Ch7.
Market Basket Analysis

박 세 진

Market Basket Analysis (MBA)

- 마케팅, 전자상거래의 데이터 마이닝
- 슈퍼마켓, 신용카드 거래내역 정보 등으로 부터 흥미로운 관계를 추출하는 것이 목표
- 이번 챕터에서는 Market Basket Analysis solution
  - MapReduce / Hadoop solution(order N에 대한 tuples)
    : frequent pattern을 찾아준다.
  - Spark solution
    : frequent pattern을 찾아주고, association rule도 generate 한다.

MapReduce solution

- 슈퍼마켓이나 전자상거래 매장에서 장바구니 안에 가장 자주 발생하는 물건들의 pair를 찾기위한 데이터 마이닝 방법
- 가장 자주 나타나는 상품들(N의 순서)의 TupleN (N = 1,2,3, …)
- "Order of N"이 맵리듀스 드라이버에 인자로 넘어가,
  하둡의 Configuration 객체를 설정하게 된다.
- 생성된, 가장 자주 나타나는 아이템 세트 Fi (i=1,2,…)을 transaction을 위한 association rule을 만드는 데 사용한다.

예제

다섯 개의 아이템

{A,B,C,D,E}

여섯 건의 transaction

```
Transaction 1:  A,  C
Transaction 2:  B,  D
Transaction 3:  A,  C,  E
Transaction 4:  C,  E
Transaction 5:  A,  B,  E
Transaction 6:  B,  E
```

- Frequent item set F1, F2를 만드는 것이 목표
  F1 (size = 1), F2 (size = 2)

$$F_1 = \{[C, 3], [A, 3], [B, 3], [E, 4]\}$$

$$F_2 = \{[<A,C>, 2], [<C,E>, 2], [<A,E>, 2], [<B,E>, 2]\}$$

- Support (지지도)
  - 최소 support(전체 transaction set에서 몇 번 패턴이 나타나는가)를 2로 적용
    - [D, 1]가 F1에서 제거

```
Transaction 1:  A,  C
Transaction 2:  B,  D
Transaction 3:  A,  C,  E
Transaction 4:  C,  E
Transaction 5:  A,  B,  E
Transaction 6:  B,  E
```

$F_1 = \{[C, 3], [A, 3], [B, 3], [E, 4]\}$

$F_2 = \{[<A,C>, 2], [<C,E>, 2], [<A,E>, 2], [<B,E>, 2]\}$

- 다음 아이템 세트 F1, F2는 transaction의 association rule을 생성한다.

F_1 = {[C, 3], [A, 3], [B, 3], [E, 4]}

F_2 = {[<A,C>, 2], [<C,E>, 2], [<A,E>, 2], [<B,E>, 2]}

[association rule]

```
LHS => RHS
coke => chips
if customers purchase coke (called LHDS - lefthand side),
they also buy chips (called RHS - righthand side).
```

다음 질문들에 대한 고객 행동에 대한 이해

- 어떤 아이템들을 함께 사는가
- 각 장바구니 안에 무엇이 있는가
- 어떤 아이템들을 판매해야 하는가
- 최대 이익을 위해서는 어떤 아이템들이 옆에 위치해야 하는가
- 전자 상거래 사이트의 카달로그

분야

- 신용카드내역 분석
- 전화 통화 패턴 분석
- 의료 보험 사기 (자주 나타나는 룰이 깨지는 경우)
- 통신서비스 구입 분석
- 아마존 같은 온라인 리테일러의 매일, 매주 구매 패턴 분석

Association rule의 metric 두 가지

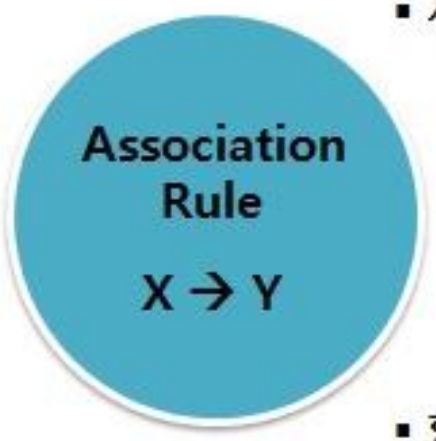
- Support (지지도)
  - 아이템 셋의 발생 빈도 (Frequency of occurrence of an item set
  - Support({ A, C }) = 2 아이템 A, C 가 두 개의 transaction에서만 함께 나타난다.
- Confidence(신뢰도)
  - 왼쪽의 룰이 오른쪽에 얼마나 자주 나타나는가

- **A. 지지도(SUPPORT)**
  - 전체 거래에서 특정 물품 A와 B가 동시에 거래되는 비중으로 해당 규칙이 얼마나 의미가 있는 규칙인지 보여준다.
  - 지지도 = P(A∩B) : *A와B가 동시에 일어난 횟수 / 전체 거래 횟수*
- **B. 신뢰도(CONFIDENCE)**
  - A를 포함하는 거래 중 A와 B가 동시에 거래되는 비중으로,
    A라는 사건이 발생했을 때 B가 발생할 확률이 얼마나 높은지를 말해줍니다.
  - 신뢰도 = P(A∩B) / P(A) : *A와B가 동시에 일어난 횟수 / A가 일어난 횟수*

