MAP-REDUCE EXAMPLES

Contents

1. Introduction to map-reduce

Operations which have more than one operation are executed in the reducers.

Map-reduce is a framework used for parallelizing tasks on a cluster of devices, managed by a central machines coordinating between other machines.

When you get a task, always try to implement the logic you want to use on a real example, in order to see if the logic you wanted to implement should do properly the task, like there are examples in examples 7 and 8.

In general, reducer as input gets a key and a list of variables, and then reduced the list of values to a single output row, or a number of rows which is less than the size of the input list (the "value" part of the input).

2. Map- reduce problems

2.1 EXAMPLE 1 (FDE exercise sheet 8, problem 1):

For the multi-sets X : [(a, b)], Y : [(c, d)] find all combinations where b = c.

```
mapX((a , b ))
        emit (b , ( 'A', a , b))

mapY((c , d ))
        emit (c , ( 'B ' , c , d))

reduce (k , list )
        listA = list.filter(_1 == 'A ')          // "_1" gets the first element of a tuple
        listB = list.filter(_1 == 'B ')
        for(x in cross(listA , listB) )
                emit("result", x)

main(X ,Y )
        mX = mapAll(mapX, X)
        mY = mapAll(mapY, Y)
        return reduceAll(reduce, mX + mY)["result"]
        // Note: We have to get the singleton result using ' ...[" result "] '.
```

The functions „mapAll" and „reduceAll" can be observed as for loops which send pair one by one from the list of pairs „X" and „Y" to mapper and reducer functions.

2.2 EXAMPLE 2 (FDE exercise sheet 8, problem 2):

For the documents D: [(name, [w])] (where D is a list of documents, in which each document is a list of words w), find the words which all documents have in common.

```
// Input variable „words“ is a list
mapCount(doc, words)
        emit(„count“ , 1)

reducerCount(k , list)
        emit(k , sum(list))

// Input variable „words“ is a list
mapWords(doc, words)
        foreach(word in words)
                emit(word, doc)

// Input variable „docs“ is a list
reduceWords(word , docs)
        emit(word, unique(docs).count() )

main(D)
        a = mapAll(mapCount, D)
        totalNumOfDocs = reduceAll (reducerCount, a)[„count“]

        b = mapAll(mapWords, D)
        wordsWithDocCount = reduceAll(reduceWords, b)

        return wordsWithDocCount.filter( totalNumOfDocs == wordsWithDocCount(_2) )
```

2.3 EXAMPLE 3 (FDE exercise sheet 8, problem 3):

Compute A*B for the two matrices A and B.

Dimensions A : n x m ; B : m x p

Matrices represented as (index row; index column; value).

```
mapA( (in_row, in_col, value) )
        for(out_col in 1 to p)
                emit( (in_row, out_col , in_col ), value)

mapB( (in_row, in_col, value) )
        for(out_row in 1 to n)
                emit ( (out_row, in_col, in_row), value )

reduceMult( (out_row, out_col, idx), values )
        emit( (out_row, out_col), prod(values) )

reduceSum( (out_col, out_row), values )
        emit( (out_row, out_col), sum(values) )

main(A,B)
        a = mapAll(mapA, A)
        b = mapAll(mapB, B)
        c = reduceAll(reduceMult, a&b)
        reduceAll(reduceSum,  c)
```

2.4 EXAMPLE 4:

Calculate the mean, $\bar{X} = 1/n * \sum x_i$ of some input list of numerical values X.

```
mapCount(const, listOfValuesInX)
        emit(„const“, 1)

//In general, reducer as input gets a key and a list of variables
reduceCount(const, listOfOnes)
        emit(„const“, sum(listOfOnes))

mapSingleValues(const, listOfValuesInX)
        emit(“value”, oneElementValue)

reduceSumValues(const, listOfValues)
        emit(„result“, sum(listOfValues))

main(X)
        a = mapAll(mapCount,  (“const”, X) )
        numberOfValues = reduceAll(reduceCount, a)[“const”]

        b = mapAll(mapSingleValues, (“const”, X) )
        sumOfAllValues = reduceAll(reduceSumValues, b)[“result”]

        if(numberOfValues != 0)
                return (1/numberOfValues)* sumOfAllValues
        else
                return 0
```

Both reducers will get all of the values, as keys which are sent to it are the same for all values.

