1. Implementation of word count problem.

mapper.py file

```
import sys
 for line in sys.stdin:
  line = line.strip()
  words = line.split()
  for word in words:
      print '%s\t%s' % (word, 1)
```

Add the below Code into reducer.py file

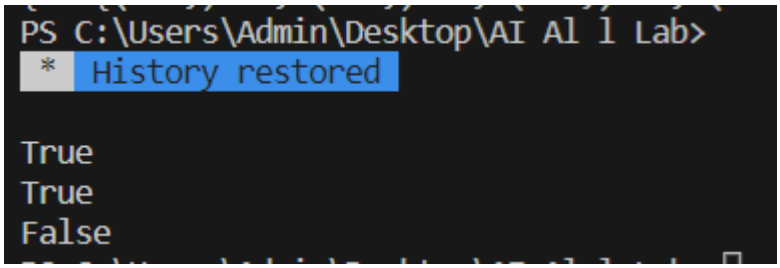
```
import sys
current_word = None
current_count = 0
for line in sys.stdin:
    line = line.strip()
    word, count = line.split('\t', 1)
    count = int(count)
    if word == current_word:
        current_count += count
    else:
        if current_word:
            print('%s\t%s' % (current_word, current_count))
        current_word = word
        current_count = count
if current_word == word:
    print('%s\t%s' % (current_word, current_count))
```

2. Implementation of Bloom Filter.

A Bloom filter is a space-efficient probabilistic data structure used to test whether an element is a member of a set. It

Code -

```
import mmh3
from bitarray import bitarray
class BloomFilter:
    def __init__(self, size, num_hash_functions):
        self.size = size
        self.num_hash_functions = num_hash_functions
        self.bit_array = bitarray(size)
        self.bit_array.setall(0)
    def add(self, item):
        for seed in range(self.num_hash_functions):
            index = mmh3.hash(item, seed) % self.size
            self.bit_array[index] = 1
    def contains(self, item):
        for seed in range(self.num_hash_functions):
            index = mmh3.hash(item, seed) % self.size
            if self.bit_array[index] == 0:
                return False
        return True
# Example usage:
bloom_filter = BloomFilter(100, 3)
bloom_filter.add("apple")
bloom_filter.add("banana")
print(bloom_filter.contains("apple"))  # True
print(bloom_filter.contains("banana"))  # True
print(bloom_filter.contains("orange"))  # False
```



3. Implementation of Apriori Algorithm.

Apriori is a popular algorithm used for frequent itemset mining and association rule learning. The goal of Apriori algorithm is to find all the frequent itemsets in a dataset. A frequent itemset is an itemset that appears frequently in a dataset.

Code -:

```
import itertools


def apriori(transactions, min_support):
    itemsets = {}
    itemset_size = 1
    while True:
        candidate_itemsets = {}
        for transaction in transactions:
            for itemset in itertools.combinations(transaction, itemset_size):
                if itemset in candidate_itemsets:
                    candidate_itemsets[itemset] += 1
                else:
                    candidate_itemsets[itemset] = 1
        frequent_itemsets = {}
        for itemset, support in candidate_itemsets.items():
            if support/len(transactions) >= min_support:
                frequent_itemsets[itemset] = support
        if not frequent_itemsets:
            break
```

```
        itemsets[itemset_size] = frequent_itemsets
        itemset_size += 1
    return itemsets
transactions = [    ['A', 'B', 'C'],
    ['A', 'C'],
    ['A', 'B', 'D'],
    ['B', 'D'],
    ['B', 'C'],
    ['C', 'D'],
    ['A', 'C', 'D'],
    ['B', 'C', 'D']
]
min_support = 0.5
itemsets = apriori(transactions, min_support)
print(itemsets)
```

```
PS C:\Users\Admin\Desktop\AI Al l Lab> python C:\Users\Admin\Music\test.py
{1: {('A',): 4, ('B',): 5, ('C',): 6, ('D',): 5}}
PS C:\Users\Admin\Desktop\AI Al l Lab>
```

4. Implementation of PCY Algorithm.

The PCY (Park-Chen-Yu) algorithm is a popular algorithm used for discovering frequent itemsets in large transactional databases. It is an improvement over the Apriori algorithm, which suffers from high computational costs.