- **C. 향상도(LIFT)**
  - A와 B가 동시에 거래된 비중을 A와B가 서로 독립된 사건일때 동시에 거래된 비중으로 나눈 값
  - 향상도 = $P(A \cap B) / P(A)*P(B) = P(B \mid A) / P(B)$
    : A와 B가 동시에 일어난 횟수 / A, B가 독립된 사건일 때 A,B가 동시에 일어날 확률
  - 품목 A와 B사이에 아무런 관계가 상호 관계가 없다면 향상도는 1

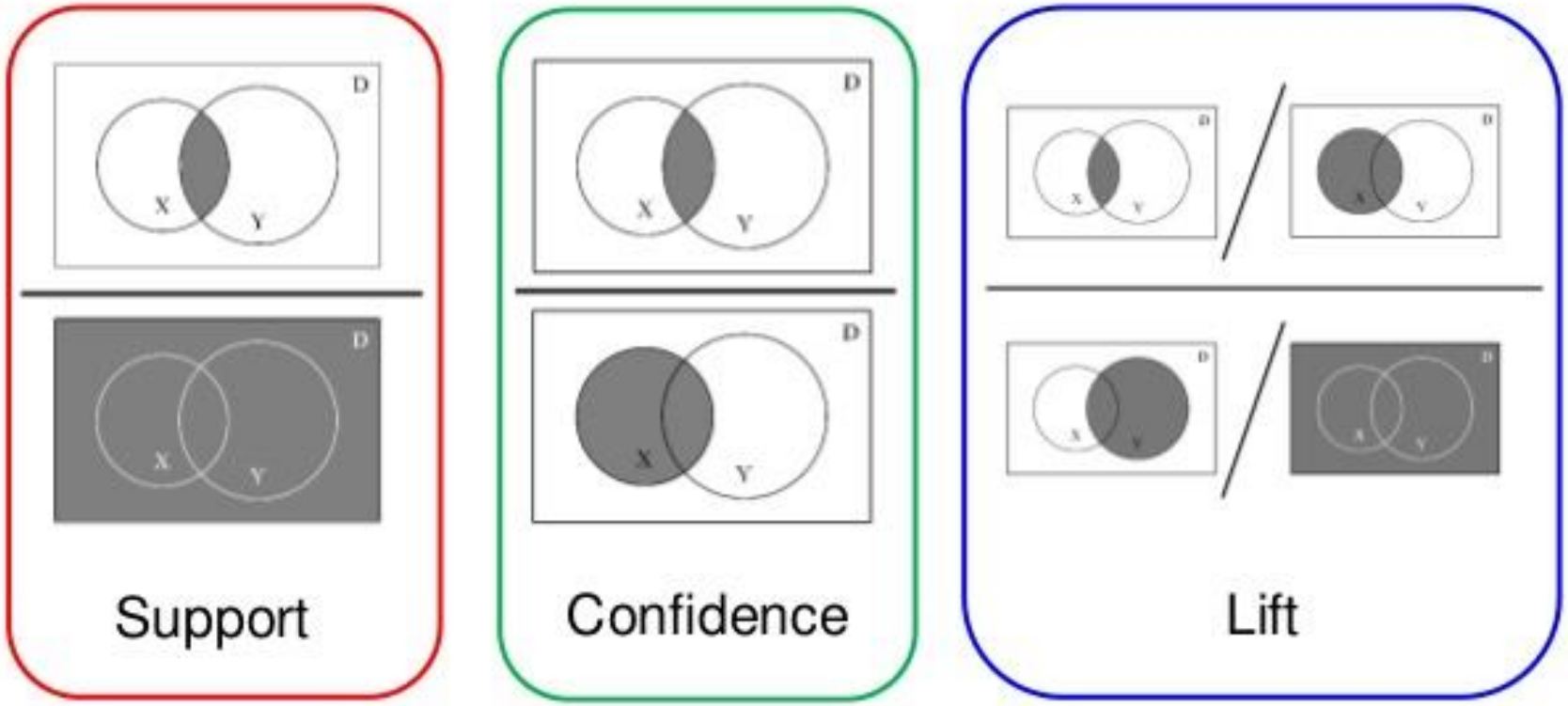
http://www.dodomira.com/2016/02/15/r%EC%9D%84-%EC%82%AC%EC%9A%A9%ED%95%9C-%EC%97%B0%EA%B4%80%EC%84%B1-%EB%B6%84%EC%84%9D-association-rules-in-r/

X와 Y를 서로 공통원소가 없는 항목들의 집합이고 (즉, X∩Y=Φ),
X → Y : 연관규칙 if X then Y
N : 전체 transaction 수 (즉, 전체 row 개수)
n(X) = : 전체 transaction에서 항목집합 X를 포함하는 transaction 수라고 하면,



- 지지도(Support) : $s(X \rightarrow Y) = \frac{n(X \cup Y)}{N}$
- 신뢰도(Confidence) : $c(X \rightarrow Y) = \frac{n(X \cup Y)}{n(X)}$
- 향상도(Lift) : $Lift(X,Y) = \frac{c(X \rightarrow Y)}{s(Y)}$

[R 분석과 프로그래밍] http://rfriend.tistory.com



とっつきにくいかもしれませんが、
これだけは覚えてください。

지지도(Support), 신뢰도(Confidence), 향상도(Lift) 예시

Customer ID	Transaction ID	Items
1131	1번	계란, 우유
2094	2번	계란, 기저귀, 맥주, 사과
4122	3번	우유, 기저귀, 맥주, 콜라
4811	4번	계란, 우유, 맥주, 기저귀
8091	5번	계란, 우유, 맥주, 콜라

