Valentin Larrieu

BigData With Hadoop

-

Implementation of Cesar coding

The problem I decided to tackle is in correlation with cryptographic security & BigData. We all know that plenty of messages are exchanges all over the world. Even though some of them are ciphered with different technics (which are sometimes opaque), some are in plain test.

The goal of this project is to implement a ciphering and deciphering method on text messages. We want that knowing the original language in which the message is written and the ciphering algorithm (here Cesar), the people receiving the message to be able to decode it in order to read the original text.

To do so I will use some heuristic properties about the reparation of the most used letter in each languages. The final goal would be to have a system able to take a plain text as an entry, then able to cipher it. After that using the same system, the receiver use heuristic analysis to guess the letter “translation” number (example “a” becomes “e” and “c” becomes “g”) and then decode the text and generate the plain text without knowing that value (it is not sent with the message).

Objectives:

- The system must be able to read text files
- The system must be able to cipher text
- The system must be able to “guess” the ciphering key (aka the number representing the translation)
- The system must be able to decode a message
- The system must be able to write the decoded message in a text file

Technologies chosen

- **Spark** (for the majority of treatment ciphering and decoding) on Databricks
- **Hadoop** (to use the effectiveness of the cluster and execute Map reduce)

Summary

Table des matières

Implementation

As it was precised previously this algorithm is supposed to be able to do the jobs: ciphering or deciphering.

The Cesar code

The text will be cipher with Cesar code. It I a simple way to code a message. The letters are simply rotated in the alphabetic table all by the same value.

Example: **Cesar** becomes with a ciphering key of **2**: **Eguct** (new letter = new letter +2).

If the resulting letter goes out of the range of a-z, it is brought back inside from the other side of the sequence. We can see this here r alphabet as a permutation circle.

Example: **zeta** with a ciphering key of **3** is **Chsd**

As we can see, our ciphering method is just a translation of letter socypherg with a key of +2 and then with a key of -2 gives us back our text

Example: **TEST** becomes with a ciphering key of +**2**: **VGUV** then ciphering it with a key of **-2** gives us our text back: **TEST**

Ciphering Deciphering

First Version on Python

We first implemented it on python

- A function “cypherletter” returns the letter ciphered by the precised offset

```
#this function can code and decode,if the entry is letter it return the letter + SHIFT letter (in a circular list)
# else it returns the letter given first. It implements the Cesar coding
def cypherLetter(x,value):
  SHIFT=value
  if ((ord(x) <= ord("Z")) and (ord(x) >= ord("A"))):  # we test if it is an upercase letter
    if ((ord(x) + SHIFT) <= ord("Z")) and ((ord(x) + SHIFT) >= ord("A")):  # the new letter stay in the interval
      return chr(ord(x) + SHIFT)
    else: # if it is outside of the interval
      if ((ord(x) + SHIFT) > ord("Z")):  # the new letter is outside the right part of the interval
        return chr(ord("A") -1 + (ord(x) + SHIFT - ord("Z")))
      else:  # the new letter is outside the left part of the interval
        return chr(ord("Z") +1 + (ord(x) + SHIFT - ord("A")))

  if ((ord(x) <= ord("z")) and (ord(x) >= ord("a"))):
    if ((ord(x) + SHIFT) <= ord("z")) and ((ord(x) + SHIFT) >= ord("a")):
      return chr(ord(x) + SHIFT)
    else:  # It is outside of the interval
      if ((ord(x) + SHIFT) > ord("z")):  # the new letter is outside the right part of the interval
        return chr(ord("a") -1 + (ord(x) + SHIFT - ord("z")))
      else: # the new letter is outside the left part of the interval
        return chr(ord("z") +1 + (ord(x) + SHIFT - ord("a")))
  else:  # it is not a letter
    return x
```

  - So as we can see we test if our letter is an uppercase or lowercase (else it is not a letter so we do not cipher it)
  - Then we verify if the offseted letter is still in the alphabet range. If it is not it is brough back inside
  - We then return the new letter

As we can see the entries are letter to change and the “offset” to apply. It can be positive or negative thus the possibility to cipher and decipher.

- A function “cypherDataframeColumn” is then able to apply this function to a full column of the dataframe

```
def cypherDataframeColumn(columnName,value,dataframe):
  udf = UserDefinedFunction(lambda x: cypherLetter(x,value), StringType())
  new_df = dataframe.select(*[udf(column).alias(columnName) if column == columnName else column for column in dataframe.columns])
  return new_df
```

Second Version with Map/reduce

We implemented the same function able to “shift letters” in java in our M/R job:

```
public char cipherLetter(char x, int value){
    int SHIFT=value%26;
    char Zvalue='Z';
    char zvalue='z';
    char Avalue='A';
    char avalue='a';
    if (((int)x <= (int)Zvalue) && ((int)x >= (int)Avalue)){  // we test if it is an upercase letter
        if ((((int)x + SHIFT) <= (int)Zvalue) && (((int)x + SHIFT) >= (int)Avalue)){  // the new letter stay in the interval
            return (char)((int)x + SHIFT);
        }
        else{ // if it is outside of the interval
            if (((int)x + SHIFT) > (int)Zvalue){  // the new letter is outside the right part of the interval
                return (char)((int)Avalue -1 + ((int)x + SHIFT - (int)Zvalue));
            }
            else{  // the new letter is outside the left part of the interval
                return (char)((int)Zvalue +1 + ((int)x + SHIFT - (int)Avalue));
            }
        }
    }
    if (((int)x <= (int)zvalue) && ((int)x >= (int)avalue)){
        if ((((int)x + SHIFT) <= (int)zvalue) && (((int)x + SHIFT) >= (int)avalue)){
            return (char)((int)x + SHIFT);
        }
        else{  // It is outside of the interval
            if (((int)x + SHIFT) > (int)zvalue){  // the new letter is outside the right part of the interval
                return (char)((int)avalue -1 + ((int)x + SHIFT - (int)zvalue));
            }
            else{ // the new letter is outside the left part of the interval
                return (char)((int)zvalue +1 +((int)x + SHIFT - (int)avalue));
            }
        }
    }
    else{  // it is not a letter
        return x;
    }
}
```

