticamente 40 vezes, o que representa um ganho de performance considerável. Aqui provou-se acertada a escolha do uso de dicionários para evitar o reprocessamento de palavras, o que diminuiu o tempo de processamento em mais de 5 vezes em relação à versão aprimorada do *stemmer* sem o uso deste dicionário (RSLP-M1).

Apesar do ganho de performance verificado, o uso do dicionário traz o ônus de aumentar drasticamente o consumo de memória do sistema. Desta forma, se executado em um sistema com poucos recursos de memória, e dependendo do tamanho da coleção processada, o dicionário de *stems* pode rapidamente ocupar toda a memória disponível. Com isto, pode ocorrer o uso freqüente de *swap* em disco para atender a demanda de memória, prejudicando o desempenho geral do sistema e possivelmente acarretando em um tempo ainda maior para processar os documentos do que o necessário caso o dicionário não estivesse sendo usado. Este problema pode ser contornado configurando o tamanho máximo de memória que pode ser ocupada pelo dicionário, como mostrado no capítulo 3.

5.3 Avaliação da Precisão do Resultado

Utilizando-se do sistema de recuperação de informações Zettair, já com o RSLP integrado, partiu-se para avaliação da precisão dos resultados obtidos com o *stemmer*. Para tanto, foram utilizadas as coleções de teste, o conjunto de consultas utilizadas nas campanhas do CLEF de 2004, 2005 e 2006 e seus respectivos resultados corretos, com o intuito de simular os resultados que seriam obtidos pelo sistema caso participasse das campanhas do CLEF.

O teste é realizado seguindo as seguintes etapas:

a) **Indexação da coleção:** a coleção completa é indexada pelo Zettair, alternando-se o uso de *stemmer* e seu tipo (RSLP aprimorado, com o uso do dicionário de nomes próprios). Além disso, alterna-se também o conjunto de passos de redução utilizados, sendo parte dos índices criados com o uso de todas os passos na seqüência normal (como visto na Figura

3.1) e outra parte o uso do passo de redução do plural somente. Todas estas configurações são realizadas no arquivo de configuração padrão do RSLP aprimorado e podem ser vistas no Apêndice C.

b) **Aplicação das consultas:** com a coleção indexada e os índices criados, o conjunto de consultas é aplicado a estes através do utilitário **zet_trec**, gerando a saída no formato esperado pelo programa **trec_eval**, que será utilizado para avaliar os resultados.

c) **Avaliação do resultados:** usando o programa **trec_eval**, os resultados na etapa anterior são confrontados com os resultados considerados corretos para aquele conjunto de consultas, que são listas de documentos da coleção que são considerados relevantes para cada consulta individual. Os documentos obtidos como resultado em nossos experimentos são procurados nestas listas de documentos relevantes, obtendo-se seus índices de relevância[18].

5.3.1 Avaliação das precisões médias

Entre as diversas medidas geradas pela avaliação do **trec_eval**, consta a MAP de cada consulta individual, bem como o seu valor para o conjunto de todas as consultas (mostrado como *MAP all* na saída gerada pelo **trec_eval**). Este último valor foi utilizado para comparar as precisões obtidas pelas diversas variações do *stemmer* testadas, como apresentado no item "a".

A Figura 5.2 mostra um gráfico comparativo, onde são apresentados os valores de MAP *(all)* obtidos no experimento como resultados das consultas utilizadas nas campanhas do CLEF de 2004, 2005 e 2006. Como citado no item "a", foram experimentadas diversas combinações entre o uso de *stemmer* e do dicionário de nomes próprios, para que fosse possível avaliar o impacto das diversas configurações nos resultados das consultas.

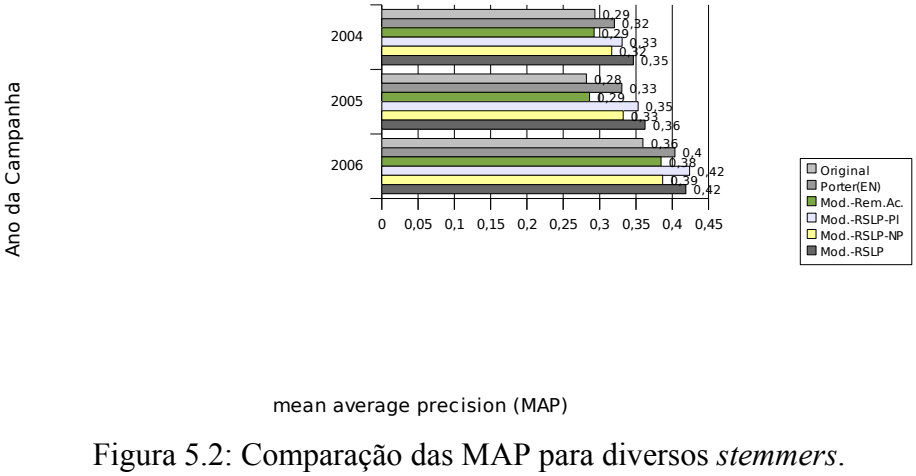

Figura 5.2: Comparação das MAP para diversos *stemmers*.

