UNIVERSIDADE FEDERAL DO CEARÁ

DESENVOLVIMENTO DE SOFTWARE PARA NUVEM

DEPARTAMENTO DE COMPUTAÇÃO

RELATÓRIO DO TRABALHO PRÁTICO DE HADOOP

Davi Almeida da Mota

UNIVERSIDADE FEDERAL DO CEARÁ

DEPARTAMENTO DE COMPUTAÇÃO

30 de Setembro de 2020

1 Ambiente de desenvolvimento

Os algoritmos utilizados para solucionar cada questão foram realizados na IDE Eclipse 2020, a máquina utilizada para a execução dos algoritmos possui as seguintes configurações: Sistema Operacional Linux ubuntu 18, 8GB de memória RAM, 120GB de armazenamento no SSD e um processador intel core i5.

Figure 1: logo da IDE Eclipse 2020

No que diz respeito a configuração do Hadoop, os algoritmos desenvolvidos foram compilados diretamente na IDE eclipse.

Com a finalidade de demonstrar a reprodutibilidade do experimento, os algoritmos desenvolvidos e os resultados brutos obtidos se encontram disponíveis no link do github https://github.com/davimota-dev/TP_Hadoop_UFC.git

Os dados brutos de resposta foram manipulados através de uma combinação do software regexr e de planilhas, de forma a buscar as estatísticas desejadas para responder as respectivas questões.

2 Questões

Nessa seção serão apresentadas as questões da lista.

Questão 1 Dado um dataset de arquivos de textos com o conteúdo de vários livros, existente no link: https://github.com/davimota-dev/TP_Hadoop_UFC.git ,deseja-se distribuir a tarefa para se descobrir:

A: Qual ou quais são as palavras com maior número de letras? Para esse item em específico, foi-se construído o algoritmo:

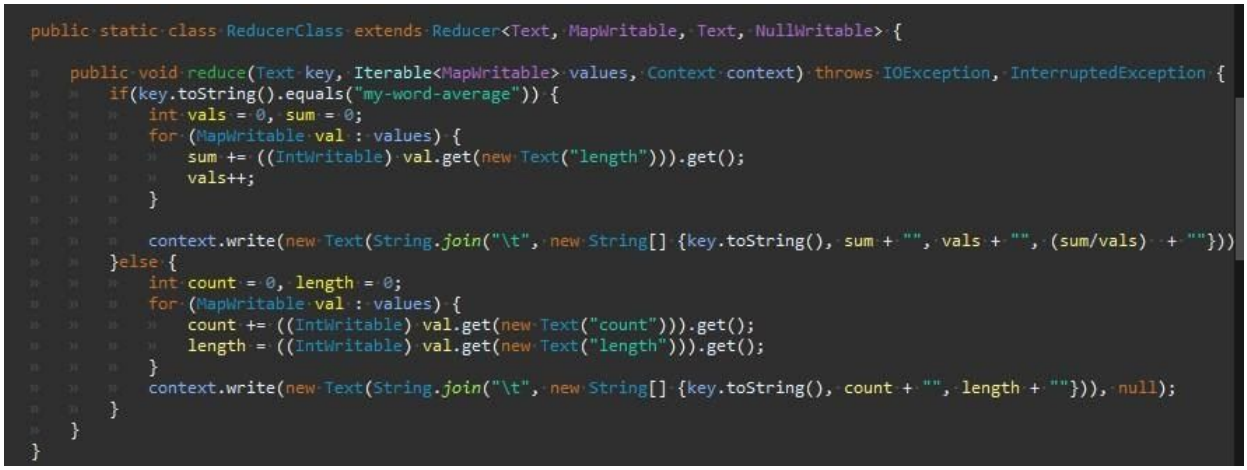
```
public static class MapperClass extends Mapper<Object, Text, Text, MapWritable> {

    public void map(Object key, Text data, Context context) throws IOException, InterruptedException {
        StringTokenizer itr = new StringTokenizer(data.toString());
        while (itr.hasMoreTokens()) {
            MapWritable map = new MapWritable();
            MapWritable map2 = new MapWritable();

            String token = itr.nextToken().trim();
            map.put(new Text("count"), new IntWritable(1));
            map.put(new Text("length"), new IntWritable(token.length()));
            context.write(new Text(token), map);

            map2.put(new Text("length"), new IntWritable(token.length()));
            context.write(new Text("my-word-average"), map2);
        }
    }
}
```

