

Map Reduce - UN17

Fundamentals – Part II

roberiogomes@gmail.com

Agenda

Top N

Moving Average

Market Basket Analysis

Recommendation Engines



TOP N List

- Given a set of (key, value) pairs, say we want to create a top N ($N > 0$);
- Top N is a design pattern for Map Reduce.

Top N, Formalized

Let N be an integer number and $N > 0$. Let L be a `List<Tuple2<T, Integer>>`, where T can be any type (such as a string or URL); $L.size() = S$; $S > N$; and elements of L be:

- $\{(K_i, V_i), 1 \leq i \leq S\}$

where K_i has a type of T and V_i is an `Integer` type (this is the frequency of K_i). Let `sort(L)` return the sorted values of L by using the frequency as a key, as follows:

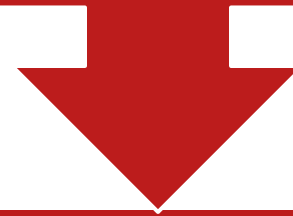
- $\{(A_j, B_j), 1 \leq j \leq S, B_1 \geq B_2 \geq \dots \geq B_S\}$

where $(A_j, B_j) \in L$. Then the top N of L is defined as:

- $topN(L) = \{(A_j, B_j), 1 \leq j \leq N, B_1 \geq B_2 \geq \dots \geq B_N \geq B_{N+1} \geq \dots \geq B_S\}$

The Map Reduce solution

Each mapper will find a local top N list (for $N > 0$) and then will pass it to a *single* reducer.



Then the single reducer will find the final top N list from all the local top N lists passed from the mappers.