We choose our **mapper to only map the line number** in order to keep the text in the same order is it was previously

- The character change by itself is done in the reducer class

```
public void reduce(IntWritable key, Iterable<Text> values,
                   Context context
) throws IOException, InterruptedException {
    String line="";
    char charact='a';
    String newLine="";
    for (Text val : values) {
        line=val.toString();
        line=line.replaceAll("(.{1})(?!$)", "$1¤");//we separate every char with a -
        StringTokenizer itr = new StringTokenizer(line,  delim: "¤"); //We cut at everyChar
        while (itr.hasMoreTokens()) {
            charact = itr.nextToken().toString().charAt(0);
            charact = cipherLetter(charact, context.getConfiguration().getInt( name: "SHIFT",  defaultValue: 0));
            newLine = newLine + charact;
        }

    }
    context.write(NullWritable.get(), new Text(newLine));

}
```

- To apply our treatment to every char we separate them using the symbol “¤”
- Then we go through every element of the splited list to change the character (the cypherLetter function only modifying letter and letting everything else intact)
- Finally we write our new elements, giving a Null Value as a key in order not to have a key displayed (we also changed the split between key and value to become nothing instead of tabulation and the output of the reducer to be a NullWritable)

Execution result on the cluster

To test our Map/Reduce we took a book as a text file (A Tale of Two Cities, by Charles Dickens, a 758ko file).

- We putted it on the cluster in an input folder called “inputCipher”
- We putted our jar on the cluster and prepared it

```
-sh-4.2$ zip -d Cipher2.jar 'META-INF/*.SF' 'META-INF/*.RSA' 'META-INF/*SF'
deleting: META-INF/MSFTSIG.SF
deleting: META-INF/MSFTSIG.RSA
```

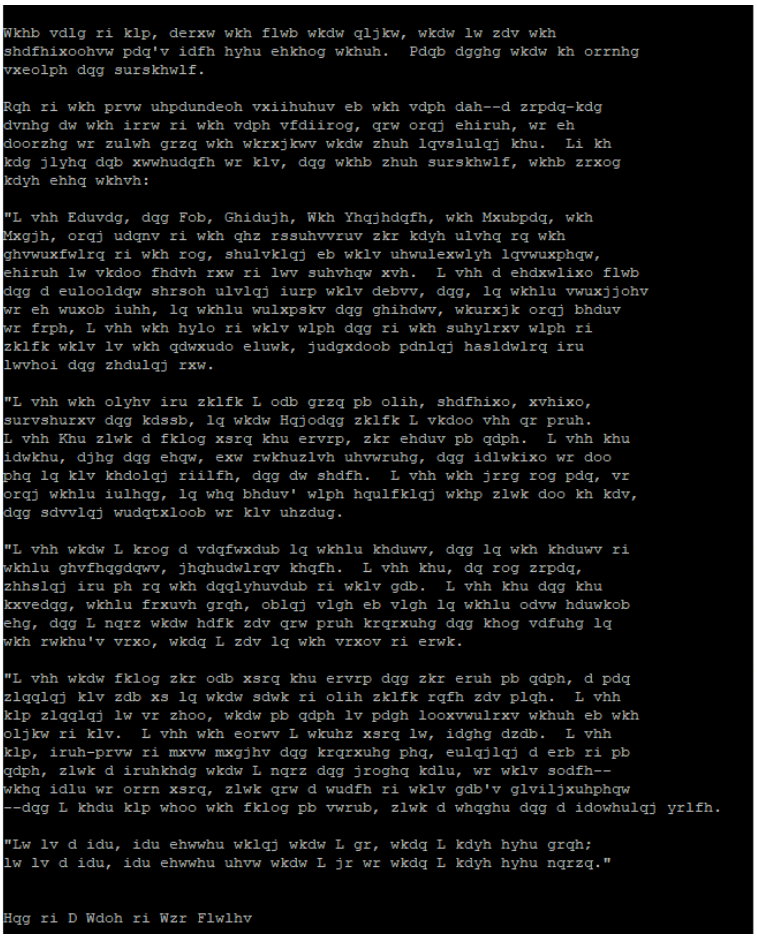
- We executed it on our book file, with a ciphering key of 3

```
-sh-4.2$ hadoop jar Cipher2.jar Cipher2 /user/vlarrieu/inputCipher /user/vlarrieu/outputCipher2 3
```

```
17/11/25 20:56:59 INFO mapreduce.Job:  map 100% reduce 0%
17/11/25 20:57:05 INFO mapreduce.Job:  map 100% reduce 100%
17/11/25 20:57:05 INFO mapreduce.Job: Job job_1507063833384_2087 completed successfully
```