As combinações utilizadas foram as seguintes:

[18] Disponível em <http://ir.iit.edu/~dagr/cs529/files/project_files/trec_eval_desc.htm>. Acessado em 03 Jun. 2007.

a) **Original:** SRI Zettair sem modificação alguma, compilado diretamente do código-fonte original, e sem o uso de qualquer algoritmo de *stemming*.

b) **Porter(EN):** Zettair original (sem modificação alguma), utilizando como *stemmer* sua implementação interna do algoritmo de Porter para a língua inglesa. Estes valores foram medidos para se verificar o desempenho de um *stemmer* próprio para a língua inglesa, que é o mais usual em SRI, aplicado em um *corpus* composto de documentos em português. Cabe lembrar que uma das premissas deste trabalho é que tais *stemmers* não apresentariam resultados satisfatórios nestas condições, por isso a necessidade de um *stemmer* para português.

c) **Mod.-Rem.Ac.**: nestas medições foi utilizado o código do Zettair modificado para reconhecer acentos nas palavras, entre outras melhorias já apresentadas, mas não foi utilizado qualquer algoritmo de *stemming* sobre os textos da coleção. A intenção é verificar se as modificações introduzidas para adaptar o Zettair às palavras da língua portuguesa trouxeram algum benefício aos resultados do sistema.

d) **Mod.-RSLP-Pl:** algoritmo RSLP aprimorado, utilizando somente o passo de redução de plural. O objetivo é comparar o desempenho se somente o uso deste passo de redução apresenta já não é suficiente para se obter bons resultados de pesquisa.

e) **Mod.-RSLP-NP:** algoritmo RSLP aprimorado, utilizando todos passos redução e o dicionário de nomes próprios.

f) **Mod.-RSLP:** Zettair utilizando o RSLP aprimorado como *stemmer*, sem o uso de dicionário de nomes próprios. Esta é a configuração padrão do RSLP.

Os valores de MAP verificados no gráfico indicam que, em relação aos valores obtidos sem o uso de *stemmer* ,ou com o uso do algoritmo de Porter nativo do Zettair, há uma melhoria na precisão quando se utiliza o RSLP em todas as suas variações, exceto com o uso de dicionário de nomes próprios. Este resultado é no mínimo curioso, pois esperava-se uma melhora na precisão com o uso deste dicionário, uma vez que nomes próprios não seriam tratados como palavras comuns da língua e, portanto, não seriam processados pelos passos de redução, permanecendo intactos.

5.3.2 Teste-T

A comparação dos valores das médias obtidas para um conjunto de consultas entre os diferentes algoritmos, como apresentado na Figura 5.2, não é suficiente para afirmar que um algoritmo é mais preciso que outro. Para embasar este tipo de afirmação, utiliza-se uma análise estatística conhecida como "Teste-T pareado", que tem por objetivo verificar se há diferença significativa entre duas médias obtidas sobre as mesmas amostras. Assim, comparando as médias obtidas por dois algoritmos distintos, é possível afirmar se as suas diferenças são estatisticamente significantes[19], ou seja, se a superioridade de uma média sobre outra é realmente relevante e deve ser considerada.

[19] Significa afirmar que as diferenças entre os resultados avaliados são grandes o suficiente para não serem atribuídas ao acaso. Disponível em: <http://www.lee.dante.br/amostragem/glossario.html>. Acesso em: 05 jun. 2007.

Como amostra de dados, foram utilizados os valores de MAP individuais das 150 consultas realizadas sobre a coleção, obtidos sobre as combinações de variações do algoritmo (as mesmas utilizadas no item 5.3.1). Sobre estes dados foi realizado o Teste-T pareado, do tipo bi-caudal, com α=0,05 (que indica um nível de significância de 5%), com auxílio do assistente de análise de dados existente no Microsoft Excel. Para estes testes, caso o valor obtido para o *P* dos dados amostrados (mostrado no campo "P(t<=T) bi-caudal" nas tabelas) seja menor do que α, confirma-se a hipótese de que a diferença das médias é estatisticamente significante, com fator de certeza de 95% (ou com 5% de chance de erro) (REIS, 2007). Os comparativos realizados e seus resultados são mostrados a seguir.

5.3.2.1 Comparativo: Zettair original e Zettair modificado (sem stemming)

Este comparativo tem por objetivo verificar se as modificações introduzidas no Zettair, com o intuito de adaptá-lo aos textos escritos em português, influenciaram de forma positiva os resultados do sistema. A Tabela 5.2 mostra os dados resultantes da aplicação do Teste-t neste experimento.

Tabela 5.2: Teste-t para Zettair Original e Zettair Modificado (sem *stemming*)

	Original	**Mod.-Red.Ac.**
Média (MAP)	0,311698000	0,321134000
Variância	0,082471488	0,081316565
Observações	150	150
Hipótese da diferença de média	0	
Stat t	-2,594844969	
P(T<=t) bi-caudal	0,010408920	
t crítico bi-caudal	1,976013145	

O resultado mostra que, apesar da pequena diferença entre os valores de MAP, esta diferença é estatisticamente significativa, uma vez que o P obtido é menor que α. Isto nos permite concluir que as alterações introduzidas no Zettair para processamento de documentos em português trouxeram uma pequena melhoria em seus resultados.

5.3.2.2 Comparativo: Zettair sem stemming e com RSLP Aprimorado

O objetivo desta comparação é verificar se o uso do RSLP realmente trouxe uma melhora nos resultados das consultas. Para tanto, utilizou-se os valores de MAP obtidas pelo Zettair sem qualquer processo de *stemming* ativado e os obtidos com a versão aprimorada do RSLP (sem o uso do dicionário de nomes próprios).Os resultados do Teste-t aplicado a estes dados podem ser vistos na Tabela 5.3.

Tabela 5.3: Teste-t para Zettair Original e com RSLP Aprimorado

	Original	**Mod.-RSLP**
Média	0,311698000	0,375903333
Variância	0,082471488	0,079246102
Observações	150	150
Hipótese da diferença de média	0	
Stat t	-5,033120979	
P(T<=t) bi-caudal	0,000001373	
t crítico bi-caudal	1,976013145	

O valor de *P* resultante deste teste é claramente menor do que α ($P < 0,05$), o que comprova que a média obtida com o uso do *stemmer* (coluna "RSLP") é estatisticamente significativa, ou seja, que o uso do RSLP no processamento da coleção efetivamente trouxe melhoria nos resultados. Esta conclusão é importante, uma vez que na bibliografia não há consenso se o uso de *stemming* é eficaz ou não, como citado por Orengo e Huyck (2001), e este experimento prova que o *stemming* trouxe melhora nos resultados gerais das consultas sobre este *corpus* em língua portuguesa.

