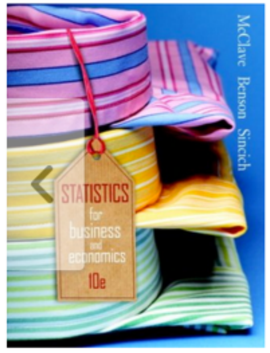
Recommendations using MapReduce

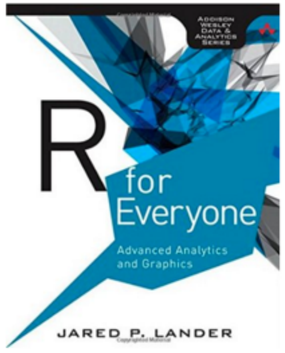
RECOMMENDATIONS

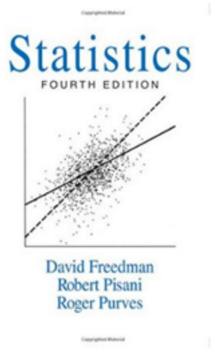
are ubiquitous

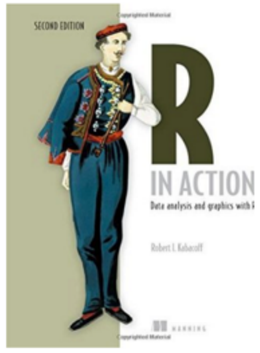
Amazon

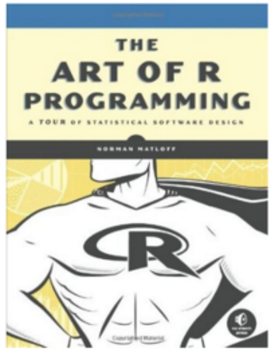
Related to Items You've Viewed See more





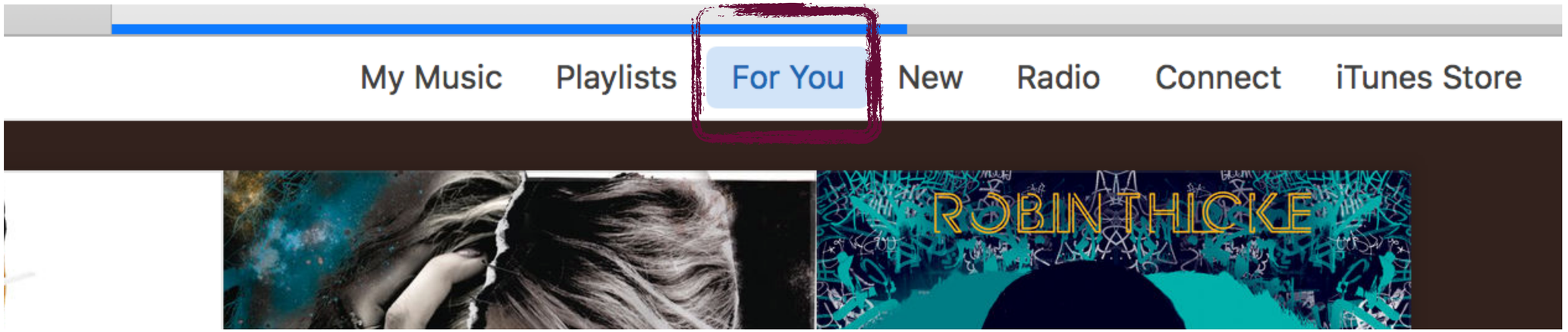




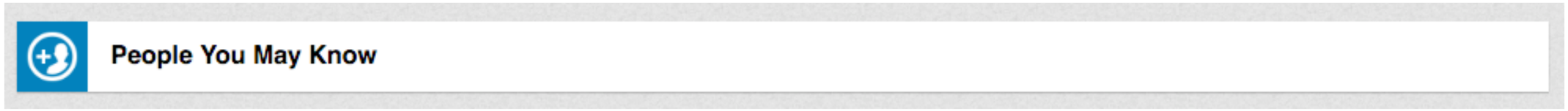


Inspired by Your Browsing History See more

iTunes


My Music
Playlists
For You
New
Radio
Connect
iTunes Store
RÖBIN THICKE

LinkedIn



RECOMMENDATIONS

are ubiquitous

There are many interesting algorithms for generating Recommendations, but most are based on a simple idea called

COLLABORATIVE FILTERING

COLLABORATIVE FILTERING

How does that work?

How do you normally find
A movie to watch?
A restaurant to go to?
A new artist to check out?
A new book to read?

SOMEONE WHO LIKES

[redacted]

COLLABORATIVE FILTERING

Let's use this idea to find **Friend Recommendations** for a Social Networking site

Friend Recommendations

We have a dataset with User name and their list of friends

User	Friends
Swetha	Janani, Pratik
Navdeep	Vitthal, Janani, Pratik
Vitthal	Navdeep, Janani
..	..

Friend Recommendations

User	Friends
Swetha	Janani, Pratik
Navdeep	Vitthal, Janani, Pratik
Vitthal	Navdeep, Janani
..	..

Swetha and Navdeep have 2 common friends

Friend Recommendations

User	Friends
Swetha	Janani, Pratik
Navdeep	Vitthal, Janani, Pratik
Vitthal	Navdeep, Janani
..	..

Recommend Navdeep as a friend for Swetha

Friend Recommendations

User	Friends
Swetha	Janani, Pratik
Navdeep	Vitthal, Janani, Pratik
Vitthal	Navdeep, Janani
..	..

This is exactly the idea behind Collaborative Filtering

Friend Recommendations

Collaborative Filtering

Find users who have a lot in common

Friend Recommendations

Collaborative Filtering

Recommend their "likes" to each other

Friend Recommendations

Collaborative Filtering

Let's express this algorithm using MapReduce

Friend Recommendations

Collaborative Filtering

MapReduce

Objective: Find the Top 10 Friend Recommendations for each user

Top 10 Friend Recommendations

This will actually require 2 MapReduce jobs

<User, List of Friends>

MapReduce 1

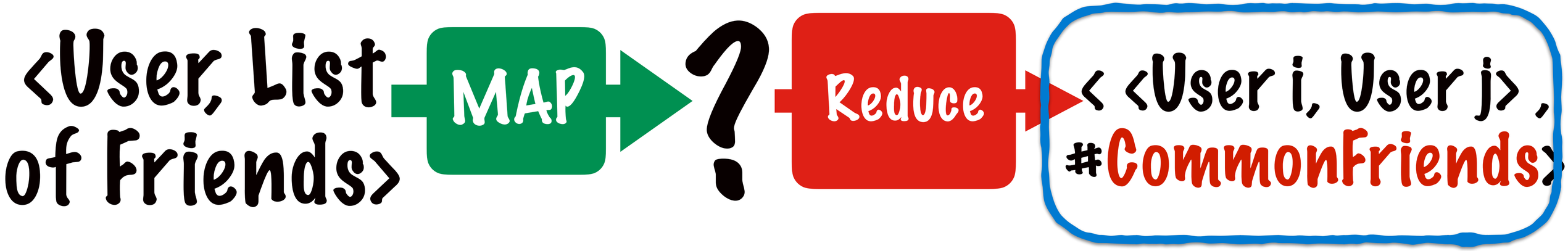
< <User i, User j> , #CommonFriends>

MapReduce 2

<User, Top 10 Recommendations>

Top 10 Friend Recommendations

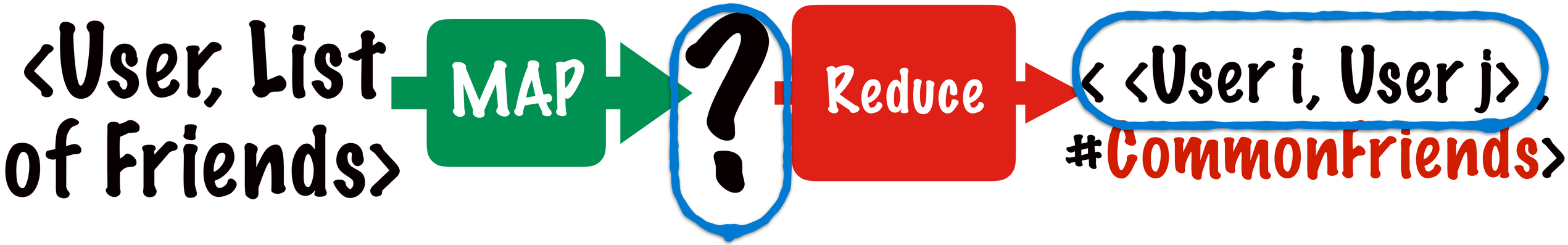
MapReduce 1



Every pair of Users who are not already friends

Top 10 Friend Recommendations

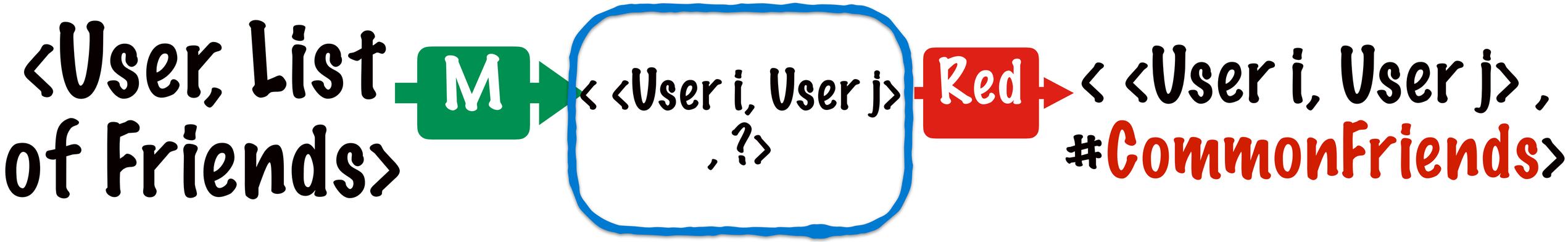
MapReduce 1



The key here will be a pair of users (same as reducer)

Top 10 Friend Recommendations

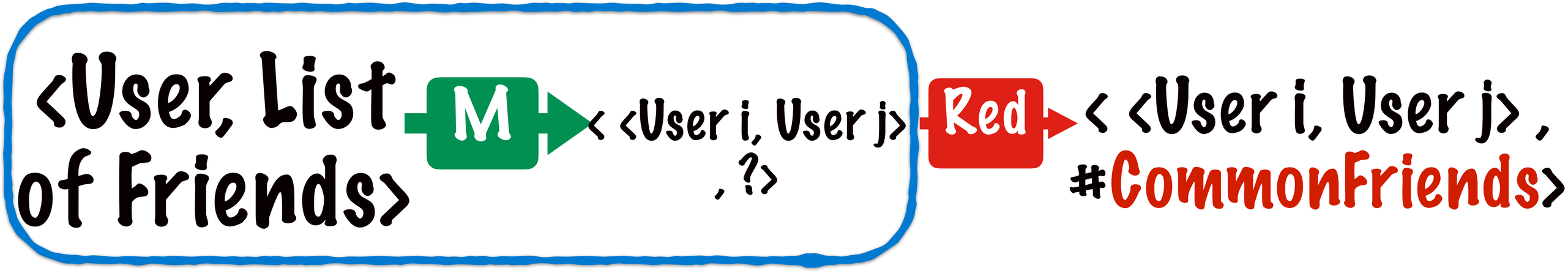
MapReduce 1



The key here will be a pair of users (same as reducer)

Top 10 Friend Recommendations

MapReduce 1



Let's understand the logic involved in this step

Top 10 Friend Recommendations

<User, List of Friends> → MAP → < <User i, User j> , ?>

Here is 1 record from the data set

User	Friends
Navdeep	Vitthal, Janani, Pratik

Top 10 Friend Recommendations

User	Friends
Navdeep	Vitthal, Janani, Pratik

< <User i, User j>
, ?>

Navdeep is a common friend for these 3 folks

Top 10 Friend Recommendations

User	Friends
Navdeep	Vitthal, Janani, Pratik

< <User i, User j> , ?>

Key	Value
Vitthal, Janani	1
Vitthal , Pratik	1
Janani, Pratik	1

One set of outputs for this record is every possible pair out these 3 folks

Top 10 Friend Recommendations

User	Friends
Navdeep	Vitthal, Janani, Pratik

< <User i, User j> , ?>

Key	Value
Vitthal, Janani	1
Vitthal , Pratik	1
Janani, Pratik	1

The value here represents 1 common friend for this pair

Top 10 Friend Recommendations

User	Friends
Navdeep	Vitthal, Janani, Pratik

< <User i, User j> , ?>

Key	Value
Vitthal, Janani	1
Vitthal , Pratik	1
Janani, Pratik	1
Janani, Vitthal	1
Pratik, Vitthal	1
Pratik, Janani	1

