1. MapReduce diagram

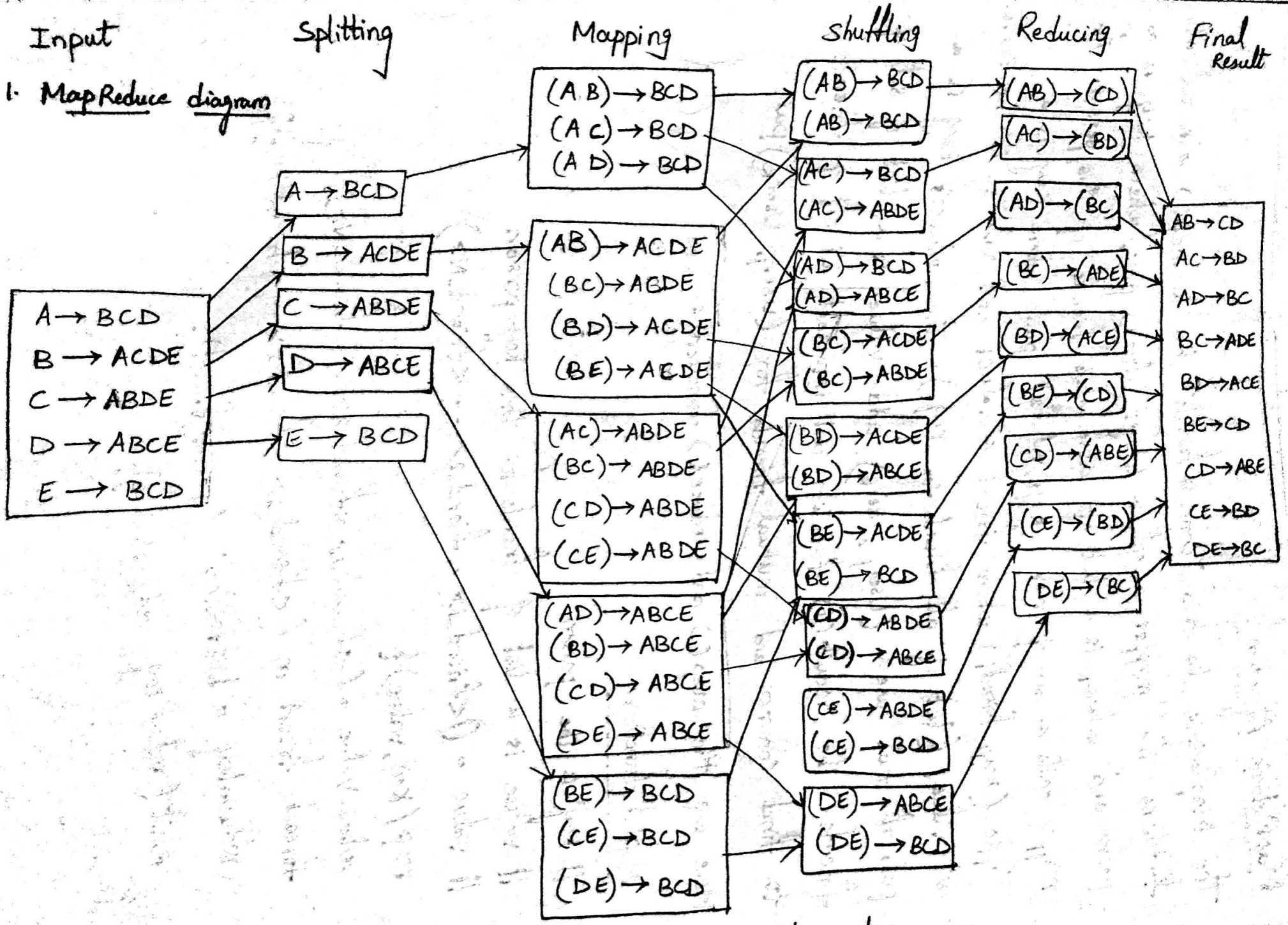

Fig: The overall MapReduce word count process

2. MapReduce Algorithm:

The MapReduce to find "common friends" has a map() and reduce() functions. The mapper accepts a $(Key_1, value_1)$ pair, where key_1 is a person and $value_1$ is a list of associated friends of this person. The mapper emits a set of new $(key_2, value_2)$ pairs where key is a Tupple2(key_1, $friend_i$) where $friend_i \in value_1$ and $value_2$ is the same as $value_1$ (list of all friends for key_1). The reducer's key is a pair of two users $(User_j, User_K)$ and value is a list of sets of friends. The reduce() function will intersect all set of friends to find common and mutual friends for $(user_j, User_K)$ pair.

Here are the map() and reduce() functions:

Map() function:

```
// key is the person
// value is a list of friends for this key=person
// value = (<friend_1> <friend_2>.... <friend_N>)
map(key, value){
reducerValue = (<friend_1> <friend_2>... <friend_N>);
foreach friend in (<friend_1> <friend_2>.. <friend_N>){
 reducerkey = buildsortedkey(person, friend);
 emit(reducerkey, reducerValue);
}
}
```

Mapper's output keys are sorted and this property will prevent duplicate keys.

buildSortedKey( ) Function :

```
Tuple2 buildSortedKey (person1, person2) {
 if (person1 < person2) {
  return Tuple2( person1, person2)
 }
 else {
  return( Tuple2 (person2, person1)
 }
}
```

The reduce( ) function finds the common friends for every pair of users by intersecting all associated friends in between.

3. Spark Scala implementation :

Input : data.txt

A → BCD
B → ACDE
C → ABDE
D → ABCE
E → BCD

Mapfunction :

```
def pairMapper( line : string) = {
 val words = line.split(" ")
 val key = words(0)
```

```
    val pairs = words.slice(1, words.size).map(friend => {
        if (key < friend)(key, friend) else (friend, key)
    })
    pairs.map(pair => (pair, words.slice(1, words.size).toSet))
}
```

Reduce function:

```
def pairReducer(accumulator: set[string], set: set[string]) = {
    accumulator intersect set
}
val data = sc.textFile("file:///data.txt")
val results = data.flatMap(pairMapper)
  .reduceByKey(pairReducer)
  .filter(!_._2.isEmpty)
  .sortByKey()
results.collect.foreach(line => {
    println(s"${line._1} ${line._2.mkString(" ")}")
})
```