*# TASK 1: Producing a Random Sub-Sample of a Big Dataset using Hadoop MapReduce:*

*"""*

*Input Key-Value Pairs:*

*Input: The input consists of lines of text. Each line is a document or record.*

*Key (for mapper): Not applicable for this task.*

*Value (for mapper): Each line of text.*

*Intermediate Key-Value Pairs:*

*Key (from mapper): Not applicable for this task.*

*Value (from mapper): Each line of text.*

*Output Key-Value Pairs:*

*Key (from reducer): Not applicable for this task.*

*Value (from reducer): Approximately 10% of the lines in the input dataset.*

*"""*

*# Mapper:*

import os

import re

import sys

import random

*# Input: Each line from the input dataset*

for line in sys.stdin:

if random.random() < 0.10: *# Randomly select approximately 10% of lines*

print(line.strip()) *# Emit the selected line*

*# Reducer:*

*# Input: Output from mappers*

for line in sys.stdin:

print(line.strip()) *# Just pass through the selected lines*

*# Hadoop Command*

hadoop jar $HADOOP\_HOME/share/hadoop/tools/lib/hadoop-streaming-\*.jar \

- input / path/to/your/large\_dataset \

- output / path/to/output/subsample \

- mapper ./mapper.py \

- reducer ./reducer.py \

- file ./mapper.py \

- file ./reducer.py

*"""*

*Justification: This solution works by randomly selecting lines from the input dataset, so there is no specific key-value pairing involved.*

*Mapper emits lines as they are, and reducer collects approximately 10% of the lines from the mappers' output.*

*Command Option Justification: The Hadoop Streaming command was used with a basic mapper and reducer,*

*as no specific key-value transformations were required.*

*"""*

*# TASK 2: Building N-grams using Hadoop MapReduce:*

*"""*

*Input Key-Value Pairs:*

*Input: The input consists of lines of text. Each line is a document or record.*

*Key (for mapper): Not applicable for this task.*

*Value (for mapper): Each line of text.*

*Intermediate Key-Value Pairs:*

*Key (from mapper): Each di-gram (a word pair).*

*Value (from mapper): A count of 1 for each di-gram.*

*Output Key-Value Pairs:*

*Key (from reducer): Unique di-grams.*

*Value (from reducer): Counts of unique di-grams.*

*"""*

*# Mapper:*

for line in sys.stdin:

*# Remove punctuation and split the line into words*

words = re.findall(r'\b\w+\b', line.lower())

*# Emit di-grams*

for i in range(len(words) - 1):

di\_gram = words[i] + " " + words[i + 1]

print(di\_gram + "\t1")

*# Reducer:*

current\_di\_gram = None

current\_count = 0

for line in sys.stdin:

line = line.strip()

di\_gram, count = line.split("\t", 1)

*# Convert count to an integer*

try:

count = int(count)

except ValueError:

continue

*# If the current di-gram is equal to the previous di-gram, increment the count*

if current\_di\_gram == di\_gram:

current\_count += count

else:

*# Output the previous di-gram and its count*

if current\_di\_gram:

print(f"{current\_di\_gram}\t{current\_count}")

current\_di\_gram = di\_gram

current\_count = count

*# Output the last di-gram*

if current\_di\_gram:

print(f"{current\_di\_gram}\t{current\_count}")

*# Hadoop Command:*

hadoop jar $HADOOP\_HOME/share/hadoop/tools/lib/hadoop-streaming-\*.jar \

- input / path/to/your/input/dataset \

- output / path/to/output/di\_gram\_counts \

- mapper ./mapper.py \

- reducer ./reducer.py \

- file ./mapper.py \

- file ./reducer.py

*"""*

*Justification: The mapper tokenizes each line into words, generates di-grams, and emits each di-gram with a count of 1.*

*The reducer then groups and sums the counts for each unique di-gram.*

*Command Option Justification: Hadoop Streaming is used, and the Mapper and Reducer scripts are provided along with the input and output paths.*

*The commands are straightforward as no advanced options or custom partitioners are required.*

*"""*

*# TASK 3: Building an Inverted Index of a Text Corpus using Hadoop MapReduce:*

*"""*

*Input Key-Value Pairs:*

*Input: The input consists of lines of text from various documents.*

*Key (for mapper): Not applicable for this task.*

*Value (for mapper): Each line of text.*

*Intermediate Key-Value Pairs:*

*Key (from mapper): The first letter of each word (initial letter).*

*Value (from mapper): Each word and the filename where it appears.*

*Output Key-Value Pairs:*

*Key (from reducer): The initial letter (first part of the multi-part key).*

*Value (from reducer): A list of words that start with that initial letter and the filenames where they appear.*