The pairs work both ways and both should be included

Top 10 Friend Recommendations

User	Friends
Navdeep	Vitthal, Janani, Pratik

< <User i, User j> , ?>

Key	Value
Vitthal, Janani	1
Vitthal , Pratik	1
Janani, Pratik	1
Janani, Vitthal	1
Pratik, Vitthal	1
Pratik, Janani	1

Vitthal has a common friend with Janani and Janani has a common friend with Vitthal

Top 10 Friend Recommendations

User	Friends
Navdeep	Vitthal, J… P…

MAP

< <User i, User j> ?>

We also need to keep track of which folks are already friends

Top 10 Friend Recommendations

User	Friends
Navdeep	Vitthal, Janani, Pratik

MAP

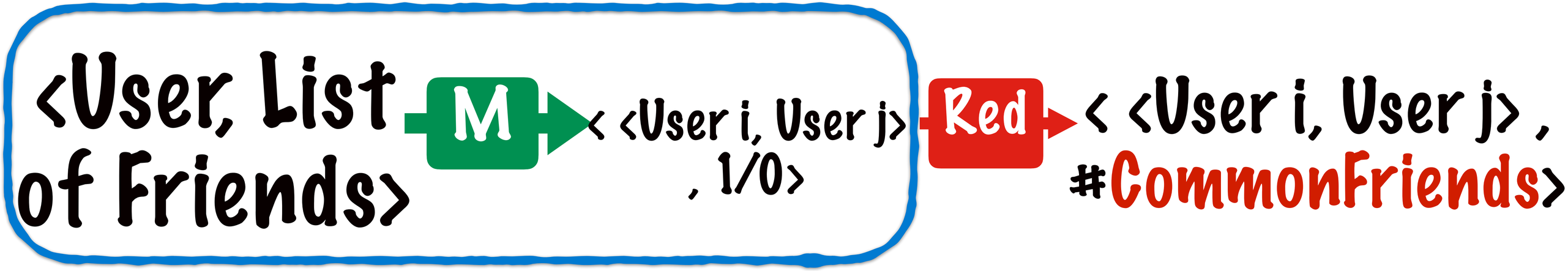
< <User i, User j> , ?>

Key	Value
Vitthal, Janani	1
Vitthal , Pratik	1
Janani, Pratik	1
Janani, Vitthal	1
Pratik, Vitthal	1
Pratik, Janani	1
Navdeep, Janani	0
Navdeep , Vitthal	0
Navdeep, Pratik	0

We've given a different value here for folks who are friends

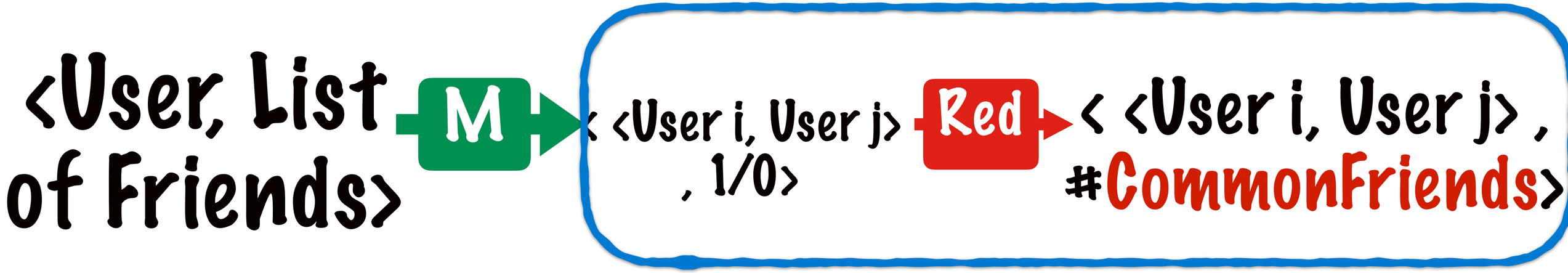
Top 10 Friend Recommendations

MapReduce 1



Top 10 Friend Recommendations

MapReduce 1



Top 10 Friend Recommendations

< <User i, User j> , 1/0>

Red

< <User i, User j> , #CommonFriends>

Now we have user pairs with value either 1 or 0

1 => a common friend

0 => already friends

Top 10 Friend Recommendations

< <User i, User j> , 1/0>

Red

< <User i, User j> , #CommonFriends>

1 => a common friend

0 => already friends

Sum= number of common friends

If Product =0 means they are already friends

Top 10 Friend Recommendations

< <User i, User j> , 1/0>

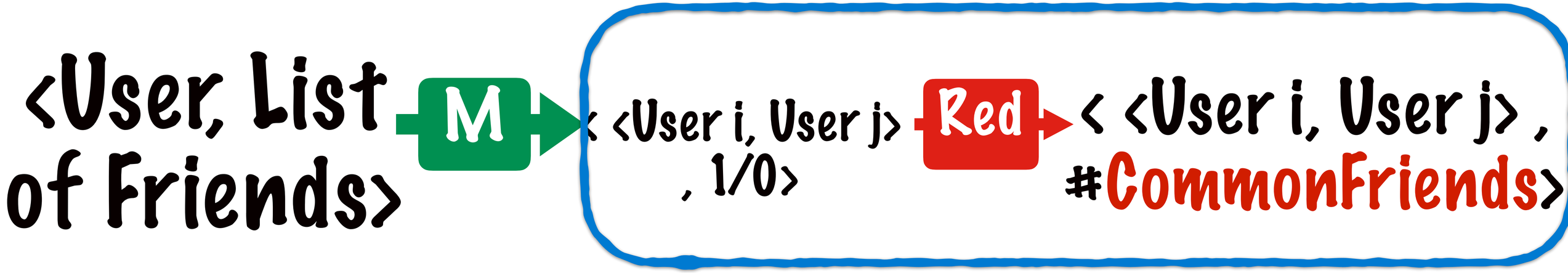
Red

< <User i, User j> , #CommonFriends>

We'll check if the product of values <> 0 and write the sum of values to output

Top 10 Friend Recommendations

MapReduce 1



Top 10 Friend Recommendations

This will actually require 2 MapReduce jobs

<User, List of Friends>

MapReduce 1

< <User i, User j> , #CommonFriends>>

MapReduce 2

<User, Top 10 Recommendations>

Top 10 Friend Recommendations

This will actually require 2 MapReduce jobs

<User, List of Friends>

MapReduce 1

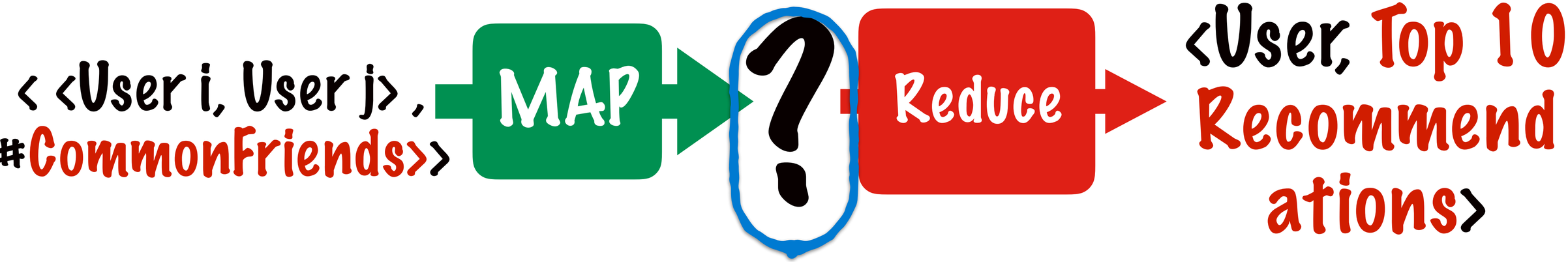
< <User i, User j> , #CommonFriends>>

MapReduce 2

<User, Top 10 Recommendations>

Top 10 Friend Recommendations

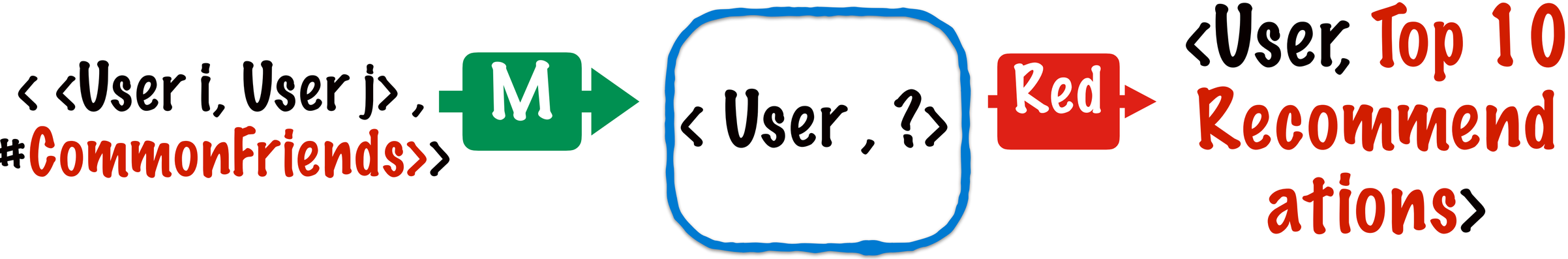
MapReduce 2



The key here will be user

Top 10 Friend Recommendations

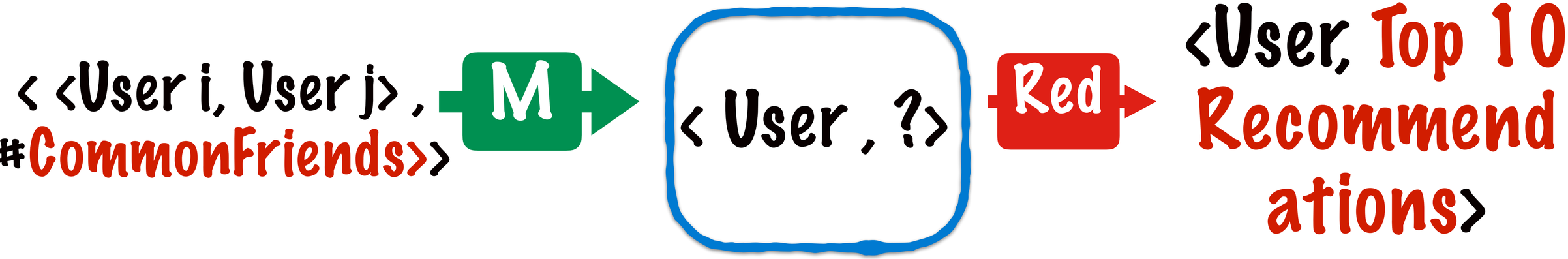
MapReduce 2



The key here will be user

Top 10 Friend Recommendations

MapReduce 2

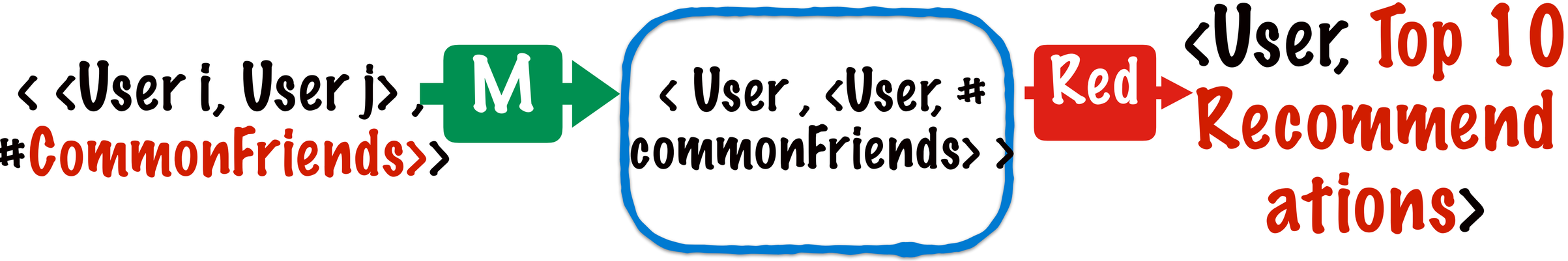


The value will be a pair

<User, # commonFriends>

Top 10 Friend Recommendations

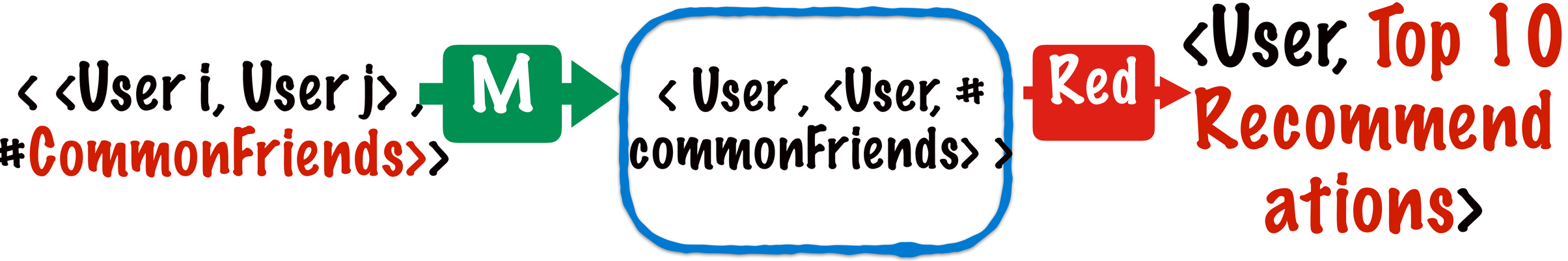
MapReduce 2



The key is the user and the value is another user along with the number of friends they have in common