2.5 EXAMPLE 5:

Calculate the sample variance $\boxed{var(X)=1/(n-1)\sum(x_i- \bar{X})^2}$ of some input list of numerical values X

```
mapCount(„X“, listOfAllValuesInX)
        for(oneElementOfList from listOfAllValuesInX)
                emit(„one“, 1)

reduceCountSum(“const”, listOfOnes)
        emit(„numberOfElements“, sum(listOfOnes))

mapSingleValues(“const”, listOfAllValuesInX)
        for(oneElementValue from listOfAllValuesInX)
                emir(“value”, oneElementValue)

reduceMean(numberOfElements, listOfvaluesInSeparateRows)
        sumOfAllValues  = sum(listOfvaluesInSeparateRows)
        emit(„mean“,sumOfAllValues/numberOfElements)

mapSingleValuesSumOfSquares(mean, listOfvaluesInSeparateRows)
        for(oneElementValue from listOfvaluesInSeparateRows)
                difference = mean-oneElementValue
                emit(“oneElementOfSumSquared”, sum(difference*difference))

reduceSumOfSquaresVariance(numberOfElements, listOfSumElements)
        sumSquared = sum(listOfSumElements)
        emit(„result“, (1/(1- numberOfElements))*sumSquared)

main(X)
        onesForEachElement = mapAll(mapCount, („X“, X))
        // or we can look at variable „X“ as a tuple of name of the list, and it's list of values
        numberOfElements = reduceAll(recudeCountSum, onesForEachElement)
        valuesInSeparateRows = mapAll(mapSingleValues, X)
        mean = reduceAll( reduceMean, (numberOfElements,
                                                valuesInSeparateRows[“value”])
        differencesWithMeanInOneRow = mapAll(mean, valuesInSeparateRows)
        variance = reduceAll(reduceSumOfSquaresVariance, (numberOfElements ,
                        differencesWithMeanInOneRow[“oneElementOfSumSquared”] )
        return variance[„result“]
```

2.6 EXAMPLE 6:

In this example, we will take a set of translation dictionaries, English-Spanish, English-Italian, English-French, and create a dictionary file that has the English word followed by all the different translations separated by the pipe (|) character. For example, looking at Figure 6, if the input files are as shown in the blue boxes, we expect the final output as shown in the green box below.

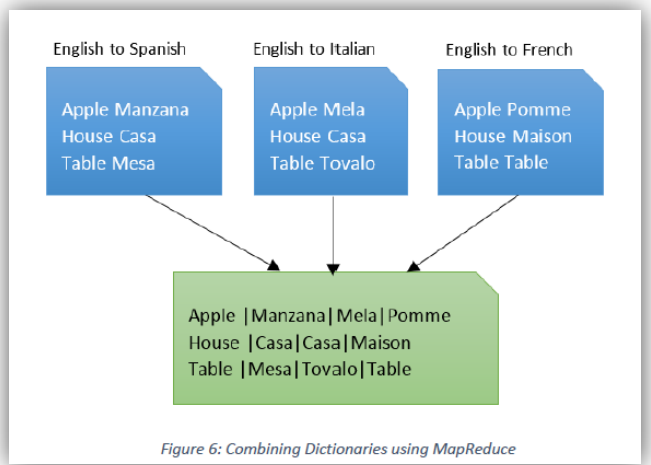

Figure 6: Combining Dictionaries using MapReduce

```
mapLanguage(A_to_B, content)
        emit(content(_1),  content(_2))

reducerLanguage(„someConst“, mappedLanguages)
        result = mappedLanguages(_1) * „|“
        for(i=0 to mappedLanguages.size)
                result += mappedLanguages(_2) + „|“
        emit(„someConst“, result)
        // Or alternatively we could of have assigned the string „result“ to be an
        // empty string at the beginning, and at the end to emit it as a key

//With „D“ we will denote a list of tuples, which are in form:
// „(from_langage_A_to_language_B, content_of_document)“
main(D)
        mappedLanguages = mapAll( mapLanguage, D)
        // Where „mapLangage(D)“ is equivalent to  „mapLangage(A_to_B, content)“
        result = reduceAll(reducerLanguage, mappedLanguages)
        return result
```

