Trabalho Prático da Matéria MapReduce - MBED-1

Objetivo: Desenvolver uma versão do contador de palavras usando os conceitos de MapReduce.

Grupo:

Leonardo Chalhoub
Mateus Luz
Murilo Arguello
William Dener

Introdução

O trabalho foi dividido em 7 etapas com o intuito de cobrir os inputs, criação das funções, e validação do script. Sendo assim, cada etapa será mostrada a seguir e o script total será apresentado no final. Além disso, o código na extensão ".ipynb" pode ser encontrado em anexo junto com esse documento.

1. Carregamento das bibliotecas

Será necessário fazer a importação das bibliotecas "re" e "reduce", as quais serão utilizadas ao longo do script.

In [ ]:
```
import re
from functools import reduce
```

1. Definição do caminho para o Texto.

Em seguida, o caminho para o texto "Beyond Good and Evil - Friedrich Nietzsche" na extensão ".txt" deve ser importado.

In [ ]:
```
nome = 'Beyond Good and Evil - Friedrich Nietzsche.txt'
```

1. Criação da função mapper

Agora é necessário criar a primeira função requisitada para o trabalho. A função "mapper" será criada para receber o caminho do arquivo na extensão ".txt" e retornar uma lista de tuplas para cada palavra encontrada.

In [ ]:
```
def mapper(nome):
    '''
    objetivo: Receber um arquivo com um texto e retornar uma lista de tuplas
    para poder fazer posteriormente a contagem.

    Input:
        nome (path): caminho para o local do texto que deve estar no formato .txt.

    Output:
        tuplas (list): lista de tuplas com as palavras do texto e o número "1".
```

```
    '''
    arquivo = open(nome, 'r', encoding='utf-8-sig')
    linhas = map(lambda linha: re.sub('[^A-Za-z0-9]+', ' ', linha).lower().split(),
                 arquivo)
    palavras = reduce(lambda x, y: x + y, linhas)
    tuplas = list(map(lambda palavra: (palavra, 1), palavras))
    return tuplas
```

1. Criação da função partitioner

O próximo passo é a criação da segunda função, partitioner, que irá converter a lista inicial criada pelo mapper para uma lista de tuplas, mas com o segundo elemento da tupla sendo uma lista com o número "1" para cada ocorrência no texto.

In [ ]:

```
def partitioner(lista):
    '''
    objetivo: Receber uma lista do tuplas e retornar uma lista de tuplas sendo o
    segundo elemento uma lista com o número "1" para cada ocorrência da palavra
    do primeiro item da tupla.

    Input:
        lista (list): lista de tuplas.

    Output:
        listaRepeticoesPalavras (list): lista de tuplas.
    '''
    listaTuplasParticao = list(map(lambda linha: (linha[0], [linha[1]]), lista))
    dicionario = {}
    for palavra in lista:
        if palavra[0] in dicionario:
            dicionario[palavra[0]].append(palavra[1])
        else:
            dicionario[palavra[0]] = [palavra[1]]
    listaRepeticoesPalavras = []
    for repeticao in dicionario.items():
        listaRepeticoesPalavras.append(repeticao)
    return listaRepeticoesPalavras
```

1. Criação da função reduce

A terceira e última função que deve ser criada é a "reduce", a qual irá transformar a lista gerada pela função partitioner em uma lista com tuplas, porém dessa vez com a contagem de ocorrências para cada palavra.

In [ ]:

```
def reducer(lista):
    '''
    objetivo: Receber uma lista do tuplas e retornar uma lista de tuplas sendo o
    segundo elemento o número de ocorrências da palavra do primeiro item da tupla.
    Além disso, a função ainda ordena a lista para apresentar da palavra com
    maior ocorrência para a menor.

    Input:
        lista (list): lista de tuplas.

    Output:
        listaReduceOrdenada (list): lista de tuplas.
    '''
    listaReduce = list(map(lambda tupla: (tupla[0], reduce(lambda x, y: x + y,
```

```
                                                    tupla[1])), lista))

    listaReduceOrdenada = sorted(listaReduce, key = lambda x: x[1], reverse = True)

    return listaReduceOrdenada
```

1. Aplicação das funções no texto

Finalmente, é necessário aplicar as funções criadas no texto para poder apresentar o resultado final.

In [ ]:
```
listaTuplas = mapper(nome)
listaTuplasParticao = partitioner(listaTuplas)
listaTuplasOcorrencia = reducer(listaTuplasParticao)
print(listaTuplasOcorrencia)
```

1. Apresentação do código inteiro

Todas as etapas do códgio estão apresentadas a seguir para caso seja desejado rodar o script de uma forma direta.

In [ ]:
```
import re
from functools import reduce

def mapper(nome):
    '''
    objetivo: Receber um arquivo com um texto e retornar uma lista de tuplas para
     poder fazer posteriormente a contagem.

    Input:
        nome (path): caminho para o local do texto que deve estar no formato .txt.

    Output:
        tuplas (list): lista de tuplas com as palavras do texto e o número "1".
    '''
    arquivo = open(nome, 'r', encoding='utf-8-sig')
    linhas = map(lambda linha: re.sub('[^A-Za-z0-9]+', ' ', linha).lower().split(),
                 arquivo)
    palavras = reduce(lambda x, y: x + y, linhas)
    tuplas = list(map(lambda palavra: (palavra, 1), palavras))
    return tuplas

def partitioner(lista):
    '''
    objetivo: Receber uma lista do tuplas e retornar uma lista de tuplas sendo o
    segundo elemento uma lista com o número "1" para cada ocorrência da palavra
    do primeiro item da tupla.

    Input:
        lista (list): lista de tuplas.

    Output:
        listaRepeticoesPalavras (list): lista de tuplas.
    '''
    listaTuplasParticao = list(map(lambda linha: (linha[0], [linha[1]]), lista))
    dicionario = {}
    for palavra in lista:
        if palavra[0] in dicionario:
            dicionario[palavra[0]].append(palavra[1])
```

```
        else:
            dicionario[palavra[0]] = [palavra[1]]
    listaRepeticoesPalavras = []
    for repeticao in dicionario.items():
        listaRepeticoesPalavras.append(repeticao)
    return listaRepeticoesPalavras

def reducer(lista):
    '''
    objetivo: Receber uma lista do tuplas e retornar uma lista de tuplas sendo o
    segundo elemento o número de ocorrências da palavra do primeiro item da tupla.
    Além disso, a função ainda ordena a lista para apresentar da palavra com
    maior ocorrência para a menor.

    Input:
        lista (list): lista de tuplas.

    Output:
        listaReduceOrdenada (list): lista de tuplas.
    '''
    listaReduce = list(map(lambda tupla: (tupla[0], reduce(lambda x, y: x + y,
                                                           tupla[1])), lista))

    listaReduceOrdenada = sorted(listaReduce, key = lambda x: x[1], reverse = True)

    return listaReduceOrdenada

if __name__ == "__main__":

  nome = 'Beyond Good and Evil - Friedrich Nietzsche.txt'

listaTuplas = mapper(nome)
listaTuplasParticao = partitioner(listaTuplas)
listaTuplasOcorrencia = reducer(listaTuplasParticao)
print(listaTuplasOcorrencia)
```