FACOM- UFU
Professor: Wendel Melo

Primeiro trabalho de Organização e Recuperação da Informação 2024-1

Descrição

Objetivos do trabalho:

1. Implementação de um gerador de índice invertido ;
2. Implementação do modelo *booleano* para recuperação de informação

Deve ser entregue apenas um **único** programa desenvolvido em Python que realize a tarefa descrita. O trabalho deve ser feito de forma individual e o código gerado deve ser entregue através de um formulário apropriado na plataforma *Teams*.

Aviso importante: se for detectado cópia ou qualquer tipo de trapaça entre diferentes alunos, todos os alunos serão punidos com a nota zero. Portanto, pense bem antes de pedir para copiar o trabalho do seu coleguinha, pois ele poderá ser punido também! É válido lembrar que essa é uma disciplina onde são estudados algoritmos e ferramentas para similaridade de documentos, e que tais ferramentas analisarão os trabalhos entregues. É uma péssima ideia tentar trapacear aqui!

É proibido usar o pacote nltk! Trabalhos que importarem qualquer parte da biblioteca nltk receberão a nota zero!

É obrigatório o uso do pacote SpaCy do Python para a lematização dos termos do vocabulário e remoção de *stopwords*. Você deve usar o modelo de língua portuguesa apresentado em aula para fazer o seu programa. Usar um modelo de linguagem diferente do da aula acarretará em perda de pontos! Estude os notebooks das aulas antes de buscar outras fontes. **O SpaCy é a única biblioteca Python fora da distribuição padrão que deve ser importada no seu código!**

Os detalhes sobre a geração do índice são descritos a seguir. **É importante ler com atenção e seguir todos os detalhes da especificação sob pena de perda de pontos na nota do trabalho!**

A base de documentos

A base de documentos é composta por um conjunto arbitrário de arquivos de texto puro. Assuma que nesses arquivos texto, palavras são separadas por um ou mais dos seguintes caracteres: espaço em branco ( ), ponto (.), reticências(...) vírgula (,), exclamação (!), interrogação (?) ou enter (\n). Seu programa deve tratar caracteres maiúsculos e minúsculos como sendo equivalentes.

As *stopwords*

As *stopwords* são termos que, tomados isoladamente, não contribuem para o entendimento do significado de um documento. Note então que, as *stopwords* **não** devem ser levadas em conta na

geração do índice invertido! Seu programa deve filtrar as *stopwords* pela classificação obtida com o modelo usado no pacote SpaCy

As consultas

As consultas a serem respondidas pelo sistema são compostas por termos conectados pelos operadores & (*AND*), | (*OR*) e ! (*NOT*). Assim, o sistema deve ser capaz de responder consultas como as seguintes:

- cão & gato — Leia-se: cão *AND* gato
- forno | fogão & cozinha — Leia-se: forno *OR* fogão *AND* cozinha
- avião | carro & !charrete | navio — Leia-se: avião *OR* carro *AND NOT* charrete *OR* navio

De modo a tornar a sua vida um pouco menos difícil, assuma de antemão que as consultas não podem conter parênteses e que o operador ! (*NOT*) tem precedência sobre & (*AND*), que por sua vez tem precedência sobre | (*OR*). Em outras palavras, você pode assumir que a consulta sempre é recebida já na forma normal disjuntiva. Portanto, a consulta:

avião | carro & !charrete | navio

pode ser respondida através de três tipos de documentos:

1. Documentos que contenham o termo “avião”;
2. Documentos que contenham o termo “carro” mas não contenham o termo “charrete”;
3. Documentos que contenham o termo “navio”.

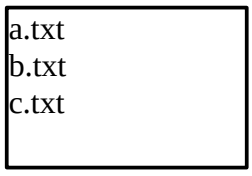
A entrada do programa

Seu programa deverá receber dois argumentos como entrada **pela linha de comando**. O primeiro argumento especifica o caminho de um arquivo texto que contém os caminhos de todos os arquivos que compõem a base, cada um em uma linha. O segundo argumento especifica o caminho de um arquivo texto que traz uma consulta a ser respondida.

