Extracting textual data from an insecure and a secure website.

import requests

from bs4 import BeautifulSoup

url = "https://huggingface.co/organizations/suggestions?emailConfirmation=true"

headers = {

"User-Agent": "Mozilla/5.0 (Windows NT 10.0; Win64; x64)

AppleWebKit/537.36 (KHTML, like Gecko) Chrome/91.0.4472.124

Safari/537.36"

}

response = requests.get(url, headers=headers)

if response.status\_code == 200:

soup = BeautifulSoup(response.text, "html.parser")

paragraphs = soup.find\_all("p")

for p in paragraphs:

print(p.text)

else:

print("Failed to retrieve the webpage.")

Calculate the matrix multiplication using MapReduce

from collections import defaultdict

A = [

[1, 2],

[3, 4]

]

B = [

[5, 6],

[7, 8]

]

# Mapper Function: Emit Key-Value Pairs

def mapper(A, B):

intermediate = []

rows\_A = len(A)

cols\_A = len(A[0])

cols\_B = len(B[0])

# Emit key-value pairs for Matrix A

for i in range(rows\_A):

for j in range(cols\_A):

for k in range(cols\_B):

intermediate.append(((i, k), ('A', A[i][j], j)))

# Emit key-value pairs for Matrix B

for j in range(cols\_A):

for k in range(cols\_B):

for i in range(rows\_A):

intermediate.append(((i, k), ('B', B[j][k], j)))

return intermediate

# Shuffle and Sort: Group by Key

def shuffle\_sort(mapped\_values):

grouped = defaultdict(list)

for key, value in mapped\_values:

grouped[key].append(value)

return grouped

# Reducer Function: Perform Multiplication and Summation

def reducer(grouped\_values):

result = defaultdict(int)

for key, values in grouped\_values.items():

a\_values = {v[2]: v[1] for v in values if v[0] == 'A'}

b\_values = {v[2]: v[1] for v in values if v[0] == 'B'}

# Multiply and sum corresponding elements

for j in a\_values.keys():

if j in b\_values:

result[key] += a\_values[j] \* b\_values[j]

return result

# Execute MapReduce Workflow

mapped = mapper(A, B)

grouped = shuffle\_sort(mapped)

reduced = reducer(grouped)

# Display Intermediate Outputs

print("Mapper Output (Key-Value Pairs):")

for item in mapped:

print(item)

print("\nGrouped Output (Shuffled & Sorted):")

for key, values in grouped.items():

print(f"{key}: {values}")

print("\nFinal Result Matrix (Reducer Output):")

result\_matrix = [[0 for \_ in range(len(B[0]))] for \_ in range(len(A))]

for (i, k), value in reduced.items():

result\_matrix[i][k] = value

for row in result\_matrix:

print(row)

Calculate the matrix multiplication using PySpark

from pyspark.sql import SparkSession

spark = SparkSession.builder.appName("MatrixMultiplication").getOrCreate()

sc = spark.sparkContext

data = [ ("A", 0, 0, 1), ("A", 0, 1, 2),

("A", 1, 0, 3), ("A", 1, 1, 4),

("B", 0, 0, 5), ("B", 0, 1, 6),

("B", 1, 0, 7), ("B", 1, 1, 8) ]

rdd = sc.parallelize(data)

def mapper(entry):

matrix, row, col, value = entry

result = []

if matrix == 'A':

for k in range(2): # Matrix A has 2 columns

result.append(((row, k), ("A", col, value)))

else:

for i in range(2): # Matrix B has 2 rows

result.append(((i, col), ("B", row, value)))

return result

mapper\_rdd = rdd.flatMap(mapper)

grouped\_rdd = mapper\_rdd.groupByKey().mapValues(list)

def reducer(key, values):

A\_values = {}

B\_values = {}

# Separate A and B values

for matrix, index, value in values:

if matrix == 'A':

A\_values[index] = value

elif matrix == 'B':

B\_values[index] = value

result\_value = sum(A\_values.get(k, 0) \* B\_values.get(k, 0) for k in A\_values.keys() if k in B\_values)

return (key, result\_value)

output\_rdd = grouped\_rdd.map(lambda x: reducer(x[0], x[1]))

for result in output\_rdd.collect():

print(result)

Write a program to implement Word count using mapReduce

import multiprocessing

from collections import defaultdict

from functools import reduce

# Sample input text

text\_data = [

"My name is Avinash kumar",

"My fav cartoon is tom and jerry",

"My fav colour is blue",

"My fav singer is udit",

"My fav singer is shreya"

]

def mapper(line):

words = line.strip().split()

return [(word, 1) for word in words]

def reducer(mapped\_data):

word\_counts = defaultdict(int)

for word, count in mapped\_data:

word\_counts[word] += count

return word\_counts

if \_\_name\_\_ == "\_\_main\_\_":

with multiprocessing.Pool() as pool:

mapped\_results = pool.map(mapper, text\_data)

shuffled\_data = [item for sublist in mapped\_results for item in sublist]

final\_counts = reducer(shuffled\_data)

for word, count in final\_counts.items():

print(f"{word}: {count}")