**Série de TD n°2**

**MapReduce sous Hadoop**

**Exercice n°1 :** Reconsidérons l’exemple du word\_count dont on a précédemment vu le pseudo-code. Ecrivez le programme Java du word\_count (avec les fonctions Map et Redue) pour être exécuté sous Hadoop.

**Exercice n°2 :** Etant donné le fichier des ventes de produits par pays (voir le fichier SalesJan2009.csv)[[1]](#footnote-1)dont voici les premières lignes.

|  |  |  |
| --- | --- | --- |
| Transaction\_date,Product,Price,Payment\_Type,Name,City,State,Country,Account\_Created,Last\_Login,Latitude,Longitude | |  |
| 1/2/09 6:17,Product1,1200,Mastercard,carolina,Basildon,England,United Kingdom,1/2/09 6:00,1/2/09 6:08,51.5,-1.1166667 | |  |
| 1/2/09 4:53,Product1,1200,Visa,Betina,Parkville ,MO,United States,1/2/09 4:42,1/2/09 7:49,39.195,-94.68194 |  |  |
| 1/2/09 13:08,Product1,1200,Mastercard,Federica e Andrea,Astoria ,OR,United States,1/1/09 16:21,1/3/09 12:32,46.18806,-123.83 | | |
| 1/3/09 14:44,Product1,1200,Visa,Gouya,Echuca,Victoria,Australia,9/25/05 21:13,1/3/09 14:22,-36.1333333,144.75 |  |  |
| 1/4/09 12:56,Product2,3600,Visa,Gerd W ,Cahaba Heights ,AL,United States,11/15/08 15:47,1/4/09 12:45,33.52056,-86.8025 | |  |

Il vous est demandé de

1. Ecrire les fonctions MAP et REDUCE qui permettent d’avoir le nombre de produits vendus par pays.

**Exercice n°3 :** On souhaite établir des statistiques sur la longueur des mots dans un document, ou un ensemble de documents, à partir d’une application Map‐Reduce. Proposez le code java dans le paradigme Map‐Reduce pour les cas suivants :

Q1. Compter le nombre de mots de chaque longueur présente dans le texte.

Q2. Compter le nombre de mots de 1 à 5 caractères (inclus), de 6 à 10 caractères (inclus), de 11 à 15 caractères (inclus) et de plus de 15 caractères présents dans le texte.

Q3. Obtenir les listes de mots de 1 à 5 caractères (inclus), de 6 à 10 caractères (inclus), de 11 à 15 caractères (inclus) et de plus de 15 caractères présents dans le texte. Il n’est pas demandé de trier les mots à l’intérieur d’une liste, ni d’éliminer les doublons.

**Corrigé**

**Exercice n°1**

**MAP**

**import** java.io.IOException;

**import** java.util.StringTokenizer;

**import** org.apache.hadoop.io.IntWritable;

**import** org.apache.hadoop.io.LongWritable;

**import** org.apache.hadoop.io.Text;

**import** org.apache.hadoop.mapreduce.Mapper;

**public** **class** MapClass **extends** Mapper{

**private** **final** **static** IntWritable ***one*** = **new** IntWritable(1);

**private** Text word = **new** Text();

@Override

**protected** **void** map(LongWritable key, Text value,

Context context)

**throws** IOException, InterruptedException {

//Get the text and tokenize the word using space as separator.

String line = value.toString();

StringTokenizer st = **new** StringTokenizer(line," ");

//For each token aka word, write a key value pair with

//word and 1 as value to context

**while**(st.hasMoreTokens()){

word.set(st.nextToken());

context.write(word,***one***);

}

}

}

**REDUCE**

**import** java.io.IOException;

**import** java.util.Iterator;

**import** org.apache.hadoop.io.IntWritable;

**import** org.apache.hadoop.io.Text;

**import** org.apache.hadoop.mapreduce.Reducer;

**public** **class** ReduceMap **extends** Reducer {

@Override

**protected** **void** reduce(Text key, Iterable values, Context context)

**throws** IOException, InterruptedException {

**int** sum = 0;

Iterator valuesIt = values.iterator();

