Name:-Preksha Patel
Sapid:-60004210126
Branch:- Computer Engineering
Div:-C2 ; Batch:-1

BIG DATA INFRASTRUCTURE(BDI)

Experiment 6

BDI

Name : Preksha A. Patil
Sapid : 60004210126
Branch : Computer Engineering
Div : C2, Batch : 1

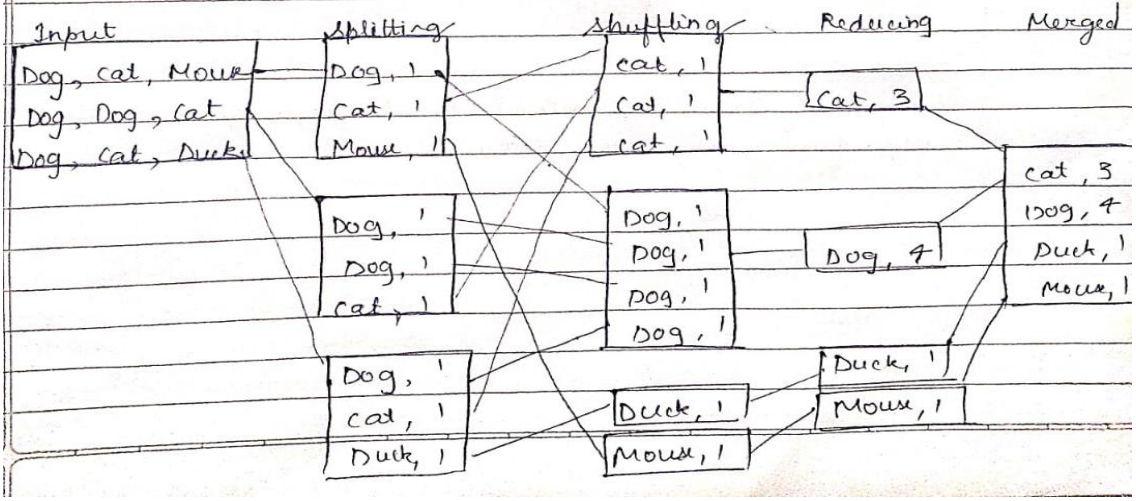
Experiment no. 6

Aim :- To implement word frequency and matrix multiplication using map reduce

Theory :-

Map reduce is a programming model used for efficient processing in parallel over large datasets in a distributed manner. The data is first split and then combined to produce the final result. The purpose for map reduce in Hadoop is to map each of the jobs and then it will reduce it to equivalent tasks for providing less overhead over the cluster network and to reduce the processing power. The map reduce ~~map phase of task~~ is mainly divided into 2 phases Map phase & Reduce phase.

Word frequency example :-



Matrix Multiplication algorithm :

Map function -

for each element m_{ij} of m do produce (key, value) pair as $((i,k), (m_{ij}, m_{ij}))$ for k = 1, 2, 3 ... upto no. of columns of N.

for each element n_{jk} of n do

produce (key, value) pair as $((i,k), (v, j, n_{jk}))$ for i = 1, 2, 3... upto the no. of rows of M.

return set of (key, value) pairs for that each key (i, k), has a list with values (m, i, m_{ij}) and (N, I, N_{jk}) for all possible values of j.

Reduce function :

for each key (i, k) do

sort values, begins with M by j in list M

sort values, begin with N by j in list N

Multiply m_{ij} and n_{jk} for j = value of each list

sum up $m_{ij} * n_{jk}$

return $(i,k), \sum_{j=1} m_{ij} \times n_{jk}$

Conclusion :

Thus, we have implemented word frequency & matrix multiplication using map reduce.

WORDCOUNT:

CODE:

```
from collections import defaultdict
documents =
[
    "Hello Hadoop",
    "Welcome to Hadoop World",
    "Hello World"
]

# Mapping Phase def
map_phase(documents):
    mapped = []     for document in
documents:         for word in
document.split():
mapped.append((word, 1))     return
mapped
# Shuffling Phase def
shuffle_phase(mapped):
    shuffled = defaultdict(list)
for key, value in mapped:
        shuffled[key].append(value)
return shuffled

# Reducing Phase def
reduce_phase(shuffled):
    reduced = {}     for key, values in
shuffled.items():
        reduced[key] = sum(values)
return reduced

# Driver code to simulate the MapReduce process if
__name__ == "__main__":     # Map Phase
mapped = map_phase(documents)     print(f"Mapped:
{mapped}")

    # Shuffle Phase     shuffled =
shuffle_phase(mapped)
print(f"Shuffled: {dict(shuffled)}")
```

```
    # Reduce Phase      reduced = reduce_phase(shuffled)      print(f"Reduced: {reduced}")
```