Result of the ciphering Job

(with ciphering of 3 compared with the original text)



```
Wkhb vdlg ri klp, derxw wkh flwb wkdw qljkw, wkdw lw zdv wkh
shdfhixoohvw pdq'v idfh hyhu ehkhog wkhuh.  Pdqb dgghg wkdw kh orrnhg
vxeolph dqg surskhwlf.

Rqh ri wkh prvw uhpdundeoh vxiihuhuv eb wkh vdph dah--d zrpdq-kdg
dvnhg dw wkh irrw ri wkh vdph vfdiirog, qrw orqj ehiruh, wr eh
doorzhg wr zulwh grzq wkh wkrxjkwv wkdw zhuh lqvslulqj khu.  Li kh
kdg jlyhq dqb xwwhudqfh wr klv, dqg wkhb zhuh surskhwlf, wkhb zrxog
kdyh ehhq wkhvh:

"L vhh Eduvdg, dqg Fob, Ghidujh, Wkh Yhqjhdqfh, wkh Mxubpdq, wkh
Mxgjh, orqj udqnv ri wkh qhz rssuhvvruv zkr kdyh ulvhq rq wkh
ghvwuxfwlrq ri wkh rog, shulvklqj eb wklv uhwulexwlyh lqvwuxphqw,
ehiruh lw vkdoo fhdvh rxw ri lwv suhvhqw xvh.  L vhh d ehdxwlixo flwb
dqg d eulooldqw shrsoh ulvlqj iurp wklv debvv, dqg, lq wkhlu vwuxjjohv
wr eh wuxob iuhh, lq wkhlu wulxpskv dqg ghihdwv, wkurxjk orqj bhduv
wr frph, L vhh wkh hylo ri wklv wlph dqg ri wkh suhylrxv wlph ri
zklfk wklv lv wkh qdwxudo eluwk, judgxdoob pdnlqj hasldwlrq iru
lwvhoi dqg zhdulqj rxw.

"L vhh wkh olyhv iru zklfk L odb grzq pb olih, shdfhixo, xvhixo,
survshurxv dqg kdssb, lq wkdw Hqjodqg zklfk L vkdoo vhh qr pruh.
L vhh Khu zlwk d fklog xsrq khu ervrp, zkr ehduv pb qdph.  L vhh khu
idwkhu, djhg dqg ehqw, exw rwkhuzlvh uhvwruhg, dqg idlwkixo wr doo
phq lq klv khdolqj riilfh, dqg dw shdfh.  L vhh wkh jrrg rog pdq, vr
orqj wkhlu iulhqg, lq whq bhduv' wlph hqulfklqj wkhp zlwk doo kh kdv,
dqg sdvvlqj wudqtxloob wr klv uhzdug.

"L vhh wkdw L krog d vdqfwxdub lq wkhlu khduwv, dqg lq wkh khduwv ri
wkhlu ghvfhqgdqwv, jhqhudwlrqv khqfh.  L vhh khu, dq rog zrpdq,
zhhslqj iru ph rq wkh dqqlyhuvdub ri wklv gdb.  L vhh khu dqg khu
kxvedqg, wkhlu frxuvh grqh, oblqj vlgh eb vlgh lq wkhlu odvw hduwkob
ehg, dqg L nqrz wkdw hdfk zdv qrw pruh krqrxuhg dqg khog vdfuhg lq
wkh rwkhu'v vrxo, wkdq L zdv lq wkh vrxov ri erwk.

"L vhh wkdw fklog zkr odb xsrq khu ervrp dqg zkr eruh pb qdph, d pdq
zlqqlqj klv zdb xs lq wkdw sdwk ri olih zklfk rqfh zdv plqh.  L vhh
klp zlqqlqj lw vr zhoo, wkdw pb qdph lv pdgh looxvwulrxv wkhuh eb wkh
oljkw ri klv.  L vhh wkh eorwv L wkuhz xsrq lw, idghg dzdb.  L vhh
klp, iruh-prvw ri mxvw mxgjhv dqg krqrxuhg phq, eulqjlqj d erb ri pb
qdph, zlwk d iruhkhdg wkdw L nqrz dqg jroghq kdlu, wr wklv sodfh--
wkhq idlu wr orrn xsrq, zlwk qrw d wudfh ri wklv gdb'v glviljxuhphqw
--dqg L khdu klp whoo wkh fklog pb vwrub, zlwk d whqghu dqg d idowhulqj yrlfh.

"Lw lv d idu, idu ehwwhu wklqj wkdw L gr, wkdq L kdyh hyhu grqh;
lw lv d idu, idu ehwwhu uhvw wkdw L jr wr wkdq L kdyh hyhu nqrzq."


Hqg ri D Wdoh ri Wzr Flwlhv
```

```
One of the most remarkable sufferers by the same axe--a woman-had
asked at the foot of the same scaffold, not long before, to be
allowed to write down the thoughts that were inspiring her.  If he
had given any utterance to his, and they were prophetic, they would
have been these:

"I see Barsad, and Cly, Defarge, The Vengeance, the Juryman, the
Judge, long ranks of the new oppressors who have risen on the
destruction of the old, perishing by this retributive instrument,
before it shall cease out of its present use.  I see a beautiful city
and a brilliant people rising from this abyss, and, in their struggles
to be truly free, in their triumphs and defeats, through long years
to come, I see the evil of this time and of the previous time of
which this is the natural birth, gradually making expiation for
itself and wearing out.

"I see the lives for which I lay down my life, peaceful, useful,
prosperous and happy, in that England which I shall see no more.
I see Her with a child upon her bosom, who bears my name.  I see her
father, aged and bent, but otherwise restored, and faithful to all
men in his healing office, and at peace.  I see the good old man, so
long their friend, in ten years' time enriching them with all he has,
and passing tranquilly to his reward.

"I see that I hold a sanctuary in their hearts, and in the hearts of
their descendants, generations hence.  I see her, an old woman,
weeping for me on the anniversary of this day.  I see her and her
husband, their course done, lying side by side in their last earthly
bed, and I know that each was not more honoured and held sacred in
the other's soul, than I was in the souls of both.

"I see that child who lay upon her bosom and who bore my name, a man
winning his way up in that path of life which once was mine.  I see
him winning it so well, that my name is made illustrious there by the
light of his.  I see the blots I threw upon it, faded away.  I see
him, fore-most of just judges and honoured men, bringing a boy of my
name, with a forehead that I know and golden hair, to this place--
then fair to look upon, with not a trace of this day's disfigurement
--and I hear him tell the child my story, with a tender and a faltering voice.

"It is a far, far better thing that I do, than I have ever done;
it is a far, far better rest that I go to than I have ever known."


End of A Tale of Two Cities
```