Top 10 Friend Recommendations

MapReduce 2



Top 10 Friend Recommendations

MapReduce 2

The reducer will sort the users in descending order of common friends and pick the Top 10

Top 10 Friend Recommendations

This will actually require 2 MapReduce jobs

<User, List of Friends>

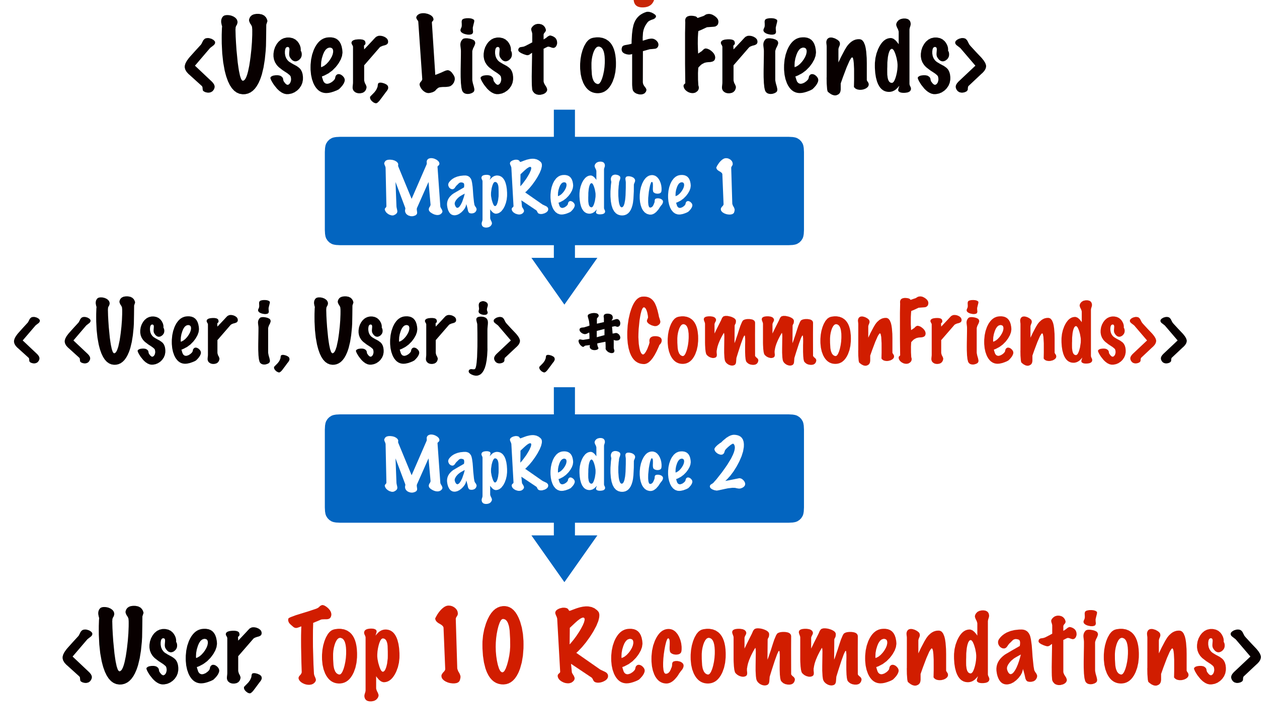
MapReduce 1

< <User i, User j> , #CommonFriends>>

MapReduce 2

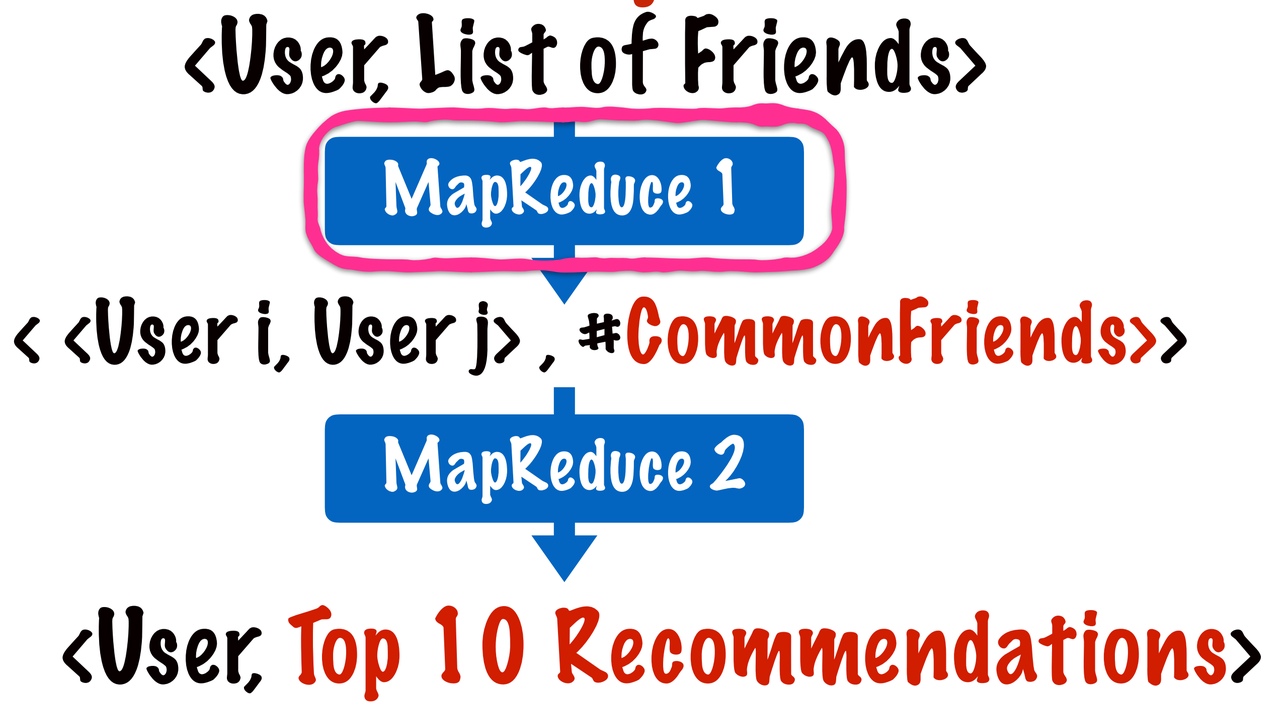
<User, Top 10 Recommendations>

Top 10 Friend Recommendations



Let's see the list of classes we'll have to write in Java

Top 10 Friend Recommendations



Java classes

Mapl

Reducel

Top 10 Friend Recommendations

Java classes

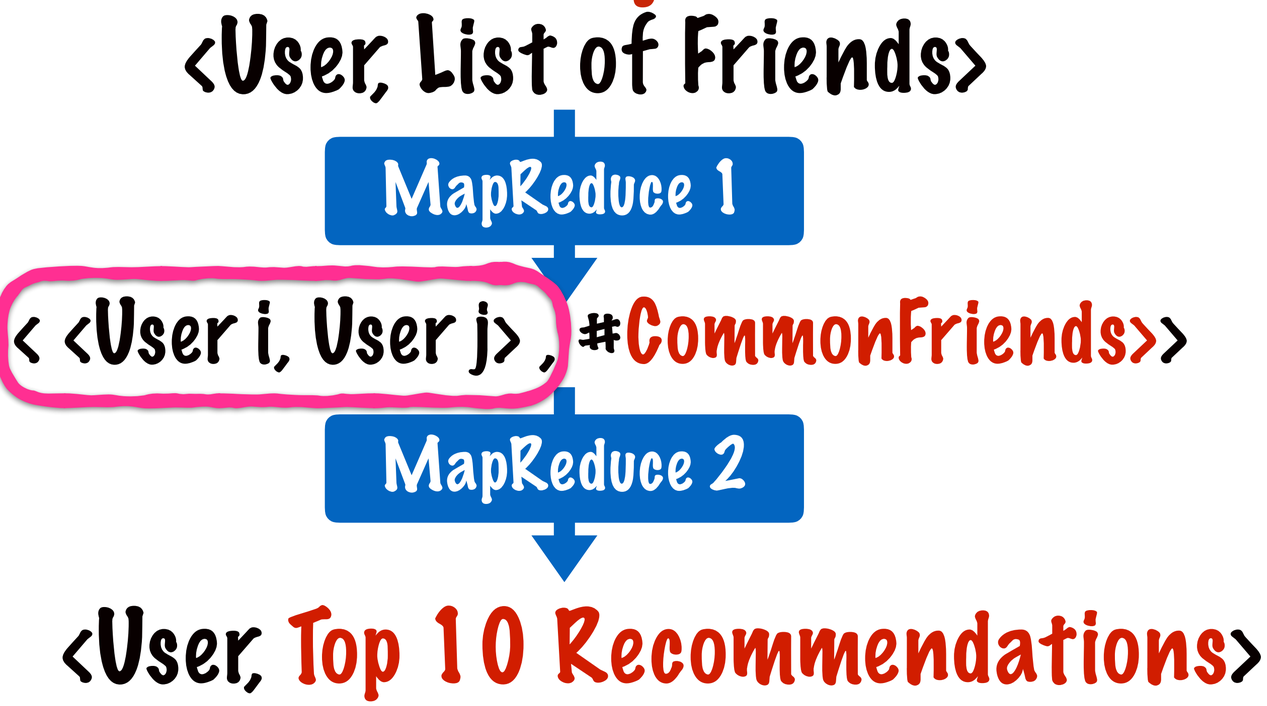
Map1

Reduce1

Map2

Reduce2

Top 10 Friend Recommendations



Java classes

Map1 Map2

Reduce1 Reduce2

TextPair

Top 10 Friend Recommendations

<User, List of Friends>

MapReduce 1

< <User i, User j> , #CommonFriends>>

M

< User , <User, # commonFriends> >

Redu

<User, Top 10 Recommendations>

Java classes

Map1 Map2

Reduce1 Reduce2

TextPair

TextIntPair

These are Writable classes

Top 10 Friend Recommendations

Java classes

Map1 Map2

Reduce1 Reduce2

TextPair

TextIntPair

Recommendations

This is the Main class where we'll chain these 2 jobs

Top 10 Friend Recommendations

Java classes

Map1
Reduce1

Map2
Reduce2

TextPair
TextIntPair

Recommendations

Top 10 Friend Recommendations

TextPair

```
public class TextPair implements
WritableComparable<TextPair>{
```

```
    private Text first;
    private Text second;

    public void set(Text first, Text second){
        this.first=first;
        this.second=second;
    }

    public Text getFirst() {
        return first;
    }
    public Text getSecond() {
        return second;
    }


    public TextPair(){
        set(new Text(),new Text());
    }

    public TextPair(String first, String second){
        set(new Text(first),new Text(second));
    }

    public TextPair(Text first, Text second){
        set(first,second);
    }

    @Override
    public void write(DataOutput out) throws IOException{
        first.write(out);
        second.write(out);
    }

    @Override
    public void readFields(DataInput in) throws IOException{
        first.readFields(in);
```

This is same as a class we've previously written (for Bigram representation)

Top 10 Friend Recommendations

TextPair

```
public class TextPair implements WritableComparable<TextPair>{

    private Text first;
    private Text second;

    public void set(Text first, Text second){
        this.first=first;
        this.second=second;
    }

    public Text getFirst() {
        return first;
    }
    public Text getSecond() {
        return second;
    }


    public TextPair(){
        set(new Text(),new Text());
    }

    public TextPair(String first, String second){
        set(new Text(first),new Text(second));
    }

    public TextPair(Text first, Text second){
        set(first,second);
    }

    @Override
    public void write(DataOutput out) throws IOException{
        first.write(out);
        second.write(out);
    }

    @Override
    public void readFields(DataInput in) throws IOException{
        first.readFields(in);
```