2.7 EXAMPLE 7:

An inverted index consists of a list of all the unique words that appear in any document, and for each word, a list of the documents in which it appears. The inverted index is useful for fast retrieval of relevant information. Let's look at building an inverted index for a set of tweets based on their hashtags and how we can map the solution as a MapReduce.

```
mapTweets(userName, tweetContent)
        emit(extarctHashtag(tweetContent), extarctWIthoutHashtag(tweetContent))

reduceTweets(hashtag, tweetContent)
        emit(hashtag, mergeString(tweetContent))

main(tweets)
        hashtagContnetPair = mapAll(mapTweets, tweets)
        // variable „tweets“ can be observed as a tuple: (userName, content)
        result = reduceAll(reduceTweets, hashtagContnetPair)
        return result
```

Example of the execution of map-reduce (which is different from the example given above):

Input Data:
"It's not too late to vote. #ElectionDay"
"Midtown polling office seeing a steady flow of voters! #PrimaryDay"
"Today's the day. Be a voter! #ElectionDay"
"Happy #PrimaryDay"
"Say NO to corruption & vote! #ElectionDay"
"About to go cast my vote...first time #ElectionDay"

Map Output:
("ElectionDay", "It's not too late to vote. #ElectionDay")
("PrimaryDay", "Midtown polling office seeing a steady flow of voters! #PrimaryDay")
("ElectionDay", "Today's the day. Be a voter! #ElectionDay ")
("PrimaryDay", "Happy #PrimaryDay")
("ElectionDay", "Say NO to corruption & vote! #ElectionDay")
("ElectionDay", "About to go cast my vote...first time #ElectionDay")

Reduce Input:

Reducer 1:
("ElectionDay", "It's not too late to vote. #ElectionDay")
("ElectionDay", "Today's the day. Be a voter! #ElectionDay ")
("ElectionDay", "Say NO to corruption & vote! #ElectionDay")
("ElectionDay", "About to go cast my vote...first time #ElectionDay")

Reducer 2:
("PrimaryDay", "Midtown polling office seeing a steady flow of voters! #PrimaryDay")
("PrimaryDay", "Happy #PrimaryDay")

Reduce Output:
("ElectionDay", ["It's not too late to vote. #ElectionDay",
"Today's the day. Be a voter! #ElectionDay ",
"Say NO to corruption & vote! #ElectionDay",
"About to go cast my vote...first time #ElectionDay"])
("PrimaryDay", ["Midtown polling office seeing a steady flow of voters! #PrimaryDay",
"Happy #PrimaryDay"])

2.8 EXAMPLE 8:

MapReduce can be used to join two database tables based on common criteria. Let's take an example.

We have two tables, where the first contains an employee's personal information (including the city where he comes from) primary keyed on SSN, and the second table includes the employee's income, again keyed on SSN. We would like to compute average income in each city.