N = 5 (전체 transaction 개 수)

s(Y) = n(Y)/ N
= n{2번, 3번, 4번}/N = 3/5 = 0.6

연관규칙 {계란, 맥주} → {기저귀} 에 대해
X Y

- **지지도(Support)**

s(X→Y) = n(X∪Y)/N
= n{2번, 4번}/ N
= 2/5 = 0.4

- **신뢰도(Confidence)**

c(X→Y) = n(X∪Y)/ n(X)
= n{2번, 4번}/ n{2번, 4번, 5번}
= 2/3 = 0.667

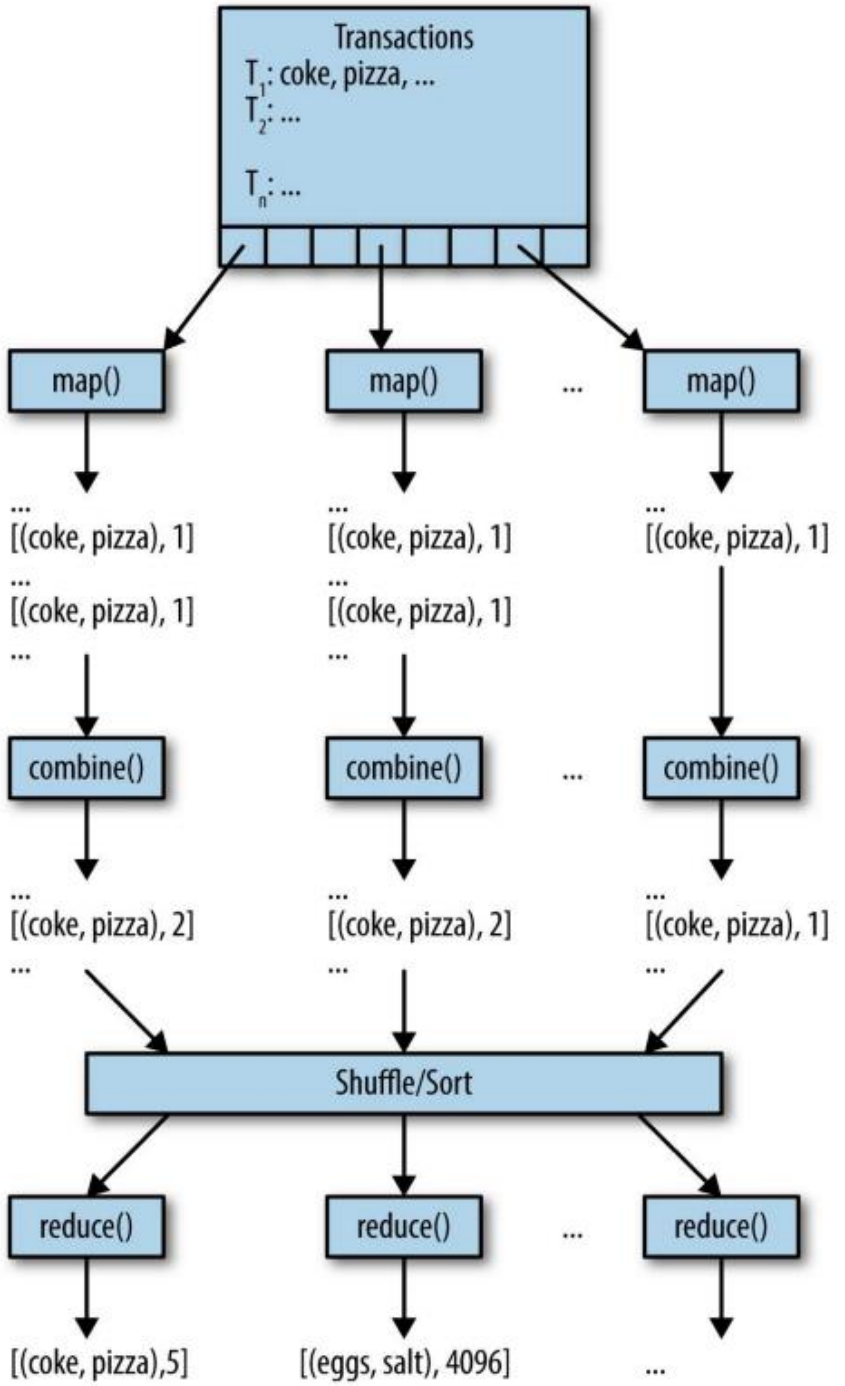
- **향상도(Lift)**

Lift(X→Y) = c(X→Y)/ s(Y)
= 0.667 / 0.6 = 1.111

[R 분석과 프로그래밍] http://rfriend.tistory.com

맵리듀스로 하는 MBA

- Map()
  :고객이 구매한
  아이템 세트인 {I1, I2, I3,…, In}로 되어있는
  transaction 을 받는다.
  - Item을 정렬하여 {s1, s2, … , sn}이 (key, 1) 쌍을 만든다.
  - Key = Tuple2 (Si, Sj) and Si < S j
  - Value = 1 (key 가 한번 나타남
  - Combiner와 reducer 는
    집계와 빈도를 카운트 한다.



input

- Input 은 sequence of transaction 이고, item은 space로 구분되어있다.

```
Transaction 1: crackers, icecream, coke, apple
Transaction 2: chicken, pizza, coke, bread
Transaction 3: baguette, soda, shampoo, crackers, pepsi
Transaction 4: baguette, cream cheese, diapers, milk
...
```

Tuples2 (order of 2) 결과 값

- 모든 unique pair (item1, item2)에 대한 아이템의 빈도를 찾는다.

