Primeira Fase do Trabalho de Recuperação de Informação — v.02/10/2020

Caros alunos, nesta primeira fase do trabalho de Recuperação de Informação vocês devem montar um primeiro Sistema de Recuperação de Informação que venha a indexar um conjunto de dados bem desestruturados: os arquivos textos que vocês possuem. Vocês deverão elaborar dois programas: o *indexador do Meu sistema de Recuperação de Informação*, `mir`; e o *Buscador do Meu sistema de Recuperação de Informação*, `mirs`.
Deve-se escrever os utilitários em python 3 e sugere-se que se use o pacote `argparse` para processar as opções de linha de comando.

***

O indexador `mir` recebe na linha de comando o nome de um `diretório` cuja sub árvore de diretório nele pendurada será indexada. Primeiramente o utilitário deve listar e enumerar recursivamente todos os arquivos com a extensão `.txt` que se encontram na sub árvore do `diretório` em questão. Por exemplo,

```
mir.py diretório
0 diretório/arq0.txt
1 diretório/arq1.txt
2 diretório/arq2.txt
3 diretório/subdirA/arq3.txt
4 diretório/subdirA/arq4.txt
5 diretório/subdirA/subdirB/arq5.txt
6 diretório/subdirC/arq6.txt
7 diretório/xarq7.txt
Foram encontradas 8 documentos.
```

Uma *lista de arquivos* armazenará os nomes. Ademais, os números desta enumeração serão seus índices na lista e *identificadores* únicos destes arquivos. Esta lista mapeia os identificadores nos caminhos completos dos arquivos no sistema de arquivos.
Em seguida, o utilitário deve indexar os documentos e criar o índice invertido, que a cada termo associa os documentos que o contêm. Adotar-se-á a solução usual de um aqruivo invertido, a estrutura proposta no livro-texto. Assim, deve-se carregar um dicionário com o vocabulário dos documentos, associando a cada token as listas de incidência com os índices únicos que identificam os documentos (os números de 0 a 7 no exemplo acima).
Nesta fase, não é necessário armazenar a frequência dos termos nos documentos nem montar um índice posicional, recomendado no caso de busca por frase.
Depois de montado o dicionário, deve ser escrito um sumário com informações como as seguintes, a menos dos números:

```
Os 7 documentos foram processados e produziram um total de 10000
tokens, que usaram um vocabulário com 900 tokens distintos.
Informações salvas em diretório/mir.pickle para carga via pickle.
```

Devem ser armazenadas via `pickle`, nesta ordem, a lista de arquivos com os caminhos completos dos documentos e o dicionário que mapeia os tokens em suas respectivas listas de incidência.

***

Já o buscador `mirs` recebe como parâmetros na linha de comando o `diretório` onde irá buscar o arquivo `pickle` produzido pelo indexador `mir` e uma lista com cada `termo` da consulta (conjuntiva). Deve-se listar os nomes de todos os documentos que contêm cada um dos termos da lista de termos, a menos que sejam muitos os arquivos. Ao final, deve-se também informar quantos são estes arquivos, como na execução abaixo:

```
mirs diretório MAC0333 é bom demais
Carregado índice diretório/mir.pickle com vocabulário de 900 tokens e
lista de 8 documentos.
Arquivos que satisfazem a consulta conjuntiva:
  5 diretório/subdirA/subdirB/arq5.txt
  6 diretório/subdirC/arq6.txt
  7 diretório/xarq7.txt
Foram encontrados 3 documentos com os termos: MAC0333 é bom demais.

mirs -l 2 diretório MAC0333 é bom demais
Carregado índice diretório/mir.pickle com vocabulário de 900 tokens e
lista de 8 documentos.
O limite de 2 arquivos por listagem da consulta foi atingido.
Foram encontrados 3 documentos com os termos: MAC0333 é bom demais.
```

***

Para produzir a lista de arquivos recomenda-se olhar a documentação do pacote `os` e métodos como `os.listdir`, `os.path.isfile`, `os.path.isdir`, `os.path.join`, `os.path.splitext`, `os.stat`, etc.
Os arquivos texto podem estar escritos em diferentes línguas e podem estender a tabela ASCII de diversas formas (UTF-8, UTF-16, ISO-8859-1, Windows-1252, etc), mas internamente os tokens devem ser todos codificados em utf-8 e serão convertidos para letras minúsculas. Adiante será fornecida informação de como tratar arquivos com outras codificações.
Como padrão, não se deve nem fazer *steaming* nem usar *stop lists*.
Serão fornecidos alguns arquivos `.zip` com estrutura de diretório de arquivos para se testar o sistema e este enuciado poderá sofrer atualizações. O material estará disponível em http:www.ime.usp.br/∼alair/mac0333 .