//For each key value pair, get the value and adds to the sum

//to get the total occurances of a word

**while**(valuesIt.hasNext()){

sum = sum + valuesIt.next().get();

}

//Writes the word and total occurances as key-value pair to the context

context.write(key, **new** IntWritable(sum));

}

}

**MAIN**

**import** org.apache.hadoop.conf.Configured;

**import** org.apache.hadoop.fs.Path;

**import** org.apache.hadoop.io.IntWritable;

**import** org.apache.hadoop.io.Text;

**import** org.apache.hadoop.mapreduce.Job;

**import** org.apache.hadoop.mapreduce.lib.input.FileInputFormat;

**import** org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;

**import** org.apache.hadoop.mapreduce.lib.output.TextOutputFormat;

**import** org.apache.hadoop.util.Tool;

**import** org.apache.hadoop.util.ToolRunner;

**public** **class** WordCount **extends** Configured **implements** Tool{

**public** **static** **void** main(String[] args) **throws** Exception{

**int** exitCode = ToolRunner.run(**new** WordCount(), args);

System.*exit*(exitCode);

}

**public** **int** run(String[] args) **throws** Exception {

**if** (args.length != 2) {

System.err.printf("Usage: %s needs two arguments, input and output

files\n", getClass().getSimpleName());

**return** -1;

}

//Create a new Jar and set the driver class(this class) as the main class of jar

Job job = **new** Job();

job.setJarByClass(WordCount.**class**);

job.setJobName("WordCounter");

//Set the input and the output path from the arguments

FileInputFormat.addInputPath(job, **new** Path(args[0]));

FileOutputFormat.setOutputPath(job, **new** Path(args[1]));

job.setOutputKeyClass(Text.**class**);

job.setOutputValueClass(IntWritable.**class**);

job.setOutputFormatClass(TextOutputFormat.**class**);

//Set the map and reduce classes in the job

job.setMapperClass(MapClass.**class**);

job.setReducerClass(ReduceClass.**class**);

//Run the job and wait for its completion

**int** returnValue = job.waitForCompletion(**true**) ? 0:1;

**if**(job.isSuccessful()) {

System.out.println("Job was successful");

} **else** **if**(!job.isSuccessful()) {

System.out.println("Job was not successful");

}

**return** returnValue;

}

}

**Exercice n°2**

MAP

import java.io.IOException;

import org.apache.hadoop.io.\*

import org.apache.hadoop.mapred.\*;

public class SalesMapper extends Mapper {

private final static IntWritable one = new IntWritable(1);

public void map(LongWritable key, Text value, Context context()) throws IOException {

String valueString = value.toString();

String[] SingleCountryData = valueString.split(",");

context.write(new Text(SingleCountryData[7]), one);

}

}

**REDUCE**

import java.io.IOException;

import java.util.\*;

import org.apache.hadoop.io.\*

import org.apache.hadoop.mapred.\*;

public class SalesCountryReducer extends Reducer {

public void reduce(Text t\_key, Iterator <IntWritable> values, Context context) throws IOException {

int frequencyForCountry = 0;

Iterator valueIt=values.Iterator();

while (valueIt.hasNext()) { frequencyForCountry += valueIt.next().get();

}

context.write(t\_key, new IntWritable(frequencyForCountry));

}

**Exercice n°3**

**MAP**

import java.io.IOException;

import org.apache.hadoop.io.\*

import org.apache.hadoop.mapred.\*;

public class ConsumeMapper extends Mapper {

private final static IntWritable[] tab = new IntWritable();

public void map(LongWritable key, Text value, Context context()) throws IOException {

String line= value.toString();

String[] conso = valueString.split(“ ");

tab[0] = yearCon[0];

tab[1] = yearCon[13];

string consAn= tab[0] + ” “ + tab[1]

context.write(“année”,new Text(consAn);