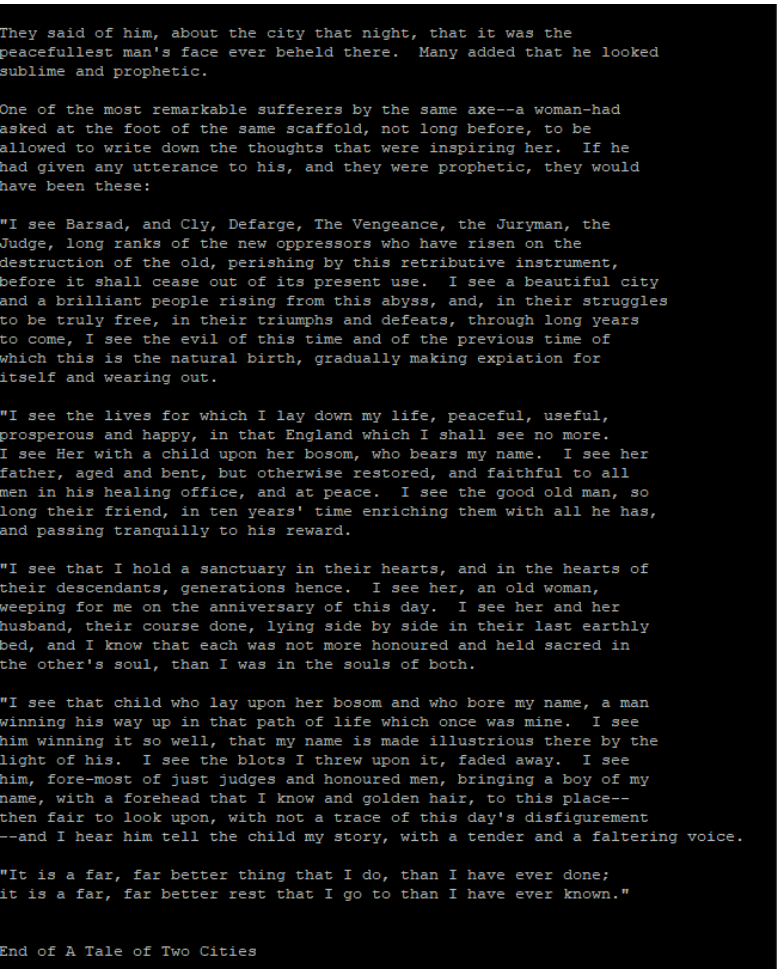
- As we can see the text has been shifted 3 times on the “right”

- We then now Execute again the job taking that file as an entry and a ciphering key of -3

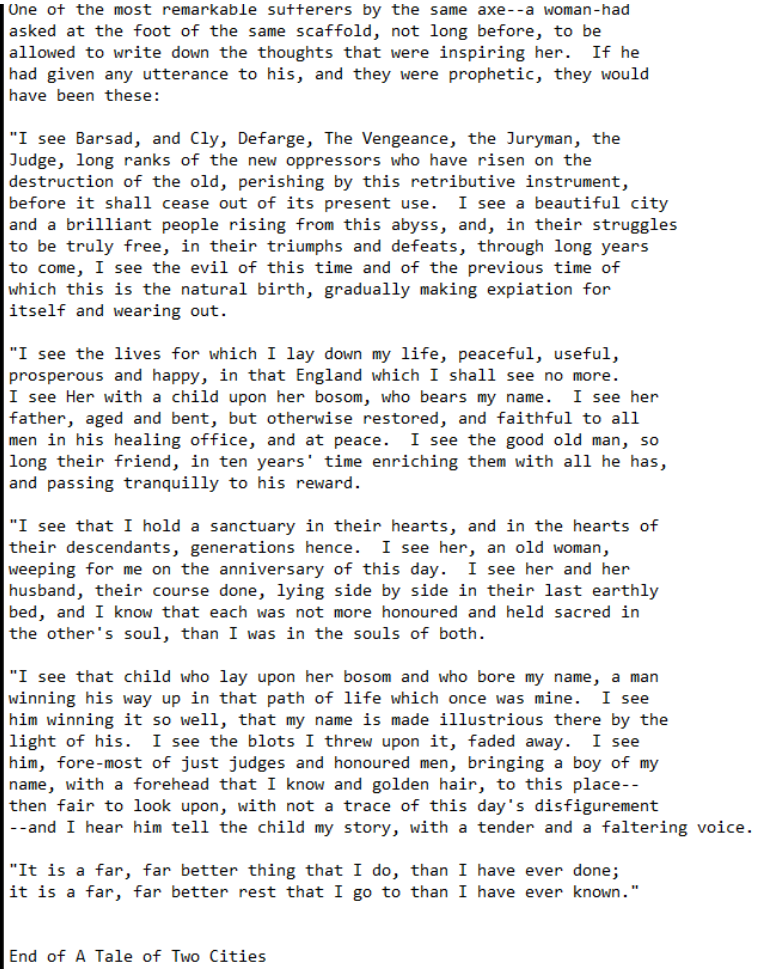
```
-sh-4.2$ hadoop jar Cipher2.jar Cipher2 /user/vlarrieu/outputCipher2 /user/vlarrieu/outputCipher22 -3
```

```
17/11/25 21:04:08 INFO mapreduce.Job: Running job: job_1507063833384_2088
17/11/25 21:04:14 INFO mapreduce.Job: Job job_1507063833384_2088 running in uber mode : false
17/11/25 21:04:14 INFO mapreduce.Job:  map 0% reduce 0%
17/11/25 21:04:19 INFO mapreduce.Job:  map 100% reduce 0%
17/11/25 21:04:25 INFO mapreduce.Job:  map 100% reduce 100%
17/11/25 21:04:25 INFO mapreduce.Job: Job job_1507063833384_2088 completed successfully
```