This class has 2 member variables

Top 10 Friend Recommendations

TextPair

```
public TextPair(){
    set(new Text(),new Text());
}

public TextPair(String first, String second){
    set(new Text(first),new Text(second));
}

public TextPair(Text first, Text second){
    set(first,second);
}

@Override
public void write(DataOutput out) throws IOException{
    first.write(out);
    second.write(out);
}

@Override
public void readFields(DataInput in) throws IOException{
    first.readFields(in);
    second.readFields(in);
}

@Override
public int compareTo(TextPair tp){
    int cmp = first.compareTo(tp.first);
    if (cmp!=0) {
        return cmp;
    }
    return second.compareTo(tp.second);
}

@Override
public int hashCode(){
    return first.hashCode()*163 + second.hashCode();
}

@Override
public boolean equals(Object o){
    if( o instanceof TextPair){
        TextPair tp = (TextPair) o;
        return first.equals(tp.first) && second.equals(tp.second);
    }
    return  false;
}

@Override
public String toString(){
    return first +"\t" +second;
}
```

And a bunch of implemented methods

Top 10 Friend Recommendations

Java classes

Map1
Reduce1

Map2
Reduce2

TextPair

TextIntPair

Recommendations

Top 10 Friend Recommendations

Java classes

Map1
Reduce1

Map2
Reduce2

TextPair
TextIntPair

Recommendations

Top 10 Friend Recommendations

TextIntPair

public class TextIntPair **implements** WritableComparable<TextIntPair> {

```
private Text first;
private IntWritable second;

public void set(Text first, IntWritable second){
    this.first=first;
    this.second=second;
}

public Text getFirst() {
    return first;
}
public IntWritable getSecond() {
    return second;
}


public TextIntPair(){
    set(new Text(),new IntWritable());
}

public TextIntPair(String first, Integer second){
    set(new Text(first),new IntWritable(second));
}

public TextIntPair(Text first, IntWritable second){
    set(first,second);
}

public TextIntPair(TextIntPair tp){
    set(tp.getFirst(),tp.getSecond());
}

@Override
public void write(DataOutput out) throws IOException {
    first.write(out);
```

This class is very similar to TextPair

Top 10 Friend Recommendations

TextIntPair

```
public class TextIntPair implements WritableComparable<TextIntPair> {

    private Text first;
     private IntWritable second;

    public void set(Text first, IntWritable second){
        this.first=first;
        this.second=second;
    }

    public Text getFirst() {
        return first;
    }
    public IntWritable getSecond() {
        return second;
    }


    public TextIntPair(){
        set(new Text(),new IntWritable());
    }

    public TextIntPair(String first, Integer second){
        set(new Text(first),new IntWritable(second));
    }

    public TextIntPair(Text first, IntWritable second){
        set(first,second);
    }

    public TextIntPair(TextIntPair tp){
        set(tp.getFirst(),tp.getSecond());
    }

    @Override
    public void write(DataOutput out) throws IOException {
        first.write(out);
        second.write(out);
    }
```

We just need to make the second member an IntWritable

Top 10 Friend Recommendations

TextIntPair

```
public class TextIntPair implements WritableComparable<TextIntPair> {
```

```
    private Text first;
    private IntWritable second;
```

```
    public void set(Text first, IntWritable second){
        this.first=first;
        this.second=second;
    }

    public Text getFirst() {
        return first;
    }
    public IntWritable getSecond() {
        return second;
    }


    public TextIntPair(){
        set(new Text(),new IntWritable());
    }

    public TextIntPair(String first, Integer second){
        set(new Text(first),new IntWritable(second));
    }

    public TextIntPair(Text first, IntWritable second){
        set(first,second);
    }

    public TextIntPair(TextIntPair tp){
        set(tp.getFirst(),tp.getSecond());
    }

    @Override
    public void write(DataOutput out) throws IOException {
        first.write(out);
        second.write(out);
    }
```

Change the type in all the methods

Top 10 Friend Recommendations

TextIntPair

```
public class TextIntPair implements WritableComparable<TextIntPair> {

    private Text first;
    private IntWritable second;

    public void set(Text first, IntWritable second){
        this.first=first;
        this.second=second;
    }

    public Text getFirst() {
        return first;
    }
    public IntWritable getSecond() {
        return second;
    }


    public TextIntPair(){
        set(new Text(),new IntWritable());
    }

    public TextIntPair(String first, Integer second){
        set(new Text(first),new IntWritable(second));
    }

    public TextIntPair(Text first, IntWritable second){
        set(first,second);
    }

    public TextIntPair(TextIntPair tp){
        set(tp.getFirst(),tp.getSecond());
    }

    @Override
    public void write(DataOutput out) throws IOException {
        first.write(out);
        second.write(out);
    }
```

The only method with a different implementation is compareTo()

Top 10 Friend Recommendations

TextIntPair

```
    set(new Text(first),new IntWritable(second));
}

public TextIntPair(Text first, IntWritable second){
    set(first,second);
}

public TextIntPair(TextIntPair tp){
    set(tp.getFirst(),tp.getSecond());
}

@Override
public void write(DataOutput out) throws IOException {
    first.write(out);
    second.write(out);
}

@Override
public void readFields(DataInput in) throws IOException {
    first.readFields(in);
    second.readFields(in);
}
```

```
@Override
public int compareTo(TextIntPair tp){
  return second.compareTo(tp.second);
}
```

```
@Override
public int hashCode(){
    return first.hashCode()*163 + second.hashCode();
}

@Override
public boolean equals(Object o){
    if( o instanceof TextIntPair){
        TextIntPair tp = (TextIntPair) o;
        return first.equals(tp.first) && second.equals(tp.second);
    }
    return  false;
}

@Override
public String toString(){
    return first +"\t" +second.toString();
}

}
```

These pairs are compared based on the Integer value i.e. the second member

Top 10 Friend Recommendations

TextIntPair

```
@Override
public int compareTo(TextIntPair tp){
  return second.compareTo(tp.second);
}
```

The integer value will represent the # common friends

Top 10 Friend Recommendations

Java classes

Map1
Reduce1

Map2
Reduce2

TextPair
TextIntPair

Recommendations

Top 10 Friend Recommendations

Mapl

```
0   1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22,23,24,25,26,27,28,29,30,31
1,42,43,44,45,46,47,48,49,50,51,52,53,54,55,56,57,58,59,60,61,62,63,64,65,66,67,68,69,7
80,81,82,83,84,85,86,87,88,89,90,91,92,93,94
1   0,5,20,135,2409,8715,8932,10623,12347,12846,13840,13845,14005,20075,21556,22939,235
691,31232,31435,32317,32489,34394,35589,35605,35606,35613,35633,35648,35678,38737,43447
629,4999,6156,13912,14248,15190,17636,19217,20074,27536,29481,29726,29767,30257,33060,3
34420,34439,34450,34651,45054,49592
```

Our input data has been taken from a dataset provided in a Stanford Data mining course

Top 10 Friend Recommendations

Mapl

```
0   1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22,23,24,25,26,27,28,29,30,31
1,42,43,44,45,46,47,48,49,50,51,52,53,54,55,56,57,58,59,60,61,62,63,64,65,66,67,68,69,7
80,81,82,83,84,85,86,87,88,89,90,91,92,93,94
1   0,5,20,135,2409,8715,8932,10623,12347,12846,13840,13845,14005,20075,21556,22939,235
691,31232,31435,32317,32489,34394,35589,35605,35606,35613,35633,35648,35678,38737,43447
629,4999,6156,13912,14248,15190,17636,19217,20074,27536,29481,29726,29767,30257,33060,3
34420,34439,34450,34651,45054,49592
```

The numbers represent userIds

Top 10 Friend Recommendations

Map1

The format of 1 record is

"userId \t (comma separated list of friends)"

```
0    1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16
1,42,43,44,45,46,47,48,49,50,51,52,53,54,5
80,81,82,83,84,85,86,87,88,89,90,91,92,93,
1    0,5,20,135,2409,8715,8932,10623,12347,
691,31232,31435,32317,32489,34394,35589,35
```

Top 10 Friend Recommendations

Map1

"userId \t (comma separated list of friends)"

```
public class Map1 extends Mapper<LongWritable, Text, TextPair, IntWritable> {

    public void map(LongWritable key, Text value, Context context) throws IOException, InterruptedException {
        String line = value.toString();
        int index = line.indexOf('\t');
        if(index == -1)
            return;
        String user_id = line.substring(0, index);
        List<String> friends_id = new ArrayList<>();
        StringTokenizer tokenizer = new StringTokenizer(line.substring(index+1), ",");
        while(tokenizer.hasMoreTokens()) {
            friends_id.add(tokenizer.nextToken());
        }

        int length = friends_id.size(), i,j;
        for(i = 0; i < length; i++) {
            context.write(new TextPair(user_id , friends_id.get(i)), new IntWritable(0));
        }
        for(i = 0; i < length; i++) {
            for(j = 0; j < length; j ++) {
                if(j == i)
                    continue;
                context.write(new TextPair(friends_id.get(i) , friends_id.get(j)), new IntWritable(1));
            }
        }
    }
}
```

Top 10 Friend Recommendations

Map1

"userId \t (comma separated list of friends)"

```
ic class Map1 extends Mapper<LongWritable, Text, TextPair, IntWritable>

public void map(LongWritable key, Text value, Context context) throws IOException, InterruptedException {
    String line = value.toString();
    int index = line.indexOf('\t');
    if(index == -1)
        return;
    String user_id = line.substring(0, index);
    List<String> friends_id = new ArrayList<>();
    StringTokenizer tokenizer = new StringTokenizer(line.substring(index+1), ",");
    while(tokenizer.hasMoreTokens()) {
        friends_id.add(tokenizer.nextToken());
    }

    int length = friends_id.size(), i,j;
    for(i = 0; i < length; i++) {
        context.write(new TextPair(user_id , friends_id.get(i)), new IntWritable(0));
    }
    for(i = 0; i < length; i++) {
        for(j = 0; j < length; j ++) {
            if(j == i)
                continue;
            context.write(new TextPair(friends_id.get(i) , friends_id.get(j)), new IntWritable(1));
        }
    }
}
```

The input key type is LongWritable (the line number)

The input value is a Text (one line of text)

Top 10 Friend Recommendations

Map1

"userId \t (comma separated list of friends)"

```
ic class Map1 extends Mapper<LongWritable, Text, TextPair, IntWritable>

public void map(LongWritable key, Text value, Context context) throws IOException, InterruptedException {
    String line = value.toString();
    int index = line.indexOf('\t');
    if(index == -1)
        return;
    String user_id = line.substring(0, index);
    List<String> friends_id = new ArrayList<>();
    StringTokenizer tokenizer = new StringTokenizer(line.substring(index+1), ",");
    while(tokenizer.hasMoreTokens()) {
        friends_id.add(tokenizer.nextToke());
    }

    int length = friends_id.size(), i,j;
    for(i = 0; i < length; i++) {
        context.write(new TextPair(user_id , friends_id.get(i)), new IntWritable(0));
    }
    for(i = 0; i < length; i++) {
        for(j = 0; j < length; j ++) {
            if(j == i)
                continue;
            context.write(new TextPair(friends_id.get(i) , friends_id.get(j)), new IntWritable(1));
        }
    }
}
```

The output key type is TextPair (a pair of users)

The output value is an Integer (1/0)

Top 10 Friend Recommendations

Map1

"userId \t (comma separated list of friends)"

```
public class Map1 extends Mapper<LongWritable, Text, TextPair, IntWritable> {

    public void map(LongWritable key, Text value, Context context) throws IOException, InterruptedException {
        String line = value.toString();
        int index = line.indexOf('\t');
        String user_id = line.substring(0, index);
        List<String> friends_id = new ArrayList<>();
        StringTokenizer tokenizer = new StringTokenizer(line.substring(index+1), ",");
        while(tokenizer.hasMoreTokens()) {
            friends_id.add(tokenizer.nextToken());
        }

        int length = friends_id.size(), i,j;
        for(i = 0; i < length; i++) {
            context.write(new TextPair(user_id , friends_id.get(i)) , new IntWritable(0));
        }
        for(i = 0; i < length; i++) {
            for(j = 0; j < length; j ++) {
                if(j == i)
                    continue;
                context.write(new TextPair(friends_id.get(i) , friends_id.get(j)) , new IntWritable(1));
            }
        }
    }
}
```

We parse the userId from the record

Top 10 Friend Recommendations

Map1

"userId \t (comma separated list of friends)"

```
List<String> friends_id = new ArrayList<>();
StringTokenizer tokenizer = new StringTokenizer(line.substring(index+1), ",");
        while(tokenizer.hasMoreTokens()) {
            friends_id.add(tokenizer.nextToken());
        }
```

Use a StringTokenizer to split up the list of friends and add them to an ArrayList