Figure 2: Algoritmo montado para resolução da questão 1- A mapper como mostrado na figura 2, o mapper criado funciona associando as palavras a um mapa de acordo com as duas caracterısticas analisadas a pedido da questão: essas são a repetição dessas palavras e seu tamanho. Dessa forma foi possível responder às duas perguntas da primeira questão de uma vez só.



```
public static class ReducerClass extends Reducer<Text, MapWritable, Text, NullWritable> {

    public void reduce(Text key, Iterable<MapWritable> values, Context context) throws IOException, InterruptedException {
        if(key.toString().equals("my-word-average")) {
            int vals = 0, sum = 0;
            for (MapWritable val : values) {
                sum += ((IntWritable) val.get(new Text("length"))).get();
                vals++;
            }

            context.write(new Text(String.join("\t", new String[] {key.toString(), sum + "", vals + "", (sum/vals) + ""})));
        }else {
            int count = 0, length = 0;
            for (MapWritable val : values) {
                count += ((IntWritable) val.get(new Text("count"))).get();
                length = ((IntWritable) val.get(new Text("length"))).get();
            }
            context.write(new Text(String.join("\t", new String[] {key.toString(), count + "", length + ""})), null);
        }
    }
}
```

Figure 3: Algoritmo montado para resolução da questão 1-A reducer

Já o reducer, como mostrado na figura 3, realizou a soma das palavras

catalogadas como iguais, além de preservar o atributo tamanho, responsável por exprimir o número de caracteres constituintes de cada palavra.

A **resposta** da primeira questão item A: dentre as palavras em português, as maiores encontradas foram magnificentissimamente (22 caracteres) e inconstitucionalidade (21 caracteres).

B: Qual a média do tamanho das palavras?

A **resposta** da segunda questão item B: o tamanho médio registrado foi de 4,70 caracteres por palavras).

Questão 2: Dado um dataset com tweets relacionados a campanha eleitoral presidencial de 2014, encontrado no link:https://github.com/davimota-dev/TP_Hadoop_UFC.git , responda:

A: Quais foram as hashtags mais usadas pela manhã, tarde e noite? Para esse item em especıfico, foi-se construıdo o algoritmo:

```
public class HashTagCountByPeriod {

    public static class MapperClass extends Mapper<Object, Text, Text, MapWritable> {
        public void map(Object key, Text data, Context context) throws IOException, InterruptedException {
            Tweet tweet = ReadTSV.parse(data.toString());
            ArrayList<String> tags = tweet.getHashTags();
            for (String tag : tags) {
                MapWritable map = new MapWritable();
                map.put(new Text("morning"), (tweet.isMorning()) ? new IntWritable(1) : new IntWritable(0));
                map.put(new Text("afternoon"), (tweet.isAfternoon()) ? new IntWritable(1) : new IntWritable(0));
                map.put(new Text("night"), (tweet.isNight()) ? new IntWritable(1) : new IntWritable(0));
                context.write(new Text(tag), map);
            }
        }
    }
```

Figure 4: Algoritmo montado para resolução da questão 2-A mapper

```
public static class ReducerClass extends Reducer<Text, MapWritable, Text, NullWritable> {
    public void reduce(Text key, Iterable<MapWritable> values, Context context) throws IOException, InterruptedException {
        int morning = 0, afternoon = 0, night = 0;
        for (MapWritable val : values) {
            morning   += ((IntWritable) val.get(new Text("morning"))).get();
            afternoon += ((IntWritable) val.get(new Text("afternoon"))).get();
            night     += ((IntWritable) val.get(new Text("night"))).get();
        }
        context.write(new Text(key + "\t" + morning + "\t" + afternoon + "\t" + night), null);
    }
}
```

Figure 5: Algoritmo montado para resolução da questão 2-A reducer

Como mostrados nas figuras 4 e 3, o procedimento realizado para a solução da segunda questão item A, foi-se criado um mapa para dividir os tweets em grandes grupos de acordo com o seu horário de registro, isto é, nos horários de manhã(00:00 - 11:59), tarde (12:00 - 17:59) e noite (18:00 - 23: 59).

A **resposta** para a segunda questão item B é: os top 3 tweets mais utilizados:

- No turno da manhã foram EMABiggestFansJustinBieber(94475 tweets), EMABiggestFans1D(88214 tweets) e , camilasayshi(9946 tweets).