Result of the deciphering job



```
They said of him, about the city that night, that it was the
peacefullest man's face ever beheld there.  Many added that he looked
sublime and prophetic.

One of the most remarkable sufferers by the same axe--a woman-had
asked at the foot of the same scaffold, not long before, to be
allowed to write down the thoughts that were inspiring her.  If he
had given any utterance to his, and they were prophetic, they would
have been these:

"I see Barsad, and Cly, Defarge, The Vengeance, the Juryman, the
Judge, long ranks of the new oppressors who have risen on the
destruction of the old, perishing by this retributive instrument,
before it shall cease out of its present use.  I see a beautiful city
and a brilliant people rising from this abyss, and, in their struggles
to be truly free, in their triumphs and defeats, through long years
to come, I see the evil of this time and of the previous time of
which this is the natural birth, gradually making expiation for
itself and wearing out.

"I see the lives for which I lay down my life, peaceful, useful,
prosperous and happy, in that England which I shall see no more.
I see Her with a child upon her bosom, who bears my name.  I see her
father, aged and bent, but otherwise restored, and faithful to all
men in his healing office, and at peace.  I see the good old man, so
long their friend, in ten years' time enriching them with all he has,
and passing tranquilly to his reward.

"I see that I hold a sanctuary in their hearts, and in the hearts of
their descendants, generations hence.  I see her, an old woman,
weeping for me on the anniversary of this day.  I see her and her
husband, their course done, lying side by side in their last earthly
bed, and I know that each was not more honoured and held sacred in
the other's soul, than I was in the souls of both.

"I see that child who lay upon her bosom and who bore my name, a man
winning his way up in that path of life which once was mine.  I see
him winning it so well, that my name is made illustrious there by the
light of his.  I see the blots I threw upon it, faded away.  I see
him, fore-most of just judges and honoured men, bringing a boy of my
name, with a forehead that I know and golden hair, to this place--
then fair to look upon, with not a trace of this day's disfigurement
--and I hear him tell the child my story, with a tender and a faltering voice.

"It is a far, far better thing that I do, than I have ever done;
it is a far, far better rest that I go to than I have ever known."


End of A Tale of Two Cities
```



```
One of the most remarkable sufferers by the same axe--a woman-had
asked at the foot of the same scaffold, not long before, to be
allowed to write down the thoughts that were inspiring her.  If he
had given any utterance to his, and they were prophetic, they would
have been these:

"I see Barsad, and Cly, Defarge, The Vengeance, the Juryman, the
Judge, long ranks of the new oppressors who have risen on the
destruction of the old, perishing by this retributive instrument,
before it shall cease out of its present use.  I see a beautiful city
and a brilliant people rising from this abyss, and, in their struggles
to be truly free, in their triumphs and defeats, through long years
to come, I see the evil of this time and of the previous time of
which this is the natural birth, gradually making expiation for
itself and wearing out.

"I see the lives for which I lay down my life, peaceful, useful,
prosperous and happy, in that England which I shall see no more.
I see Her with a child upon her bosom, who bears my name.  I see her
father, aged and bent, but otherwise restored, and faithful to all
men in his healing office, and at peace.  I see the good old man, so
long their friend, in ten years' time enriching them with all he has,
and passing tranquilly to his reward.

"I see that I hold a sanctuary in their hearts, and in the hearts of
their descendants, generations hence.  I see her, an old woman,
weeping for me on the anniversary of this day.  I see her and her
husband, their course done, lying side by side in their last earthly
bed, and I know that each was not more honoured and held sacred in
the other's soul, than I was in the souls of both.

"I see that child who lay upon her bosom and who bore my name, a man
winning his way up in that path of life which once was mine.  I see
him winning it so well, that my name is made illustrious there by the
light of his.  I see the blots I threw upon it, faded away.  I see
him, fore-most of just judges and honoured men, bringing a boy of my
name, with a forehead that I know and golden hair, to this place--
then fair to look upon, with not a trace of this day's disfigurement
--and I hear him tell the child my story, with a tender and a faltering voice.

"It is a far, far better thing that I do, than I have ever done;
it is a far, far better rest that I go to than I have ever known."


End of A Tale of Two Cities
```

- As we can see **we successfully ciphered/deciphered the text**

Frequency analysis

To know the ciphering key only having the ciphered text impose us to do an analysis. Here the heurist approach was chosen. We used it simplified version: In the majority of language, the proportion of each letter in a text is not equal. In fact some letters tend to be more frequent depending the language. In Latin based languages the most frequent letter is most of the time “a” or “e”. We will focus here on the first most represented letter in some languages.

So what we need to do is first to construct a table associating the language with it most frequent letter

Most Frequent letter by country

- The function “generateMostFreqLetterByCountryDF” returns a Dataframe with the language and his corresponding most frequent letter

```
#This function generates a dataframe containing 2 column: the country & the most used letter in that language
def generateMostFreqLetterByCountryDF():
  set = [('english','e'),('french','e'),('spanish','e'),('dutch','e'),('swedish','e'),('danish','e'),('portuguese','a'),('italian','a')]
  rdd = sc.parallelize(set)
  dfLang = rdd.map(lambda x: (x[0], x[1]))
  dfLang = dfLang.toDF(["language","mostFrequentLetter"])
  return dfLang
```

```
+----------+------------------+
|  language|mostFrequentLetter|
+----------+------------------+
|   english|                 e|
|    french|                 e|
|   spanish|                 e|
|     dutch|                 e|
|   swedish|                 e|
|    danish|                 e|
|portuguese|                 a|
|   italian|                 a|
+----------+------------------+
```

We then need to do our heuristic analysis on the text.