Top 10 Friend Recommendations

```
public class Map1 extends Mapper<LongWritable, Text, TextPair, IntWritable> {

    public void map(LongWritable key, Text value, Context context) throws IOException, InterruptedException {
        String line = value.toString();
        int index = line.indexOf('\t');
        if(index == -1)
            return;
        String user_id = line.substring(0, index);
        List<String> friends_id = new ArrayList<>();
        StringTokenizer tokenizer = new StringTokenizer(line.substring(index+1), ",");
        while(tokenizer.hasMoreTokens()) {
            friends_id.add(tokenizer.nextToken());
        }

        int length = friends_id.size(), i,j;
        for(i = 0; i < length; i++) {

            context.write(new TextPair(user_id , friends_id.get(i)), new IntWritable(0));
        }
```

Map1

```
for(i = 0; i < length; i++) {
  for(j = 0; j < length; j ++) {
        if(j == i)
        continue;
```

For every pair that can be generated from the list of friends, we add an output to context

```
context.write(new TextPair(friends_id.get(i), friends_id.get(j)), new IntWritable(1));
```

```
        }
    }
}
```

There is one entry for A, B and another for B, A

Top 10 Friend Recommendations

```
public class Map1 extends Mapper<LongWritable, Text, TextPair, IntWritable> {

    public void map(LongWritable key, Text value, Context context) throws IOException, InterruptedException {
        String line = value.toString();
        int index = line.indexOf('\t');
        if(index == -1)
            return;
        String user_id = line.substring(0, index);
        List<String> friends_id = new ArrayList<>();
        StringTokenizer tokenizer = new StringTokenizer(line.substring(index+1), ",");
        while(tokenizer.hasMoreTokens()) {
            friends_id.add(tokenizer.nextToken());
        }

        int length = friends_id.size(), i,j;
        for(i = 0; i < length; i++) {

            context.write(new TextPair(user_id , friends_id.get(i)), new IntWritable(0));
        }
```

Map1

```
for(i = 0; i < length; i++) {
   for(j = 0; j < length; j ++) {
            if(j == i)
            continue;

context.write(new TextPair(friends_id.get(i), friends_id.get(j)), new IntWritable(1));


                                    }
        }
    }
}
```

Skip over if the indices are the same, we don't want outputs like A, A

Top 10 Friend Recommendations

```
public class Map1 extends Mapper<LongWritable, Text, TextPair, IntWritable> {

    public void map(LongWritable key, Text value, Context context) throws IOException, InterruptedException {
        String line = value.toString();
        int index = line.indexOf('\t');
        if(index == -1)
            return;
        String user_id = line.substring(0, index);
        List<String> friends_id = new ArrayList<>();
        StringTokenizer tokenizer = new StringTokenizer(line.substring(index+1), ",");
        while(tokenizer.hasMoreTokens()) {
            friends_id.add(tokenizer.nextToken());
        }

        int length = friends_id.size(), i,j;
        for(i = 0; i < length; i++) {

            context.write(new TextPair(user_id , friends_id.get(i)), new IntWritable(0));
        }
```

Map1

```
for(i = 0; i < length; i++) {
   for(j = 0; j < length; j ++) {
          if(j == i)
          continue;

context.write(new TextPair(friends_id.get(i), friends_id.get(j)), new IntWritable(1));


                                    }

        }
    }
}
```

The value is 1 to represent this pair has a common friend

Top 10 Friend Recommendations

Map1

```
public class Map1 extends Mapper<LongWritable, Text, TextPair, IntWritable> {

    public void map(LongWritable key, Text value, Context context) throws IOException, InterruptedException {
        String line = value.toString();
        int index = line.indexOf('\t');
        if(index == -1)
            return;
        String user_id = line.substring(0, index);
        List<String> friends_id = new ArrayList<>();
        StringTokenizer tokenizer = new StringTokenizer(line.substring(index+1), ",");
        while(tokenizer.hasMoreTokens()) {
            friends_id.add(tokenizer.nextToken());
        }

        int length = friends_id.size(), i,j;
        for(i = 0; i < length; i++) {
            context.write(new TextPair(user_id , friends_id.get(i)), new IntWritable(0));
        }
        for(i = 0; i < length; i++) {
            for(j = 0; j < length; j ++) {
                if(j == i)
                    continue;
                context.write(new TextPair(friends_id.get(i) , friends_id.get(j)), new IntWritable(1));
            }
        }
    }
```

For the pair <current user, friend>, we add a pair to the output

The value is 0 to represent that this pair is already friends with each other

Top 10 Friend Recommendations

Java classes

Map1
Reduce1

Map2
Reduce2

TextPair

TextIntPair

Recommendations

Top 10 Friend Recommendations

Java classes

Map1

Reduce1

Map2

Reduce2

TextPair

TextIntPair

Recommendations

Top 10 Friend Recommendations

Reducer1

```
public  class Reduce1 extends Reducer<TextPair, IntWritable, TextPair, IntWritable> {
    public void reduce(TextPair key, Iterable<IntWritable> values, Context context) throws
IOException, InterruptedException {
        int sum = 0, prod = 1;
        for (IntWritable val : values) {
            sum += val.get();
            prod *= val.get();
        }
        if( prod!=0 )
            context.write(key, new IntWritable(sum));
    }
}
```

Top 10 Friend Recommendations

Reducel

```
public  class Reduce1 extends Reducer<TextPair, IntWritable, TextPair, IntWritable> {
    public void reduce(TextPair key, Iterable<IntWritable> values, Context context) throws
IOException, InterruptedException {
        int sum = 0, prod = 1;
            for (IntWritable val : values) {
                sum += val.get();
                prod *= val.get();
            }
        if( prod!=0 )
            context.write(key, new IntWritable(sum));
    }
}
```

For each user pair, we compute the sum and product of the values

Top 10 Friend Recommendations

Reducer1

```
public  class Reduce1 extends Reducer<TextPair, IntWritable, TextPair, IntWritable> {
    public void reduce(TextPair key, Iterable<IntWritable> values, Context context) throws
IOException, InterruptedException {
        int sum = 0, prod = 1;
        for (IntWritable val : values) {
            sum += val.get();
            prod *= val.get();
        }
         if( prod!=0 )
                context.write(key, new IntWritable(sum));
    }
}
```

If product <> 0, this pair is not already friends

Top 10 Friend Recommendations

Reducer1

```
public  class Reduce1 extends Reducer<TextPair, IntWritable, TextPair, IntWritable> {
    public void reduce(TextPair key, Iterable<IntWritable> values, Context context) throws
IOException, InterruptedException {
        int sum = 0, prod = 1;
        for (IntWritable val : values) {
            sum += val.get();
            prod *= val.get();
        }
         if( prod!=0 )
                context.write(key, new IntWritable(sum));
    }
}
```

So, we keep the pair to generate a possible recommendation

Top 10 Friend Recommendations

Reducer1

```
public  class Reduce1 extends Reducer<TextPair, IntWritable, TextPair, IntWritable> {
    public void reduce(TextPair key, Iterable<IntWritable> values, Context context) throws
IOException, InterruptedException {
        int sum = 0, prod = 1;
        for (IntWritable val : values) {
            sum += val.get();
            prod *= val.get();
        }
         if( prod!=0 )
                context.write(key, new IntWritable(sum));
    }
}
```

The output value is the # common friends this user pair has

Top 10 Friend Recommendations

Java classes

Map1

Reduce1

Map2

Reduce2

TextPair

TextIntPair

Recommendations

Top 10 Friend Recommendations

Java classes

Mapl
Reducel

The output of this MR step is written to an intermediate text file in HDFS

The format will be

"user1 \t user2 \t #commonFriends"

Top 10 Friend Recommendations

Java classes

"user1 \t user2 \t #commonFriends"

Map1

Reduce1

Map2

Reduce2

TextPair

TextIntPair

Recommendations

Top 10 Friend Recommendations

Map2

"user1 \t user2 \t #commonFriends"

```
public class Map2 extends Mapper<LongWritable, Text, Text, TextIntPair> {
    public void map(LongWritable key, Text value, Context context) throws
IOException, InterruptedException {
        String line = value.toString();
        String[] data = line.split("\t");


        Text user_id = new Text(data[0]);
        Text friend_id = new Text(data[1]);
        Integer commonFriend = Integer.parseInt(data[2]);
        context.write(user_id, new TextIntPair(friend_id, new
IntWritable(commonFriend)));
    }
}
```

Top 10 Friend Recommendations

Map2

"user1 \t user2 \t #commonFriends"

```
public class Map2 extends Mapper<LongWritable, Text, Text, TextIntPair> {

        public void map(LongWritable key, Text value, Context context) throws IOException, InterruptedException {
        String line = value.toString();
        String[] data = line.split("\t");


        Text user_id = new Text(data[0]);
        Text friend_id = new Text(data[1]);
        Integer commonFriend = Integer.parseInt(data[2]);
        context.write(user_id, new TextIntPair(friend_id, new IntWritable(commonFriend)));
    }
}
```

The output will be

<user1, <user2, #commonFriends>>

Top 10 Friend Recommendations

Map2

"user1 \t user2 \t #commonFriends"

```
public class Map2 extends Mapper<LongWritable, Text, Text, TextIntPair> {
    public void map(LongWritable key, Text value, Context context) throws IOException, InterruptedException {
        String line = value.toString();
        String[] data = line.split("\t");

        Text user_id = new Text(data[0]);
        Text friend_id = new Text(data[1]);
        Integer commonFriend = Integer.parseInt(data[2]);
        context.write(user_id, new TextIntPair(friend_id, new IntWritable(commonFriend)));
    }
}
```

Split the data using tab

Top 10 Friend Recommendations

Map2

"user1 \t user2 \t #commonFriends"

```
public class Map2 extends Mapper<LongWritable, Text, Text, TextIntPair> {
    public void map(LongWritable key, Text value, Context context) throws IOException, InterruptedException {

        String line = value.toString(); String[] data = line.split("\t");

    Text user_id = new Text(data[0]);
    Text friend_id = new Text(data[1]);
      Integer commonFriend = Integer.parseInt(data[2]);
      context.write(user_id, new TextIntPair(friend_id, new IntWritable(commonFriend)));
    }
}
```

The first 2 elements are the userIds

Top 10 Friend Recommendations

Map2

"user1 \t user2 \t #commonFriends"

```
public class Map2 extends Mapper<LongWritable, Text, Text, TextIntPair> {
    public void map(LongWritable key, Text value, Context context) throws IOException, InterruptedException {
        String line = value.toString(); String[] data = line.split("\t");


  Text user_id = new Text(data[0]);
  Text friend_id = new Text(data[1]);
        Integer commonFriend = Integer.parseInt(data[2]);
        context.write(user_id, new TextIntPair(friend_id, new IntWritable(commonFriend)));
    }
}
```

The 3rd column is the # common friends which needs to be converted to Integer

Top 10 Friend Recommendations

Map2

```
public class Map2 extends Mapper<LongWritable, Text, Text, TextIntPair> {
    public void map(LongWritable key, Text value, Context context) throws IOException, InterruptedException {
        String line = value.toString(); String[] data = line.split("\t");


  Text user_id = new Text(data[0]);
  Text friend_id = new Text(data[1]);


         Integer commonFriend = Integer.parseInt(data[2]);
     context.write(user_id, new TextIntPair(friend_id, new
IntWritable(commonFriend)));
    }
}
```

We write the output to context

Top 10 Friend Recommendations

Java classes

Map1
Reduce1

Map2
Reduce2

TextPair
TextIntPair

Recommendations

Top 10 Friend Recommendations

Java classes

Map1
Reduce1

Map2
Reduce2

TextPair
TextIntPair

Recommendations

Top 10 Friend Recommendations

Reduce2

```
public  class Reduce2 extends Reducer<Text, TextIntPair, Text, Text> {

    public void reduce(Text key, Iterable<TextIntPair> values, Context context) throws IOException, InterruptedException {


        List<TextIntPair> common  = new ArrayList<>();
        for (TextIntPair val:values){
            Text first = new Text(val.getFirst());
            IntWritable second = new IntWritable(val.getSecond().get());

            if (common.size()<10){
                common.add(new TextIntPair(first, second));
                continue;
            }

            for (int i=0; i<10; i++){
                if(common.get(i).getSecond().get()<second.get()){
                    common.set(i, new TextIntPair(first, second));
                    break;
                }
            }
        }


        StringBuilder stringBuilder = new StringBuilder();

        int j;
        for (j=0 ; j < Math.min(common.size(),11) ;j++) {

            stringBuilder.append(common.get(j).getFirst()) ;

            if (j < Math.min(common.size(),11)-1){
                stringBuilder.append("|");}

        };

        context.write(key, new Text(stringBuilder.toString()));
    }
}
```

For every user this class will sort the value <user, #common> pairs by # common and pick the top 10

