

## Segundo trabalho de Organização e Recuperação da Informação 2020-02

### Descrição

Objetivo do trabalho:

1. Implementação do modelo booleano de RI que deverá utilizar o índice invertido gerado no trabalho 1 para responder a uma consulta.

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 indicado pelo professor para tal até o dia 09/07/2021.

**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 obrigatório o uso do pacote nltk para a extração de radicais dos termos do vocabulário e obtenção de uma lista válida de *stopwords*. Os detalhes sobre a geração do índice invertido e o processamento das consultas 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 considerar a lista de stopwords para a língua portuguesa disponível no pacote nltk, conforme visto em aula. (veja os exemplos da aula do dia 24/09/2019).

### 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 e evitar chororô, assuma de ante-mão 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:

a.txt  
b.txt  
c.txt

*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*

Vamos supor que a nossa lista de *stopwords*, obtida do pacote nltk, contenha os seguintes termos:

o  
de  
na  
em  
não  
uma

*lista de stopwords*

## A saída do programa

O programa deverá gerar dois arquivos 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

### 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, de forma idêntica ao trabalho 1.

Para cada um desses radicais no índice, é preciso apontar o número do arquivo em que o mesmo aparece, e a quantidade de vezes em que a 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. Não tinha teto,  
não tinha nada.

*a.txt*

quem casa quer casa.  
QUEM não mora em  
casa, também quer casa!

*b.txt*

quer casar comigo, amor?  
quer casar comigo,  
faça o favor!  
mora na minha casa!

*c.txt*

```
am: 3,1
cas: 1,1 2,4 3,3
comig: 3,2
engraç: 1,1
faç: 3,1
favor: 3,1
mor: 2,1 3,1
nad: 1,1
qu: 2,2 3,2
tet: 1,1
```

*indice.txt (com extração de radicais)*

### **O arquivo *resposta.txt***

O arquivo *resposta.txt* contém a resposta à consulta contida no arquivo de consulta, no nosso exemplo, *consulta.txt*. A primeira linha desse arquivo deve conter a quantidade de documentos que satisfazem a consulta. As demais linhas contém os arquivos da base que atendem a consulta, um por linha, conforme o exemplo a seguir, onde respondemos a consulta especificada em nosso arquivo *consulta.txt*. Note que será preciso considerar todas as disjunções da consulta, além de desconsiderar *stopwords* presentes na mesma e também extrair os radicais de seus termos:

```
2
a.txt
c.txt
```

*resposta.txt*

Para construir a resposta à consulta, seu programa deve se basear exclusivamente no conteúdo do índice invertido gerado! Isto quer dizer que, após a construção do índice invertido, o conteúdo dos documentos da base não deve mais ser diretamente utilizado! Para impressionar seu professor e mostrar que compreendeu a utilidade do índice invertido, procure usar a estrutura do índice gerado em memória para obter o conjunto de documentos que atendem a consulta (Não é preciso ler o índice invertido do arquivo de saída *indice.txt*).

Não deixe de testar seu código. Você pode usar a ferramenta de teste disponibilizada pelo professor nos arquivos da disciplina no Teams:

Para rodar o corretor, baixe e descompacte o arquivo corretor\_modelo\_booleano.zip . Mova os arquivos \*.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, o arquivo de consulta seja consulta.txt e seu código esteja em um arquivo chamado modelo\_booleano.py, faça:

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

ou

```
py waxm_corretor_modelo_booleano.py c base.txt consulta.txt modelo_booleano.py
```

Você também pode baixar as bases de exemplo nos arquivos da disciplina