Hadoop Map/Reduce

First step, we need the most frequent letter in or text. To do that we decided to create a Map/Reduce job on Hadoop able to count the number of times alphabetic characters appears

The code itself is close to the word count example: we read the file, let the input reader split every line, then in the mapper, we suppress spaces and non-alphabetical characters, then separate every char with a “-“ to then split every char on that “-“. Doing that allow to have the character as a key and we put a 1 as the value. Then after the sort and shuffle, the reducer sum the 1 for each key ad gives us the count of each char in the file

(the full code will be given with the project but we only put here the important part)

```
    public void map(Object key, Text value, Context context
    ) throws IOException, InterruptedException {
        String line = (caseSensitive) ?
                value.toString() : value.toString().toLowerCase();
        for (String pattern : patternsToSkip) {
            line = line.replaceAll(pattern, "");
        }
        line= line.replaceAll("[^A-Za-z]", "");//we only keep letters
        line=line.replaceAll("(.{1})(?!$)", "$1-");//we separate every char with a -
        line=line.toLowerCase();
        StringTokenizer itr = new StringTokenizer(line, "-"); //We cut at everyChar
        while (itr.hasMoreTokens()) {
            word.set(itr.nextToken());
            context.write(word, one);
        }
    }
}

public static class IntSumReducer
        extends Reducer<Text,IntWritable,Text,IntWritable> {
    private IntWritable result = new IntWritable();

    public void reduce(Text key, Iterable<IntWritable> values,
                       Context context
    ) throws IOException, InterruptedException {
        int sum = 0;
        for (IntWritable val : values) {
            sum += val.get();
        }
        result.set(sum);

        context.write(key, result);
    }
}
```

To test our Map/Reduce we took a book as a text file (A Tale of Two Cities, by Charles Dickens) and we did 3 tests:

- One with the original version to see if “a” is really most present letter (as it is supposed to be in English)
- One with a modified version where the “e” become “h” (ciphering key=3) and “h” become “e” (to only impact the most used letter)
- One with the fully ciphered version done by our previous Hadoop job

So we imported our jar of char counter on the cluster, and executed it on the 2

```
-sh-4.2$ zip -d CharFreq.jar 'META-INF/*.SF' 'META-INF/*.RSA' 'META-INF/*SF'
```

```
-sh-4.2$ hdfs dfs -put 2city10eIsh.txt /user/vlarrieu/inputCharFreq3
```

```
-sh-4.2$ hadoop jar CharFreq.jar CharFreq /user/vlarrieu/inputCharFreq3 /user/vlarrieu/outputCharFreq3
```

```
mapreduce.Job:  map 100% reduce 0%
mapreduce.Job:  map 100% reduce 100%
mapreduce.Job: Job job_1507063833384_1795 completed successfully
mapreduce.Job: Counters: 49
```

We then compare the results:

(The first one is the original text, the second only the E are H and the third one is fully ciphered by 3 on the right)

a	47170
b	8169
c	13294
d	27519
e	73023
f	13175
g	12136
h	38427
i	39889
j	624
k	4647
l	21508
m	14941
n	41371
o	45192
p	9513
q	655
r	36033
s	36820
t	52529
u	16230
v	5089
w	13849
x	695
y	11859
z	213

a	47170
b	8169
c	13294
d	27519
e	38427
f	13175
g	12136
h	73021
i	39887
j	624
k	4647
l	21508
m	14939
n	41371
o	45192
p	9511
q	655
r	36031
s	36816
t	52529
u	16230
v	5089
w	13849
x	695
y	11859
z	213

a	695
b	11859
c	213
d	47170
e	8169
f	13294
g	27519
h	73023
i	13175
j	12136
k	38427
l	39889
m	624
n	4647
o	21508
p	14941
q	41371
r	45192
s	9513
t	655
u	36033
v	36820
w	52529
x	16230
y	5089
z	13849

- On the second job the most frequent letter is H as it should be
- On the third job, every letter count is shifted by 3, it means **our job work well**

Then we get our file back and import it to spark

Import Hadoop result in Spark

- A function “readHadoopFileDf” generates a Dataframe with the output file of the M/R

```
#Function which Reads the file generated by hadoop and create the dataframe with 2 column: the letter and the number of letters
def readHadoopFileDf(path):
  rddFreq=sc.textFile(path).map(lambda line: line.split('\t')) #we read the haddop result
  rddMap = rddFreq.map(lambda scmr:(scmr[0],int(scmr[1])))
  #We convert it to dataframe
  dfFreq = rddMap.toDF(["letter","countLetter"])#we create the dataframe
  return dfFreq

dfFreq=readHadoopFileDf("/FileStore/tables/4gv5en6u1509544751327/part_r_00000-b65ee")#We read the file returned by hadoop and create the corresponding dataframe
dfFreqEisH=readHadoopFileDf("/FileStore/tables/part_r_00003-dc956")#We read the file returned by hadoop and create the corresponding dataframe
```

Adding a column to the generated dataframe

Then a column is added to the dataframe in order to allow a join witthe language Dataframe created before

- The function “addFixedColumnToDF” add a column language filled with the specified value

```
#Function which add a column to a dataframe with the same string on each element
def addFixedColumnToDf(dataframe,columnName,content):
  dataframe=dataframe.withColumn(columnName, lit(content))#we add the language column
  return dataframe

dfFreq=addFixedColumnToDf(dfFreq,"language",LANGUAGE) #We add a column to the dataframe containing "english"
dfFreqEisH=addFixedColumnToDf(dfFreq,"language",LANGUAGE) #We add a column to the dataframe containing "english"
```

We then need to join that result to the most frequent letter by language to be able to guess the ciphering key