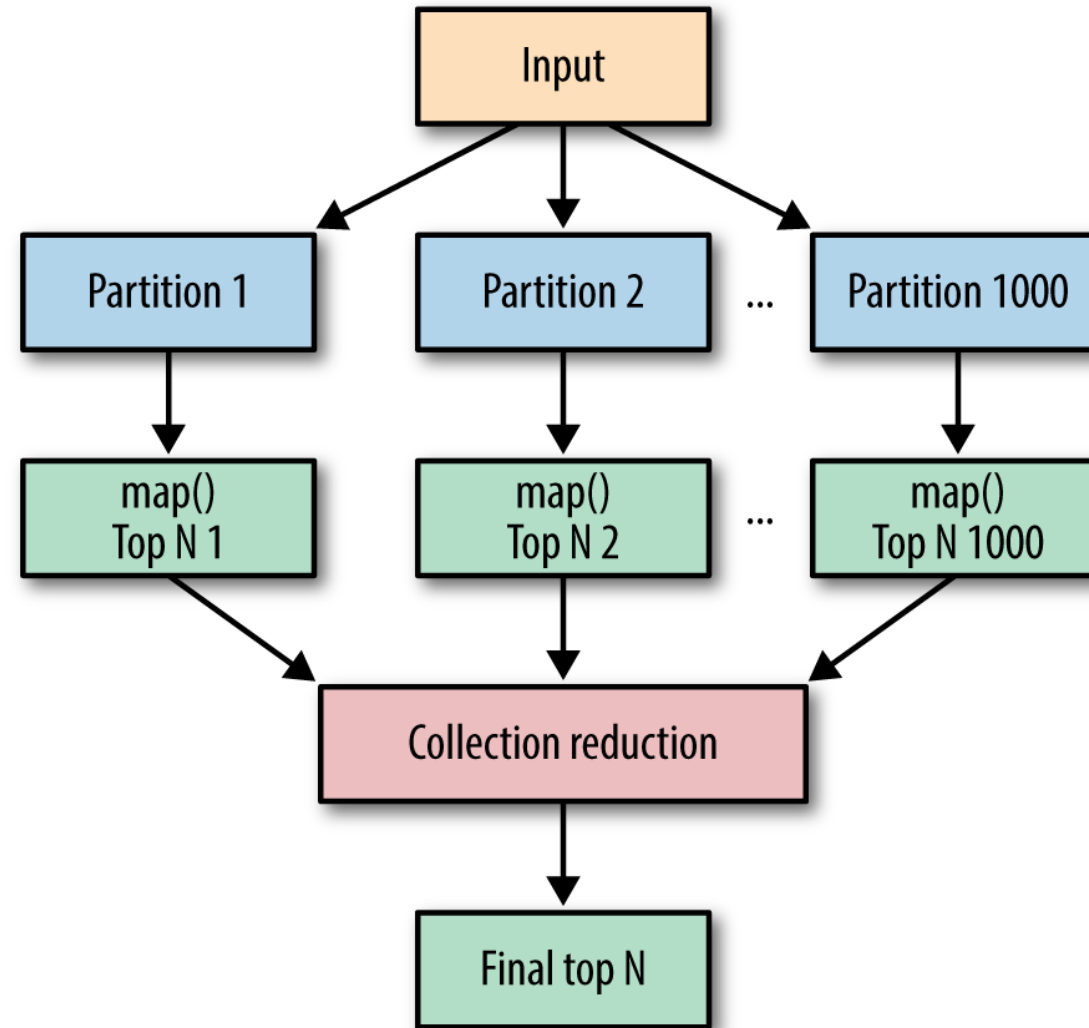
Solution

In general, in most of the MapReduce algorithms, having a single reducer is problematic and will cause a performance bottleneck;

Here, our single reducer will not cause a performance problem. Why?

Let's assume that we have 1,000 mappers, so each mapper will generate only 10 key-value pairs. Therefore, our single reducer will get only 10,000 ($10 \times 1,000$) records—which is not nearly enough data to cause a performance bottleneck!

Top N algorithm





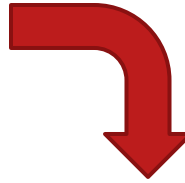
FOOTBALL TOURNAMENT



```

1 /**
2  * @param key is generated by MapReduce framework, ignored here
3  * @param value as a String has the format:
4  *   <cat_weight><,><cat_id><,><cat_name>
5  */
6 map(key, value) {
7     String[] tokens = value.split(",");
8     // cat_weight = tokens[0];
9     // <cat_id><,><cat_name> = tokens[1]
10    Double weight = Double.parseDouble(tokens[0]);
11    top10cats.put(weight, value);
12
13    // keep only top N
14    if (top10cats.size() > N) {
15        // remove the element with the smallest frequency
16        top10cats.remove(top10cats.firstKey());
17    }
18 }

```



```

1 /**
2  * cleanup() function will be executed once at the end of each mapper
3  * Here we set up the "cats top N list" as top10cats
4  */
5 cleanup(Context context) {
6     // now we emit top N from this mapper
7     for (String catAttributes : top10cats.values() ) {
8         context.write(NullWritable.get(), catAttributes);
9     }
10 }

```



Let's Practice!!!

Moving Average

G1

CEARÁ 50 ANOS

Fortaleza reduz em 98% a média móvel de casos de Covid-19 em comparação ao pico da pandemia

Diário do Nordeste

HOME CORONAVÍRUS DIAS MELHORES METRO POLÍTICA JOGADA

Fortaleza reduz em mais de 90% as médias móveis de casos e óbitos de Covid-19 desde o pico da doença



Moving Average

First, though, we need to understand *time series* data.

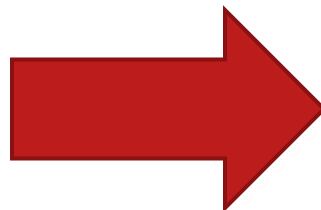


Time series data represents the values of a variable over a period of time, such as a second, minute, hour, day, week, month, quarter, or year.



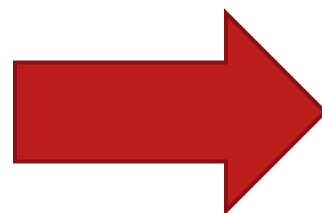
Semiformally, we can represent time series data as a sequence of triplets: (k, t, v)

Time series	Date	Closing price
1	2013-10-01	10
2	2013-10-02	18
3	2013-10-03	20
4	2013-10-04	30
5	2013-10-07	24
6	2013-10-08	33
7	2013-10-09	27
...