- No turno da tarde foram EMABiggestFans1D(57711 tweets), EMABiggest-FansJustinBieber(48951 tweets) e QueroNoTVZ(4430 tweets).

- No turno da noite foram EMABiggestFans1D(64075 tweets), EMABiggest-FansJustinBieber(54028 tweets) e StealMyGirl(4816 tweets).

B: Quais as hashtags mais usadas em cada dia?

para esse item em especıfico, foi-se construıdo o algoritmo:

```
public static class MapperClass extends Mapper<Object, Text, Text, Text> {
    public void map(Object key, Text data, Context context) throws IOException, InterruptedException {
        Tweet tweet = ReadTSV.parse(data.toString());
        ArrayList<String> tags = tweet.getHashTags();
        for (String tag : tags) {
            context.write(new Text(tag), new Text(tweet.getFormattedDate()));
        }
    }
}
```

Figure 6: Algoritmo monstado para a resolução da questão 2-B mapper

```
public static class ReducerClass extends Reducer<Text, Text, Text, NullWritable> {
    public void reduce(Text key, Iterable<Text> values, Context context) throws IOException, InterruptedException {
        HashMap<String, Integer> map = new HashMap<>();
        for (Text val : values) {
            String mkey = val.toString();
            if(map.containsKey(mkey)) {
                map.put(mkey, map.get(mkey) + 1);
            }else {
                map.put(mkey, 1);
            }
        }
        for(String mkey : map.keySet()) {
            context.write(new Text(key + "\t" + mkey + "\t" + map.get(mkey)), null);
        }
    }
}
```

Figure 7: Algoritmo montado para a resolução da questão 2-B reducer

Para a segunda questão item, os mapper e o reducer utilizados se encontram ilustrados nas figuras 6 e 7 respectivamente. O mapper em conjunto com o reducer nessa quest˜ao, separam e em seguida realizam a soma dos tweets de acordo com sua data de registro.

A **resposta** para a segunda questão item B, as hashtags mais usadas a cada dia são:

- Dia 15/10 - EMABiggestFans1D
- Dia 16/10 - EMABiggestFans1D
- Dia 17/10 - EMABiggestFansJusnBieber
- Dia 18/10 - EMABiggestFans1D
- Dia 19/10 - EMABiggestFansJusnBieber
- Dia 20/10 - EMABiggestFansJusnBieber

C: Qual o número de tweets por hora a cada dia?

Para esse item em específico, foi-se construído o algoritmo:

```
public static class MapperClass extends Mapper<Object, Text, Text, IntWritable> {
    public void map(Object key, Text data, Context context) throws IOException, InterruptedException {
        Tweet tweet = ReadTSV.parse(data.toString());
        context.write(new Text(tweet.getDayAndHour()), new IntWritable(1));
    }
}
```

Figure 8: Algoritmo montado para resolução da questão 2-C mapper

```
public static class ReducerClass extends Reducer<Text, IntWritable, Text, NullWritable> {
    public void reduce(Text key, Iterable<IntWritable> values, Context context) throws IOException, InterruptedException {
        int total = 0;
        for (IntWritable val : values) {
            total += val.get();
        }
        context.write(new Text(key + "\t" + total), null);
    }
}
```

Figure 9: Algoritmo montado para resolu¸c˜ao da quest˜ao 2-C reducer

O mapper e o reducer da segunda questão item C, ilustrados pelas figuras 6 e 7 atuam de maneira simular aos mapper e reducer da questão 2 B, com o diferencial de que

esses somam o total de tweets mapeados por dia, gerando assim o número total de tweets. Uma pequena amostra da **resposta** do item C da questão 2, o número de tweets por hora a cada dia é:

:

15/10/2014 11:00:00 34380

15/10/2014 12:00:00 79168

15/10/2014 13:00:00 78363

15/10/2014 14:00:00 83956

15/10/2014 15:00:00 77716

15/10/2014 16:00:00 65103

15/10/2014 17:00:00 66820

15/10/2014 18:00:00 79278

15/10/2014 19:00:00 86037

15/10/2014 20:00:00 97589

15/10/2014 21:00:00 110250

15/10/2014 22:00:00 163357

15/10/2014 23:00:00 176225

16/10/2014 00:00:00 124614

16/10/2014 01:00:00 77749

O restante encontra-se presente no link do github:

https://github.com/davimota-dev/TP_Hadoop_UFC.git

D: Quais as principais sentenças relacionadas a palavra ”Dilma”? para esse item em específico, foi-se construído o algoritmo:

```
public static class MapperClass extends Mapper<Object, Text, Text, IntWritable> {
    public void map(Object key, Text data, Context context) throws IOException, InterruptedException {
        Tweet tweet = ReadTSV.parse(data.toString());
        if(tweet.isRelatedDilma()) {
            ArrayList<String> nGrams = tweet.getNGrams();
            for (String ngram : nGrams) {
                context.write(new Text(ngram), new IntWritable(1));
            }
        }
    }
}
```

Figure 10: Algoritmo montado para a resolução da questão 2-D mapper

E: Quais as principais sentenças relacionadas a palavra ”Aécio”? para esse item em específico, foi-se construído o algoritmo:

A atuação dos mappers e reducers para os itens D e E da questão 2, ilustrados pelas figuras 10 e 11 para o item D e 12 e 13 para o item E, são bem semelhante. os tweets foram mapeados por filtros de caracteres de acordo

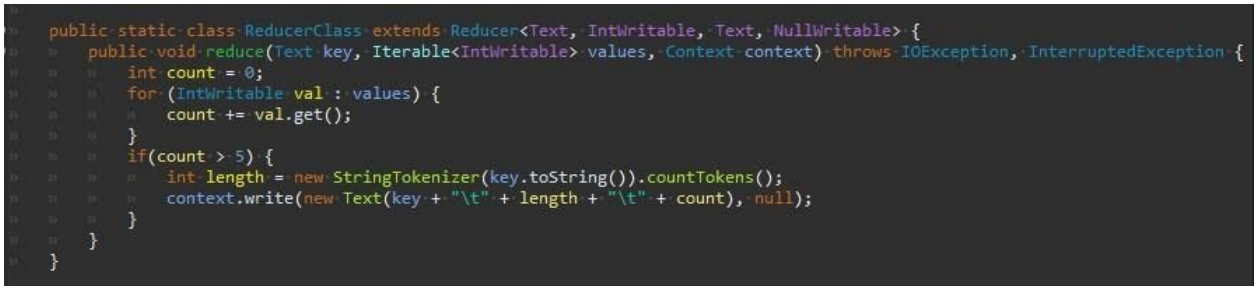


Figure 11: Algoritmo monstado para a resolução da questão 2-D reducer

Figure 12: Algoritmo monstado para a resolução da questão 2-E mapper

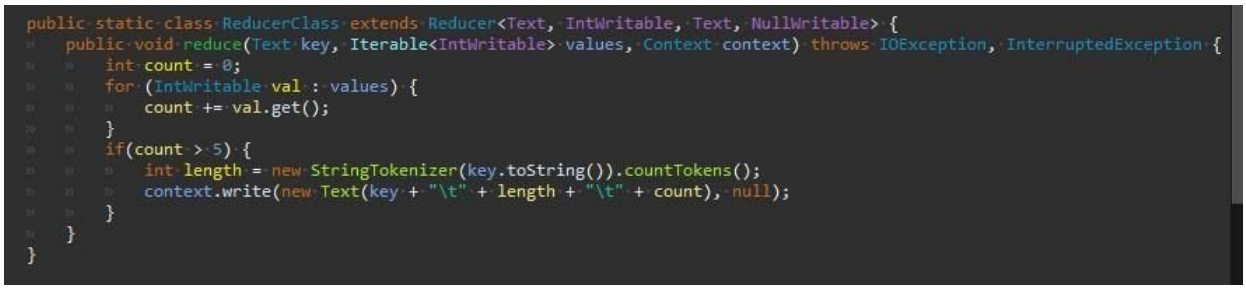


Figure 13: Algoritmo monstado para a resolu¸c˜ao da quest˜ao 2-E reducer

com os nomes e o grau de proximidade dos candidatos, além de que foram filtrados caracteres especiais, URLs e palavras menores que 3 caracteres.

A **resposta** para o item D da questão 2 é: Os tweets mais comuns associados à **Dilma**

são: dilma aécio, aécio dilma, dilma fala, votar dilma, dilma rousself, entre dilma aécio, aprovado pela dilma, cala boca dilma, dilma sabe falar, dilma nocauteada vivo debate, entre dilma aécio prefiro, dilma orientada pelo marqueteiro.

