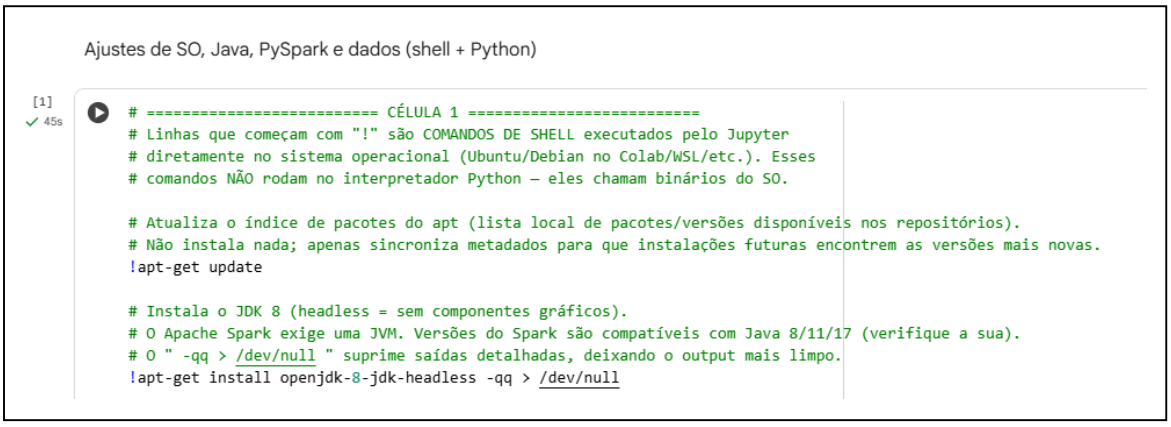
Pedro Sergio Gomes Quinderé

Operação Passo 1 durou 45 segundos:



Resultando em:

========================= CÉLULA 1 =========================

Linhas que começam com "!" são COMANDOS DE SHELL executados pelo Jupyter

diretamente no sistema operacional (Ubuntu/Debian no Colab/WSL/etc.). Esses

comandos NÃO rodam no interpretador Python — eles chamam binários do SO.

Atualiza o índice de pacotes do apt (lista local de pacotes/versões disponíveis nos repositórios).

Não instala nada; apenas sincroniza metadados para que instalações futuras encontrem as versões mais novas.

!apt-get update

Instala o JDK 8 (headless = sem componentes gráficos).

O Apache Spark exige uma JVM. Versões do Spark são compatíveis com Java 8/11/17 (verifique a sua).

O " -qq > /dev/null " suprime saídas detalhadas, deixando o output mais limpo.