Top 10 Friend Recommendations

Reduce2

```
public  class Reduce2 extends Reducer<Text, TextIntPair, Text, Text> {

    public void reduce(Text key, Iterable<TextIntPair> values, Context context) throws IOException, InterruptedException {

            List<TextIntPair> common  = new ArrayList<>();
            for (TextIntPair val:values){
        Text first = new Text(val.getFirst());
        IntWritable second = new IntWritable(val.getSecond().get());

        if (common.size()<10){
            common.add(new TextIntPair(first, second));
            continue;
        }

        for (int i=0; i<10; i++){
            if(common.get(i).getSecond().get()<second.get()){
                common.set(i, new TextIntPair(first, second));
                break;
            }
        }
    }


    StringBuilder stringBuilder = new StringBuilder();

    int j;
    for (j=0 ; j < Math.min(common.size(),11) ;j++) {

        stringBuilder.append(common.get(j).getFirst()) ;

        if (j < Math.min(common.size(),11)-1){
            stringBuilder.append("|");}

    };

    context.write(key, new Text(stringBuilder.toString()));
    }
}
```

We'll iterate through the values and keep the ones we want in a list

Top 10 Friend Recommendations

Reduce2

```
public  class Reduce2 extends Reducer<Text, TextIntPair, Text, Text> {

    public void reduce(Text key, Iterable<TextIntPair> values, Context context) throws IOException, InterruptedException {


        List<TextIntPair> common  = new ArrayList<>();
        for (TextIntPair val:values){
            Text first = new Text(val.getFirst());
            IntWritable second = new IntWritable(val.getSecond().get());

            if (common.size()<10){
                        common.add(new TextIntPair(first, second));
                        continue;
                    }

            for (int i=0; i<10; i++){
                if(common.get(i).getSecond().get()<second.get()){
                    common.set(i, new TextIntPair(first, second));
                    break;
                }
            }
        }


        StringBuilder stringBuilder = new StringBuilder();

        int j;
        for (j=0 ; j < Math.min(common.size(),11) ;j++) {

            stringBuilder.append(common.get(j).getFirst()) ;

            if (j < Math.min(common.size(),11)-1){
                stringBuilder.append("|");}

        };

        context.write(key, new Text(stringBuilder.toString()));
    }
}
```

The first 10 values we encounter will get added to the list

Top 10 Friend Recommendations

Reduce2

```
public  class Reduce2 extends Reducer<Text, TextIntPair, Text, Text> {

    public void reduce(Text key, Iterable<TextIntPair> values, Context context) throws IOException, InterruptedException {


        List<TextIntPair> common  = new ArrayList<>();
        for (TextIntPair val:values){
            Text first = new Text(val.getFirst());
            IntWritable second = new IntWritable(val.getSecond().get());

            if (common.size()<10){
                        common.add(new TextIntPair(first, second));
                        continue;
                    }

            for (int i=0; i<10; i++){
                if(common.get(i).getSecond().get()<second.get()){
                    common.set(i, new TextIntPair(first, second));
                    break;
                }
            }
        }


        StringBuilder stringBuilder = new StringBuilder();

        int j;
        for (j=0 ; j < Math.min(common.size(),11) ;j++) {

            stringBuilder.append(common.get(j).getFirst()) ;

            if (j < Math.min(common.size(),11)-1){
                stringBuilder.append("|");}

        };

        context.write(key, new Text(stringBuilder.toString()));
    }
}
```

Notice how we add a new TextIntPair

Top 10 Friend Recommendations

Reduce2

```
public  class Reduce2 extends Reducer<Text, TextIntPair, Text, Text> {

    public void reduce(Text key, Iterable<TextIntPair> values, Context context) throws IOException, InterruptedException {

        List<TextIntPair> common  = new ArrayList<>();
        for (TextIntPair val:values){
                Text first = new Text(val.getFirst());
                IntWritable second = new IntWritable(val.getSecond().get());

          if (common.size()<10){
                        common.add(new TextIntPair(first, second));
                        continue;
                    }

            for (int i=0; i<10; i++){
                if(common.get(i).getSecond().get()<second.get()){
                    common.set(i, new TextIntPair(first, second));
                    break;
                }
            }
        }

        StringBuilder stringBuilder = new StringBuilder();

        int j;
        for (j=0 ; j < Math.min(common.size(),11) ;j++) {

            stringBuilder.append(common.get(j).getFirst()) ;

            if (j < Math.min(common.size(),11)-1){
                stringBuilder.append("|");}

        };

        context.write(key, new Text(stringBuilder.toString()));
    }
}
```

This is created by unboxing the val in the current iteration

Top 10 Friend Recommendations

Reduce2

```
public  class Reduce2 extends Reducer<Text, TextIntPair, Text, Text> {

    public void reduce(Text key, Iterable<TextIntPair> values, Context context) throws IOException, InterruptedException {

        List<TextIntPair> common  = new ArrayList<>();
        for (TextIntPair val:values){
                    Text first = new Text(val.getFirst());
                    IntWritable second = new IntWritable(val.getSecond().get());

          if (common.size()<10){
                        common.add(new TextIntPair(first, second));
                        continue;
                    }
            for (int i=0; i<10; i++){
                if(common.get(i).getSecond().get()<second.get()){
                    common.set(i, new TextIntPair(first, second));
                    break;
                }
            }
        }

        StringBuilder stringBuilder = new StringBuilder();

        int j;
        for (j=0 ; j < Math.min(common.size(),11) ;j++) {

            stringBuilder.append(common.get(j).getFirst()) ;

            if (j < Math.min(common.size(),11)-1){
                stringBuilder.append("|");}

        };

        context.write(key, new Text(stringBuilder.toString()));
    }
}
```

This is a peculiarity of Hadoop

Top 10 Friend Recommendations

Reduce2

```
public  class Reduce2 extends Reducer<Text, TextIntPair, Text, Text> {

    public void reduce(Text key, Iterable<TextIntPair> values, Context context) throws IOException, InterruptedException {

        List<TextIntPair> common  = new ArrayList<>();
        for (TextIntPair val:values){
                    Text first = new Text(val.getFirst());
                    IntWritable second = new IntWritable(val.getSecond().get());


          if (common.size()<10){
                        common.add(new TextIntPair(first, second));
                        continue;
                    }

            for (int i=0; i<10; i++){
                if(common.get(i).getSecond().get()<second.get()){
                    common.set(i, new TextIntPair(first, second));
                    break;
                }
            }
        }


        StringBuilder stringBuilder = new StringBuilder();

        int j;
        for (j=0 ; j < Math.min(common.size(),11) ;j++) {

            stringBuilder.append(common.get(j).getFirst()) ;

            if (j < Math.min(common.size(),11)-1){
                stringBuilder.append("|");}

        };

        context.write(key, new Text(stringBuilder.toString()));
    }
}
```

When you iterate through the values, the same Object is reused

Top 10 Friend Recommendations

Reduce2

```
public  class Reduce2 extends Reducer<Text, TextIntPair, Text, Text> {

    public void reduce(Text key, Iterable<TextIntPair> values, Context context) throws IOException, InterruptedException {

        List<TextIntPair> common  = new ArrayList<>();
        for (TextIntPair val:values){
                    Text first = new Text(val.getFirst());
                    IntWritable second = new IntWritable(val.getSecond().get());

          if (common.size()<10){
                        common.add(new TextIntPair(first, second));
                        continue;
                    }
            for (int i=0; i<10; i++){
                if(common.get(i).getSecond().get()<second.get()){
                    common.set(i, new TextIntPair(first, second));
                    break;
                }
            }
        }

        StringBuilder stringBuilder = new StringBuilder();

        int j;
        for (j=0 ; j < Math.min(common.size(),11) ;j++) {

            stringBuilder.append(common.get(j).getFirst()) ;

            if (j < Math.min(common.size(),11)-1){
                stringBuilder.append("|");}

        };

        context.write(key, new Text(stringBuilder.toString()));
    }
}
```

So, if you directly add the value to the List, in the end all the elements will be the same

Top 10 Friend Recommendations

Reduce2

```
public  class Reduce2 extends Reducer<Text, TextIntPair, Text, Text> {

    public void reduce(Text key, Iterable<TextIntPair> values, Context context) throws IOException, InterruptedException {

        List<TextIntPair> common  = new ArrayList<>();
        for (TextIntPair val:values){
                    Text first = new Text(val.getFirst());
                    IntWritable second = new IntWritable(val.getSecond().get());

          if (common.size()<10){
                        common.add(new TextIntPair(first, second));
                        continue;
                    }
            for (int i=0; i<10; i++){
                if(common.get(i).getSecond().get()<second.get()){
                    common.set(i, new TextIntPair(first, second));
                    break;
                }
            }
        }

        StringBuilder stringBuilder = new StringBuilder();

        int j;
        for (j=0 ; j < Math.min(common.size(),11) ;j++) {

            stringBuilder.append(common.get(j).getFirst()) ;

            if (j < Math.min(common.size(),11)-1){
                stringBuilder.append("|");}

        };

        context.write(key, new Text(stringBuilder.toString()));
    }
}
```

Instead, we create a new TextIntPair and add that to our List

Top 10 Friend Recommendations

Reduce2

```
public  class Reduce2 extends Reducer<Text, TextIntPair, Text, Text> {

    public void reduce(Text key, Iterable<TextIntPair> values, Context context) throws IOException, InterruptedException {


        List<TextIntPair> common  = new ArrayList<>();
        for (TextIntPair val:values){
            Text first = new Text(val.getFirst());
            IntWritable second = new IntWritable(val.getSecond().get());

            if (common.size()<10){
                common.add(new TextIntPair(first, second));
                continue;
            }

              for (int i=0; i<10; i++){
                            if(common.get(i).getSecond().get()<second.get()){
                                common.set(i, new TextIntPair(first, second));
                                break;
                            }
                        }
        }


        StringBuilder stringBuilder = new StringBuilder();

        int j;
        for (j=0 ; j < Math.min(common.size(),11) ;j++) {

            stringBuilder.append(common.get(j).getFirst()) ;

            if (j < Math.min(common.size(),11)-1){
                stringBuilder.append("|");}

        };

        context.write(key, new Text(stringBuilder.toString()));
    }
}
```

If # values > 10, we check each new value against the 10 we kept

Top 10 Friend Recommendations

Reduce2

```
public  class Reduce2 extends Reducer<Text, TextIntPair, Text, Text> {

    public void reduce(Text key, Iterable<TextIntPair> values, Context context) throws IOException, InterruptedException {


        List<TextIntPair> common  = new ArrayList<>();
        for (TextIntPair val:values){
            Text first = new Text(val.getFirst());
            IntWritable second = new IntWritable(val.getSecond().get());

            if (common.size()<10){
                common.add(new TextIntPair(first, second));
                continue;
            }

              for (int i=0; i<10; i++){
                            if(common.get(i).getSecond().get()<second.get()){
                                common.set(i, new TextIntPair(first, second));
                                break;
                            }
                        }
                    }

        }


        StringBuilder stringBuilder = new StringBuilder();

        int j;
        for (j=0 ; j < Math.min(common.size(),11) ;j++) {

            stringBuilder.append(common.get(j).getFirst()) ;

            if (j < Math.min(common.size(),11)-1){
                stringBuilder.append("|");}

        };

        context.write(key, new Text(stringBuilder.toString()));
    }
}
```

If the new value is greater than an old one, we store the new one instead

Top 10 Friend Recommendations

Reduce2

```
public  class Reduce2 extends Reducer<Text, TextIntPair, Text, Text> {

    public void reduce(Text key, Iterable<TextIntPair> values, Context context) throws IOException, InterruptedException {


        List<TextIntPair> common  = new ArrayList<>();
        for (TextIntPair val:values){
            Text first = new Text(val.getFirst());
            IntWritable second = new IntWritable(val.getSecond().get());

            if (common.size()<10){
                common.add(new TextIntPair(first, second));
                continue;
            }

            for (int i=0; i<10; i++){
                if(common.get(i).getSecond().get()<second.get()){
                    common.set(i, new TextIntPair(first, second));
                    break;
                }
            }
        }


        StringBuilder stringBuilder = new StringBuilder();

        int j;
        for (j=0 ; j < Math.min(common.size(),11) ;j++) {

            stringBuilder.append(common.get(j).getFirst()) ;

            if (j < Math.min(common.size(),11)-1){
                stringBuilder.append("|");}

        };

        context.write(key, new Text(stringBuilder.toString()));
    }
}
```

At the end of this we've iterated through all the values and kept only the top 10