5.3.2.3 Comparativo: uso do dicionário de nomes próprios

A radicalização de nomes próprios é uma das dificuldades de *stemming* em português, como citado por Orengo e Huyck (2001), e com este comparativo pretende-se verificar se a solução criada para evitar este problema (um dicionário de nomes próprios) apresentou um efeito positivo nos resultados das consultas. Neste teste foram utilizadas as MAP obtidas pelo algoritmo RSLP aprimorado com e sem o uso deste dicionário. Os resultados da aplicação do Teste-t sobre os dados são apresentados na Tabela 5.4.

Tabela 5.4: Teste-t para RSLP com e sem uso de dicionário de nomes próprios

	Mod.-RSLP-NP	**Mod.-RSLP**
Média	0,345282000	0,375903333
Variância	0,075115140	0,079246102
Observações	150	150
Hipótese da diferença de média	0	
Stat t	-3,615308838	
P(T<=t) bi-caudal	0,000409801	
t crítico bi-caudal	1,976013145	

Da análise do valor de *P*, conclui-se que a diferença entre as médias é estatisticamente significante, ou seja, os resultados obtidos sem o uso do dicionário de nomes são melhores do que os obtidos com seu uso. Esta conclusão confirma a suspeita levantada no item 5.3.1 deste capítulo, e vem de encontro ao que se esperava, pois esperava-se que a não radicalização de nomes próprios tivesse impacto positivo na precisão do resultado, resultando em médias de precisão maiores do que as verificadas quando ocorre o *stemming* dos nomes próprios. Este resultado pode ser parcialmente explicado com a qualidade da base de nomes próprios, pois cabe lembrar que a qualidade dos resultados obtidos neste teste estão intimamente ligados a esta base. Assim, este resultado provavelmente possa ser melhorado, através do refino da lista de nomes utilizadas no dicionário, o que merece um estudo mais aprofundado, não realizado neste trabalho.

5.3.2.4 Comparativo: uso da redução do plural e de todas as reduções

Em seus experimentos, Orengo (2006) concluiu que os melhores resultados foram obtidos utilizando somente a redução de plural, configuração esta chamada em seu trabalho de "*light stemming*", e não utilizando-se todos os passos de redução. Porém, apesar da coleção de documentos ser a mesma utilizada neste trabalho, em seu artigo foram utilizadas somente as consultas do CLEF para 2006. Em vista disso, foi realizado um experimento semelhante, porém também avaliando as consultas das campanhas de 2004 e 2005 do CLEF, para verificar se a conclusão de Orengo se aplica em experimentos com maior quantidade de consultas. Em tempo: no trabalho citado, foi

utilizado como Sistema de Recuperação de Informação o SMART[20] para indexação dos documentos.

Tabela 5.5: Teste-t das MAP para o uso da redução do plural e todas as reduções

	Mod.-RSLP	Mod.-RSLP-Pl.
Média	0,375903333	0,369247330
Variância	0,079246102	0,080052954
Observações	150	150
Hipótese da diferença de média	0	
Stat t	0,618209033	
P(T<=t) bi-caudal	0,537381390	
t crítico bi-caudal	1,976013145	

Como se nota na Tabela 5.5, a diferença entre as médias não é estatisticamente significante (o valor P não é menor que o α de 0,05), portanto não há uma diferença relevante entre o uso exclusivo da redução de plural ou de todas as reduções em termos de MAP.

5.4 Publicações

No decorrer deste trabalho, foram escritos 2 artigos a ele relacionados:

a) **"A Study on the use of Stemming for Monolingual Ad-Hoc Portuguese Information Retrieval"** (2006), Viviane Moreira Orengo. Publicado no *Working Note*s do CLEF 2006. A ser publicado no *Lecture Notes in Computing Science.*

b) **"RSLP: Uma Ferramenta para a Remoção de Sufixos da Língua Portuguesa"** (2007), Alexandre Ramos Coelho, Viviane Moreira Orengo e Luciana Salete Buriol. Submetido ao XXII Simpósio Brasileiro de Banco de Dados (SBBD) em 10 de junho de 2007, para a sessão de ferramentas.

Ambos artigos citados utilizaram a nova implementação do RSLP em seu experimentos.

5.5 Resumo do Capítulo

Este capítulo apresentou experimentos realizados com a integração do RSLP ao Zettair. Com os resultados obtidos nestes experimentos, espera-se mostrar a validade de todo o trabalho aqui exposto, mostrando que o uso do *stemmer* traz, em geral, benefícios ao sistema ao qual ele está integrado, melhorando a qualidade de seus resultados.

É muito importante frisar que, apesar de ter-se verificado que determinado método apresentava melhoria na precisão em relação a outro, isso não implica necessariamente que esta melhoria seja observada em todas consultas realizadas.

Na verdade, em vários experimentos observou-se que, para algumas consultas, os métodos que usavam *stemming* obtiveram valores menores de MAP do que as mesmas consultas realizadas sem o uso do *stemmer*. Por outro lado, observou-se em várias consultas uma melhora substancial nos resultados obtidos com o uso de *stemming*. Esta

[20] Disponível em <ftp://ftp.cs.cornell.edu/pub/smart/>.

observação somente vem a confirmar o que já havia sido concluído por Hull (1996), ou seja, o uso de *stemming* é quase sempre benéfico.

6 CONCLUSÃO

O "Removedor de Sufixos da Língua Portuguesa" é um dos poucos algoritmos existentes de *stemming* para nossa língua, e neste trabalho foram propostas melhorias em sua implementação e forma de funcionamento, tendo como objetivo melhorar o desempenho de *stemmer*, tanto em termos de performance de processamento quanto de qualidade de resultado. Entre as melhorias, propôs-se o uso de dicionários, para aumentar o desempenho e resolver a problemática de tratamento de nomes próprios, a flexibilização de configuração das funcionalidades do *stemmer*, permitindo alterar sua forma de processamento conforme desejado, adequando as necessidades do utilizador.