Time series	Date	Moving average	How calculated
1	2013-10-01	10.00	$= (10)/(1)$
2	2013-10-02	14.00	$= (10+18)/(2)$
3	2013-10-03	16.00	$= (10+18+20)/(3)$
4	2013-10-04	22.66	$= (18+20+30)/(3)$
5	2013-10-07	24.66	$= (20+30+24)/(3)$
6	2013-10-08	29.00	$= (30+24+33)/(3)$
7	2013-10-09	28.00	$= (24+33+27)/(3)$

URL	Date	Unique visitors
URL1	2013-10-01	400
URL1	2013-10-02	200
URL1	2013-10-03	300
URL1	2013-10-04	700
URL1	2013-10-05	800
URL2	2013-10-01	10
URL2	2013-10-02	20
URL2	2013-10-03	30



URL	Date	Moving average
URL1	2013-10-01	400
URL1	2013-10-02	300
URL1	2013-10-03	300
URL1	2013-10-04	400
URL1	2013-10-05	600
URL2	2013-10-01	10
URL2	2013-10-02	15
URL2	2013-10-03	20
URL2	2013-10-04	40

POJO Version

```
1 import java.util.Queue;
2 import java.util.LinkedList;
3 public class SimpleMovingAverage {
4
5     private double sum = 0.0;
6     private final int period;
7     private final Queue<Double> window = new LinkedList<Double>();
8
9     public SimpleMovingAverage(int period) {
10         if (period < 1) {
11             throw new IllegalArgumentException("period must be > 0");
12         }
13         this.period = period;
14     }
15
16     public void addNewNumber(double number) {
17         sum += number;
18         window.add(number);
19         if (window.size() > period) {
20             sum -= window.remove();
21         }
22     }
23 }
```


The input to our MapReduce solution will have the following format:

`<name-as-string><,><date-as-timestamp><,><value-as-double>`

In this simple example, I've provided stock prices for three companies/stock symbols: GOOG, AAPL, and IBM. The first column is the stock symbol, the second is the timestamp, and the third is the adjusted closing price of the stock on that day:

GOOG,2004-11-04,184.70
GOOG,2004-11-03,191.67
GOOG,2004-11-02,194.87
AAPL,2013-10-09,486.59
AAPL,2013-10-08,480.94
AAPL,2013-10-07,487.75
AAPL,2013-10-04,483.03
AAPL,2013-10-03,483.41
IBM,2013-09-30,185.18
IBM,2013-09-27,186.92
IBM,2013-09-26,190.22
IBM,2013-09-25,189.47
GOOG,2013-07-19,896.60
GOOG,2013-07-18,910.68
GOOG,2013-07-17,918.55