!apt-get install openjdk-8-jdk-headless -qq > /dev/null

---------- A partir daqui é Python ----------

import os # Módulo padrão do Python para interagir com o Sistema Operacional (variáveis de ambiente, paths, etc.)

```
# Variável de ambiente JAVA_HOME: aponta para o diretório raiz do JDK.
# Diversas ferramentas (incluindo o Spark) usam isso para localizar 'java' e 'javac'.
os.environ["JAVA_HOME"] = "/usr/lib/jvm/java-8-openjdk-amd64"

# update-alternatives permite escolher qual versão do 'java' será o padrão do sistema
# quando existem múltiplas instalações. Aqui, fixamos a versão instalada acima.
!update-alternatives --set java /usr/lib/jvm/java-8-openjdk-amd64/jre/bin/java

# Checa a versão ativa do Java para confirmar que a configuração está OK.
!java -version

# Instala o cliente PySpark (biblioteca Python que conversa com o engine do Spark).
# Sem isso, "from pyspark import ..." não funcionaria.
!pip install pyspark

# Baixa (clona) o repositório com os arquivos de exemplo e os datasets (logs da NASA).
# "git clone <url>" cria uma pasta local "aulapython/" com o conteúdo remoto.
!git clone https://github.com/leonardoamorim/aulapython.git

# Lista os arquivos dentro de "aulapython/" para verificar o clone com sucesso.
! ls aulapython/

# Descompacta os arquivos de log (.gz) para .txt (sem compressão), assim o Spark e o shell
# conseguem ler normalmente. 'gunzip' substitui o arquivo .gz pelo extraído.
! gunzip aulapython/NASA_access_log_Jul95.gz
! gunzip aulapython/NASA_access_log_Aug95.gz

# Mostra o tamanho (uso de disco) de cada arquivo (opção -h = "human readable":
KB/MB/GB).
! du -h aulapython/NASA_access_log_Jul95
```

```
! du -h aulapython/NASA_access_log_Aug95


# Conta quantas linhas existem (cada linha = 1 requisição registrada).
! wc -l aulapython/NASA_access_log_Jul95
! wc -l aulapython/NASA_access_log_Aug95


# Mostra as 10 primeiras linhas do arquivo de Julho para inspecionar o formato:
# Formato comum (Common Log Format estendido):
# host ident authuser [date:time zone] "request" status bytes
! head -n10 aulapython/NASA_access_log_Jul95
# ========================= CÉLULA 1 =========================
# Linhas que começam com "!" são COMANDOS DE SHELL executados pelo Jupyter
# diretamente no sistema operacional (Ubuntu/Debian no Colab/WSL/etc.). Esses
# comandos NÃO rodam no interpretador Python — eles chamam binários do SO.


# Atualiza o índice de pacotes do apt (lista local de pacotes/versões disponíveis nos repositórios).
# Não instala nada; apenas sincroniza metadados para que instalações futuras encontrem as versões mais novas.
!apt-get update


# Instala o JDK 8 (headless = sem componentes gráficos).
# O Apache Spark exige uma JVM. Versões do Spark são compatíveis com Java 8/11/17 (verifique a sua).
# O " -qq > /dev/null " suprime saídas detalhadas, deixando o output mais limpo.
!apt-get install openjdk-8-jdk-headless -qq > /dev/null


# ---------- A partir daqui é Python ----------
import os  # Módulo padrão do Python para interagir com o Sistema Operacional (variáveis de ambiente, paths, etc.)


# Variável de ambiente JAVA_HOME: aponta para o diretório raiz do JDK.
```

```
# Diversas ferramentas (incluindo o Spark) usam isso para localizar 'java' e 'javac'.
os.environ["JAVA_HOME"] = "/usr/lib/jvm/java-8-openjdk-amd64"

# update-alternatives permite escolher qual versão do 'java' será o padrão do sistema
# quando existem múltiplas instalações. Aqui, fixamos a versão instalada acima.
!update-alternatives --set java /usr/lib/jvm/java-8-openjdk-amd64/jre/bin/java

# Checa a versão ativa do Java para confirmar que a configuração está OK.
!java -version

# Instala o cliente PySpark (biblioteca Python que conversa com o engine do Spark).
# Sem isso, "from pyspark import ..." não funcionaria.
!pip install pyspark

# Baixa (clona) o repositório com os arquivos de exemplo e os datasets (logs da NASA).
# "git clone <url>" cria uma pasta local "aulapython/" com o conteúdo remoto.
!git clone https://github.com/leonardoamorim/aulapython.git

# Lista os arquivos dentro de "aulapython/" para verificar o clone com sucesso.
! ls aulapython/

# Descompacta os arquivos de log (.gz) para .txt (sem compressão), assim o Spark e o shell
# conseguem ler normalmente. 'gunzip' substitui o arquivo .gz pelo extraído.
! gunzip aulapython/NASA_access_log_Jul95.gz
! gunzip aulapython/NASA_access_log_Aug95.gz

# Mostra o tamanho (uso de disco) de cada arquivo (opção -h = "human readable":
KB/MB/GB).
! du -h aulapython/NASA_access_log_Jul95
! du -h aulapython/NASA_access_log_Aug95
```

Conta quantas linhas existem (cada linha = 1 requisição registrada).