A **resposta** para o item E da questão 2 é:

Os tweets mais comuns associados ao **Aécio** são: aecio neves, votar aécio, minas gerais, aecio ganhar, entre dilma aecio, debates aecio neves, aecio neves a conversa, vivo debate aécio, dilma nocauteada vivo debate, vivo debate aecio neves, aecio neves postos acovardar.

Questão 3: Dado um dataset com tweets relacionados a visita da Torre Eiffel em Paris, encontrado no link:https://github.com/davimota-dev/TP_Hadoop_UFC.git , responda:

A: Encontre as palavras mais usadas nas avaliações.

Para esse item em específico, foi-se construído o algoritmo:

```
public static class MapperClass extends Mapper<Object, Text, Text, IntWritable> {
    public void map(Object key, Text data, Context context) throws IOException, InterruptedException {
        Review review = ReadJson.getReview(data.toString());
        if(review.getText() != null) {
            StringTokenizer tokenizer = new StringTokenizer(review.getCleanedContent());
            while (tokenizer.hasMoreTokens()) {
                String token = tokenizer.nextToken();
                context.write(new Text(token), new IntWritable(1));
            }
        }
    }
}
```

Figure 14: Algoritmo montado para resolução da questão 3-A mapper

```
public static class ReducerClass extends Reducer<Text, IntWritable, Text, NullWritable> {
    public void reduce(Text key, Iterable<IntWritable> values, Context context) throws IOException, InterruptedException {
        int count = 0;
        for (IntWritable val : values) {
            count += val.get();
        }
        context.write(new Text(key + "\t" + count), null);
    }
}
```

Figure 15: Algoritmo montado para resolução da questão 3-A reducer

O mapper e o recuder utilizados para a terceira questão item A podem ser visualizados nas figuras 14 e 15, e atuam como um filtro de palavras chaves, agrupando e em seguida contando as palavras repetidas.

A **resposta** para a terceira questão item A é:
As 8 palavras mais utilizadas nas avaliações da torre eifel são:

- tower(5916 repetições)
- paris(3461 repetições)
- eiffel(3395 repetições)
- there(2524 repetições)
- from(2521 repetições)
- this(2223 repetições)
- time(2154 repetições)
- that(2042 repetições)

B: Encontre as expressões mais usadas.
para esse item em específico, foi-se construído o algoritmo:

```
public static class MapperClass extends Mapper<Object, Text, Text, IntWritable> {
    public void map(Object key, Text data, Context context) throws IOException, InterruptedException {
        Review review = ReadJson.getReview(data.toString());
        if(review.getText() != null) {
            ArrayList<String> nGrams = review.getNGrams();
            for (String ngram : nGrams) {
                context.write(new Text(ngram), new IntWritable(1));
            }
        }
    }
}
```

Figure 16: Algoritmo montado para a resolução da questão 3-B mapper

```
public static class ReducerClass extends Reducer<Text, IntWritable, Text, NullWritable> {
    public void reduce(Text key, Iterable<IntWritable> values, Context context) throws IOException, InterruptedException {
        int count = 0;
        for (IntWritable val : values) {
            count += val.get();
        }
        if(count > 5) {
            int length = new StringTokenizer(key.toString()).countTokens();
            context.write(new Text(key + "\t" + length + "\t" + count), null);
        }
    }
}
```

Figure 17: Algoritmo montado para a resolução da questão 3-B reducer

A atuação do mapper e reducer para o itens B da questão 3, ilustrados pelas figuras 16 e 17 funcionam em conjunto da seguinte forma: as tweets foram mapeados por filtros de caracteres de acordo com os nomes e por tamanho de sentenças, além de que

foram filtrados caracteres especiais, URLs e palavras menores que 3 caracteres.
A **resposta** para a terceira questão item B é:
As 3 maiores sentenças de tamanho 2, 3 e 4 são:

- Sentenças de tamanho 2: eiffel tower(2938 repetições), view from(362 repetições), second floor(329 repetições).

- Sentenças de tamanho 3: visit eiffel tower(200 repetições), visited eiffel tower(135 repetições), eiffel tower must(112 repetições).

- Sentenças de tamanho 4: without visiting eiffel tower(23 repetições), must when visiting paris(20 repetições), paris visit eiffel tower(20 repetições).