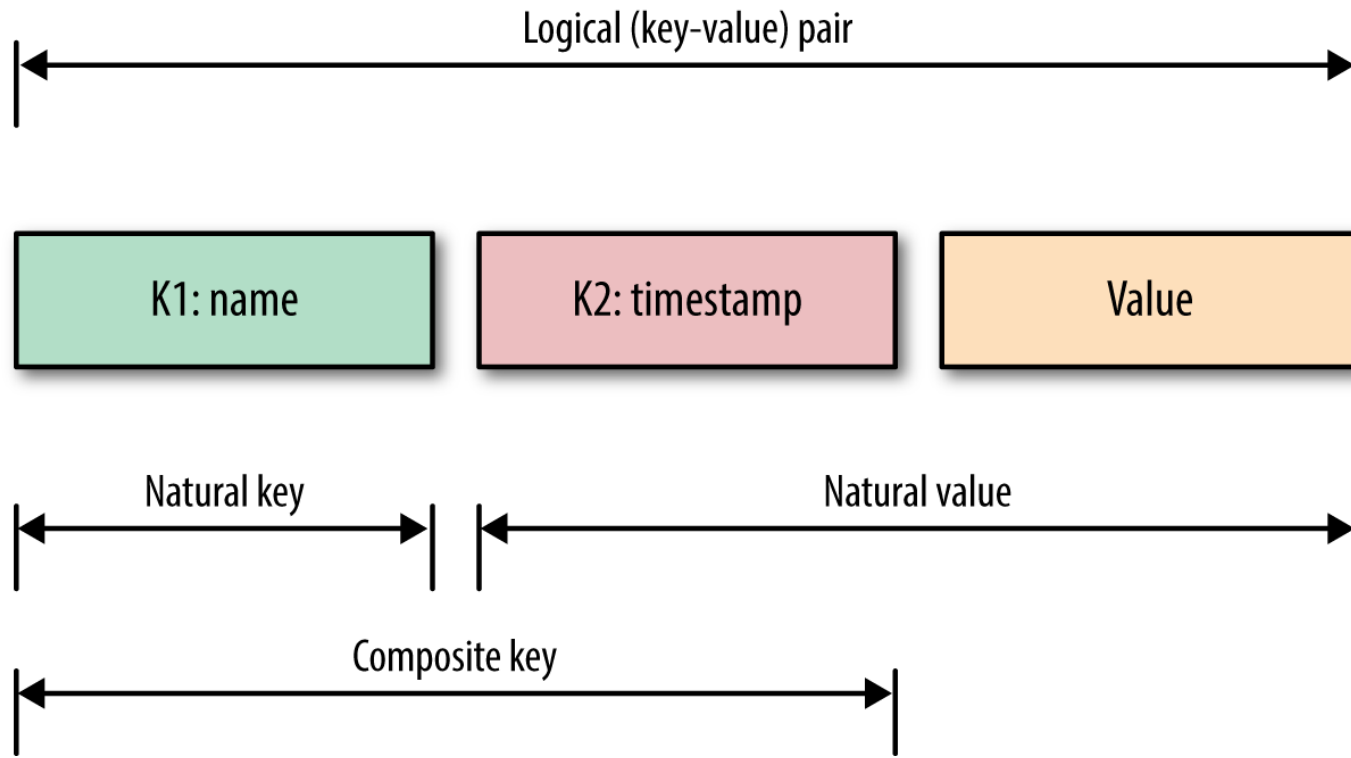


Name: GOOG,	Date: 2013-07-17	Moving Average: 912.52
Name: GOOG,	Date: 2013-07-18	Moving Average: 912.01
Name: GOOG,	Date: 2013-07-19	Moving Average: 916.39



Let's Practice!!!

Let's try
using
Secondary
Sort!





Let's Practice!!!

MBA


Market Basket Analysis (MBA) is a popular data mining technique, frequently used by marketing and ecommerce professionals to reveal affinities between individual products or product groupings.

The general goal of data mining is to extract interesting correlated information from a large collection of data—for example, millions of supermarket or credit card sales transactions.