! wc -l aulapython/NASA_access_log_Jul95

! wc -l aulapython/NASA_access_log_Aug95

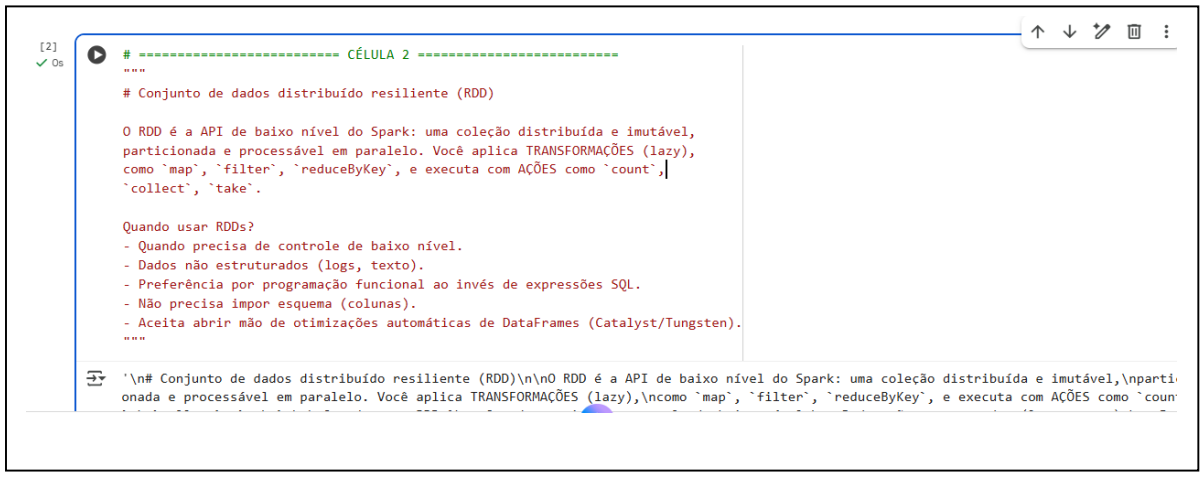
Mostra as 10 primeiras linhas do arquivo de Julho para inspecionar o formato:

Formato comum (Common Log Format estendido):

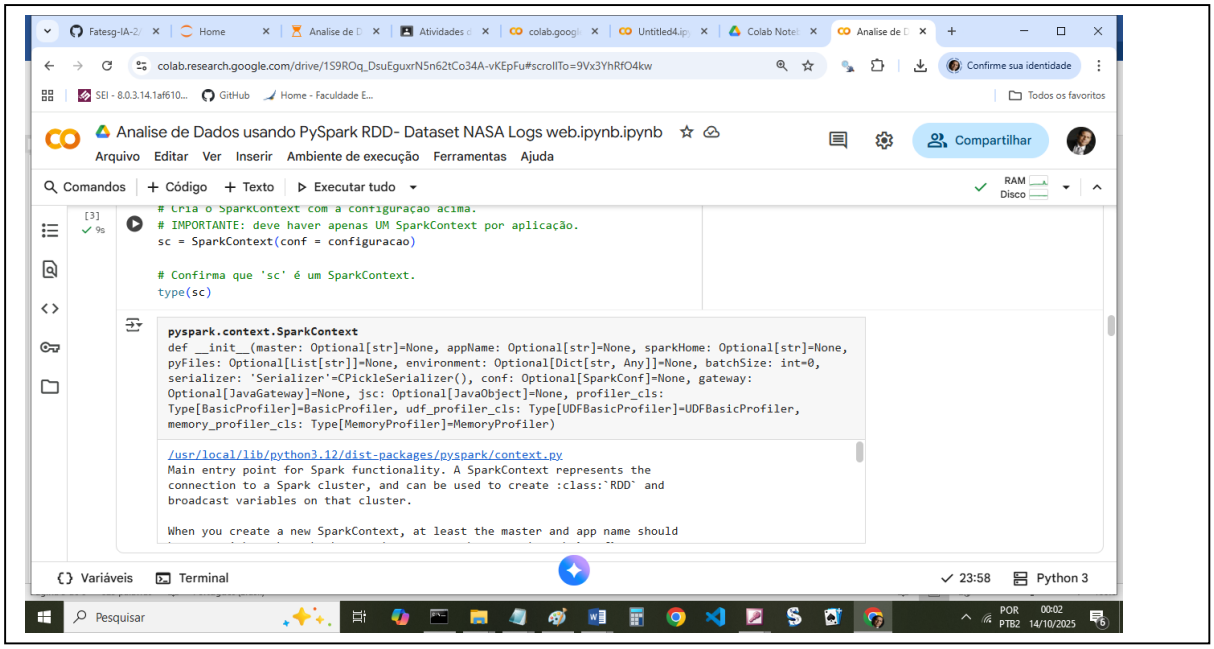
host ident authuser [date:time zone] "request" status bytes