Pair of items	Frequency
...	...
(bread, crackers)	8,709
(baguette, eggs)	7,524
(crackers, coke)	4,300
...	...

Tuples3 (order of 3) 결과 값

- 모든 unique set (item1, item2, item3) 의 모든 unique set에 대한 items의 빈도를 구한다.

Triplet of items	Frequency
...	...
(bread, crackers, eggs)	1,940
(baguette, diapers, eggs)	1,900
(crackers, coke, meat)	1,843
...	...

Informal Mapper

- Map()으로 하나의 transaction을 받아 reduce로 key-value 쌍의 셋트를 생성

```
[<crackers, icecream>, 1]
[<crackers, coke>, 1]
[<crackers, apple>, 1]
[<icecream, coke>, 1]
[<icecream, apple>, 1]
[<coke, apple>, 1]
```

Transaction 1에서 map()으로 생성된
key-value pair

Mapper는 transaction item을 키로
장바구니에 나타난 key의 숫자를 짝지어준다.

```
Transaction 1: crackers, icecream, coke, apple
Transaction 2: chicken, pizza, coke, bread
Transaction 3: baguette, soda, shampoo, crackers, pepsi
Transaction 4: baguette, cream cheese, diapers, milk
...
```

- 두개씩 쌍을 짓는다면 다음과 같은 transaction에는 오류가 생긴다.
- 정렬해서 해결.
- Combiner() 제대로된 결과출력

```
T1: crackers, icecream, coke
T2: icecream, coke, crackers
```

then for transaction T_1, `map()` will generate:

```
[(crackers, icecream), 1]
[(crackers, coke), 1]
[(icecream, coke), 1]
```

and for transaction T_2, `map()` will generate:

```
[(icecream, coke), 1]
[(icecream, crackers), 1]
[(coke, crackers), 1]
```

Formal Mapper

- Input data를 읽고
- 각 transaction에 대한 아이템 리스트를 생성
- Transaction list의 item 들을 정렬
  - (cracker, coke) (coke, cracker)와 같은 키 중복을 막는다.
- 아이템 쌍의 키로 변환

Example 7-1. MBA map() function

```
1 // key is transaction ID and ignored here
2 // value = transaction items (I1, I2, ...,In).
3 map(key, value) {
4     (S1, S2, ..., Sn) = sort(I1, I2, ...,In);
5     // now, we have: S1 < S2 < ... < Sn
6     List<Tuple2<Si, Sj>> listOfPairs =
7         Combinations.generateCombinations(S1, S2,
..., Sn);
8     for ( Tuple2<Si, Sj> pair : listOfPairs) {
9         // reducer key is: Tuple2<Si, Sj>
10        // reducer value is: integer 1
11        emit([Tuple2<Si, Sj>, 1]);
12    }
13 }
```

Combinations

- 주어진 아이템 리스트와 숫자 쌍에 대한 장바구니 아이템의 조합을 생성하는 자바 클래스
- generateCombinations(S1, s2, s3, s4)

```
(S1, S2)
(S1, S3)
(S1, S4)
(S2, S3)
(S2, S4)
(S3, S4)
```

Example 7-1. MBA map() function

```
1  // key is transaction ID and ignored here
2  // value = transaction items (I1, I2, ...,In).
3  map(key, value) {
4       (S1, S2, ..., Sn) = sort(I1, I2, ...,In);
5       // now, we have: S1 < S2 < ... < Sn
6       List<Tuple2<Si, Sj>> listOfPairs =
7            Combinations.generateCombinations(S1, S2,
...,  Sn);
8       for ( Tuple2<Si, Sj> pair : listOfPairs) {
9            // reducer key is: Tuple2<Si, Sj>
10           // reducer value is: integer 1
11           emit([Tuple2<Si, Sj>, 1]);
12      }
13 }
```

Reducer

- 각 reducer key에 대한 숫자를 합산

Example 7-2. MBA reduce() function

```
1 // key is in form of Tuple2(Si, Sj)
2 // value = List<integer>, where each element is an
integer number
3 reduce(Tuple2(Si, Sj) key, List<integer> values) {
4     integer sum = 0;
5     for (integer i : values) {
6         sum += i;
7     }
8
9     emit(key, sum);
10 }
```

MapReduce/Hadoop Implementation Classes

Table 7-3. Implementation classes in MapReduce/Hadoop

Class name	Class description
`Combination`	A utility class to create a combination of items
`MBADriver`	Submits the job to Hadoop
`MBAMapper`	Defines `map()`
`MBAReducer`	Defines `reduce()`