Com as melhorias propostas efetivamente implementadas, o novo algoritmo foi integrado a um Sistema de Recuperação de Informação, para realização de experimentos visando medir seu impacto na qualidade dos resultados obtidos. Apesar das dificuldades enfrentadas nesta integração, principalmente relacionadas a adaptação do sistema originalmente desenvolvido para uso com documentos escritos em língua inglesa, como resultado final obteve-se um poderoso sistema de RI perfeitamente adaptado ao uso em documentos escritos em nosso língua.

Da análise dos resultados obtidos nos experimentos, concluiu-se que a nova implementação do RSLP apresentou um ganho considerável em performance de processamento, bem como confirmou-se que o uso de *stemming* traz ganhos de qualidade ao sistema de RI, como já concluído em outros estudos. Além do ganho em desempenho, manteve-se o bom nível de precisão obtido pela versão original do algoritmo. Estes resultados serviram, então, para validar o trabalho realizado, uma vez que os objetivos propostos foram atingidos.

Apesar das conclusões positivas, este trabalho ainda oferece espaço para melhorias futuras, principalmente na pesquisa do uso o dicionário de nomes próprios, que apresentou resultados abaixo do esperado, bem como em um estudo aprofundado dos resultados obtidos em certas consultas, que apresentaram uma queda na qualidade quando usado o *stemming*. Com isso, pode-se melhorar ainda mais os resultados obtidos neste trabalho.

REFERÊNCIAS

ALVARES, R. V.; GARCIA, A. C. B.; FERRAZ, I. STEMBR: A Stemming Algorithm for the Brazilian Portuguese Language. *In*: PORTUGUESE CONFERENCE ON ARTIFICIAL INTELLIGENCE, 12., 2005. **Proceedings...** Covilhã, Portugal.

BAEZA-YATES, R. **Modern information retrieval.** New York, N.Y: Addison-Wesley, 1999.

CHAVES, M. S. Um estudo e apreciação sobre algoritmos de stemming. *In:* JORNADAS IBEROAMERICANAS DE INFORMÁTICA, 9., 2003. **Proceedings...** Cartagena de Indias, Colômbia. Disponível em: <http://xldb.di.fc.ul.pt/~mchaves/pg_portugues/public/stemming.pdf>. Acesso em: 20 maio 2007.

DIAS, M. A. L. **Extração automática de palavras-chave na língua portuguesa aplicada a dissertações e teses da área das engenharias.** 2004. 127 f. Dissertação (Mestrado em Engenharia Elétrica) – Faculdade de Engenharia Elétrica e de Computação, Universidade Estadual de Campinas, Campinas, 2004.

FRAKES, W.B.; R. BAEZA-YATES. **Information Retrieval:** data structures and algorithms**.** London, UK: Prentice Hall, 1992.

HAYDEN, P. Data Structures and algorithms: topic 7: tries and suffix tress. Class notes. McGill University: Montreal, 1997. Disponível em: <http://www.cs.mcgill.ca/~cs251/OldCourses/1997/topic7/>. Acesso em: 10 jun. 2007.

HONRADO, A. *et al*. A Word Stemming Algorithm for the Spanish Language. *In:* SYMPOSIUM ON STRING PROCESSING AND INFORMATION RETRIEVAL, 7., 2000. **Proceedings...** Washington, DC: IEEE Computer Society, 2000.

HULL, D. A. Stemming Algorithms : a case study for detailed evaluation. **Journal of the American Society for Information Science**, New York, v. 47, n. 1, p. 70-84, jan. 1996. Disponível em: <http://citeseer.ist.psu.edu/509573.html>. Acesso em 10 jun. 2007.

MONTEIRO JR, A. **LexWeb:** um léxico da Língua Portuguesa extraído automaticamente da Internet**.** 2004. 78 f . Dissertação (Mestrado em Engenharia Elétrica) - Programa de Pós-Graduação em Engenharia Elétrica , Centro Tecnológico, Universidade Federal do Pará, Belém, 2004.

ORENGO, V. M. **Avaliação de desempenho em sistemas de recuperação de informação:** tópicos especiais em recuperação de informação. 2006. 40 p. Notas de aula.

ORENGO, V. M. **Rslp.cpp** [código-fonte]. Versão 1.0. Londres, Middlesex University 2001.

ORENGO, V. M.; HUYCK C. A Stemming Algorithm for Portuguese Language. *In*: SYMPOSIUM ON STRING PROCESSING AND INFORMATION RETRIEVAL, 8., 2001. **Proceedings...** Chile, 2001.

ORENGO, V.M. A study on the use of Stemming for Monolingual Ad-Hoc Portuguese Information Retrieval . *In:* WORKING NOTES OF THE WORKSHOP OF THE CROSS-LANGUAGE EVALUATION FORUM. 2006. **Proceedings...** CLEF 2006, Alicante.

PALLES, B. R.; FUEZI, C. C. R. **Estudo comparativo de dois algoritmos de Stemming para a língua portuguesa.** 2005. 79 f. Monografia (Graduação em Ciência da Computação) – Curso de Bacharelado em Ciência da Informação, Universidade Católica de Salvador, Salvador, 2005.

PORTER, M. F. The stemming algorithm. 2005. Disponível em <http://snowball.tartarus.org/algorithms/portuguese/stemmer.html>. Acesso em: 20 maio 2007.

PORTER, M. F. An algorithm for suffix stripping. **Program**, London, v. 14, n. 3, p. 130-137, 1980. Disponível em <http://www.tartarus.org/~martin/PorterStemmer/def.txt>. Acesso em: 20 maio 2007.

SEARCH ENGINE GROUP. The Zettair Search Engine. Melbourne, Austrália: RMIT University, 2006. Disponível em: <http://www.seg.rmit.edu.au/zettair/>. Acesso em: 15 out. 2006.