}

}

**Reduce**

import java.io.IOException;

import java.util.\*;

import org.apache.hadoop.io.\*

import org.apache.hadoop.mapred.\*;

public class ConsumReducer extends Reducer {

public void reduce(Text t\_key, Iterator[] <IntWritable> values, Context context) throws IOException {

int max = 0;

Iterator valueIt=values.Iterator();

while (valueIt.hasNext()) {

int[] T = parseInt(valueIt.split())

if (T[1] > max) {

max = T[1] ;

annee=T[0]

}

}

context.write(new IntWritable(annee), new IntWritable(max));

}

Exercice n°3

**public class CaptTemperatureMapper**

**extends** Mapper<LongWritable, Text, Text, IntWritable> {

**private static final int** MISSING = 9999;

@Override

**public void** map(LongWritable key, Text value, Context context)

**throws** IOException, InterruptedException {  
String line = value.toString();  
String id\_station = line.substring(5,10);

**int** airTemperature;  
**if** (line.charAt(87) == '+') { *// parseInt doesn't like leading plus signs* airTemperature = Integer.parseInt(line.substring(88, 92));  
} **else** {  
 airTemperature = Integer.parseInt(line.substring(87, 92));  
}

String quality = line.substring(92, 93);  
**if** (airTemperature = MISSING || quality.matches("[23678]")) {  
context.write(**new** Text(id\_station), **new** Text(“CapTemp”));  
}

}

}

public class CaptTempReducer extends Reducer {

    public Void reducer(Text Id\_Station, Iterator <Text> Capts,

    Context contex()) throws IOExeption{

               contex.write(Id\_Station, new Text(“Capt\_temp\_defaillant”));

    }

}

public class Mapper extends Mapper {

    private static final int MISSING = 9999 ;

public Void map (LongWritable key, Text value, Context contex())

throws IOExeption{

        int Id\_Station;

        int CaptEtat;

string

        Id\_Station = value.SubString(0,5);

        Temperature = value.SubString(88,92);

        Context.write(Id\_Station,Temperature);

    }

}

Q1

import java.io.IOException;

import java.util.HashMap;

import java.util.Map;

import org.apache.hadoop.io.IntWritable;

import org.apache.hadoop.io.Text;

import org.apache.hadoop.mapreduce.Mapper;

import org.apache.hadoop.mapreduce.Reducer;

public class WordLengthCounterMapper extends Mapper<Object, Text, IntWritable, IntWritable> {

@Override

public void map(Object key, Text value, Context context) throws IOException, InterruptedException {

String line = value.toString();

List<String> words = splitWords(line);

for (String word : words) {

int length = word.length();

context.write(new IntWritable(length), new IntWritable(1));

}

}

private List<String> splitWords(String line) {

List<String> words = new ArrayList<>();

for (String word : line.split(" ")) {

if (word.length() > 0) {

words.add(word);

}

}

return words;

}

}

public class WordLengthCounterReducer extends Reducer<IntWritable, IntWritable, IntWritable, Text> {

@Override

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

int count = 0;

for (IntWritable value : values) {

count += value.get();

}

context.write(key, new Text(String.valueOf(count)));

}

}

Q2.

import java.io.IOException;

import java.util.HashMap;

import java.util.Map;

import org.apache.hadoop.io.IntWritable;

import org.apache.hadoop.io.Text;

import org.apache.hadoop.mapreduce.Mapper;

import org.apache.hadoop.mapreduce.Reducer;

public class WordLengthCounterMapper extends Mapper<Object, Text, IntWritable, IntWritable> {

private static final int MIN\_LENGTH\_1\_TO\_5 = 1;

private static final int MAX\_LENGTH\_1\_TO\_5 = 5;

private static final int MIN\_LENGTH\_6\_TO\_10 = 6;

private static final int MAX\_LENGTH\_6\_TO\_10 = 10;

private static final int MIN\_LENGTH\_11\_TO\_15 = 11;

private static final int MAX\_LENGTH\_11\_TO\_15 = 15;

private static final int MIN\_LENGTH\_MORE\_THAN\_15 = 16;