This computation requires a JOIN operation on these two tables. We will map the problem to a two-phase MapReduce solution. The first phase effectively creates a JOIN on the two tables using two map functions (one for each table), and the second phase gathers the relevant data for calculating desired statistics.

```
mapTable1(„table1“, table1)
        emit(table1(_1), table1(_2))

mapTable2(„table2“, table2)
        emit(table2(_1), table2(_2))

//all people who hame same SNN from emit phases will get to a single reduce
reduceJoin(„someConst“, listOfRowsINBothTables)
        rowsFromTable1 = listOfRowsINBothTables.filter(isString(_2))
        rowsFromTable2 = listOfRowsINBothTables.filter(isNumeric(_2))
        emit(„someString“, joinOnKey(rowsFromTable1, rowsFromTable2))

mapCity(„someString“, joinedTables)
        emit(extractColumn(joinedTables, „city“), extractColumn(joinedTables, „income“))

reduceAvgIncomeByCity(city, income)
        emit(city, avg(income))


// „table1“ can be seen as pair (SNN, city, other_info)
// „table2“ can be seen as pair (SNN, income)
main(table1, table2)
        a = mapAll( mapTable1, („table1“, table1) )
        b = mapAll( mapTable2, („table2“, table2) )
        joinedTables = reduceAll(reduceJoin, a&b)
        // Sending both variables „a“ and „b“ to the reducer
        mappedByCities = mapAll(mapCity, joinedTables)
        result = mapAll(reduceAvgIncomeByCity, mappedByCities)
        return result
```

-This example is very similar to „EXAMPLE 1“

Example of the execution of map-reduce (which is different from the example given above):

Stage 1

Map Output:

Mapper 1a: (SSN, city)
(111222, "Sacramento, CA")
(333444, "San Diego, CA)
(555666, "San Diego, CA)

Mapper 1b: (SSN, income 2016)
(111222, $70000)
(333444, $72000)
(555666, $80000)

Reduce Input: (SSN, city), (SSN, income)
(111222, "Sacramento, CA")
(111222, $70000)
(333444, "San Diego, CA")
(333444, $72000)
(555666, "San Diego, CA")
(555666, $80000)

Reduce Output: (SSN, [city, income])
(111222, ["Sacramento, CA", 70000])
(333444, ["San Diego, CA", 72000])
(555666, ["San Diego, CA", 80000])

Stage 2:

Map Input: (SSN, [city, income])
(111222, ["Sacramento, CA", 70000])
(333444, ["San Diego, CA", 72000])
(555666, ["San Diego, CA", 80000])

Map Output: (city, income)
("Sacramento, CA", 70000)
("San Diego, CA", 72000)
("San Diego, CA", 80000)

Reduce Input: (city, income)

Reducer 2a:
("Sacramento, CA", 70000)

Reducer 2b:
("San Diego, CA", 72000)
("San Diego, CA", 80000)

Reduce Output: (city, average [income])

Reducer 2a:
("Sacramento, CA", 70000)

Reducer 2b:
("San Diego, CA", 76000)

The reader is encouraged to think how the solution will differ if the employee is allowed to have multiple addresses, i.e., there can be multiple addresses per SSN in Table 1.

2.9 EXAMPLE 9 (DS, assignment 6, exercise 2b):

Data is set of files consisting of text. Output the file names that contain a given pattern.

```
mapContaintsPattern(pattern, D)
        if (D.contains(pattern))
                emit(„yes“, D(_2))

// Variable „D“ can be viewed as pair (fileName, contentOfTheFile)
main(D, pattern)
        docsWhichCotnainPattern = mapAll(mapContaintsPattern, (pattern, D))
        return docsWhichCotnainPattern
```

2.10 EXAMPLE 10 (DS, assignment 6, exercise 2c):

Given a set of files which contain one value per line, sort the values

```
//it is important that in this example only one line of the document is emitted at a time
map(„D“, D)
        for(oneLine in D.valus)
                emit(oneLine, „someConst“)

reduce(D, „someConst“)
        emit(D, „someConst“)

main(D)
        a = mapAll(map, D)
        reduceAll(reduce, a)
```

This example is very interesting, as it doesn't do anything in the code, but we manage to get the correct final result (although it's executing will not be very fast).

Here we take advantage of the reducer:

The (key, value) pairs are processed in the order which depend on their keys, and reducers themselves are ordered.

The (key, value) pairs from mappers are sent to a particular reducer based on **hash(key)** . You should just pick a hash function such that for your data k1<k2 it outputs hash(k1) < hash(k2)

2.11 EXAMPLE 11 (FDE exam 20.06.2020, problem 5):

You are given a data set S of shipping container travel information. Every row denotes that a specific container either arrived at a port, or left a port. These columns are available:

- ContainerID (Id of shipping container)
- Port (Name of the port)
- RecordType (Departure or Arrival)
- Timestamp (Time of departure or arrival)

Write Map-Reduce pseudo code to answer the following questions.