REIS, M. M. **Tópicos de Estatística:** testes de diferenças entre médias. 2007. Disponível em: <http://www.inf.ufsc.br/~marcelo/testes2.html>. Acesso em: 03 jun. 2007.

TRIES. 1999. Disponível em: <http://www.csse.monash.edu.au/~lloyd/tildeAlgDS/Tree/Trie>. Acesso em: 10 jun. 2007.

VANRIJSBERGEN, C. J. **Information Retrieval**. London: Butterworths, 1979. Disponível em: <http://www.dcs.gla.ac.uk/Keith/Preface.html>. Acesso em: 23 maio 2007.

WITTEN, I. H.; MOFFAT, A.; BELL, T. C. **Managing Gigabytes**. São Francisco: Morgan Kaufmann Publishing, 1999.

ANEXO A - Exemplo de documento no formato TREC

O formato de arquivo utilizado para experimentos TREC prevê vários documentos em um mesmo arquivo, todos separados pelos marcadores (*tags*) *<DOC> </DOC>* e identificados através das *tags <DOCNO>* e *<DOCID>*. Segue abaixo um trecho de um arquivo neste formato:

```
<DOC>
<DOCNO>FSP940213-001</DOCNO>
<DOCID>FSP940213-001</DOCID>
<DATE>940213</DATE>
<TEXT>
O desfile das escolas de samba no Rio tem um duelo especial hoje: a Mangueira, que homenageia Caetano Veloso, Gilberto Gil, Gal Costa e Maria Bethânia, enfrenta o mais criativo dos carnavalescos, Joãosinho Trinta, da Unidos do Viradouro. Sete vezes campeão, Joãosinho trata da loucura no enredo sobre um quilombo destruído no século 18. Em Salvador, um homem de 24 anos foi assassinado com um tiro na cabeça durante o desfile do Olodum ontem. O crime ocorreu perto da ladeira de São Bento, no centro. Caetano desfilou com o bloco, que teve como tema o tropicalismo. As estradas de acesso ao litoral de São Paulo ficaram congestionadas. A velocidade média ontem era inferior a 5 km/h. Leia no caderno Carnaval e na pág. 1-11.


Leia na pág. 1-3 artigos dos escritores Lygia Fagundes Telles e José Paulo Paes sobre o Carnaval.
</TEXT>
</DOC>
<DOC>
<DOCNO>FSP940213-002</DOCNO>
<DOCID>FSP940213-002</DOCID>
<DATE>940213</DATE>
<TEXT>
O 44º All Star Game, partida de confraternização dos astros do basquete americano, está marcado para hoje à noite em Minneapolis, Minnesota. O jogo reúne as equipes da Costa Leste e daCosta Oeste. A TV transmite. O pivô Shaquille O'Neal(foto), do Leste, é o destaque. O ala Charles Barkley é a principal ausência. Não joga por contusão. Leia no caderno especial All Star Game.
</TEXT>
</DOC>
[...]
```

ANEXO B - Exemplo de tópicos no formato TREC

Os tópicos são as consultas aplicadas pelo SRI ao índice de documentos. Um tópico consiste de:

a) um número de consulta, indicado entre as marcas <*num*> <*/num*>.

b) um título, indicado pelo marcador <*title*>.

c) uma descrição do tópico (marcador <*desc*>).

d) uma narrativa do tópico (marcador <*narr*>).

Segue abaixo um trecho de um arquivo neste formato:

```
<top>

<num> 201 </num>

<title> Fogos domésticos </title>

<desc> Quais são as principais causas de fogos no lar? </desc>

<narr> Documentos relevantes devem mencionar pelo menos uma causa (provável) de fogos em residencias privadas em geral ou referências a casos específicos. </narr>

</top>


<top>

<num> 202 </num>

<title> Prisão de Nick Leeson </title>

<desc> Encontrar documentos sobre a prisão de Nick Leeson e as causas que o levaram à cadeia. </desc>

<narr> Documentos relevantes devem relatar as razões da prisão de Nick Leeson e sua subsequente prisão. </narr>

</top>

[...]
```

ANEXO C - Exemplo de arquivo de avaliação

Aqui é mostrado um trecho de um arquivo utilizado para avaliação dos resultados obtidos por um Sistema de Recuperação de Informações, já no formato esperado pela ferramenta *trec_eval*. As colunas são as que seguem:

a) primeira coluna: o número do tópico ou consulta.

b) segunda coluna: não utilizado, normalmente fixo em 0.

c) terceira coluna: o código do documento (indicado entre as marcas *<DOCNO>*).

d) quarta coluna: relevância do documento, 0 indica não relevante, 1 indica relevante.

Documentos que não aparecem no arquivo de resposta não foram julgados pela comissão da campanha, e são assumidos como não-relevantes.

```
301 0 FSP950904-052 1
301 0 FSP951023-051 1
301 0 FSP951030-037 1
301 0 PUBLICO-19940603-137 1
301 0 PUBLICO-19940603-138 1
301 0 PUBLICO-19940910-114 1
301 0 PUBLICO-19941106-038 1
301 0 PUBLICO-19941106-040 1
301 0 PUBLICO-19941202-051 1
301 0 PUBLICO-19950811-122 1
301 0 FSP940107-073 0
301 0 FSP940115-030 0
301 0 FSP940120-078 0
301 0 FSP940122-079 0
301 0 FSP940124-042 0
301 0 FSP940131-035 0
```

APÊNDICE A - Regras de remoção de sufixos

Segue abaixo a listagem das regras de remoção de sufixos de todos os passos. As exceções de cada regra foram omitidas, devido ao espaço limitado, mas podem ser obtidas online no endereço:

<*http://www.inf.ufrgs.br/~arcoelho/rslp/steprules_modificado.txt*>.