Top 10 Friend Recommendations

Reduce2

```
public  class Reduce2 extends Reducer<Text, TextIntPair, Text, Text> {

    public void reduce(Text key, Iterable<TextIntPair> values, Context context) throws IOException, InterruptedException {


        List<TextIntPair> common  = new ArrayList<>();
        for (TextIntPair val:values){
            Text first = new Text(val.getFirst());
            IntWritable second = new IntWritable(val.getSecond().get());

            if (common.size()<10){
                common.add(new TextIntPair(first, second));
                continue;
            }

            for (int i=0; i<10; i++){
                if(common.get(i).getSecond().get()<second.get()){
                    common.set(i, new TextIntPair(first, second));
                    break;
                }
            }
        }


        StringBuilder stringBuilder = new StringBuilder();

        int j;
        for (j=0 ; j < Math.min(common.size(),11) ;j++) {

            stringBuilder.append(common.get(j).getFirst()) ;

            if (j < Math.min(common.size(),11)-1){
                stringBuilder.append("|");}

        };

        context.write(key, new Text(stringBuilder.toString()));
    }
}
```

The top 10 are not necessarily in sorted order

Top 10 Friend Recommendations

Reduce2

```
StringBuilder stringBuilder = new StringBuilder();

        int j;
        for (j=0 ; j < Math.min(common.size(),11) ;j++) {

            stringBuilder.append(common.get(j).getFirst()) ;

            if (j < Math.min(common.size(),11)-1){
                stringBuilder.append("|");}

        };
```

We use StringBuilder to create a "|" separated list of the top 10 userIds

Top 10 Friend Recommendations

Reduce2

```
        Text first = new Text(val.getFirst());
        IntWritable second = new IntWritable(val.getSecond().get());

        if (common.size()<10){
            common.add(new TextIntPair(first, second));
            continue;
        }

        for (int i=0; i<10; i++){
            if (common.get(i).getSecond().get()<second.get()){
                common.add(i, new TextIntPair(first, second));
                break;
            }
        }
    }


    StringBuilder stringBuilder = new StringBuilder();

    int j;
    for (j=0 ; j < Math.min(common.size(),11) ;j++) {

        stringBuilder.append(common.get(j).getFirst()) ;

        if (j < Math.min(common.size(),11)-1){
            stringBuilder.append("|");}

    };


context.write(key, new Text(stringBuilder.toString()));
    }
}
```

We write out the UserId and the list of recommendations

Top 10 Friend Recommendations

Java classes

Map1
Reduce1

Map2
Reduce2

TextPair
TextIntPair

Recommendations

Top 10 Friend Recommendations

Java classes

Map1
Reduce1

Map2
Reduce2

TextPair

TextIntPair

Recommendations

Top 10 Friend Recommendations

Recommendations

```
public class Recommendations{
    private static final String OUTPUT_PATH = "/test/intermediate_output";
    public static void main(String[] args) throws Exception {
        // --------- first Map1-Reduce1 Job ------------------------
        // For each user, calculates the number of common friends
        // with other users
        Configuration conf = new Configuration();

        FileSystem fs = FileSystem.get(conf);
        if(fs.exists(new Path(OUTPUT_PATH))){
            fs.delete(new Path(OUTPUT_PATH),true);
        }

        Job job = new Job(conf, "FriendRecom-MR1");
        job.setJarByClass(Recommendations.class);
        job.setOutputKeyClass(TextPair.class);
        job.setOutputValueClass(IntWritable.class);

        job.setMapOutputKeyClass(TextPair.class);
        job.setMapOutputValueClass(IntWritable.class);

        job.setMapperClass(Map1.class);
        job.setReducerClass(Reduce1.class);

        FileInputFormat.addInputPath(job, new Path(args[0]));
        FileOutputFormat.setOutputPath(job, new Path(OUTPUT_PATH));

        job.waitForCompletion(true);
        // -------- Second Map1 Reduce1 Job -------
        // For each user, find top recommendations
        Configuration conf2 = new Configuration();

        Job job2 = new Job(conf2, "FriendRecom-MR2");
        job2.setJarByClass(Recommendations.class);
        job2.setOutputKeyClass(Text.class);
        job2.setOutputValueClass(Text.class);

        job2.setMapOutputKeyClass(Text.class);
        job2.setMapOutputValueClass(TextIntPair.class);

        job2.setMapperClass(Map2.class);
        job2.setReducerClass(Reduce2.class);

        FileInputFormat.addInputPath(job2, new Path(OUTPUT_PATH));
        FileOutputFormat.setOutputPath(job2, new Path(args[1]));

        job2.waitForCompletion(true);
    }

}
```

This is our Main Class where we will set up our 2 MR jobs

The first one needs to feed into the other

Top 10 Friend Recommendations

Recommendations

```
public class Recommendations{
    private static final String OUTPUT_PATH = "/test/intermediate_output";
    public static void main(String[] args) throws Exception {
        // --------- first Map1-Reduce1 Job ------------------------
        // For each user, calculates the number of common friends
        // with other users
        Configuration conf = new Configuration();

        FileSystem fs = FileSystem.get(conf);
        if(fs.exists(new Path(OUTPUT_PATH))){
            fs.delete(new Path(OUTPUT_PATH),true);
        }

        Job job = new Job(conf, "FriendRecom-MR1");
        job.setJarByClass(Recommendations.class);
        job.setOutputKeyClass(TextPair.class);
        job.setOutputValueClass(IntWritable.class);

        job.setMapOutputKeyClass(TextPair.class);
        job.setMapOutputValueClass(IntWritable.class);

        job.setMapperClass(Map1.class);
        job.setReducerClass(Reduce1.class);

        FileInputFormat.addInputPath(job, new Path(args[0]));
        FileOutputFormat.setOutputPath(job, new Path(OUTPUT_PATH));

        job.waitForCompletion(true);
        // -------- Second Map1 Reduce1 Job -------
        // For each user, find top recommendations
        Configuration conf2 = new Configuration();

        Job job2 = new Job(conf2, "FriendRecom-MR2");
        job2.setJarByClass(Recommendations.class);
        job2.setOutputKeyClass(Text.class);
        job2.setOutputValueClass(Text.class);

        job2.setMapOutputKeyClass(Text.class);
        job2.setMapOutputValueClass(TextIntPair.class);

        job2.setMapperClass(Map2.class);
        job2.setReducerClass(Reduce2.class);

        FileInputFormat.addInputPath(job2, new Path(OUTPUT_PATH));
        FileOutputFormat.setOutputPath(job2, new Path(args[1]));

        job2.waitForCompletion(true);
    }

}
```

A path to store the intermediate output from 1st job

Top 10 Friend Recommendations

Recommendations

Within the main() method, we'll set up 2 Job objects

```
public class Recommendations{
    private static final String OUTPUT_PATH = "/test/intermediate_output";
    public static void main(String[] args) throws Exception {
        // --------- first Map1-Reduce1 Job ------------------------
        // For each user, calculates the number of common friends
        // with other users
        Configuration conf = new Configuration();

        FileSystem fs = FileSystem.get(conf);
        if(fs.exists(new Path(OUTPUT_PATH))){
            fs.delete(new Path(OUTPUT_PATH),true);
        }

        Job job = new Job(conf, "FriendRecom-MR1");

        ob.setJarByClass(Recommendations.class);
        job.setOutputKeyClass(TextPair.class);
        job.setOutputValueClass(IntWritable.class);

        job.setMapOutputKeyClass(TextPair.class);
        job.setMapOutputValueClass(IntWritable.class);

        job.setMapperClass(Map1.class);
        job.setReducerClass(Reduce1.class);

        FileInputFormat.addInputPath(job, new Path(args[0]));
        FileOutputFormat.setOutputPath(job, new Path(OUTPUT_PATH));

        job.waitForCompletion(true);
        // -------- Second Map1 Reduce1 Job -------
        // For each user, find top recommendations
        Configuration conf2 = new Configuration();

        Job job2 = new Job(conf2, "FriendRecom-MR2");
        job2.setJarByClass(Recommendations.class);
        job2.setOutputKeyClass(Text.class);
        job2.setOutputValueClass(Text.class);

        job2.setMapOutputKeyClass(Text.class);
        job2.setMapOutputValueClass(TextIntPair.class);

        job2.setMapperClass(Map2.class);
        job2.setReducerClass(Reduce2.class);

        FileInputFormat.addInputPath(job2, new Path(OUTPUT_PATH));
        FileOutputFormat.setOutputPath(job2, new Path(args[1]));

        job2.waitForCompletion(true);
    }

}
```

Top 10 Friend Recommendations

Recommendations

```
public class Recommendations{
    private static final String OUTPUT_PATH = "/test/intermediate_output";
    public static void main(String[] args) throws Exception {
        // --------- first Map1-Reduce1 Job ------------------------
        // For each user, calculates the number of common friends
        // with other users
        Configuration conf = new Configuration();

        FileSystem fs = FileSystem.get(conf);
        if(fs.exists(new Path(OUTPUT_PATH))){
            fs.delete(new Path(OUTPUT_PATH),true);
        }

        Job job = new Job(conf, "FriendRecom-MR1");
        job.setJarByClass(Recommendations.class);
        job.setOutputKeyClass(TextPair.class);
        job.setOutputValueClass(IntWritable.class);

        job.setMapOutputKeyClass(TextPair.class);
        job.setMapOutputValueClass(IntWritable.class);

        job.setMapperClass(Map1.class);
        job.setReducerClass(Reduce1.class);

        FileInputFormat.addInputPath(job, new Path(args[0]));
        FileOutputFormat.setOutputPath(job, new Path(OUTPUT_PATH));

        job.waitForCompletion(true);
        // -------- Second Map1 Reduce1 Job -------
        // For each user, find top recommendations
        Configuration conf2 = new Configuration();

        Job job2 = new Job(conf2, "FriendRecom-MR2");
        job2.setJarByClass(Recommendations.class);
        job2.setOutputKeyClass(Text.class);
        job2.setOutputValueClass(Text.class);

        job2.setMapOutputKeyClass(Text.class);
        job2.setMapOutputValueClass(TextIntPair.class);

        job2.setMapperClass(Map2.class);
        job2.setReducerClass(Reduce2.class);

        FileInputFormat.addInputPath(job2, new Path(OUTPUT_PATH));
        FileOutputFormat.setOutputPath(job2, new Path(args[1]));

        job2.waitForCompletion(true);
    }

}
```

The output path of Job = input path of Job2

Top 10 Friend Recommendations

Recommendations

```
public class Recommendations{
    private static final String OUTPUT_PATH = "/test/intermediate_output";
    public static void main(String[] args) throws Exception {
        // --------- first Map1-Reduce1 Job ------------------------
        // For each user, calculates the number of common friends
        // with other users
        Configuration conf = new Configuration();

        FileSystem fs = FileSystem.get(conf);
        if(fs.exists(new Path(OUTPUT_PATH))){
            fs.delete(new Path(OUTPUT_PATH),true);
        }

        Job job = new Job(conf, "FriendRecom-MR1");
        job.setJarByClass(Recommendations.class);
        job.setOutputKeyClass(TextPair.class);
        job.setOutputValueClass(IntWritable.class);

                job.setMapOutputKeyClass(TextPair.class);
                job.setMapOutputValueClass(IntWritable.class);

        job.setMapperClass(Map1.class);
        job.setReducerClass(Reduce1.class);

        FileInputFormat.addInputPath(job, new Path(args[0]));
        FileOutputFormat.setOutputPath(job, new Path(OUTPUT_PATH));

        job.waitForCompletion(true);
        // -------- Second Map1 Reduce1 Job -------
        // For each user, find top recommendations
        Configuration conf2 = new Configuration();

        Job job2 = new Job(conf2, "FriendRecom-MR2");
        job2.setJarByClass(Recommendations.class);
        job2.setOutputKeyClass(Text.class);
        job2.setOutputValueClass(Text.class);

        job2.setMapOutputKeyClass(Text.class);
        job2.setMapOutputValueClass(TextIntPair.class);

        job2.setMapperClass(Map2.class);
        job2.setReducerClass(Reduce2.class);

        FileInputFormat.addInputPath(job2, new Path(OUTPUT_PATH));
        FileOutputFormat.setOutputPath(job2, new Path(args[1]));

        job2.waitForCompletion(true);
    }
}
```