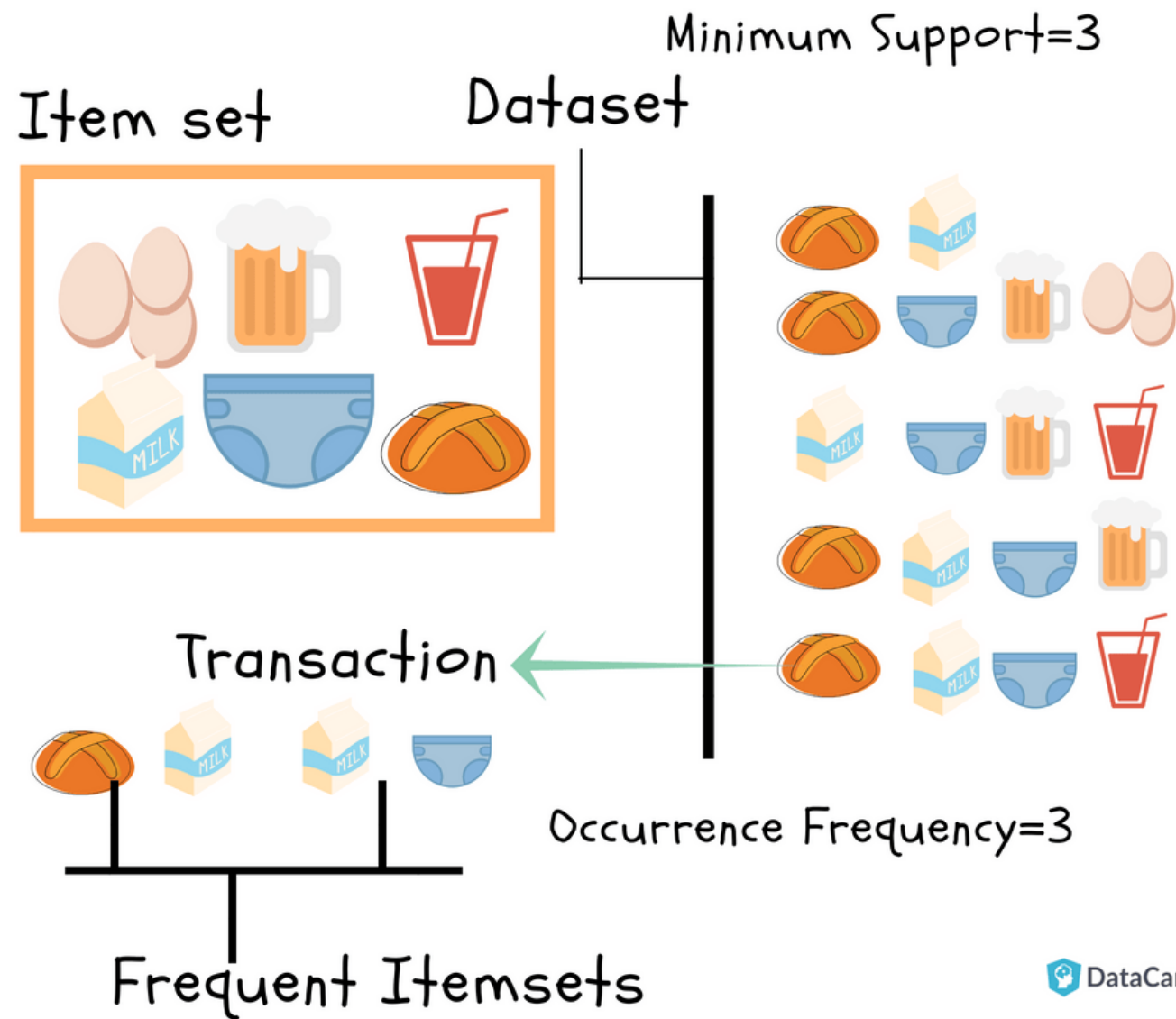
Market Basket Analysis helps us identify items likely to be purchased together, and association rule mining finds correlations between items in a set of transactions.

Market Basket Analysis


Marketers may then use these association rules to place correlated products next to each other on store shelves or online so that customers buy more items.



Finding *frequent sets* in mining association rules for Market Basket Analysis is a computationally intensive problem, making it an ideal case for MapReduce.



Once we have the most frequent item sets F_i (where $i = 1, 2, \dots$), we can use them to produce an association rule of the transaction. For example, if we have five items $\{A, B, C, D, E\}$ with the following six transactions:



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



then our goal is to build frequent item sets F_1 (size = 1) and F_2 (size = 2) as:

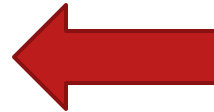
- $F_1 = \{[C, 3], [A, 3], [B, 3], [E, 4]\}$
- $F_2 = \{[<A,C>, 2], [<C,E>, 2], [<A,E>, 2], [<B,E>, 2]\}$

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
 ...