Redução do Plural

Tamanho mínimo da palavra: 3
Comparação de exceções: palavra inteira
Letra final: s

Sufixo a Remover	Tamanho Mínimo do Stem	Sufixo de Substituição
ns	1	m
ões	3	ão
ães	1	ão
ais	1	al
éis	2	el
eis	2	el
óis	2	ol
is	2	il
les	3	l
res	3	r
s	2	-

Remoção de Vogais

Tamanho mínimo da palavra: 0
Comparação de exceções: somente sufixo

Sufixo a Remover	Tamanho Mínimo do Stem	Sufixo de Substituição
bil	2	vel
gue	2	g
á	3	
ê	3	
a	3	
e	3	
o	3	

Redução do Aumentativo

Tamanho mínimo da palavra: 0
Comparação de exceções: palavra inteira

Sufixo a Remover	Tamanho Mínimo do Stem	Sufixo de Substituição
díssimo	5	
abilíssimo	5	
íssimo	3	
ésimo	3	
érrimo	4	
zinho	2	
quinho	4	c
uinho	4	
adinho	3	
inho	3	
alhão	4	
uça	4	
aço	4	
aça	4	
adão	4	cidadão
idão	4	
ázio	3	
arraz	4	
zarrão	3	
arrão	4	
arra	3	
zão	2	
ão	3	

Redução do Feminino

Tamanho mínimo da palavra: 3
Comparação de exceções: palavra inteira
Letra final: a,ã

Sufixo a Remover	Tamanho Mínimo do Stem	Sufixo de Substituição
ona	3	ão
ora	3	or
na	4	no
inha	3	inho
esa	3	ês
osa	3	oso
íaca	3	íaco
ica	3	ico
ada	2	ado
ida	3	ido
ída	3	ido
ima	3	imo
iva	3	ivo
eira	3	eiro
ã	2	ão

Redução Adverbial

Tamanho mínimo da palavra: 0
Comparação de exceções: somente sufixo

Sufixo a Remover	Tamanho Mínimo do Stem	Sufixo de Substituição
mente	4	

Redução Verbal

Tamanho mínimo da palavra: 0
Comparação de exceções: somente sufixo

Sufixo a Remover	Tamanho Mínimo do Stem	Sufixo de Substituição
aríamo	2	
ássemo	2	
eríamo	2	
êssemo	2	
iríamo	3	
íssemo	3	
áramo	2	
árei	2	
aremo	2	
ariam	2	
aríei	2	
ássei	2	
assem	2	
ávamo	2	
êramo	3	
eremo	3	
eriam	3	
eríei	3	
êssei	3	
essem	3	
íramo	3	
iremo	3	
iriam	3	
iríei	3	
íssei	3	
issem	3	
ando	2	
endo	3	
indo	3	
ondo	3	
aram	2	
arão	2	
arde	2	
arei	2	
arem	2	
aria	2	
armo	2	
asse	2	
aste	2	
avam	2	
ável	2	
eram	3	
erão	3	
erde	3	
erei	3	
êrei	3	
erem	3	
eria	3	
ermo	3	
esse	3	
este	3	
íamo	3	
iram	3	
íram	3	
irão	2	
irde	2	
irei	3	
irem	3	
iria	3	
irmo	3	
isse	3	
iste	4	
iava	4	
amo	2	
iona	3	

ara	2	
ará	2	
are	2	
ava	2	
emo	2	
era	3	
erá	3	
ere	3	
iam	3	
íei	3	
imo	3	
ira	3	
ído	3	
irá	0	
tizar	4	
izar	5	
itar	5	
ire	3	
omo	3	
ai	2	
am	2	
ear	4	
ar	2	
uei	3	
uía	5	u
ei	3	
guem	3	g
em	2	
er	2	
eu	3	
ia	3	
ir	3	
iu	3	
eou	5	
ou	3	
i	3	

Redução Nominal

Tamanho mínimo da palavra: 0

Comparação de exceções: somente sufixo

Sufixo a Remover	Tamanho Mínimo do Stem	Sufixo de Substituição
encialista	4	
alista	5	
agem	3	
iamento	4	
amento	3	
imento	3	
mento	6	
alizado	4	
atizado	4	
tizado	4	
izado	5	
ativo	4	
tivo	4	
ivo	4	
ado	2	
ido	3	
ador	3	
edor	3	
idor	4	
dor	4	
sor	4	
atoria	5	
tor	3	
or	2	
abilidade	5	
icionista	4	
cionista	5	
ionista	5	
ionar	5	
ional	4	
ência	3	
ância	4	
edouro	3	
queiro	3	c
adeiro	4	
eiro	3	
uoso	3	
oso	3	
alizaç	5	
atizaç	5	
tizaç	5	
izaç	5	
aç	3	
iç	3	
ário	3	
atório	3	
rio	5	
ério	6	
ês	4	
eza	3	
ez	4	
esco	4	
ante	2	
ástico	4	
alístico	3	
áutico	4	
êutico	4	
tico	3	

ico	4	
ividade	5	
idade	4	
oria	4	
encial	5	
ista	4	
auta	5	
quice	4	c
ice	4	
íaco	3	
ente	4	
ense	5	
inal	3	
ano	4	
ável	2	
ível	3	
vel	5	
bil	3	vel
ura	4	
ural	4	
ual	3	
ial	3	
al	4	
alismo	4	
ivismo	4	
ismo	3	

APÊNDICE B - Modificações no código-fonte do sistema Zettair para integração do RSLP

Para integrar o código-fonte do RSLP ao Zettair, foram necessárias algumas modificações no sistema. Estas modificações se deram em vários arquivos do código-fonte original do sistema, e são mostradas a seguir. Os trechos em **negrito** destacam as alterações realizadas. Os códigos-fonte completos do Zettair com o RSLP integrado estão disponíveis *online* em: <*http://www.inf.ufrgs.br/~arcoelho/rslp/zettair-rslp.zip*>.