! head -n10 aulapython/NASA_access_log_Jul95

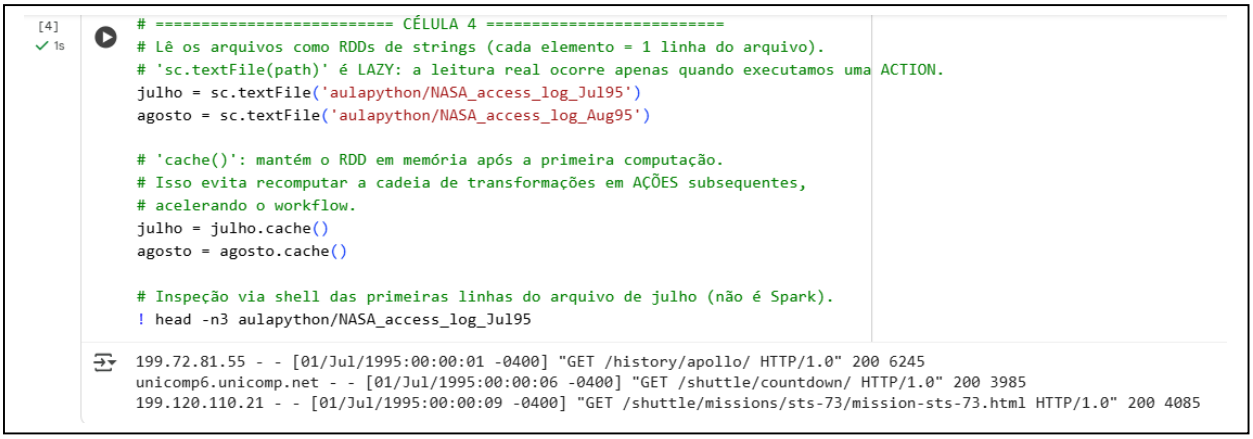
Operação Passo 2 durou 0 segundo:



Operação Passo 3 durou 9 segundos:

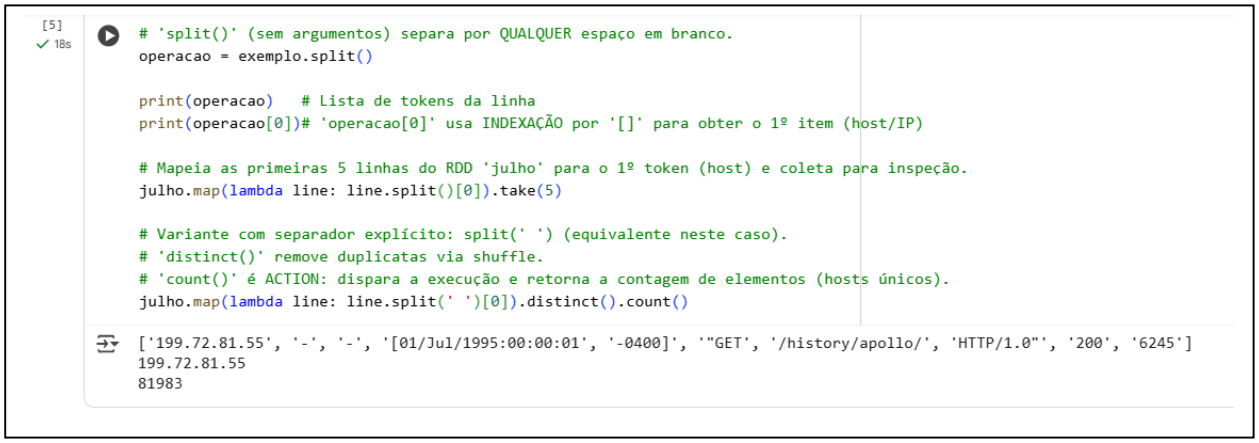


Operação Passo 4 durou 4 segundos:



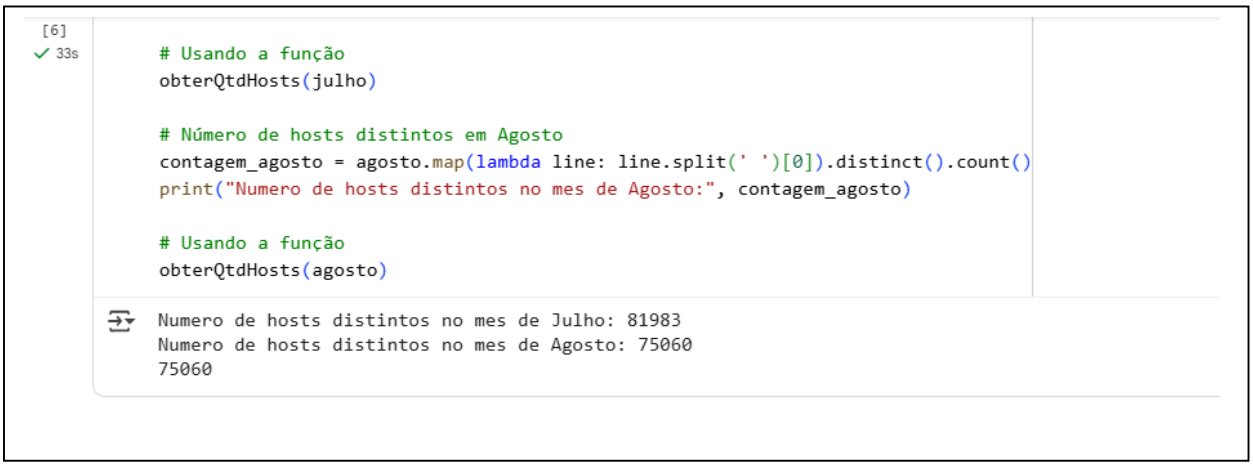
```
[4]
✓ 1s
# ========================== CÉLULA 4 ==========================
# Lê os arquivos como RDDs de strings (cada elemento = 1 linha do arquivo).
# 'sc.textFile(path)' é LAZY: a leitura real ocorre apenas quando executamos uma ACTION.
julho = sc.textFile('aulapython/NASA_access_log_Jul95')
agosto = sc.textFile('aulapython/NASA_access_log_Aug95')

# 'cache()': mantém o RDD em memória após a primeira computação.
# Isso evita recomputar a cadeia de transformações em AÇÕES subsequentes,
# acelerando o workflow.
julho = julho.cache()
agosto = agosto.cache()

# Inspeção via shell das primeiras linhas do arquivo de julho (não é Spark).
! head -n3 aulapython/NASA_access_log_Jul95

199.72.81.55 - - [01/Jul/1995:00:00:01 -0400] "GET /history/apollo/ HTTP/1.0" 200 6245
unicomp6.unicomp.net - - [01/Jul/1995:00:00:06 -0400] "GET /shuttle/countdown/ HTTP/1.0" 200 3985
199.120.110.21 - - [01/Jul/1995:00:00:09 -0400] "GET /shuttle/missions/sts-73/mission-sts-73.html HTTP/1.0" 200 4085
```

Operação Passo 5 durou 18 segundos:



```
[5]
✓ 18s
# 'split()' (sem argumentos) separa por QUALQUER espaço em branco.
operacao = exemplo.split()

print(operacao)   # Lista de tokens da linha
print(operacao[0])# 'operacao[0]' usa INDEXAÇÃO por '[]' para obter o 1º item (host/IP)

# Mapeia as primeiras 5 linhas do RDD 'julho' para o 1º token (host) e coleta para inspeção.
julho.map(lambda line: line.split()[0]).take(5)

# Variante com separador explícito: split(' ') (equivalente neste caso).
# 'distinct()' remove duplicatas via shuffle.
# 'count()' é ACTION: dispara a execução e retorna a contagem de elementos (hosts únicos).
julho.map(lambda line: line.split(' ')[0]).distinct().count()

['199.72.81.55', '-', '-', '[01/Jul/1995:00:00:01', '-0400]', '"GET', '/history/apollo/', 'HTTP/1.0"', '200', '6245']
199.72.81.55
81983
```

Operação Passo 6 durou 33 segundos:



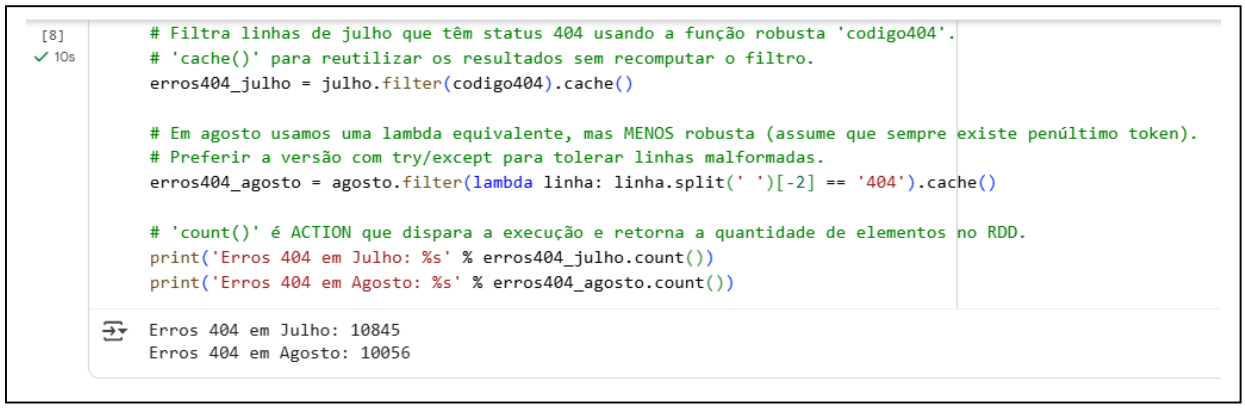
```
[6]
✓ 33s
# Usando a função
obterQtdHosts(julho)

# Número de hosts distintos em Agosto
contagem_agosto = agosto.map(lambda line: line.split(' ')[0]).distinct().count()
print("Numero de hosts distintos no mes de Agosto:", contagem_agosto)

# Usando a função
obterQtdHosts(agosto)

Numero de hosts distintos no mes de Julho: 81983
Numero de hosts distintos no mes de Agosto: 75060
75060
```

Operação Passo 7 durou 0 segundo:

```
[7]
✓ 0s
print(exemplo)

# Linha sintética com status 404 (para validar True)
exemplo2 = '199.72.81.55 - - [01/Jul/1995:00:00:01 -0400] "GET /history/apollo/ HTTP/1.0" 404 6245'
codigo404(exemplo2)  # Esperado: True

199.72.81.55 - - [01/Jul/1995:00:00:01 -0400] "GET /history/apollo/ HTTP/1.0" 200 6245
True
```

Filtrando os 404 por mês e contando

Operação Passo 8 durou 10 segundos:



```
[8]
✓ 10s
# Filtra linhas de julho que têm status 404 usando a função robusta 'codigo404'.
# 'cache()' para reutilizar os resultados sem recomputar o filtro.
erros404_julho = julho.filter(codigo404).cache()

# Em agosto usamos uma lambda equivalente, mas MENOS robusta (assume que sempre existe penúltimo token).
# Preferir a versão com try/except para tolerar linhas malformadas.
erros404_agosto = agosto.filter(lambda linha: linha.split(' ')[-2] == '404').cache()

# 'count()' é ACTION que dispara a execução e retorna a quantidade de elementos no RDD.
print('Erros 404 em Julho: %s' % erros404_julho.count())
print('Erros 404 em Agosto: %s' % erros404_agosto.count())

Erros 404 em Julho: 10845
Erros 404 em Agosto: 10056
```

Operação Passo 9 durou 1 segundo:

```
[9]
✓ 1s
counts.take(10)  # amostra de 10 pares (url, contagem)
type(counts)     # RDD[Tuple[str, int]]

199.72.81.55 - - [01/Jul/1995:00:00:01 -0400] "GET /history/apollo/ HTTP/1.0" 200 6245
['199.72.81.55 - - [01/Jul/1995:00:00:01 -0400] ', 'GET /history/apollo/ HTTP/1.0', ' 200 6245']
/history/apollo/

pyspark.rdd.PipelinedRDD
def __init__(prev: RDD[T], func: Callable[[int, Iterable[T]], Iterable[U]], preservesPartitioning:
bool=False, isFromBarrier: bool=False)

/usr/local/lib/python3.12/dist-packages/pyspark/rdd.py
Examples
--------
Pipelined maps:

>>> rdd = sc.parallelize([1, 2, 3, 4])
```