OUTPUT:

```
PS E:\Sem6\BDI> python word.py
Mapped: [('Hello', 1), ('Hadoop', 1), ('Welcome', 1), ('to', 1), ('Hadoop', 1), ('World', 1), ('Hello', 1), ('World', 1)]
Shuffled: {'Hello': [1, 1], 'Hadoop': [1, 1], 'Welcome': [1], 'to': [1], 'World': [1, 1]}
Reduced: {'Hello': 2, 'Hadoop': 2, 'Welcome': 1, 'to': 1, 'World': 2}
```

MATRIX MULTIPLICATION:

CODE:

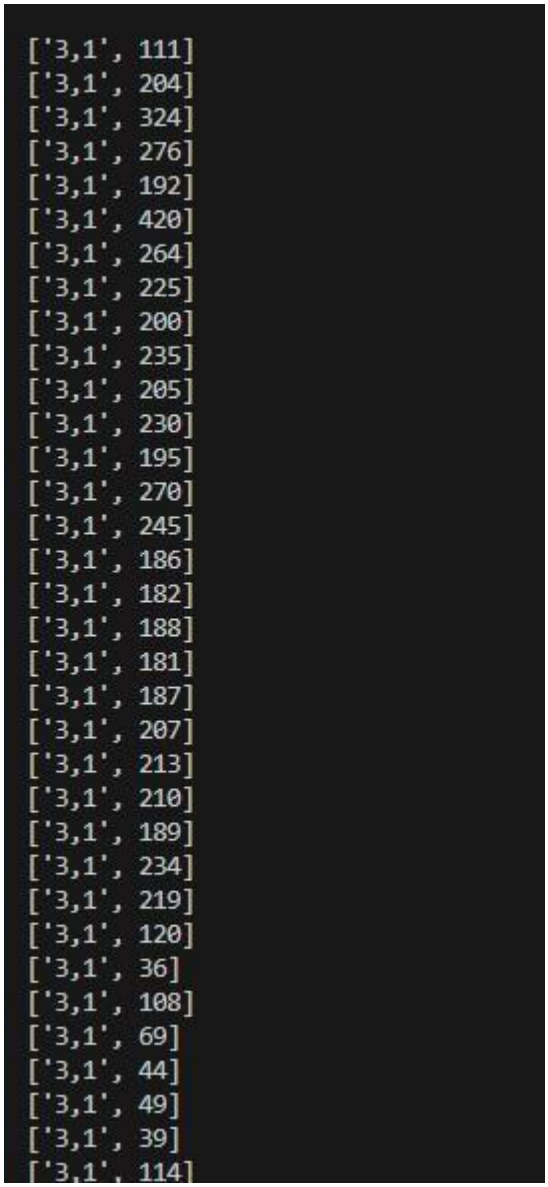
```
with open("cache.txt") as cache_file:
cache = cache_file.readline().split(",")
row_a, col_b = map(int, cache) mapperOutput
= open("mapperOutput.txt", "w") for line in
open("input.txt"):
    matrix_index, row, col, value = line.rstrip().split(",")     if
matrix_index == "A":         for i in range(0, col_b):             key
= row + "," + str(i)             mapperOutput.write("%s\t%s\t%s" %
(key, col, value) + "\n")     else:         for j in range(0, row_a):
key = str(j) + "," + col             mapperOutput.write("%s\t%s\t%s" %
(key, row, value) + "\n") mapperOutput.close()

listMultiply1 = list()
listMultiply2 = list()
listAdd1 = list() listAdd2 =
list() reducerTemp = list()
reducerOutput =
list()  for line in
open("mapperOutput.txt"):
    key, index, value = line.rstrip().split("\t")     index,
value = map(int, [index, value])
listMultiply1.append((key, index, value))
```

```
 listMultiply2 =
listMultiply1   for i in
listMultiply1:
for j in listMultiply2:
if i != j:             if
i[1] == j[1]:
                listAdd1.append([i[0], i[2] * j[2]])
for sublist in listAdd1:     if sublist not in
listAdd2:
listAdd2.append(sublist)
listAdd1 = listAdd2
for i in listAdd1:
for j in listAdd2:
if i != j:             if i[0]
== j[0]:
                reducerTemp.append([i[0], i[1] + j[1]])
for sublist in reducerTemp:     if sublist not in
reducerOutput:
reducerOutput.append(sublist)

# Print the result of this reducer for i
in reducerOutput:
    print(i)
```

OUTPUT:


['3,1', 111]
['3,1', 204]
['3,1', 324]
['3,1', 276]
['3,1', 192]
['3,1', 420]
['3,1', 264]
['3,1', 225]
['3,1', 200]
['3,1', 235]
['3,1', 205]
['3,1', 230]
['3,1', 195]
['3,1', 270]
['3,1', 245]
['3,1', 186]
['3,1', 182]
['3,1', 188]
['3,1', 181]
['3,1', 187]
['3,1', 207]
['3,1', 213]
['3,1', 210]
['3,1', 189]
['3,1', 234]
['3,1', 219]
['3,1', 120]
['3,1', 36]
['3,1', 108]
['3,1', 69]
['3,1', 44]
['3,1', 49]
['3,1', 39]
['3,1', 114]