Seguem as alterações:

1 – Adição da opção de um novo *stemmer*: isto é feito no arquivo *index.h*, presente no diretório *include*, onde é criado mais um define para *index_stem*:

```
enum index_stem {
    INDEX_STEM_NONE = 0,            /* no stemming algorithm */
    INDEX_STEM_PORTERS = 1,         /* use Porter's stemming algorithm */
    INDEX_STEM_EDS = 2,             /* use 'eds' stemming algorithm */
    INDEX_STEM_LIGHT = 3,           /* use light stemming algorithm */
    INDEX_STEM_RSLP = 4             /* use RSLP stemming algorithm */
};
```

2 – Adicionar o novo *stemmer* na estrutura *index_flags*, no arquivo *src/include/_index.h*:

```
enum index_flags {
    INDEX_BUILT = (1 << 0),             /* the index has been constructed */
    INDEX_SORTED = (1 << 1),            /* the index is sorted by vocabulary
                                         * order (more or less) */
    INDEX_SOURCE = (1 << 2),            /* source text for the documents is
                                         * available */
    INDEX_STEMMED = (3 << 3),           /* mask to tell whether index is
                                         * stemmed */
    INDEX_STEMMED_PORTERS = (1 << 3),   /* whether its stemmed with porter's
                                         * stemmer */
    INDEX_STEMMED_EDS = (1 << 4),       /* whether its stemmed with eds
                                         * stemmer */
    INDEX_STEMMED_LIGHT = (3 << 3),     /* whether its stemmed with the light
                                         * stemmer */
    INDEX_STEMMED_RSLP  = (1 << 5)      /* RSLP stemmer */
};
```

3 – Adicionar o protótipo da nova função de *stemming* no arquivo *src/include/stem.h*:

```
/* RSLP stemmer */
void stem_rslp(void *opaque, char *term);
```

4 – Definir a função do novo *stemmer* em *src/stem.c*. No caso do RSLP, esta função se limita a chamar a função *rslpProcessWord*, passando como parâmetros a palavra obtida pelo Zettair e a estrutura de dados principal do RSLP. :

```
/* RSLP stemmer */
void stem_rslp(void *opaque, char *term)
{
        rslpProcessWord(term, &rslpMainStruct);

        return;
}
```

Por usar a função *rslpProcessWord*, é necessário adicionar a inclusão do arquivo onde a mesma está definida:

```
#include "rslpStemmer.h"
```

5 – No arquivo *src/index.c*, além de incluir o arquivo *rslpStemmer.h*, deve-se incluir a chamada à função *stem_rslp* na função *index_stemmer*:

```
/* function to return the stemming function used by an index */
void (*index_stemmer(struct index *idx))(void *, char *) {
    if ((idx->flags & INDEX_STEMMED_LIGHT) == INDEX_STEMMED_LIGHT) {
        return (void (*)(void *, char *)) stem_light;
    } else if (idx->flags & INDEX_STEMMED_PORTERS) {
        return (void (*)(void *, char *)) stem_porters;
    } else if (idx->flags & INDEX_STEMMED_EDS) {
        return (void (*)(void *, char *)) stem_eds;
    } else if (idx->flags & INDEX_STEMMED_RSLP) {
        return (void (*)(void *, char *)) stem_rslp;
    } else {
        return NULL;
    }
}
```

No mesmo arquivo, é necessário adicionar a inicialização do *stemmer* na função *index_new*:

```
        ...
    /* initialise stemming algorithm if requested */
    if (opts & INDEX_NEW_STEM) {
        if ((opt->stemmer == INDEX_STEM_PORTERS)
          && (idx->stem
            = stem_cache_new(stem_porters, NULL, 200))) {
            idx->flags |= INDEX_STEMMED_PORTERS;
        } else if ((opt->stemmer == INDEX_STEM_EDS)) {
            idx->flags |= INDEX_STEMMED_EDS;
            idx->stem = NULL;
```

```
        } else if ((opt->stemmer == INDEX_STEM_LIGHT)) {
            idx->flags |= INDEX_STEMMED_LIGHT;
            idx->stem = NULL;
        } else if ((opt->stemmer == INDEX_STEM_RSLP)) {
            idx->flags |= INDEX_STEMMED_RSLP;
            idx->stem = NULL;
        } else {
         ...
```

6 – No arquivo *src/commandline.c*, é necessário incluir a inicialização do RSLP na função *parse_args*, caso o mesmo seja escolhido como método de *stemming*. Nesta função é feita a chamada à função *rslpLoadStemmer*, que carrega todas as estruturas necessárias para o funcionamento do RSLP:

```
...
        case OPT_STEM:
            args->nopts |= INDEX_NEW_STEM;
            stem = 1;
            if (!str_casecmp(arg, "eds")) {
                args->nopt.stemmer = INDEX_STEM_EDS;
            } else if (!str_casecmp(arg, "light")) {
                args->nopt.stemmer = INDEX_STEM_LIGHT;
            } else if (!str_casecmp(arg, "none")) {
                /* in fact, don't stem */
                args->nopts ^= INDEX_NEW_STEM;
            } else if (!str_casecmp(arg, "porters")
              || (!str_casecmp(arg, "porter"))) {
                args->nopt.stemmer = INDEX_STEM_PORTERS;
                 } else if (!str_casecmp(arg, "rslp")) {
                         args->nopt.stemmer = INDEX_STEM_RSLP;
                         // Loads the RSLP Stemmer
                         rslpLoadStemmer(&rslpMainStruct, "rslpconfig.txt");
            } else {
                fprintf(output, "unrecognised stemming algorithm '%s'\n", arg);
                err = 1;
            }
            break;
...
```

Neste mesmo arquivo, na função *main*, é necessário incluir uma chamada à função *rslpUnloadStemmer*, garantindo assim que toda memória utilizada pelo RSLP seja devidamente liberada na saída do programa:

```
...
            /* build or add to a index */
            if (!build(args, output)) {
                free_args(args);
                return EXIT_FAILURE;
            }
        }
                 if(args->nopt.stemmer)
```

```
                        if(args->nopt.stemmer == INDEX_STEM_RSLP)
                                rslpUnloadStemmer(&rslpMainStruct);
        free_args(args);
    }
    return EXIT_SUCCESS;
}
```