2.11.1 EXAMPLE 11.1:

a) Find decommissioned (isključen iz upotrebe) containers (not used in the last year).

```
mapNumberOfTimesContainerUsedNotLastYear(containerID, remainingInfoAboutContrainter)
        if(remainingInfoAboutContrainter[„Timestamp“] != 2020) //2020 is last year
                emit(containerID, 1)

reduceNumberOfTimesContainerUsedNotLastYear(containerID, listOfOnes)
        emit(containerID, (“usedNotLastYear”, sum(listOfOnes)))

mapNumberOfTimesContainerUsedAnyYear(containerID, remainingInfoAboutContrainter)
        emit(containerID, 1)

reduceNumberOfTimesContainerUsedAnyYear(containerID, listOfOnes)
        emit(containerID, (“usedAnyYear”, sum(listOfOnes)))

reduceFinal(containerID, list)
        numContainerUsedLastYear = list.find(_1 == “usedNotLastYear”)
        numContainerUsedAnyYear = list.find(_1 == “usedAnyYear”)
        if(numContainerUsedLastYear == 0)
                emit(containerID, “const”)

main(S)
        a = mapAll(mapUsedContainerInLastYear, S)
        numContainerUsedLastYear =
                                reduceAll(reduceNumberOfTimesContainerUsedNotLastYear, a)

        b = mapAll(mapNumberOfTimesContainerUsedAnyYear, S)
         numContainerUsedAnyYear =
                                reduceAll(reduceNumberOfTimesContainerUsedAnyYear, b)

        return reduceAll(reduceFinal, union(numContainerUsedLastYear,
                                                numContainerUsedAnyYear))
```

Note that you could easily make a mistake in solving this problem, by thinking that you can just make a mapper and corresponding reducer for calculating the number of departures a container was on during the last year, and making another matter and reducer to calculate the number of arrivals a container was part of in the last year, and at the end making a final reducer in which you would check if both number departures in the last year and number of arrivals in the last year is zero. The reason why such a solution would not give out a correct solution is because if there is a container which was not included in any of the departures last year the first mapper would not emit anything, and consequently the reducer for that container would never be called. Same stands for the case if a container was not included in any arrivals in the last year, where an emit for that container would never happen..

2.11.2 EXAMPLE 11.2:

b) Find containers that have been to all ports.

```
mapTotalNumberOfPorts(containerID, remainingInfoAboutContrainter)
        emit("const", remainingInfoAboutContrainter[„port"])

//Different reducers will be assigned with different port number
reduceTotalNumberOfPorts(const, listOfPorts)
        emit("const", unique(listOfPorts).count())

mapVisitedPorts(containerID, remainingInfoAboutContrainter)
        emit(containerID, remainingInfoAboutContrainter[„port"])

// each machine is going to work on one port, which will get pair:
// (containerID,  listOfVisitedPorts)
reduceVisitedPorts(containerID, listOfVisitedPorts)
        emit(continerID, ("visited", unique(listOfVisitedPorts).count))

main(S)
        numberOfPorts = mapAll(mapTotalNumberOfPorts, S)
        totalNumberOfPorts = reduceAll(reduceTotalNumberOfPorts, numberOfPorts)["const"]

        visitedPorts = mapAll(mapVisitedPorts, S)
        numberOfVisitedPorts = reduceAll(reduceVisitedPorts, visitedPorts)

        return numberOfVisitedPorts.filter( totalNumberOfPorts == numberOfVisitedPorts(_2))
```

-Only one value („total number of something");

-"$variable_{SINGLE\_VALUE}$"

-A list of pairs

-„$variable_{PAIRS}$"

When we have map-reduce problems which have at the end in the main function two variables, of which one is a single value (we will denote it with "$variable_{SINGLE\_VALUE}$"), which is most often referring to „total number of something", and the other variable is a list of pairs (we will denote it with "$variable_{PAIRS}$") where the „value" part of the pairs is something we need to compare to the "$variable_{SINGLE\_VALUE}$", then we should not make a "reduceFinal" reducer, but instead get the result in the main function using the following syntax:

return $variable_{PAIRS}$.filter($variable_{SINGLE\_VALUE}$ == $variable_{PAIRS}$(_2))

The similar trick is used in the [FDE exercise sheet 8, problem 2] (which is in this PDF file in "section 2.2 example 2").

