MAC323 EXERCÍCIO-PROGRAMA 3

LOCALIZAÇÃO DE PALAVRAS I

Y. KOHAYAKAWA

Data de entrega: 3/6/2013 (23:55)

1. Introdução

Nos EPs 3 e 4, você fará alguns exercícios simples envolvendo textos e processamento de linguagem natural (PLN). A funcionalidade básica de seu EP3 será a localização de palavras em textos grandes. Basicamente, dada uma palavra w e um texto T, seu programa deve devolver todas as sentenças que contêm a palavra w. Para tornar seu programa mais útil, o usuário poderá especificar se a busca deve ser por w ou se variantes de w devem ser consideradas.

Vários problemas envolvendo PLN (por exemplo, dado T, determinar as sentenças que compõe T) serão resolvidos usando um sistema disponibilizado pelo grupo de PLN de Stanford. Veja `http://www-nlp.stanford.edu/`

Os EPs 3 e 4 devem ser projetados para a língua inglesa.

2. Descrição do problema

Já descrevemos o problema principal: localizar palavras w e suas variantes em um texto T, devolvendo as sentenças de T que as contêm.

2.1. **Variantes de palavras; *stemming e lemmatization*.** Para entender o que temos em mente quando falamos sobre 'variantes' de palavras, leia sobre *stemming* e *lemmatization* no livro *Introduction to Information Retrieval*, de Christopher D. Manning, Prabhakar Raghavan and Hinrich Schütze, Cambridge University Press, 2008. Veja

- `http://www-nlp.stanford.edu/IR-book/`
- `http://nlp.stanford.edu/IR-book/html/htmledition/stemming-and-lemmatization-1.html`

Quando o usuário fizer uma busca por uma palavra w, ele dirá se a busca deve ser por w exatamente, ou se deve ser por todas as *variantes* de w, que *definimos como sendo todas as palavras que são mapeadas para a mesma palavra que w quando lematizadas.*

2.2. **Sentenças.** Seu programa deve ter como saída uma certa coleção de sentenças do texto T. Para tanto, dado T, será necessário identificar as sentenças em T.

2.3. **Sistema de PLN de Stanford.** Bons algoritmos de lematização são difíceis de implementar. Ademais, mesmo o desenvolvimento de um bom algoritmo para identificar as sentenças que formam um dado texto dá certo trabalho. Para resolver estes problemas, você usará um sistema do grupo de PLN de Stanford. Veja

- `http://nlp.stanford.edu/software/corenlp.shtml`

Versão de 23 de maio de 2013, 12:20.

De fato, para processar um texto T, você primeiro executará o *Stanford CoreNLP* em T, produzindo um texto T^{S}, e você então fornecerá T^{S} para seu programa como entrada. O texto T^{S} contém todo o texto T, mas com várias informações adicionais, como a quebra de T em sentenças e a lematização das palavras que ocorrem em T.

3. Textos anotados produzidos pelo CoreNLP

Uma boa fonte de textos para experimentação é o projeto Gutenberg: `http://www.gutenberg.org/`. Em particular, neste EP, você deve experimentar trabalhar com os arquivos em `http://www.ime.usp.br/~yoshi/2013i/mac323/EPs/EP3/DATA`.

Este é um trecho da saída de uma execução do CoreNLP com entrada `manifesto.txt`:

```
Sentence #470 (8 tokens):
WORKING MEN OF ALL COUNTRIES, UNITE!
[Text=WORKING CharacterOffsetBegin=73516 CharacterOffsetEnd=73523
PartOfSpeech=VBG Lemma=work] [Text=MEN CharacterOffsetBegin=73524
CharacterOffsetEnd=73527 PartOfSpeech=NNS Lemma=man] [Text=OF
CharacterOffsetBegin=73528 CharacterOffsetEnd=73530 PartOfSpeech=IN
Lemma=of] [Text=ALL CharacterOffsetBegin=73531
CharacterOffsetEnd=73534 PartOfSpeech=NN Lemma=all] [Text=COUNTRIES
CharacterOffsetBegin=73535 CharacterOffsetEnd=73544 PartOfSpeech=NNS
Lemma=country] [Text=, CharacterOffsetBegin=73544
CharacterOffsetEnd=73545 PartOfSpeech=, Lemma=,] [Text=UNITE
CharacterOffsetBegin=73546 CharacterOffsetEnd=73551 PartOfSpeech=VB
Lemma=unite] [Text=! CharacterOffsetBegin=73551
CharacterOffsetEnd=73552 PartOfSpeech=. Lemma=!]
```

Estude a saída acima com cuidado. Note que, se o usuário quiser localizar todas as ocorrências de 'man' em `manifesto.txt`, incluindo variantes, a sentença acima deve fazer parte da saída.

A seguinte chamada foi executada para processar `manifesto.txt`:

```
yoshi@delta-8 ~/TMP
$ java -cp \
  stanford-corenlp-1.3.4.jar:stanford-corenlp-1.3.4-models.jar:xom.jar:joda-time.jar:jollyday.jar \
  -Xmx3g edu.stanford.nlp.pipeline.StanfordCoreNLP -annotators \
  tokenize,ssplit,pos,lemma -file manifesto.txt -outputFormat text
```

A saída produzida pela chamada acima chama-se `manifesto.txt.out`.