Find sorted combinations

- Combination.findSortedCombinations()

```
1 /**
2  * If elements = { a, b, c, d },
3  * then findCollections(elements, 2) will return:
4  *      { [a, b], [a, c], [a, d], [b, c], [b, d], [c,
d] }
5  *
6  * and findCollections(elements, 3) will return:
7  *      { [a, b, c], [a, b, d], [a, c, d], [b, c, d] }
8  *
```

```java
public static <T extends Comparable<? super T>>
List<List<T>>
        findSortedCombinations(Collection<T> elements,
int n) {
    List<List<T>> result = new ArrayList<List<T>>();

    // handle initial step for recursion
    if (n == 0) {
        result.add(new ArrayList<T>());
        return result;
    }

    // handle recursion for n-1
    List<List<T>> combinations =
findSortedCombinations(elements, n - 1);
    for (List<T> combination: combinations) {
        for (T element: elements) {
            if (combination.contains(element)) {
                continue;
            }

            List<T> list = new ArrayList<T>();
            list.addAll(combination);

            if (list.contains(element)) {
                continue;
            }

            list.add(element);
            // sort items to avoid duplicate items
            // example: (a, b, c) and (a, c, b) might
be counted as
            // different items if not sorted
            Collections.sort(list);

            if (result.contains(list)) {
                continue;
            }
            result.add(list);
        }
    }
    return result;
}
```

Market Basket Analysis driver : MBADriver

```
1  public class MBADriver extends Configured implements Tool {
2      ...
3      public int run(String args[]) throws Exception {
4          String inputPath = args[0];
5          String outputPath = args[1];
6          int numberOfPairs = Integer.parseInt(args[2]);
7          ...
8          // job configuration
9          Job job = new Job(getConf());
10         ...
11         job.getConfiguration().setInt("number.of.pairs",
numberOfPairs);
12
13         // set input/output path
14         FileInputFormat.setInputPaths(job, new Path(inputPath));
15         FileOutputFormat.setOutputPath(job, new Path(outputPath));
16
17         // mapper K, V output
18         job.setMapOutputKeyClass(Text.class);
19         job.setMapOutputValueClass(IntWritable.class);
20         // output format
21         job.setOutputFormatClass(TextOutputFormat.class);
22
23         // reducer K, V output
24         job.setOutputKeyClass(Text.class);
25         job.setOutputValueClass(IntWritable.class);
26
27         // set mapper/reducer/combiner
28         job.setMapperClass(MBAMapper.class);
29         job.setCombinerClass(MBAReducer.class);
30         job.setReducerClass(MBAReducer.class);
31
32         //delete the output path if it exists to avoid "existing
dir/file" error
33         Path outputDir = new Path(outputPath);
34         FileSystem.get(getConf()).delete(outputDir, true);
35
36         // submit job
37         boolean status = job.waitForCompletion(true);
38         ...
39     }
```

Market Basket Analysis mapper : MBAMapper

Example 7-5. MBAMapper

```
 1 public class MBAMapper extends Mapper<LongWritable,
Text, Text, IntWritable> {
 2
 3     public static final int DEFAULT_NUMBER_OF_PAIRS =
2;
 4
 5     // output key2: list of items paired; can be 2 or
3 ...
 6     private static final Text reducerKey = new Text();
 7
 8     // output value2: number of the paired items in
the item list
 9     private static final IntWritable NUMBER_ONE = new
IntWritable(1);
10
11     int numberOfPairs; // will be read by setup(), set
by driver
12
13     protected void setup(Context context)
14         throws IOException, InterruptedException {
15         this.numberOfPairs =
context.getConfiguration()
16                 .getInt("number.of.pairs",
DEFAULT_NUMBER_OF_PAIRS);
17     }
18
19     public void map(LongWritable key, Text value,
Context context)
20         throws IOException, InterruptedException {
21         String line = value.toString().trim();
22         List<String> items = convertItemsToList(line);
23         if ((items == null) || ( items.isEmpty())) {
24            // no mapper output will be generated
25             return;
26         }
27         generateMapperOutput(numberOfPairs, items,
context);
28     }
29
30     private static List<String>
convertItemsToList(String line) {
31          // see Example 7-6
32     }
33
34     private void generateMapperOutput(...) {
35        // see Example 7-7
36     }
37 }
```

MBAMapper helper methods

Example 7-6. MBAMapper helper methods: convertItemsToList

```
1     private static List<String> convertItemsToList(String
line) {
2         if ((line == null) || ( line.length() == 0)) {
3             // no mapper output will be generated
4             return null;
5         }
6         String[] tokens = StringUtils.split(line, ",");
7         if (( tokens == null) || ( tokens.length == 0) ) {
8             return null;
9         }
10        List<String> items = new ArrayList<String>();
11        for (String token : tokens) {
12            if (token != null) {
13                items.add(token.trim());
14            }
15        }
16        return items;
17    }
```

Example 7-7. MBAMapper helper methods: generateMapperOutput

```
1     /**
2      *
3      * This method builds a set of key-value pairs by
sorting the input list.
4      * key is a combination of items in a transaction, and
value = 1.
5      * Sorting is required to make sure that (a, b) and (b,
a)
6      * represent the same key.
7      * @param numberOfPairs is the number of pairs
associated
8      * @param items is a list of items (from input line)
9      * @param context is the Hadoop job context
10     * @throws IOException
11     * @throws InterruptedException
12     */
13    private void generateMapperOutput(int numberOfPairs,
14                                      List<String> items,
15                                      Context context)
16        throws IOException, InterruptedException {
17        List<List<String>> sortedCombinations =
18            Combination.findSortedCombinations(items,
numberOfPairs);
19        for (List<String> itemList: sortedCombinations) {
20            reducerKey.set(itemList.toString());
21            context.write(reducerKey, NUMBER_ONE);
22        }
23    }
24 }
```