Exemplo: Vamos supor que nossa base é composta pelos arquivos *a.txt*, *b.txt* e *c.txt*. Vamos supor também que nosso programa se chama *modelo_booleano.py*. Assim, chamaríamos nosso programa pela linha de comando fazendo:

> python modelo_booleano.py base.txt consulta.txt

onde o arquivo *base.txt* contém os caminhos para os arquivos que compõem a base de documentos (ressalta-se que o arquivo *base.txt* pode conter um número arbitrário de caminhos para os arquivos que compõem a base de documentos, não necessariamente 3), conforme a seguir:



base.txt

, e o arquivo *consulta.txt* possui uma consulta a ser respondida pelo sistema de RI, escrita em uma única linha no formato especificado anteriormente.

```
casa & amor | casa & !mora
```

consulta.txt

A saída do programa

O programa deverá gerar dois arquivo de saída, com nomes e conteúdo **exatamente** como a seguir:

- *indice.txt* : arquivo que contem o índice invertido gerado a partir dos documentos da base
- *resposta.txt* : arquivo com os **nomes** dos documentos que atendem a consulta do usuário segundo o modelo booleano

Obs: Os nomes dos arquivos de saída são *indice.txt* e *resposta.txt*. Note que não é *index.txt*, nem *answer.txt*, nem qualquer outra coisa. Siga os nomes especificados à risca.

O arquivo *indice.txt*:

O programa deve gerar um arquivo chamado *indice.txt*, que contem o índice invertido gerado a partir dos documentos da base.

Para cada um dos termos no índice, é preciso apontar o número do arquivo em que o mesmo aparece, e a quantidade de vezes em que o mesmo aparece no arquivo. Os arquivos são numerados segundo a ordem em que aparecem no arquivo que indica os documentos da base, que, para o nosso exemplo, foi denominado como *base.txt*. Assim, o arquivo *a.txt* é o arquivo 1, o arquivo *b.txt* é o arquivo 2 e, por fim, o arquivo *c.txt* é o arquivo 3. Suponha que estes arquivos estejam preenchidos conforme abaixo:

```
era uma CASA muito
engracada. nao tinha teto,
nao tinha nada.
```

a.txt

```
quem casa quer casa.
quem nao mora em casa,
tambem quer casa!
```

b.txt

```
quer casar comigo, amor?
quer casar comigo,
   faca o favor! Mora na
minha casa!
```

c.txt

```
amor: 3,1
casa: 1,1 2,4 3,1
casar: 3,2
comigo: 3,2
engracado: 1,1
faca: 3,1
morar: 2,1 3,1
nao: 1,2 2,1
tambem: 2,1
ter: 1,2
teto: 1,1
```

indice.txt (com lematizacao)

Para a geração de índice, recomenda-se usar uma estrutura de dicionários. Todavia, você é livre para buscar outras alternativas desde que sejam usados apenas elementos da instalação padrão do Python.

Não deixe de testar seu código. Você pode usar a ferramenta de teste disponibilizada pelo professor :

Para rodar o corretor, baixe e descompacte o arquivo corretor_indice.zip . Mova os aqruivos *.pyc para a pasta onde seu código está salvo. Abra um terminal do sistema operacional nessa mesma pasta (sim, o do sistema operacional e não o do python), e execute o comando:

python3 waxm_corretor_modelo_booleano.pyc <ARQUIVO DA BASE> <ARQUIVO DE CONSULTA> <ARQUIVO COM SEU CÓDIGO>

Se o seu sistema for Windows, talvez o comando seja esse:

py waxm_corretor_modelo_booleano.pyc <ARQUIVO DA BASE> <ARQUIVO DE CONSULTA> <ARQUIVO COM SEU CÓDIGO>

Por exemplo, supondo que o arquivo que especifica a base se chame base.txt e seu código esteja em um arquivo chamado indice.py, faça:

python3 waxm_corretor_modelo_booleano.pyc base.txt consulta.txt modelo_booleano.py

ou

py waxm_corretor_modelo_booleano.pyc base.txt consulta.txt modelo_booleano.py

Você também pode baixar as bases de exemplo para testar seu código.

Atenção: se o seu código não passar satisfatoriamente pelo corretor automático, seu trabalho já começa a ser corrigido com desconto de pontuação. USE O CORRETOR AUTOMÁTICO! E caso tenha qualquer problema em usar o corretor, entre em contato (em alguns sistemas, pode ocorrer algum problema de compatibilidade)!