Code -:

```
from collections import defaultdict
import itertools


def apply_pcy(dataset, support_threshold, bitmap_size):
    # Step 1: Count item frequencies
```

```
    item_counts = defaultdict(int)
    for transaction in dataset:
        for item in transaction:
            item_counts[item] += 1
    # Step 2: Generate frequent itemsets
    frequent_itemsets = []
    for item, count in item_counts.items():
        if count >= support_threshold:
            frequent_itemsets.append([item])
    # Step 3: Generate item pairs and count their occurrences
    pair_counts = defaultdict(int)
    for transaction in dataset:
        transaction = sorted(transaction)
        pairs = list(itertools.combinations(transaction, 2))
        for pair in pairs:
            pair_counts[pair] += 1
    # Step 4: Filter frequent item pairs using bitmap
    frequent_pairs = []
    bitmap = [0] * bitmap_size
    for pair, count in pair_counts.items():
        if count >= support_threshold:
            bitmap[hash(pair) % bitmap_size] += 1
            frequent_pairs.append(pair)
    return frequent_itemsets, frequent_pairs
# Example usage
dataset = [
    ['A', 'B', 'C', 'D'],
    ['A', 'C', 'D'],
```

```
    ['A', 'B', 'C'],
    ['B', 'D'],
    ['A', 'B', 'C', 'D'],
    ['B', 'C', 'D']
]
support_threshold = 3
bitmap_size = 10
frequent_itemsets, frequent_pairs = apply_pcy(dataset, support_threshold, bitmap_size)
print("Frequent Itemsets:")
for itemset in frequent_itemsets:
    print(itemset)
print("\nFrequent Pairs:")
for pair in frequent_pairs:
    print(pair)
```

```
PS C:\Users\Admin\Desktop\AI Al l Lab> python C:\Users\Admin\Music\test.py
Frequent Itemsets:
['A']
['B']
['C']
['D']

Frequent Pairs:
('A', 'B')
('A', 'C')
('A', 'D')
('B', 'C')
('B', 'D')
('C', 'D')
PS C:\Users\Admin\Desktop\AI Al l Lab>
```

4. Commands of Hadoop File System.

Hadoop works on its own File System which is distributed in nature known as “Hadoop distributed File System HDFS”. In order to perform various operations at the file level, HDFS provides its own set of commands Known as Hadoop File System Commands. Let us explore those commands. In this topic, we are going to learn about Hadoop FS Command.Any HDFS command has the prefix

of “hdfs dfs”. It means that we are specifying that the default file system is HDFS. Let us explore Hadoop FS Commands list one by one

1. Versions

The version command is used to find the version of the Hadoop installed in the system.

Syntax:

Hadoop version

2. ls Command

ls command in Hadoop is used to specify the list of directories in the mentioned path. ls command takes hdfs path as parameter and returns a list of directories present in the path.

Syntax:

hdfs dfs -ls <hdfs file path>

Example: hdfs dfs -ls /user/harsha

3. Cat Command

Cat command is used to display the contents of the file to the console. This command takes the hdfs file path as an argument and displays the contents of the file.

Syntax:

hdfs dfs -cat <hdfs file path>

Example: hdfs dfs -cat /user/harsha/empnew.txt

4. mkdir command

mkdir command is used to create a new directory in the hdfs file system. It takes the hdfs path as an argument and creates a new directory in the path specified.

Syntax:

hdfs dfs -mkdir <hdfs path>

Example: hdfs dfs -mkdir /user/example

5. put command

put the command in HDFS is used to copy files from given source location to the destination hdfs path. Here source location can be a local file system path. put command takes two arguments, first one is source directory path and the second one is targeted HDFS path

Syntax:

hdfs dfs -put <source path> <destination path>

Example: hdfs dfs -put /home/harsha/empnew.txt /user/test/example2

6. get Command

get command in hdfs is used to copy a given hdfs file or directory to the target local file system path. It takes two arguments, one is source hdfs path and other is target local file system path

Syntax:

hdfs dfs -get <source hdfs> <destination local file system>

Example: hdfs dfs -get /user/test/example2 /home/harsha

7. count command

count command in hdfs is used to count the number of directories present in the given path. count command takes a given path as an argument and gives the number of directories present in that path.

Syntax:

hdfs dfs -count <path>

Example: hdfs dfs -count /user

8. mv command

mv command in hdfs is used to move a file in between hdfs. mv command takes file or directory from given source hdfs path and moves it to target hdfs path.

Syntax:

hdfs dfs -mv <hdfs path> <hdfs path>

Example: hdfs dfs -mv /user/test/example2 /user/harsha

9. du command

du command in hdfs shows disk utilization for the hdfs path given. It takes the hdfs path as input and returns disk utilization in bytes.

Syntax:

hdfs dfs -du <hdfs path>

Example: hdfs dfs -du /user/harsha/empnew.txt

10. rm command

rm command in hdfs is used to remove files or directories in the given hdfs path. This command takes the hdfs path as input and removes the files present in that path.

Syntax:

hdfs dfs -rm <hdfs path>

Example: hdfs dfs -rm /user/harsha/example