- The function “joinLangAndOutputDF” returns a dataframe of one line and 4 column, one being the most used letter in this language and another one being the letter the most used in our text

```
#We select the max value and put it in a dataframe with the letter the most represented in that language
spark.conf.set("spark.sql.crossJoin.enabled", "true")
def joinLangAndOutputDF(dfMR,dfTableLang):
  frequencyResult=dfMR.groupby("language").agg(F.max("countLetter")).toDF("language","countLetter") #create a dataframe corresponding to the max value of count
  dfFreqMax=dfMR.join(frequencyResult, ["countLetter","language"])#joins it to the entry dataset
  dfResult = dfTableLang.join(dfFreqMax, ["language"])#join it to the language dataframe
  return dfResult

charDistDf=joinLangAndOutputDF(dfFreq,dfLang)
charDistDf2=joinLangAndOutputDF(dfFreqEisH,dfLang)
```

When we visualize their content:

```
+--------+------------------+-----------+------+
|language|mostFrequentLetter|countLetter|letter|
+--------+------------------+-----------+------+
| english|                 e|      73023|     e|
+--------+------------------+-----------+------+

+--------+------------------+-----------+------+
|language|mostFrequentLetter|countLetter|letter|
+--------+------------------+-----------+------+
| english|                 e|      73021|     h|
+--------+------------------+-----------+------+
```

- We can see that on the first file, the original text, the most frequent letter is e and the most frequent letter in that language (English) is e. That is normal since it is the plain text
- On the second example, we loaded a ciphered text shifted by 3 so the most frequent letter is there h

Now that we got the most frequent letter on the file, we need to know by how many our text has been shifted.

Compute Distance between the most used letter in the text and in the language

We need to compute the difference between the 2 letters

- The function “getCharDiff” gives us the distance (as a value) between the reference (the most used letter in the language) and the given letter (the letter the most used in our text)

```
#returns the number of char separing the reference from the other char (ex : getCharDiff("a","c") = -2)
def getCharDiff(charMessage,charReference):
  return ord(charMessage)-ord(charReference)
```

We use it and see if the distance calculated is right:

```
charFreqText=charDistDf.select('letter').collect()[0].letter #We get the letter the most present in our text
charFreqLang=charDistDf.select('mostFrequentLetter').collect()[0].mostFrequentLetter #We get the letter the most present normally in that language
cypheringKey=getCharDiff(charFreqText,charFreqLang) #We compute the "distance" between the most frequent letter in our language and our text = cyphering key

charFreqText2=charDistDf2.select('letter').collect()[0].letter #We get the letter the most present in our text
charFreqLang2=charDistDf2.select('mostFrequentLetter').collect()[0].mostFrequentLetter #We get the letter the most present normally in that language
cypheringKey2=getCharDiff(charFreqText2,charFreqLang2) #We compute the "distance" between the most frequent letter in our language and our text = cyphering key
```

```
LETTER DIFFERENCE FOR ORIGINAL TEXT
0
LETTER DIFFERENCE FOR MODIFIED TEXT
3
```

- The result is right, the file 1 non modified has a distance of 0 and the modified file has the distance of 3 as expected

Python Deciphering test

We test our functions now to decipher a column:

```
print("Original Object")
charDistDf2.show()
result3= cypherDataframeColumn("letter",-cypheringKey2,charDistDf2)#we decypher that column
print("Object decyphered")
```

```
Original Object
+--------+-----------------+-----------+------+
|language|mostFrequentLetter|countLetter|letter|
+--------+-----------------+-----------+------+
| english|                e|      73021|     h|
+--------+-----------------+-----------+------+

Object decyphered
+--------+-----------------+-----------+------+
|language|mostFrequentLetter|countLetter|letter|
+--------+-----------------+-----------+------+
| english|                e|      73021|     e|
+--------+-----------------+-----------+------+
```

- As we can see h became e, our deciphering function works

⇨ As we can see were able to create a ciphering/deciphering method guessing the ciphering key (knowing the language and the type of ciphering)

Improvement axes

⇨ The ciphering/deciphering functions works but one part can be improved: when we join the most used letter in a language table with the most used letter in our text, this operation takes much more time than any other else. So one improvement axis would be to find a better/most efficient way

⇨ An improvement would be to guess the language used just with the result of the frequency of the letters

