

## Segunda Fase do Trabalho de Recuperação de Informação — v.06/11/2020

Caros alunos,

Tal como visto na aula do dia 4 de novembro,<sup>1</sup> na segunda fase do trabalho da disciplina vocês devem estender o *My Information Retrieval* construído na primeira fase<sup>2</sup>, de forma a:

- Fazer uma **indexação dinâmica** (tal como descrito no capítulo 4). Por simplicidade, na indexação dinâmica adotaremos a solução de *um único índice auxiliar*, que indexa os arquivos recentemente modificados na sub-árvore do diretório-raiz. Assim, o sistema deve ser capaz de combinar os resultados da busca feita nos dois índices, o principal e o auxiliar.
- Construir um **índice posicional** (tal como descrito nos capítulos 1 e 2). Para a construção do índice posicional, *cada elemento da lista de incidências é uma tripla* contendo:
  - o *identificador* do documento;
  - a *frequência* do termo no documento;
  - o índice do *início* da lista de posições dentro de uma lista geral, da qual a lista de posições corresponde à fatia *[início:início+frequência]*. Esta lista geral é formada pela concatenação de todas as listas de posições, ordenadas pelo identificador do termo, como primeira chave, e pelo identificador de documento, como segunda chave. Esta lista geral de posições é volumosa e nem sempre é usada, sendo guardada num arquivo à parte.

Observe que se produz uma lista com todos os termos do vocabulário ao se aplicar o método `keys()` ao dicionário que mapeia cada termo em sua lista de incidências. (A propósito, em versões mais recentes da Python 3 ( $\geq 3.6.9$ ), esta lista é ordenada temporalmente: novos termos inseridos no dicionário são acrescentados ao final da lista.) Assim sendo, podemos associar a cada termo do vocabulário um número identificador: o índice deste termo nesta lista produzida pelo método `keys()` e que mapeia um identificador de termo ao próprio termo. Este processo é em parte semelhante ao que foi feito para associar a cada documento um identificador.

Porque o índice é posicional e permite atualização dinâmica, devemos introduzir mudanças nas informações armazenadas via `pickle`, que obedecerão a um novo formato. As seguintes informações devem ser gravadas nesta ordem nos arquivos de índice gerados:

1. a *cadeia* de caracteres ‘MIR 2.0’;
2. a *lista de arquivos*, com o *caminho* relativo ao **diretório** raiz deste arquivo;

---

<sup>1</sup>Com vídeo amador em <https://drive.google.com/file/d/1JPGIr7XDBFRVQ9Dev0qCjEYFkWCxk1WY> .

<sup>2</sup>Disponível em <http://www.ime.usp.br/~alair/mac0333/MeuPrimeiroRecInfo20201030.pdf> .

3. o *dicionário* que mapeia os termos do *vocabulário* às triplas detalhadas acima, através das quais obtêm-se suas respectivas listas de incidência;
4. um *dicionário* que complementa as informações presentes na lista de arquivos supra citada. A cada arquivo, ele associa: `None`, caso o arquivo tenha sido ‘removido’ no índice atualizado dinamicamente; ou um *dicionário* com as seguintes informações do arquivo:
  - seu respectivo ‘encoding’;
  - a respectiva ‘confidence’ desta classificação;
  - seu tratamento de ‘errors’;
  - seu ‘tamanho’ relatado por `os.path.getsize()`;
  - o instante em que foi ‘modificado’ pela última vez, (via `os.path.getmtime()`);
5. o instante do *início da indexação* devolvido pela função `time.time()`.