Operação Passo 10 durou 5 segundos:

```
[10]
✓ 5s
        1) extrair URL: linha.split('"')[1].split(' ')[1]
        2) map para (url, 1)
        3) reduceByKey(add) para somar
        4) sortBy(lambda par: -par[1]) para ordenar por contagem desc
        5) take(5)
    """
    urls = rdd.map(lambda linha: linha.split('"')[1].split(' ')[1])   # '/history/apollo/'
    counts = urls.map(lambda urls: (urls, 1)).reduceByKey(add)        # (url, soma)
    top5 = counts.sortBy(lambda par: -par[1]).take(5)                 # ordena por -contagem (descendente)
    return top5

# Retorna lista com 5 tuplas (url, contagem), ordenadas por maior contagem
top5_hosts404(erros404_julho)
top5_hosts404(erros404_agosto)
```

```
[('/pub/winvn/readme.txt', 1337),
 ('/pub/winvn/release.txt', 1185),
 ('/shuttle/missions/STS-69/mission-STS-69.html', 683),
 ('/images/nasa-logo.gif', 319),
 ('/shuttle/missions/sts-68/ksc-upclose.gif', 253)]
```

Operação Passo 11 durou 2 segundos:

```
[11]
✓ 2s
def contador_dias_404(rdd):
    """
    1) Extrai o dia no formato 'dd/Mon/yyyy' do trecho entre colchetes.
    2) Mapeia para (dia, 1) e soma com reduceByKey(add).
    3) collect() traz o resultado (lista de tuplas) para o Driver.
    """
    dias = rdd.map(lambda linha: linha.split('[')[1].split(':')[0])  # '01/Jul/1995'
    counts = dias.map(lambda dia: (dia, 1)).reduceByKey(add).collect()
    return counts

print(contador_dias_404(erros404_julho))  # lista Python de ("01/Jul/1995", 123), ...
```

```
199.72.81.55 - - [01/Jul/1995:00:00:01 -0400] "GET /history/apollo/ HTTP/1.0" 200 6245
01/Jul/1995:00:00:01 -0400] "GET /history/apollo/ HTTP/1.0" 200 6245
01/Jul/1995
[('01/Jul/1995', 316), ('09/Jul/1995', 348), ('21/Jul/1995', 334), ('26/Jul/1995', 336), ('27/Jul/1995', 336), ('17/Jul/1995', 406), ('20
```

Operação Passo 12 durou 7 segundos:

```
[12]
✓ 7s
# Ordenação por TEXTO (dia), desc: para strings, use reverse=True (não faz sentido negar string).
sorted(contador_dias_404(erros404_julho), key=lambda x: x[0], reverse=True)

# Ordenação por TEXTO (dia), asc:
sorted(contador_dias_404(erros404_julho), key=lambda x: x[0])

# Reaproveita a função para agosto (lista não ordenada):
contador_dias_404(erros404_agosto)

# Conversão de string para inteiro (exemplo auxiliar):
int('10')  # retorna 10 (int)
```

```
['a', 'b', 'c', 'd', 'e', 'f', 'g', 'h']
10
```

Operação Passo 13 durou 12 segundos:

[13] ✓ 12s

```
                    raise ValueError()
                return count
            except:
                # Retorna 0 para linhas malformadas ou com '-' em BYTES.
                return 0

        # 'map(contador)': aplica a função acima em cada linha, produzindo um RDD de inteiros (
        # 'reduce(add)': soma todos os inteiros de forma DISTRIBUÍDA (reduceByKey é para pares
        count = rdd.map(contador).reduce(add)
        return count

print('Quantidade de bytes total em Julho: %s' % quantidade_bytes_acumulados(julho))
print('Quantidade de bytes total em Agosto: %s' % quantidade_bytes_acumulados(agosto))
```

```
Quantidade de bytes total em Julho: 38695973491
Quantidade de bytes total em Agosto: 26828341424
```

Operação Passo 14 durou 0 segundo e encerrou a aplicação:

[14] ✓ 0s

```
# ========================== CÉLULA 14 ==========================
# Encerra o contexto do Spark e libera recursos locais/cluster.
sc.stop()
```