*"""*

*# Mapper:*

*# Get the filename from the input path*

current\_file = os.environ.get("map\_input\_file", "unknown")

for line in sys.stdin:

*# Remove punctuation and split the line into words*

words = re.findall(r'\b\w+\b', line.lower())

*# Emit (word, filename) pairs*

for word in words:

print(f"{word}\t{current\_file}")

*# Reducer:*

current\_word = None

current\_files = []

for line in sys.stdin:

line = line.strip()

word, filename = line.split("\t", 1)

if current\_word == word:

current\_files.append(filename)

else:

*# Output the inverted index for the previous word*

if current\_word:

print(f"{current\_word}\t{', '.join(current\_files)}")

current\_word = word

current\_files = [filename]

*# Output the last inverted index entry*

if current\_word:

print(f"{current\_word}\t{', '.join(current\_files)}")

*# Hadoop Command:*

hadoop jar $HADOOP\_HOME/share/hadoop/tools/lib/hadoop-streaming-\*.jar \

- input / path/to/your/input/dataset \

- output / path/to/output/inverted\_index \

- mapper ./mapper.py \

- reducer ./reducer.py \

- file ./mapper.py \

- file ./reducer.py

*'''*

*Justification: The mapper tokenizes the text, emits key-value pairs with the initial letter and word as the key, and the filename as the value.*

*The reducer groups words by their initial letter, resulting in a sorted list of words for each letter.*

*Command Option Justification: Hadoop Streaming is used with custom partitioning for sorting and ordering.*

*The partitioner is defined to use the initial letter as the partition key,*

*which ensures that the words starting with the same letter go to the same reducer, achieving a total order sort.*

*'''*

*# TASK 4: Sorting Using Hadoop MapReduce:*

*'''*

*Input Key-Value Pairs:*

*Input: The input consists of lines of text.*

*Key (for mapper): Not applicable for this task.*

*Value (for mapper): Each line of text.*

*Intermediate Key-Value Pairs:*

*Key (from mapper): The first letter of each word (initial letter).*

*Value (from mapper): Each word.*

*Output Key-Value Pairs:*

*Key (from reducer): The initial letter (first part of the multi-part key).*

*Value (from reducer): A sorted list of words starting with that initial letter.*

*'''*

*# Mapper:*

for line in sys.stdin:

*# Remove punctuation and split the line into words*

words = re.findall(r'\b\w+\b', line.lower())

for word in words:

*# Emit key-value pairs with a multi-part key*

first\_letter = word[0]

print(f"{first\_letter}\t{word}\t1")

*# Partitioner:*

for line in sys.stdin:

first\_letter, word, \_ = line.strip().split("\t", 2)

print(f"{first\_letter}\t{word}")

*# Reducer:*

current\_letter = None

current\_words = []

for line in sys.stdin:

letter, word = line.strip().split("\t", 1)

if current\_letter == letter:

current\_words.append(word)

else:

*# Sort the words and output*

if current\_letter:

sorted\_words = sorted(current\_words)

print(f"{current\_letter}\t{' '.join(sorted\_words)}")

current\_letter = letter

current\_words = [word]

*# Output the last group of words*

if current\_letter:

sorted\_words = sorted(current\_words)

print(f"{current\_letter}\t{' '.join(sorted\_words)}")

*# Hadoop Command:*

hadoop jar $HADOOP\_HOME/share/hadoop/tools/lib/hadoop-streaming-\*.jar \

- input / path/to/your/input/dataset \

- output / path/to/output/sorted\_words \

- mapper ./mapper.py \

- reducer ./reducer.py \

- partitioner org.apache.hadoop.mapred.lib.KeyFieldBasedPartitioner \

- D stream.num.map.output.key.fields = 2 \

- D mapred.text.key.partitioner.options = -k1, 1 \

- D map.output.key.field.separator =\t \

- file ./mapper.py \

- file ./reducer.py \

- file ./partitioner.py

*'''*

*Justification: The mapper tokenizes the text, emits key-value pairs with the initial letter and word.*

*The reducer groups words by their initial letter and sorts them in ascending order.*

*Command Option Justification: Hadoop Streaming is used with a custom partitioner.*

*The custom partitioner partitions the data by the initial letter, and the reducer then sorts the words within each partition, achieving the required total order sort.*

*The -D options are used to specify the partitioning and key-field options to achieve this custom sorting behavior.*

*'''*