2.11.3 EXAMPLE 11.3:

c) Find ports that send out more containers than they receive.

```
mapCountSentOutFromPort(containerID, remainingInfoAboutContrainter)
        if(remainingInfoAboutContrainter[„RecordType“] == „Departure“)
                emit(remainingInfoAboutContrainter[„Port“], 1)

reduceSumPortDepartures(portName, listOfOnes)
        emit(portName, (“sentOut”, sum(listOfOnes)))

mapCountReceiveFromPort(containerID, remainingInfoAboutContrainter)
        if(remainingInfoAboutContrainter[„RecordType“] == „Arrival“)
                emit(remainingInfoAboutContrainter[„Port“], 1)

reduceSumPortArrivals(portName, listOfOnes)
        emit(portName, (“arrivals”,sum(listOfOnes)))

reduceFinal(portID, list)
        portDepatures = list.find(-1 == ” sentOut”)
        portArrivals = list.find(_1 == “arrivals”)
        if(portDepatures > portArrivals)
                emit(portID, “const”)

main(S)
        a = mapAll(mapCountSentOutFromPort, S)
        portDepartures = reduceAll(reduceSumPortDepartures , a)

        b = mapAll(mapCountReceiveFromPort, S)
        portArrivals = reduceAll(reduceSumPortArrivals, b)

        return reduceAll(reduceFinal, union(portDepartures, portArrivals))
```

3. Question about map-reduce problems for the tutors

3.1. The mail sent to the tutors

EXAMPLE 11.4 (similar to task 11.3):

Assume you have dataset of boxing fighters which contained: „fighterID“, „oponent“, „outcome“, where column „outcome“ can be either „won“ or „lost“.

Find all fighters which won more boxing matches than they have lost.

```
mapCountFighterWon(fighterID, restOfInfo)
        if(restOfInfo[„outcome“] == „won“)
                emit(fighterID, 1)

reduceCountFighterWon(fighterID, listOfOnes)
        emit(fighterID, sum(listOfOnes))

mapCountFighterLost(fighterID, restOfInfo)
        if(restOfInfo[„outcome“] == „lost“)
                emit(fighterID, 1)

reduceCountFighterLost(fighterID, listOfOnes)
        emit(fighterID, sum(listOfOnes))

??????????????????????????

main(F)
        a = mapAll(mapCountFighterWon, F)
        numFightsWon = reduceAll(reduceCountFighterWon, a)

        b = mapAll(mapCountFighterLost, F)
        numFightsLost = reduceAll(reduceCountFighterLost, b)
```

As you see from the code I wrote for the proposed problem that I found on the internet (where no solution was provided), using the map-reduce approach to solve it, I come to a point where I am not sure how should I proceed in the solution. I am wondering, as I have both the number of fights a boxer won saved in the variable „numFightsWon“, and the number of fights a boxer lost saved in the variable „numFightsLost“, can I do the comparison of these two variables in the „main“ function by simply adding lines of code:

```
if(numFightsWon(_2) > numFIghtsLost(_2))
        return numFightsWon(_1)
```

, or look for a different approach?
One of the possible solutions I was thinking of was making another map function, which would do more or less the same thing, like for an example:

```
//Let's say that the number of won matches is the first element of „pairOfWonLostMatches“, while
// the number of matches lost is the second element of a pair „pairOfWonLostMatches“
    mapResult(fighterID, pairOfWonLostMatches)
        if(pairOfWonLostMatches(_1) > pairOfWonLostMatches(_2))
            emit(fighterID, „wonMoreThanLost“)
```

The third solution, the only which I am confident that is not the correct way (but not 100% sure either), is to use a reduce function instead of a map function, or doing it in the “main”. Maybe I am wrong about the reducer as well, so to be honest I do not know which of the three proposed solutions should be used.