실행예제

Input

```
# hadoop fs -cat /market_basket_analysis/input/input.txt
crackers,bread,banana
crackers,coke,butter,coffee
crackers,bread
crackers,bread
crackers,bread,coffee
butter,coke
butter,coke,bread,crackers
```

Log of sample run

```
# INPUT=/market_basket_analysis/input
# OUTPUT=/market_basket_analysis/output
# $HADOOP_HOME/bin/hadoop fs -rmr $OUTPUT
# $HADOOP_HOME/bin/hadoop jar $JAR MBADriver  $INPUT $OUTPUT 2
...
Deleted hdfs://localhost:9000/market_basket_analysis/output
14/05/02 13:36:36 INFO MBADriver: inputPath:
/market_basket_analysis/input
14/05/02 13:36:36 INFO MBADriver: outputPath:
/market_basket_analysis/output
14/05/02 13:36:36 INFO MBADriver: numberOfPairs: 2
...
14/05/02 13:36:37 INFO mapred.JobClient: Running job:
job_201405021309_0003
14/05/02 13:36:38 INFO mapred.JobClient:  map 0% reduce 0%
...
14/05/02 13:37:29 INFO mapred.JobClient:  map 100% reduce 100%
14/05/02 13:37:30 INFO mapred.JobClient: Job complete:
job_201405021309_0003
...
14/05/02 13:37:30 INFO mapred.JobClient:   Map-Reduce Framework
14/05/02 13:37:30 INFO mapred.JobClient:     Map input records=7
...
```

```
14/05/02 13:37:30 INFO mapred.JobClient:     Combine input
records=21
14/05/02 13:37:30 INFO mapred.JobClient:     Reduce input
records=12
14/05/02 13:37:30 INFO mapred.JobClient:     Reduce input groups=12
14/05/02 13:37:30 INFO mapred.JobClient:     Combine output
records=12
14/05/02 13:37:30 INFO mapred.JobClient:     Reduce output
records=12
14/05/02 13:37:30 INFO mapred.JobClient:     Map output records=21
14/05/02 13:37:30 INFO MBADriver: job status=true
14/05/02 13:37:30 INFO MBADriver: Elapsed time: 52737 milliseconds
14/05/02 13:37:30 INFO MBADriver: exitStatus=0
```

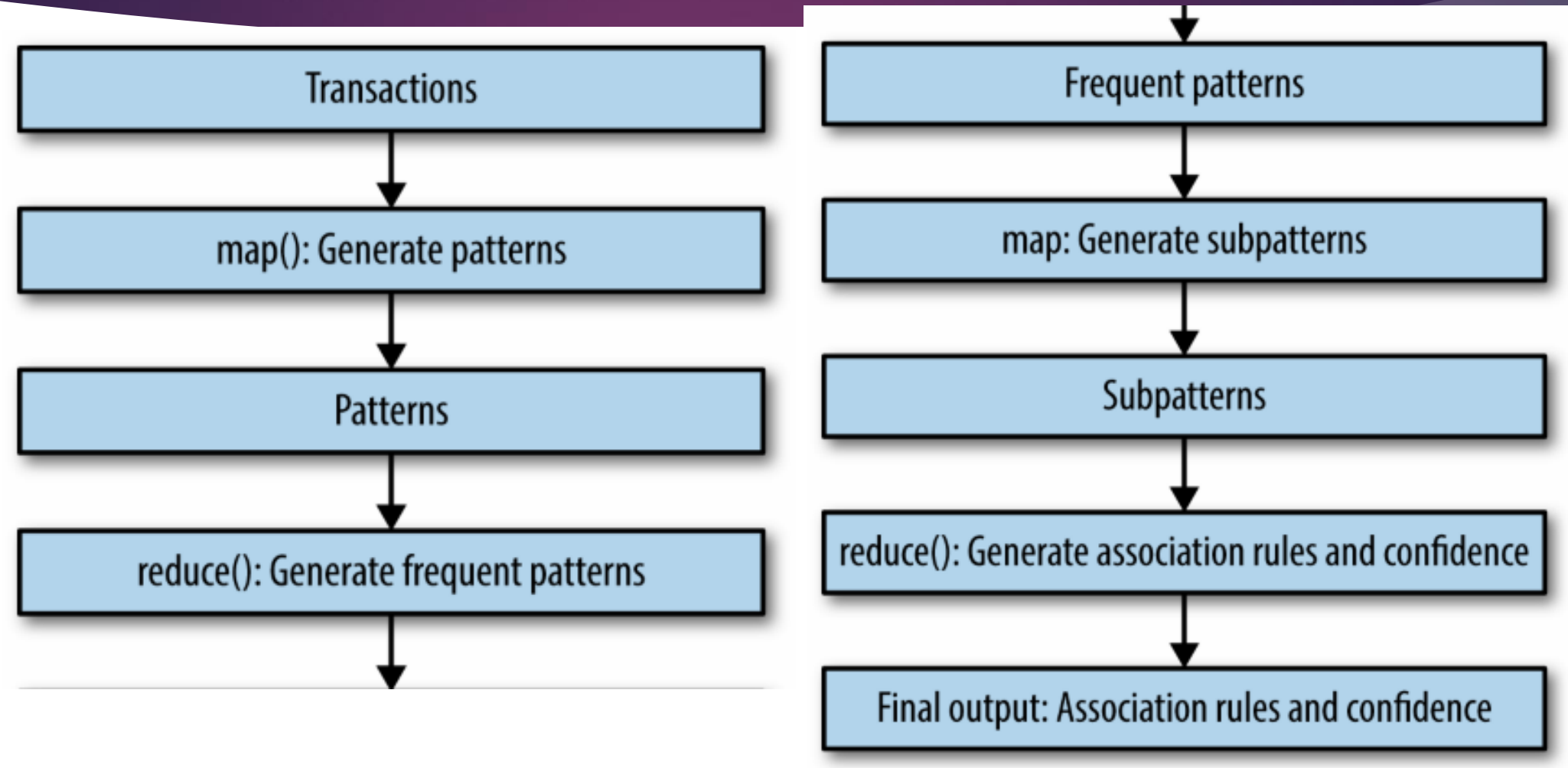
Output

```
# hadoop fs -cat /market_basket_analysis/output/part*
[bread, coffee]     1
[butter, coffee]    1
[banana, crackers]   1
[butter, coke]      3
[coffee, crackers]   2
[bread, butter]     1
[banana, bread]     1
[bread, crackers]    5
[coke, crackers]     2
[bread, coke]       1
[coffee, coke]      1
[butter, crackers]   2
```