****

Adendo 6/10: Com relação à codificação de arquivos texto, tem se tornado padrão o uso da extensão da tabela ASCII conhecida como UTF-8, que quebra a convensão de que cada caractere usa apenas um byte para permitir a representação dos caracteres acentuados das línguas que usam o alfabeto latino ou mesmo outros alfabetos. Em python 3, quando se abre para leitura um arquivo em modo texto com `open( NomeDoArquivo, 'r' )`, é implícita a codificação padrão `encoding='utf-8'`.
Quando fomos preparar este trabalho e codificar uma solução para ele, deparamo-nos com uma dificuldade: em muitos arquivos, a leitura do arquivo texto produzia palavras que acusavam erros do tipo `UnicodeDecodeError` em muitas palavras. Ocorre que dos milhares de arquivos `.txt` que temos acumulando há algumas décadas, apenas os 7% que usam o formato ASCII e os 49% que usam o formato UTF-8 foram decodificados. Nos 44% que usam o formato ISO-8859-1 e nos 0.2% que usam sua extensão Windows-1252, uma exceção com o erro acima era levantada e o indexador abortava. Mesmo quem tenha uma história computacional não tão longa pode vir a enfrentar o mesmo problema caso tenha baixado algum arquivo na internet produzido a partir de material mais antigo. Esta é uma dificuldade inerente à manipulação de textos escritos em diversas línguas e diversas codificações.
Parte da solução que propomos faz uso do pacote `chardet`.

```
import chardet
def GetFileEncoding(file_path):
    """ Get the encoding of file_path using chardet package  """
    with open(file_path, 'rb') as f:
        return chardet.detect( f.read() )
```

Assim, a leitura em modo texto pode ser feita usando a codificação detectada.

```
...
enc = GetFileEncoding( ListaDeArquivos[ifile] )
encoding = enc['encoding']
confidence = float(enc['confidence'])*100
print( '%5d %-10s %4.1f%% %s' % (ifile, encoding, confidence, ListaDeArquivos[ifile])
if confidence < 63: myerr = 'replace'
else: myerr = 'strict'
with open( ListaDeArquivos[ifile], 'r', encoding=encoding, errors=myerr ) as filehand
    # Lê arquivo texto, quebrando em linhas e em palavras, gera tokens,
    # insere-os no dicionário e  atualiza listas de incidência
    ...
```

Infelizmente, a detecção acima ainda não garante a corretude da decodificação em todas as situações, particularmente em arquivos com codificação reconhecida como `'Windows-1252` ou assemelhados e nos quais se observa a inserção de caracteres inválidos. Isto também diminui a taxa confiança atribuída à codificação detectada e, por esta razão, quando a confiança for abaixo de um limiar (`63`), relaxamos o rigor no tratamento de erros de decodificação `'strict'` de modo a substituir caracteres inválidos pelo símbolo `'?'` (`'replace'`). Além

de quebrar a linha em palavras e convertê-las para minúsculas (vide `str.lower()`), faz-se também necessária a eliminação de caracteres que não são letras de uma palavra. O código abaixo usa o pacote de expressões regulares da Python 3 para eliminação de símbolos de pontuação, tanto caracteres estritamente da tabela ASCII quanto do alfabeto estendido. A função substitui (`re.sub`) por espaço (`' '`) qualquer sequência de caracteres que: não (`[^`) façam parte de identificadores (`\w`) nem sejam tratados como espaços (`\s]`); ou ainda (`|`) que sejam dígitos (`\d`) ou (`|`) sublinhado (`_`). (Juntamente com as letras do alfabeto, acentuadas ou não, os dígitos e o sublinhado são caracteres aceitos como parte dos identificadores.)

```
def StripPonctuation(s):
    # return #s.translate(mytable)
    tokens = re.sub(r'([^\w\s]|\d|_)+', ' ', s)
    return tokens.split()
```

Outras soluções podem ser tentadas caso se queira, inclusive com o uso de `re.finditer`, `re.split`, `str.translate`, etc.
Observe que nesta implementação nós não estamos inserindo nem números nem datas no vocabulário.
Oportunamente será fornecido um arquivo `.zip` de uma árvore de diretório com diversos arquivos-exemplo. Deve-se também implementar a opção `-t nn` no buscador `mirs`, que lista os `nn` tokens mais frequentes nos documentos. (Como será visto no capítulo VI, a frequência de documento (df) de um termo/token é também o comprimento de sua lista de incidências.)