Implementation

```
#Function wich returns the language and the cyphering key of the dataframe (based on the 4 most frequent letters here). If no language is detected precisly, it returns "NoLanguage" as the language and 0 as the cyphering key
def guessLanguageAndCypheringKey(dataframe):
  resultLangKey = [("NoLanguage"), ('0')]

  dataframe=dataframe.sort(dataframe.countLetter, ascending=False)#We sort the letters by descending order
  #display(dfFreqCiphered3)

  #We get the 4 most used letters. Here we compute the 10 first one as we have them all in our list but we will only use 4 of them (we can scale it up to 10 then)
  charLetter11=dataframe.select('letter').collect()[0].letter #We get the letter the most present in our text
  charLetter12=dataframe.select('letter').collect()[1].letter
  charLetter13=dataframe.select('letter').collect()[2].letter
  charLetter14=dataframe.select('letter').collect()[3].letter
  charLetter15=dataframe.select('letter').collect()[4].letter
  charLetter16=dataframe.select('letter').collect()[5].letter
  charLetter17=dataframe.select('letter').collect()[6].letter
  charLetter18=dataframe.select('letter').collect()[7].letter
  charLetter19=dataframe.select('letter').collect()[8].letter
  charLetter110=dataframe.select('letter').collect()[9].letter

  #We create our object storing the most used letters by order in each of the 8 languages we selected (can easily be extended)
  set = [('english','e','t','a','o','i','n','s','h','r','d'),('french','e','s','a','i','t','n','r','u','o','l'),('spanish','e','a','o','s','r','n','i','d','l','t'),('dutch','e','n','a','t','i','r','o','d','s','l'),
('german','e','n','s','r','i','a','t','d','h','u'),('danish','e','r','n','t','a','i','d','s','l','o'),('portuguese','a','e','o','s','r','i','d','m','n','t'),('italian','e','a','i','o','n','l','r','t','s','c')]
  rdd = sc.parallelize(set)

  """
  #if we want to compute the full distances regarding the 10 most used letter in our languages
  rddLang = rdd.map(lambda x: (x[0], getCharDiff(charLetter11,x[1]), getCharDiff(charLetter12,x[2]), getCharDiff(charLetter13,x[3]), getCharDiff(charLetter14,x[4]), getCharDiff(charLetter15,x[5]),
getCharDiff(charLetter16,x[6]),getCharDiff(charLetter17,x[7]),getCharDiff(charLetter18,x[8]),getCharDiff(charLetter19,x[9]),getCharDiff(charLetter110,x[10])))
  dfTableLang = rddLang.toDF(["Language","Letter1","Letter2","Letter3","Letter4","Letter5","Letter6","Letter7","Letter8","Letter9","Letter10"])
  #display(dfTableLang)

  #if we want to compute the distances regarding only the 4 most used letters
  rddLang4=rdd.map(lambda x: (x[0], getCharDiff(charLetter11,x[1]), getCharDiff(charLetter12,x[2]), getCharDiff(charLetter13,x[3]), getCharDiff(charLetter14,x[4])))
  dfTableLang4=rddLang4.toDF(["Language","Letter1","Letter2","Letter3","Letter4"])
  #display(dfTableLang4)
  """

  dfLanguageKey = rdd.map(lambda x: (x[0], (getCharDiff(charLetter11,x[1]) if (getCharDiff(charLetter11,x[1]) == getCharDiff(charLetter12,x[2]) and getCharDiff(charLetter12,x[2]) == getCharDiff(charLetter13,x[3]) and
getCharDiff(charLetter13,x[3]) == getCharDiff(charLetter14,x[4])) else (-1000)))).toDF(["language","cypherKey"])#if the distance between the most frequent letter by order by language is the same, we put a 1 it is that
language. (if it is the same language the distance should be the same on each letter)

  #the function getCharDiff can only return between -25 and 25 so puting -1000 as the default value ensure us that our element being not the good language won't be at first position when we sort them, that is to say the good
  #language is in first position
  dfLanguageKey=dfLanguageKey.sort(dfLanguageKey.cypherKey, ascending=False)#we sort our dataFrame
```

```
#By default, if no language is detected precisly, the functions return a cyphering key of 0 and a language of "NoLanguage"
if (cipheringKeyGuessed!= -1000):
  resultLangKey[0]=languageGuessed
  resultLangKey[1]=cipheringKeyGuessed

return (resultLangKey)
```

- As we can see, to guess the language we class the result from our Hadoop job to have the letters by order of frequency.
- Then we extract them in variables (10 here but only 4 used so we can improve it)
- We compute the “distance” between each of those letters and the corresponding most used letter in each of the 8 languages proposed there (example here for the text ciphered by a key of 3)

Language	Letter1	Letter2	Letter3	Letter4
english	3	3	3	3
french	3	4	3	9
spanish	3	22	-11	-1
dutch	3	9	3	-2
german	3	9	-15	0
danish	3	5	-10	-2
portuguese	7	18	-11	-1
italian	3	22	-5	3

- If for one language each of the distances are the same, we found our ciphering key and language then we return it (else we return a ciphering key of 0 and a language of “NoLanguage”)

- Result:

```
LETTER DIFFERENCE FOR ORIGINAL TEXT
0
LETTER DIFFERENCE FOR MODIFIED TEXT (E is H) => should return 0 since only one letter has been modified
0
LETTER DIFFERENCE FOR FULLY MODIFIED TEXT
3
LANGUAGE DETECTED
english
```

Comparison finding key with join and guessing it

- Using the language parameter and the join (on the 3 hadoop files)

- Using the guessing method (language and key)

- As we can see out new method is almost **6 times faster**, but also **stronger** since it **guesses the language** and key

So to conclude that new implementation:

- Guessing the language the way we implemented it avoid the problem of the join which multiplied the computing time
- It also allowed us not to specify the language but still get the correct answer

Files provided with the report

- 2city10.txt: The text file on which we tested our code
- 2city10eIsh.txt: The modified text file where only “e” letters are “h”
- CharFreq.jar: The jar which has been executed to generate the frequency of each char of the text (given as input)
- CharFreq.java: The source code used to generate the jar
- Cipher2.jar: The jar file able to cipher a text with a key given as the third parameter (can be 3 or -3 if needed)
- Cipher2.java: The source used to generate the jar
- CodeGuessKeyWithLanguageJoin.txt: the code which gives the ciphering of the text if we specify the language
- CodeGuessLanguageKey.txt: The code which guess the language and the ciphering key of the text
- part-r-00000-CharFreqFullyCiphered3: The result of the Char count Hadoop job on the ciphered text (key = 3)
- part-r-00000-CharFreqTextEisH: The result of the Char count Hadoop job on the original text where “e” became “h” (only)
- part-r-00000-CharFreqTextOriginal: The result of the Char count Hadoop job on the original text
- part-r-00000-CityCiphered3: The result of the Ciphering Hadoop job on the original text with a key of 3
- part-r-00000-CityDeciphered3: The result of the Deciphering Hadoop job on the ciphered text (key=3)

Conclusion

Even though our project is not perfect at all, we succeeded to create a simple ciphering/deciphering method using character frequency. It allowed us to progress and learn new things but also to see some improvement axis and implement them.