7 – De forma similar às modificações realizadas no arquivo *commandline.c*, o arquivo *src/trecrun.c* também deve ser modificado. Isso se deve ao fato deste arquivo fonte gerar executável *zet_trec*, que é utilizado para realizar consultas em índices criados à partir de documentos formatados no padrão TREC. Na função *main*, é necessário incluir chamadas às funções de inicialização e encerramento do RSLP:

```
int main(int argc, char **argv) {
    struct args *args;
    unsigned int i;
    FILE *fp,
         *output;
    struct treceval *results = NULL;

    if (isatty(STDOUT_FILENO)) {
        output = stdout;
    } else {
        output = stderr;
    }

        rslpLoadStemmer(&rslpMainStruct, "rslpconfig.txt");

        args = parse_args(argc, argv, output);

    if (args == NULL) {
                rslpUnloadStemmer(&rslpMainStruct);
        return EXIT_FAILURE;
    }

    if (args->qrels) {
        if (!(results = treceval_new())) {
            fprintf(stderr, "failed to initialise results structure\n");
        }
    }



    if (args) {
        for (i = 0; i < args->topic_files; i++) {
            if ((fp = fopen((const char *) args->topic_file[i], "rb"))) {
                if (!process_topic_file(fp, args, stdout, results)) {
                    if (results) {
                        treceval_delete(&results);
                    }
                    fprintf(output, "failed to process topic file '%s'\n",
```

```
                    args->topic_file[i]);
                fclose(fp);
                free_args(args);

                                        rslpUnloadStemmer(&rslpMainStruct);
                return EXIT_FAILURE;
            }
            fclose(fp);
        } else {
            if (results) {
                treceval_delete(&results);
            }
            fprintf(output, "couldn't open topic file '%s': %s\n",
              args->topic_file[i], strerror(errno));
            free_args(args);
                              rslpUnloadStemmer(&rslpMainStruct);
            return EXIT_FAILURE;
        }
    }

    if (results) {
        struct treceval_results eval;

        treceval_evaluate(results, args->qrels, &eval);

        treceval_print_results(&eval, 1, stdout, 0);

        treceval_delete(&results);
    }

    free_args(args);

            rslpUnloadStemmer(&rslpMainStruct);
}

return EXIT_SUCCESS;
}
```

8 – Com todas as modificações realizadas no código-fonte do Zettair, é necessário copiar os arquivos do RSLP para os locais adequados: *rslpStemmer.c* e *rslpTrie.c* para o diretório *src*, e os demais arquivos (*rslpStemmer.h*, *rslpTrie.h* e *rslpStructs.h*) para o diretório *src/include*.

9 – Após os arquivos com o código-fonte estarem nos diretórios adequados, é necessário adicioná-los ao *Makefile* do Zettair, para que sejam compilados junto com o sistema.

Com as modificações apresentadas, ao compilarmos o código fonte do Zettair, o RSLP estará automaticamente integrado ao sistema, podendo ser utilizado passando-se o argumento "*-stem=rslp*" na chamada ao Zettair.

APÊNDICE C – Arquivo de configuração do RSLP

Aqui são mostradas todas as possíveis variáveis no arquivo de configuração do RSLP, seus usos e valores possíveis.

Variável	Valores possíveis
STEPS_FILE	Uma *string* com o nome do arquivo de passos e regras
FLOW_FILE	Uma *string* com o nome do arquivo com o fluxo de aplicação das regras. Caso não exista, o RSLP automaticamente carrega o fluxo de regras padrão.
DO_STEMMING	`YES` ou `NO`. Indica se o processo de stemming deve ser realizado. O padrão é `YES`. Caso seja selecionado `NO`, a função `rslpProcessWord` não aplica as regras de *stemming*, deixando a palavra intocada. Útil para depuração de código.
REPLACE_ISO_CHARS	`YES` ou `NO`. Indica se deve substituir caracteres acentuados na palavra pelos equivalentes sem acento (por exemplo, `á` por `a`).
USE_STEM_DICTIONARY	`YES` ou `NO`. Indica se deve ser utilizado o dicionário de stems. O uso deste dicionário aumenta drasticamente o desempenho do *stemmer*, porém às custas de um uso maior de memória.
STEM_DICT_MAX_SIZE	Um valor inteiro, à partir de `1`. Indica o tamanho máximo de memória (em *megabytes*) que o dicionário de stems pode utilizar. Caso este limite seja atingido, novas palavras não são mais adicionadas ao dicionário, obrigando o seu reprocessamento a cada nova ocorrência.
USE_NAMED_ENTITIES	`YES` ou `NO`. Indica se deve ser utilizado o dicionário de Nomes Próprios. Se ativado este dicionário, sempre que uma palavra iniciada por letras maiúsculas é recebida, o RSLP primeiro verifica sua presença neste dicionário. Caso exista, a palavra não é processada pelos passos do *stemmer*, não sendo modificada (à exceção do *case* das letras e à substituição de acentos, caso ativada).

NAMED_ENTITIES_FILE	*String* com o nome do arquivo com a listagem de Nomes Próprios.
NAMED_ENTITIES_DICT_MAX_SIZE	Idem a STEM_DICT_MAX_SIZE, porém diz respeito ao dicionário de Nomes Próprios.

Segue abaixo um exemplo de arquivo de configuração do RSLP:

```
STEPS_FILE=steprules.txt
DO_STEMMING=YES
REPLACE_ISO_CHARS=YES
USE_STEM_DICTIONARY=YES
USE_NAMED_ENTITIES=YES
NAMED_ENTITIES_FILE=nomes.xml
STEM_DICT_MAX_SIZE=123
NAMED_ENTITIES_DICT_MAX_SIZE=50
FLOW_FILE=rslpflow.original
```