3.2 The answer from a tutor on a proposed question:

If you think that the number of elements is not too big, you can put your final processing step into the main function. This is, e.g., done by us on sheet 8, exercise 2.2. The exact syntax is not too important as long as it's clear what you mean. For your problem, you could use something like this:

```
result = []
for (fighterID in noFightsWon.keys)
        if (noFightsWon[fighterID] > noFightsLost[fighterID])
            result += fighterID
return result
```

Of course, your solution will be better (and may get you more points) if the processing in the main function is minimized, because such processing can by definition not be distributed. For example, a better solution would be the following map-reduce program (I have marked the changes to your program in bold):

```
mapCountFighterWon(fighterID, restOfInfo)
        if(restOfInfo["outcome"] == "won")
            emit(fighterID, 1)

reduceCountFighterWon(fighterID, listOfOnes)
        emit(fighterID, ("won", sum(listOfOnes))
```

```
mapCountFighterLost(fighterID, restOfInfo)
        if(restOfInfo["outcome"] == "lost")
                emit(fighterID, 1)

reduceCountFighterLost(fighterID, listOfOnes)
        emit(fighterID, ("lost", sum(listOfOnes)))

reduceFinal(fighterID, list)
        numWon = list.find(_1 == "won")
        numLost = list.find(_1 == "lost")
         if (numWon > numLost)
                // Note that the 0 is just some arbitrary value which is not important;
                 // it's only required to fulfill the function signature "emit(key, value)".
                emit(fighterId, 0)

main(F)
        a = mapAll(mapCountFighterWon, F)
        numFightsWon = reduceAll(reduceCountFighterWon, a)

        b = mapAll(mapCountFighterLost, F)
        numFightsLost = reduceAll(reduceCountFighterLost, b)

        return reduceAll(reduceFinal, union(numFightsWon, numFightsLost))
```

I hope this pseudo code makes sense to you?

By the way, there's an even more compact solution to this problem (and in the exam, you should of course strive towards a more compact solution, if only to reduce the time needed for writing it down):

```
map(fighterID, restOfInfo)
         if (restOfInfo["outcome"] == "won")
                emit(fighterID, 1)
         else
                emit(fighterID, -1)

reduce(fighterID, list)
        if (sum(list) > 0)
                emit(fighterID, 0)

main(F)
        a = mapAll(map, F)
        return reduceAll(reduce, a)
```

Note that with all of the solutions you provided, you run the risk that we do not clearly understand what exactly you're trying to accomplish. With a more clear solution like one of the above (preferably the last one, of course), you will increase your chances of getting full points in the exam.

3.2.1 Ilustration on an example

To better illustrate what is going on, let's consider an example:

„numFightsWon“ and „numFightsLost“ are sets of the form:

{("Bruce", ("won", 2)), ("Alex", ("won", 8)), ...}

respectively

{("Bruce", ("lost", 7)), ("Alex", ("lost", 3)), ...}.

Hence, we can create a union of them:

union(numFightsWon, numFightsLost) = *{("Bruce", ("won", 2)), ("Bruce", ("lost", 7)), ("Alex", ("won", 8)), ("Alex", ("lost", 3)), ...}*

We then reduce the union set using „reduceFinal“.

Firsly, „reduceAll“ will, as usual, group the key-value-pairs from the union by key. Hence, for our example, „reduceFinal“ will be called once for each of the following argument pairs:

reduceFinal(*"Bruce", {("won", 2), ("lost", 7)}*)

reduceFinal(*"Alex", {("won", 8), ("lost", 3)}*)

...

Now, „reduceFinal“ will find the pair in the list whose first element is "won" and store its value in the „numWon“ variable. Same for „numLost“. So, for the first call
(with "Bruce"), „numWon“ and „numLost“ will have the values:

numWon = 2

numLost = 7