A seguinte reescritura da função `get_file_encoding` (antiga `GetFileEncoding`) fornece uma solução suficiente para a montagem do dicionário com informações complementares para os arquivos.

```
def get_file_encoding(ref_path, file_rel_path, file_encoding):
    """
    Get the encoding of ref_path / file_rel_path using chardet package
    Besides 'encoding' e 'confidence', it sets an extra field: 'errors'.
    FILE_ENCODING salva instruções prescritas e serve de cache de resultados
    """
    if file_rel_path in file_encoding:
        return file_encoding[ file_rel_path ]
    file_path = os.path.join( ref_path, file_rel_path)

    with open(file_path, 'rb') as f:
        file_encoding[ file_rel_path ] = chardet.detect( f.read(MAXSIZE) )
        enc = file_encoding[ file_rel_path ]['encoding']
        size = os.stat( file_path ).st_size
        if size > MAXSIZE and enc=='ascii':
            file_encoding[ file_rel_path ]['encoding'] = 'UTF-8'
            file_encoding[ file_rel_path ]['confidence'] = 0.4
            myerr = 'mixed'
        elif file_encoding[ file_rel_path ]['confidence'] < .63 :
            myerr = 'replace'
        else:
            myerr = 'strict'
        file_encoding[ file_rel_path ]['errors'] = myerr
```

```

        file_encoding[ file_rel_path ]['tamanho'] = size
        file_encoding[ file_rel_path ]['modificado'] = os.path.getmtime( file_path )
    return file_encoding[ file_rel_path ]

```

Ademais, pode-se supor que uma vez escrito o índice principal, ele não mais será alterado, até que seja refeito por completo, nem mesmo o campo `file_encoding`. No caso de haver um índice auxiliar, é tarefa de quem carrega os dois índices colocar `None` nalgumas entradas do dicionário `file_encoding` do índice principal, conforme o arquivo tenha sido removido ou atualizado.

Por conta da atualização dinâmica, o sistema deve ser capaz de combinar os resultados da busca feita nos dois índices: o principal e o auxiliar. Ademais, deve-se ser capaz de detectar que arquivos foram alterados e que arquivos foram inseridos nalgum subdiretório desde a última indexação, vindo a definir uma nova lista com os arquivos que precisam ser indexados no índice auxiliar.

Além do índice principal presente em `mir.pickle`, deve ser gravado o índice auxiliar em `mira.pickle`, no mesmo diretório. Ademais, para especificar arquivos que foram removidos do sistema desde a criação do índice principal, deve-se armazenar seus nomes no arquivo de instruções de exclusão `mira.rem`, também no mesmo diretório.

Recomendamos que primeiro se faça uma segunda versão da solução da primeira fase que seja uma reescritura do código feito, com a definição de uma interface sobre a qual o sistema de RI da primeira fase constitui uma implementação. Nesta segunda fase acrescenta-se então uma nova implementação desta interface, agora incorporando a união do índice auxiliar ao índice principal bem como a infraestrutura do índice posicional.

Este enunciado está disponível na página da disciplina e deve sofrer acréscimos.

Neste mesmo sítio foi deixado o arquivo `shell.out` com a execução de vários exemplos de execução de uma implementação do algoritmo solicitado na corrente fase do trabalho. Estes exemplos são comentados em vídeo. Em particular, é comentado que não há necessidade de se implementar a opção `-g 0`, que tem o efeito de produzir lista dos arquivos a serem indexados no índice auxiliar sem entretanto indexá-los. Nem tampouco é necessário incluir *no indexador mir* a opção `-t`.

O buscador `mirs` deve primeiro procurar pelo índice no diretório fornecido, verificando se há a presença de um índice auxiliar. Caso houver, a busca deve se dar no índice combinado. Já o indexador `mir` deve produzir o índice principal `diretório/mir.pickle`. Contudo, com a opção `-A`, ele deve produzir o índice auxiliar `diretório/mir.pickle`, deixando intacto o principal. Para produzi-lo, use o mesmo procedimento que recursivamente produz uma listagem com todos os arquivos da sub-árvore do diretório usado na produção do índice principal. Os arquivos devem ser comparados, de forma que os arquivos a serem indexados e presentes no índice auxiliar são os da listagem atual que não constam no índice principal e os que constam mas possuem tamanho ou última hora de modificação diferentes.

O índice posicional deve ser arquivado à parte como discutido, num arquivo `mirp.pickle`. Sua implementação não será cobrada nesta fase. Bom trabalho a todos.