Map1 output key and value types

Top 10 Friend Recommendations

Recommendations

```
public class Recommendations{
    private static final String OUTPUT_PATH = "/test/intermediate_output";
    public static void main(String[] args) throws Exception {
        // --------- first Map1-Reduce1 Job ------------------------
        // For each user, calculates the number of common friends
        // with other users
        Configuration conf = new Configuration();

        FileSystem fs = FileSystem.get(conf);
        if(fs.exists(new Path(OUTPUT_PATH))){
            fs.delete(new Path(OUTPUT_PATH),true);
        }

        Job job = new Job(conf, "FriendRecom-MR1");
        job.setJarByClass(Recommendations.class);
        job.setOutputKeyClass(TextPair.class);
        job.setOutputValueClass(IntWritable.class);

        job.setMapOutputKeyClass(TextPair.class);
        job.setMapOutputValueClass(IntWritable.class);

        job.setMapperClass(Map1.class);
        job.setReducerClass(Reduce1.class);

        FileInputFormat.addInputPath(job, new Path(args[0]));
        FileOutputFormat.setOutputPath(job, new Path(OUTPUT_PATH));

        job.waitForCompletion(true);
        // -------- Second Map1 Reduce1 Job -------
        // For each user, find top recommendations
        Configuration conf2 = new Configuration();

        Job job2 = new Job(conf2, "FriendRecom-MR2");
        job2.setJarByClass(Recommendations.class);
        job2.setOutputKeyClass(Text.class);
        job2.setOutputValueClass(Text.class);

        job2.setMapperClass(Map2.class);
        job2.setReducerClass(Reduce2.class);

        FileInputFormat.addInputPath(job2, new Path(OUTPUT_PATH));
        FileOutputFormat.setOutputPath(job2, new Path(args[1]));

        job2.waitForCompletion(true);
    }
}
```

Map2 output key and value types

```
job2.setMapOutputKeyClass(Text.class);
job2.setMapOutputValueClass(TextIntPair.class);
```

Top 10 Friend Recommendations

Recommendations

```
public class Recommendations{
    private static final String OUTPUT_PATH = "/test/intermediate_output";
    public static void main(String[] args) throws Exception {
        // --------- first Map1-Reduce1 Job -------------------------
        // For each user, calculates the number of common friends
        // with other users
        Configuration conf = new Configuration();

        FileSystem fs = FileSystem.get(conf);
        if(fs.exists(new Path(OUTPUT_PATH))){
            fs.delete(new Path(OUTPUT_PATH),true);
        }

        Job job = new Job(conf, "FriendRecom-MR1");
        job.setJarByClass(Recommendations.class);
        job.setOutputKeyClass(TextPair.class);
        job.setOutputValueClass(IntWritable.class);

        job.setMapOutputKeyClass(TextPair.class);
        job.setMapOutputValueClass(IntWritable.class);

        job.setMapperClass(Map1.class);
        job.setReducerClass(Reduce1.class);

        FileInputFormat.addInputPath(job, new Path(args[0]));
        FileOutputFormat.setOutputPath(job, new Path(OUTPUT_PATH));

        job.waitForCompletion(true);
        // -------- Second Map1 Reduce1 Job -------
        // For each user, find top recommendations
        Configuration conf2 = new Configuration();

        Job job2 = new Job(conf2, "FriendRecom-MR2");
        job2.setJarByClass(Recommendations.class);
        job2.setOutputKeyClass(Text.class);
        job2.setOutputValueClass(Text.class);

        job2.setMapOutputKeyClass(Text.class);
        job2.setMapOutputValueClass(TextIntPair.class);

        job2.setMapperClass(Map2.class);
        job2.setReducerClass(Reduce2.class);

        FileInputFormat.addInputPath(job2, new Path(OUTPUT_PATH));
        FileOutputFormat.setOutputPath(job2, new Path(args[1]));

        job2.waitForCompletion(true);
    }
}
```

Job will run first and then Job2

Top 10 Friend Recommendations

Recommendations

```
public class Recommendations{
    private static final String OUTPUT_PATH = "/test/intermediate_output";
    public static void main(String[] args) throws Exception {
        // --------- first Map1-Reduce1 Job ------------------------
        // For each user, calculates the number of common friends
        // with other users
        Configuration conf = new Configuration();

        FileSystem fs = FileSystem.get(conf);
        if(fs.exists(new Path(OUTPUT_PATH))){
            fs.delete(new Path(OUTPUT_PATH),true);
        }

        Job job = new Job(conf, "FriendRecom-MR1");
        job.setJarByClass(Recommendations.class);
        job.setOutputKeyClass(TextPair.class);
        job.setOutputValueClass(IntWritable.class);

        job.setMapOutputKeyClass(TextPair.class);
        job.setMapOutputValueClass(IntWritable.class);

        job.setMapperClass(Map1.class);
        job.setReducerClass(Reduce1.class);

        FileInputFormat.addInputPath(job, new Path(args[0]));
        FileOutputFormat.setOutputPath(job, new Path(OUTPUT_PATH));

        job.waitForCompletion(true);
        // -------- Second Map1 Reduce1 Job -------
        // For each user, find top recommendations
        Configuration conf2 = new Configuration();

        Job job2 = new Job(conf2, "FriendRecom-MR2");
        job2.setJarByClass(Recommendations.class);
        job2.setOutputKeyClass(Text.class);
        job2.setOutputValueClass(Text.class);

        job2.setMapOutputKeyClass(Text.class);
        job2.setMapOutputValueClass(TextIntPair.class);

        job2.setMapperClass(Map2.class);
        job2.setReducerClass(Reduce2.class);

        FileInputFormat.addInputPath(job2, new Path(OUTPUT_PATH));
        FileOutputFormat.setOutputPath(job2, new Path(args[1]));

        job2.waitForCompletion(true);
    }

}
```

One last thing, Hadoop always expects that the output path directory does not exist

Top 10 Friend Recommendations

Recommendations

```
public class Recommendations{
    private static final String OUTPUT_PATH = "/test/intermediate_output";
    public static void main(String[] args) throws Exception {
        // --------- first Map1-Reduce1 Job ------------------------
        // For each user, calculates the number of common friends
        // with other users
        Configuration conf = new Configuration();

        FileSystem fs = FileSystem.get(conf);
        if(fs.exists(new Path(OUTPUT_PATH))){
            fs.delete(new Path(OUTPUT_PATH),true);
        }

        Job job = new Job(conf, "FriendRecom-MR1");
        job.setJarByClass(Recommendations.class);
        job.setOutputKeyClass(TextPair.class);
        job.setOutputValueClass(IntWritable.class);

        job.setMapOutputKeyClass(TextPair.class);
        job.setMapOutputValueClass(IntWritable.class);

        job.setMapperClass(Map1.class);
        job.setReducerClass(Reduce1.class);

        FileInputFormat.addInputPath(job, new Path(args[0]));
        FileOutputFormat.setOutputPath(job, new Path(OUTPUT_PATH));

        job.waitForCompletion(true);
        // -------- Second Map1 Reduce1 Job -------
        // For each user, find top recommendations
        Configuration conf2 = new Configuration();

        Job job2 = new Job(conf2, "FriendRecom-MR2");
        job2.setJarByClass(Recommendations.class);
        job2.setOutputKeyClass(Text.class);
        job2.setOutputValueClass(Text.class);

        job2.setMapOutputKeyClass(Text.class);
        job2.setMapOutputValueClass(TextIntPair.class);

        job2.setMapperClass(Map2.class);
        job2.setReducerClass(Reduce2.class);

        FileInputFormat.addInputPath(job2, new Path(OUTPUT_PATH));
        FileOutputFormat.setOutputPath(job2, new Path(args[1]));

        job2.waitForCompletion(true);
    }

}
```

So, before we start we need to delete any existing directory at the intermediate output path

Top 10 Friend Recommendations

Recommendations

```
public class Recommendations{
    private static final String OUTPUT_PATH = "/test/intermediate_output";
    public static void main(String[] args) throws Exception {
        // --------- first Map1-Reduce1 Job ---------------------------
        // For each user, calculates the number of common friends
        // with other users
        Configuration conf = new Configuration();

        FileSystem fs = FileSystem.get(conf);
                if(fs.exists(new Path(OUTPUT_PATH))){
                    fs.delete(new Path(OUTPUT_PATH),true);
                }



        Job job = new Job(conf, "FriendRecom-MR1");
        job.setJarByClass(Recommendations.class);
        job.setOutputKeyClass(TextPair.class);
        job.setOutputValueClass(IntWritable.class);

        job.setMapOutputKeyClass(TextPair.class);
        job.setMapOutputValueClass(IntWritable.class);

        job.setMapperClass(Map1.class);
        job.setReducerClass(Reduce1.class);

        FileInputFormat.addInputPath(job, new Path(args[0]));
        FileOutputFormat.setOutputPath(job, new Path(OUTPUT_PATH));

        job.waitForCompletion(true);
        // -------- Second Map1 Reduce1 Job -------
        // For each user, find top recommendations
        Configuration conf2 = new Configuration();

        Job job2 = new Job(conf2, "FriendRecom-MR2");
        job2.setJarByClass(Recommendations.class);
        job2.setOutputKeyClass(Text.class);
        job2.setOutputValueClass(Text.class);

        job2.setMapOutputKeyClass(Text.class);
        job2.setMapOutputValueClass(TextIntPair.class);

        job2.setMapperClass(Map2.class);
        job2.setReducerClass(Reduce2.class);

        FileInputFormat.addInputPath(job2, new Path(OUTPUT_PATH));
        FileOutputFormat.setOutputPath(job2, new Path(args[1]));

        job2.waitForCompletion(true);
    }
}
```

This FileSystem object will access HDFS

Top 10 Friend Recommendations

Recommendations

```
public class Recommendations{
    private static final String OUTPUT_PATH = "/test/intermediate_output";
    public static void main(String[] args) throws Exception {
        // --------- first Map1-Reduce1 Job ------------------------
        // For each user, calculates the number of common friends
        // with other users
        Configuration conf = new Configuration();

        FileSystem fs = FileSystem.get(conf);
                if(fs.exists(new Path(OUTPUT_PATH))){
                    fs.delete(new Path(OUTPUT_PATH),true);
                }


        Job job = new Job(conf, "FriendRecom-MR1");
        job.setJarByClass(Recommendations.class);
        job.setOutputKeyClass(TextPair.class);
        job.setOutputValueClass(IntWritable.class);

        job.setMapOutputKeyClass(TextPair.class);
        job.setMapOutputValueClass(IntWritable.class);

        job.setMapperClass(Map1.class);
        job.setReducerClass(Reduce1.class);

        FileInputFormat.addInputPath(job, new Path(args[0]));
        FileOutputFormat.setOutputPath(job, new Path(OUTPUT_PATH));

        job.waitForCompletion(true);
        // -------- Second Map1 Reduce1 Job -------
        // For each user, find top recommendations
        Configuration conf2 = new Configuration();

        Job job2 = new Job(conf2, "FriendRecom-MR2");
        job2.setJarByClass(Recommendations.class);
        job2.setOutputKeyClass(Text.class);
        job2.setOutputValueClass(Text.class);

        job2.setMapOutputKeyClass(Text.class);
        job2.setMapOutputValueClass(TextIntPair.class);

        job2.setMapperClass(Map2.class);
        job2.setReducerClass(Reduce2.class);

        FileInputFormat.addInputPath(job2, new Path(OUTPUT_PATH));
        FileOutputFormat.setOutputPath(job2, new Path(args[1]));

        job2.waitForCompletion(true);
    }
}
```

If the directory already exists, it will delete it

Top 10 Friend Recommendations

Java classes

Map1
Reduce1

Map2
Reduce2

TextPair
TextIntPair

Recommendations

Top 10 Friend Recommendations

9978	12361|12650|10275|14288|10498|11383|19469|22097|22946|23065
9979	21308|25866|8457|9970|10738|10824|10909|11299|11612|11635
998	1000|1001|1002|1003|1004|1005|1006|1007|1008|1009
9980	11612|14246|17435|351|4981|5069|52|7611|7651|8760
9981	30409|44358|127|13854|13913|14004|14105|14173|14182|14872
9982	31270|17999|30818|40610|40896|12060|12067|12179|12187|12195
9983	17199|4509|1973|17193|23993|37830|7327|4522|10622|18035
9984	4498|4509|11757|1421|10620|17198|17270|17272|1772|1382
9985	14411|16185|20050|22147|33461|40358|42471|11191|45825|522
9986	522|579|30860|4839|4841|14390|14411|15761|20500|21273
9987	34485|13134|37941|9992|34642|13478|13877|34299
9988	35667
9989	34299|34485|34642|37941|9992|13134|13478|13877
999	950|1000|1001|1002|1003|1004|1005|1006|1007|1008
9990	13134|13478|13877|34299|34485|34642|37941
9991	13478|13134|13877|34299|34485|34642|37941|9992|9993|9994
9992	9987|35667|9989|9991
9993	13134|13478|13877|34299|34485|34642|37941|9991
9994	13134|13478|13877|34299|34485|34642|37941|9991
9995	37356|37188|37135|10097|36679|40748|36905|37812|37597|10103
9996	36679|36854|10096|10000|44050|36669|37156|10008|36703|36765
9997	10018|10066|10086|19365|10077|36672|9998|9999|10022|10023
9998	25256|27564|27670|27586|27590|27617|27655|35613|49546|7785
9999	36764|44132|36909|44068|44076|44057|44075|36933|37145|37634