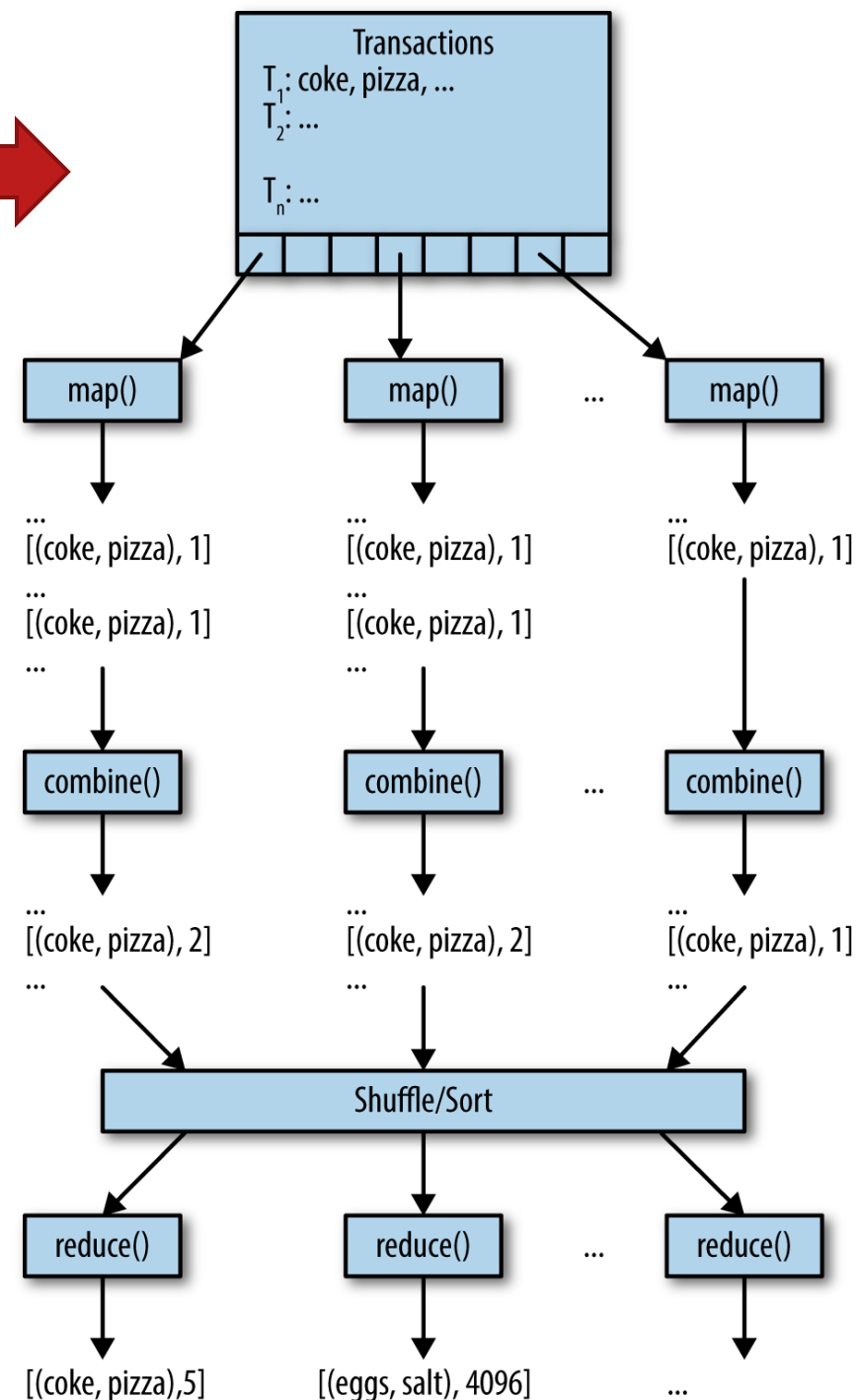
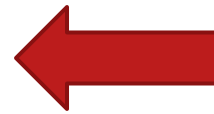
Pair of items Frequency

...	...
(bread, crackers)	8,709
(baguette, eggs)	7,524
(crackers, coke)	4,300
...	...



Triplet of items Frequency

...	...
(bread, crackers, eggs)	1,940
(baguette, diapers, eggs)	1,900
(crackers, coke, meat)	1,843
...	...



The map() function will take a single transaction and generate a set of key-value pairs to be consumed by reduce().

The mapper pairs the transaction items as a key and the number of key occurrences in the basket as its value (for all transactions and without the transaction numbers).



```
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  
...
```



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



```
 $T_1$ : crackers, icecream, coke  
 $T_2$ : 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 ]
```

Commutative Property



The diagram shows the equation $a + b = b + a$ on a chalkboard. Two red curved arrows illustrate the commutative property: one arrow starts above the 'a' and points to the 'a' in the second expression, and another arrow starts below the 'b' and points to the 'b' in the second expression.

$$a + b = b + a$$

We can avoid this problem
if we sort the transaction
items in alphabetical order
before generating key-
value pairs.



```

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 }

```



```

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 }

```




Let's Practice!!!