4. Seu programa

Seu programa deve receber na linha de comando o nome do arquivo texto produzido pela execução do corenlp (por exemplo, `manifesto.txt.out`), precedido por `-f`. Por exemplo,

```
prompt$ ep3 -fmanifesto.txt.out
```

Após interpretar o arquivo de entrada (por exemplo, `manifesto.txt.out`), seu programa deve ler instruções no `stdin`. Cada instrução tem a forma `-<opções> <palavra>`. A instrução

```
-e man
```

pede que as sentenças que contêm 'man' (exatamente) sejam enviadas para o `stdout`. Por outro lado,

```
-a man
```

pede que todas as sentenças que contêm 'man' e suas variantes (como por exemplo 'men') sejam enviadas para o `stdout`. Como padrão, ao enviar uma sentença para o `stdout`, seu programa deve enviar o texto da sentença e nada mais (no exemplo acima, `WORKING MEN OF ALL COUNTRIES, UNITE!`). Caso a opção `v` seja dada, devem também ser impressos os identificadores das sentenças (`Sentence #470 (8 tokens):` no exemplo acima). A opção `V` significa que toda informação sobre a sentença (isto é, sua identificação, a sentença em si e a sentença anotada) deve ser enviada para o `stdout`. Assim, possíveis instruções são

```
-e men
-av men
-aV workers
```

Implemente também a instrução `-F`, que indica o fim da lista de instruções.

Instruções adicionais. A instrução `-t` deve produzir a lista de todos os *tokens* presentes no texto, ordenados em ordem alfabética (um *token* por linha, sem repetições). A instrução `-d` deve produzir a lista de todas as palavras presentes no texto (uma palavra é um *token* constituído de letras), ordenadas em ordem alfabética (uma palavra por linha, sem repetições). A instrução `-l` deve produzir a lista de todas as palavras presentes no texto em sua versão lematizada (um lema por linha, sem repetições, em ordem alfabética). A instrução `-L` é semelhante: ela deve produzir a lista de todos os lemas, mas com cada lema seguido de todas as palavras no texto que têm aquele lema; esta instrução deve produzir uma linha de saída por lema.

A instrução `-s` deve produzir estatísticas do texto: número de sentenças no texto, número total de *tokens* no texto (isto é, contando repetições), número total de palavras no texto, número total de *tokens* distintos, número total de *palavras* distintas, e número total de lemas distintos.

4.1. **Estruturas de dados.** Seu programa deve usar tabelas de símbolos para organizar as informações contidas nos arquivos de entrada. Por exemplo, é natural implementar uma tabela que 'traduza' palavras w para sua forma lematizada. Também é natural implementar uma estrutura para dizer quais palavras w que ocorrem no texto têm uma dada versão lematizada.

Informações adicionais. Expandimos aqui as sugestões acima. Implemente uma tabela de símbolos T_1, cujos objetos têm como chave as palavras w que ocorrem no texto. O objeto ('item') de chave w deve conter a versão lematizada de w e deve também conter informações sobre as localizações de w no texto (por exemplo, um conjunto de ponteiros para as primeiras e últimas letras das sentenças que contêm w ou para as primeiras letras dos identificadores das sentenças que contêm w, etc).

Implemente também uma tabela de símbolos T_2 cujos objetos têm como chaves os lemas das palavras que ocorrem no texto (as versões lematizadas das palavras que ocorrem no texto). O objeto de chave ℓ em T_2 deve conter as variantes de ℓ que ocorrem no texto. Caso o usuário dê uma instrução do tipo `-a` com uma palavra w, seu programa pode usar T_1 para obter a versão lematizada ℓ de w, e então usar T_2 e novamente T_1.

Tanto para T_1 como para T_2, neste EP, *você deve usar árvores rubro-negras esquerdistas.* Estas tabelas de símbolos devem ser implementadas como tipos abstratos. *No próximo EP, você deverá implementar estas tabelas de outra forma* (ainda a ser especificado).

Observação. É possível e também é natural 'integrar' as tabelas T_1 e T_2 acima em uma estrutura mais complicada, tornando a segunda busca em T_1 na discussão acima desnecessária. Entretanto, tal implementação mais complexa (e mais eficiente) não será exigida neste EP.

Ressalva. Suponha que `working` e `worked` ocorram no texto, mas `works` não ocorra. O que acontece se o usuário der a instrução `-a works`? Para tratar esse caso (*isto é opcional neste EP*), você pode usar `morpha` em `http://www.informatics.susx.ac.uk/research/nlp/carroll/morph.html` ou usar componentes adequados do Stanford CoreNLP (veja Morphology).

Observações. Seguem as observações usuais.

1. *Este EP é estritamente individual.* Programas semelhantes receberão nota 0.
2. Seja cuidadoso com sua programação (correção, documentação, apresentação, clareza do código, etc), dando especial atenção a suas estruturas de dados. A correção será feita levando isso em conta.
3. Comparem entre vocês o desempenho de seus programas.
4. Entregue seu EP no Paca.
5. Não deixe de incluir em seu código um *relatório* para discutir seu EP: discuta as estruturas de dados usadas, os algoritmos usados, etc. *Se você escrever claramente como funciona seu EP, o monitor terá pouca dificuldade em corrigi-lo, e assim você terá uma nota mais alta.* (Se o monitor sofrer para entender seu código, sua nota será baixa.)

Observação final. Enviem dúvidas para a lista de discussão da disciplina.

Instituto de Matemática e Estatística, Universidade de São Paulo, Rua do Matão 1010, 05508–090 São Paulo, SP

Endereço eletrônico: `yoshi@ime.usp.br`