@Override

public void map(Object key, Text value, Context context) throws IOException, InterruptedException {

String line = value.toString();

List<String> words = splitWords(line);

for (String word : words) {

int length = word.length();

if (length >= MIN\_LENGTH\_1\_TO\_5 && length <= MAX\_LENGTH\_1\_TO\_5) {

context.write(new IntWritable(1), new IntWritable(1));

} else if (length >= MIN\_LENGTH\_6\_TO\_10 && length <= MAX\_LENGTH\_6\_TO\_10) {

context.write(new IntWritable(2), new IntWritable(1));

} else if (length >= MIN\_LENGTH\_11\_TO\_15 && length <= MAX\_LENGTH\_11\_TO\_15) {

context.write(new IntWritable(3), new IntWritable(1));

} else if (length >= MIN\_LENGTH\_MORE\_THAN\_15) {

context.write(new IntWritable(4), new IntWritable(1));

}

}

}

private List<String> splitWords(String line) {

List<String> words = new ArrayList<>();

for (String word : line.split(" ")) {

if (word.length() > 0) {

words.add(word);

}

}

return words;

}

}

public class WordLengthCounterReducer extends Reducer<IntWritable, IntWritable, IntWritable, Text> {

@Override

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

int count = 0;

for (IntWritable value : values) {

count += value.get();

}

switch (key.get()) {

case 1:

context.write(new IntWritable(1), new Text("1-5: " + String.valueOf(count)));

break;

case 2:

context.write(new IntWritable(2), new Text("6-10: " + String.valueOf(count)));

break;

case 3:

context.write(new IntWritable(3), new Text("11-15: " + String.valueOf(count)));

break;

case 4:

context.write(new IntWritable(4), new Text(">15: " + String.valueOf(count)));

break;

}

}

}

Q3

import java.io.IOException;

import java.util.ArrayList;

import java.util.List;

import org.apache.hadoop.io.IntWritable;

import org.apache.hadoop.io.Text;

import org.apache.hadoop.mapreduce.Mapper;

public class WordLengthMapper extends Mapper<Object, Text, IntWritable, Text> {

private static final int MIN\_LENGTH\_1\_TO\_5 = 1;

private static final int MAX\_LENGTH\_1\_TO\_5 = 5;

private static final int MIN\_LENGTH\_6\_TO\_10 = 6;

private static final int MAX\_LENGTH\_6\_TO\_10 = 10;

private static final int MIN\_LENGTH\_11\_TO\_15 = 11;

private static final int MAX\_LENGTH\_11\_TO\_15 = 15;

private static final int MIN\_LENGTH\_MORE\_THAN\_15 = 16;

@Override

public void map(Object key, Text value, Context context) throws IOException, InterruptedException {

String line = value.toString();

List<String> words = splitWords(line);

for (String word : words) {

int length = word.length();

if (length >= MIN\_LENGTH\_1\_TO\_5 && length <= MAX\_LENGTH\_1\_TO\_5) {

context.write(new IntWritable(1), new Text("1-5: " + word));

} else if (length >= MIN\_LENGTH\_6\_TO\_10 && length <= MAX\_LENGTH\_6\_TO\_10) {

context.write(new IntWritable(2), new Text("6-10: " + word));

} else if (length >= MIN\_LENGTH\_11\_TO\_15 && length <= MAX\_LENGTH\_11\_TO\_15) {

context.write(new IntWritable(3), new Text("11-15: " + word));

} else if (length >= MIN\_LENGTH\_MORE\_THAN\_15) {

context.write(new IntWritable(4), new Text(">15: " + word));

}

}

}

private List<String> splitWords(String line) {

List<String> words = new ArrayList<>();

for (String word : line.split(" ")) {

if (word.length() > 0) {

words.add(word);

}

}

return words;

}

}

1. Un fichier CSV est un fichier tableur, contenant des données sur chaque ligne séparées par un caractère de séparation (ici la virgule). [↑](#footnote-ref-1)