Spark solution

- MapReduce Algorithm Workflow
  - MapReduce phase 1: mappers convert transactions to patterns, and reducers find frequent patterns.
  - MapReduce phase 2: mappers convert frequent patterns into subpatterns, and finally reducers generate association rules and their associated confidences.

MapReduce Algorithm Workflow



Input

- Input : transaction 셋트

```
transaction-1:  a,b,c
transaction-2:  a,b,d
transaction-3:  b,c
transaction-4:  b,c
```

Spark Implementation

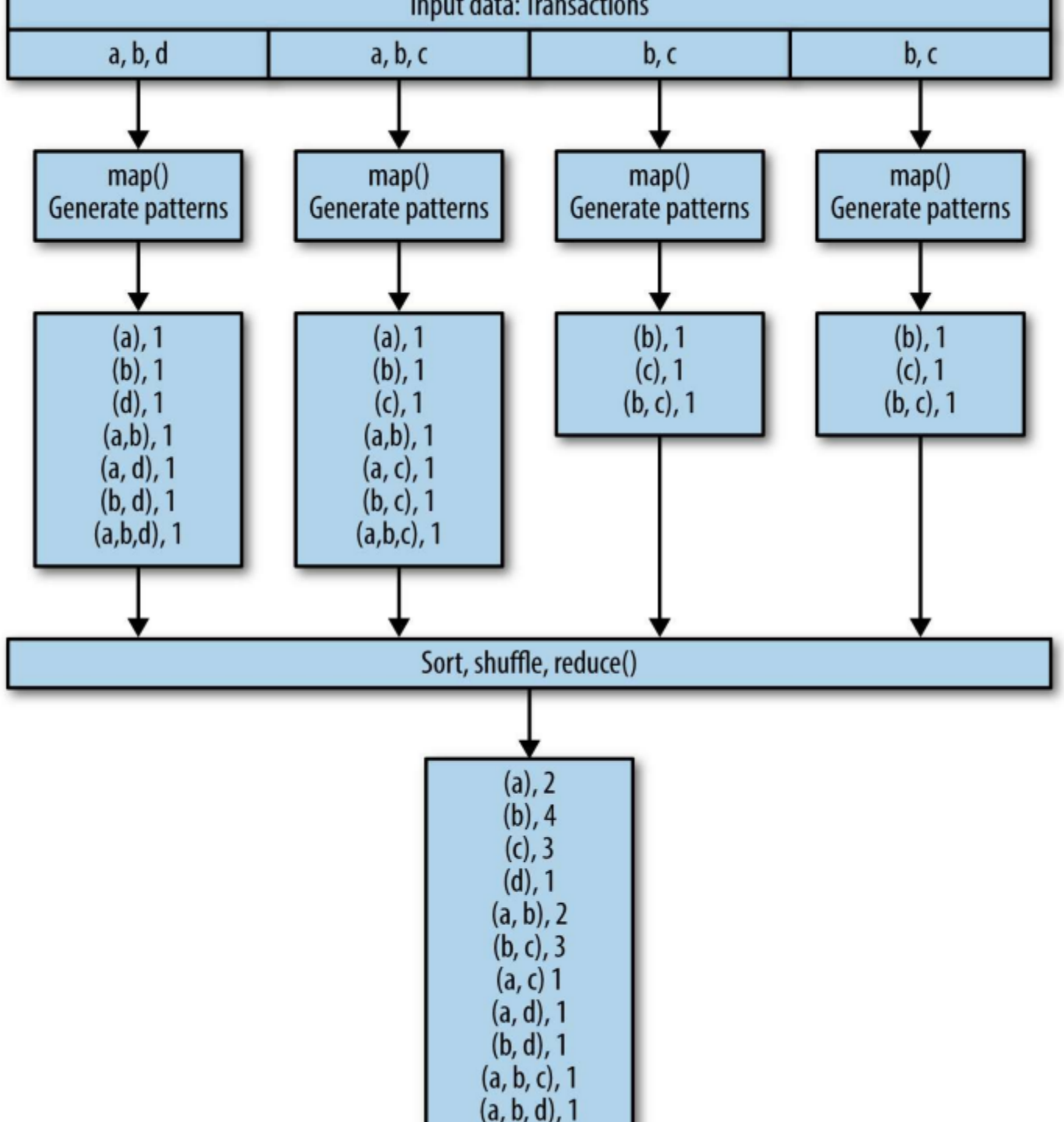
- 2 단계의 MapReduce 알고리즘 사용
  - 첫단계 : high-level step

Example 7-8. High-level steps

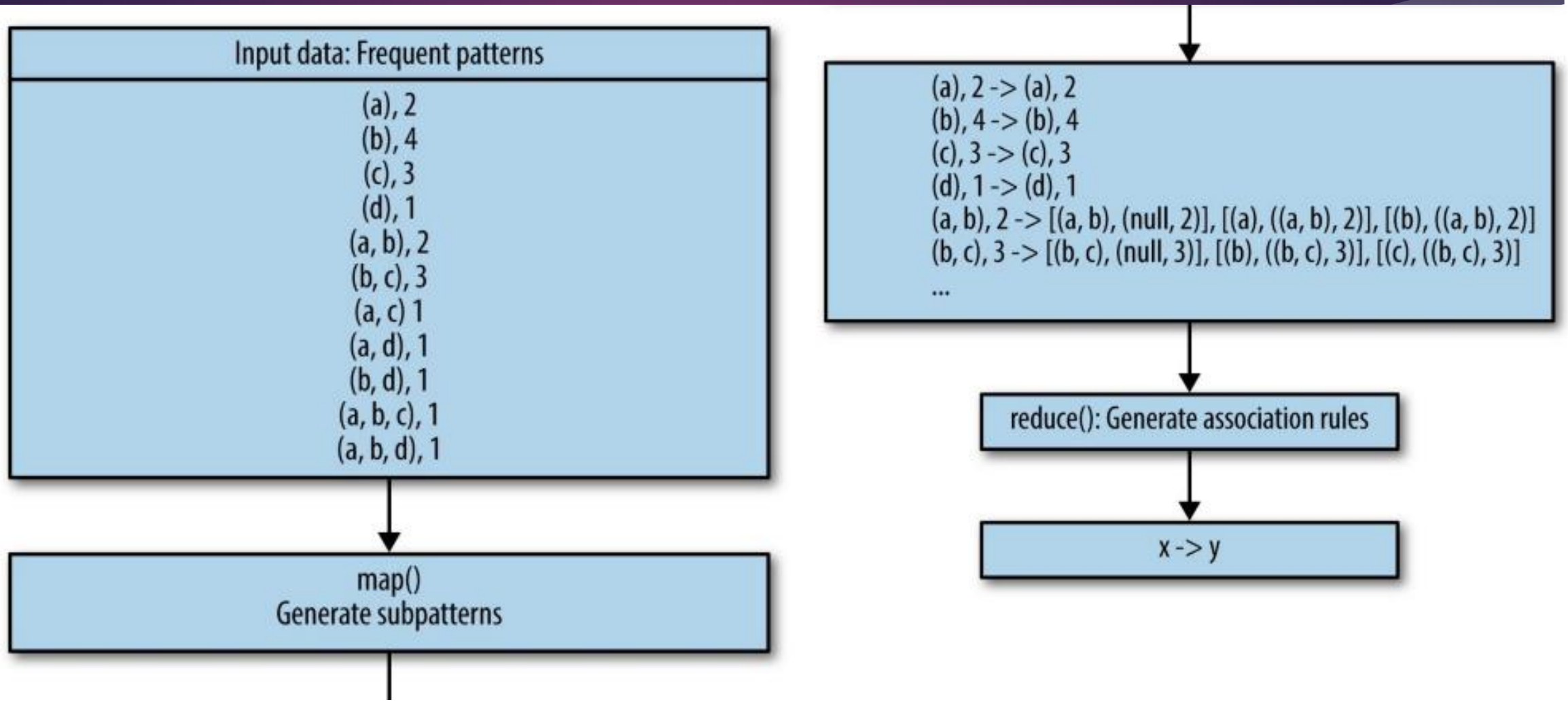
```
1  // Step 1: import required classes and interfaces
2  public class FindAssociationRules {
3
4     static JavaSparkContext createJavaSparkContext() {...}
5     static List<String> toList(String transaction) {...}
6     static List<String> removeOneItem(List<String> list, int i)
{...}
7
8     public static void main(String[] args) throws Exception {
9        // Step 2: handle input parameters
10       // Step 3: create a Spark context object
11       // Step 4: read all transactions from HDFS and create the
first RDD
12       // Step 5: generate frequent patterns (map() phase 1)
13       // Step 6: combine/reduce frequent patterns (reduce() phase
1)
14       // Step 7: generate all subpatterns (map() phase 2)
15       // Step 8: combine subpatterns
16       // Step 9: generate association rules (reduce() phase 2)
17       System.exit(0);
18    }
19 }
```

Mapreduce phase I

- Transaction을 patter으로 변환
- Frequent pattern 을 찾는다.



MapReduce phase 2 : association rule 생성



Spark Solution - 1

- Step 1 : Import required classes and interfaces
  - Create a spark context object
  - Utility functions
- Step 2 : Handle input parameters
- Step 3 : Create a Spark context object
- Step 4 : Read transaction from HDFS and create an RDD

Spark Solution - 2

- Step 5 : Generate frequent patterns
- Step 6 : Combine/reduce frequent patterns : reduceByKey()
- Step 7 : Generate all subpatterns
- Step 8 : Combine subpatterns : groupByKey()
- Step 9 : Generate association rules : JavaPairRDD.map()