C: Encontre os principais tópicos relacionados
as revisões. Para esse item em específico, foi-se construído o algoritmo:

```
public static class MapperClass extends Mapper<LongWritable, Text, LongWritable, Text> {

    public static int reviewCount = 0;
    private static List<String> stopWords = Arrays.asList(new String[]{"n't", "'ll", "'ve", "1-1", "a", "a's", "able", "about", "above", "ab

    public void map(LongWritable key, Text value, Context context) throws IOException, InterruptedException {
        Review review = ReadJson.getReview(value.toString());
        if(review.getText() != null && !review.getText().isEmpty()) {
            ArrayList<String> list = new ArrayList<>();

            Properties props = new Properties();
            props.setProperty("annotators", "tokenize, ssplit, pos, lemma, ner");
            StanfordCoreNLP pipeline = new StanfordCoreNLP(props);

            Annotation document = new Annotation(review.getText());

            pipeline.annotate(document);

            List<CoreMap> sentences = document.get(SentencesAnnotation.class);

            for (CoreMap sentence : sentences) {

                for (CoreLabel token : sentence.get(TokensAnnotation.class)) {

                    String word = token.get(TextAnnotation.class);
                    String ne = token.get(NamedEntityTagAnnotation.class);
                    if(!ne.equals("NUMBER") && !ne.equals("ORDINAL")
                            && !ne.equals("PERCENT") && !ne.equals("DATE")
                            && !ne.equals("EMAIL") && !ne.equals("MONEY")
                            && !ne.equals("TIME") && !ne.equals("URL")
                            && !word.startsWith("http://")
                            && word.length() > 2
                            && !stopWords.contains(word)) {
                        list.add(word.toLowerCase().trim());
                    }
                }
            }
            context.write(key, new Text(String.join(",", list)));
            reviewCount++;
        }
    }
}
```

Figure 18: Algoritmo montado para resolução da questão 3-C mapper Para a realização

do que foi pedido na questão 3 item C, cujos mapper e reducer foram ilustrados nas figuras 18 e 19 respectivamente, foi-se realizado um estudo mais aprofundado acerca da técnica LDA(Latent Dirichlet Allocation)[1], na qual esse algoritmo foi adaptado para a realização da atividade, além do uso da biblioteca coreNLP[2] para o devido pré processamento do banco de dados.

```
public static class ReducerClass extends Reducer<LongWritable, Text, Text, NullWritable> {

    public static int resultCount = 0;
    private static Corpus corpus = new Corpus();

    public void reduce(LongWritable key, Iterable<Text> values, Context context) throws IOException, InterruptedException {
        Text array = values.iterator().next();
        ArrayList<String> document = new ArrayList<>();
        String[] words = array.toString().split(",");
        for (String text : words) {
            document.add(text.toString().trim());
        }
        resultCount++;

        if(MapperClass.reviewCount != resultCount) {
            return;
        }

        corpus.addDocument(document);

        LdaGibbsSampler new_ldagbs = new LdaGibbsSampler(corpus.getDocument(), corpus.getVocabularySize());

        new_ldagbs.gibbs(10);

        double[][] phi = new_ldagbs.getPhi();
        Map<String, Double>[] topicMap = LdaUtil.translate(phi, corpus.getVocabulary(), 10);

        int i = 0;
        for (Map<String, Double> topics : topicMap) {
            ArrayList<String> result = new ArrayList<>();
            result.add(String.format("Topic %d", i++));

            for (Map.Entry<String, Double> entry : topics.entrySet()) {
                result.add(entry.getKey() + " (" + String.format("%.4f", entry.getValue()) + ")");
            }

            context.write(new Text(String.join("\t", result)), null);
        }
        //LdaUtil.explain(topicMap);
    }
}
```

Figure 19: Algoritmo montado para resolução da questão 3-C reducer

A **resposta** da questão 3 item C é:

- Topic 0 the (0,0701) patient (0,0481) eiffel (0,0477) apartment (0,0474) spectacular (0,0469) make (0,0451) visited (0,0450) tiny (0,0450) great (0,0449) thomas (0,0448)
- Topic 1 the (0,0813) eiffel (0,0476) buy (0,0468) there (0,0467) patient (0,0463) visited (0,0457) level (0,0447) make (0,0446) great (0,0445)

experience (0,0444)

- Topic 2 the (0,0792) eiffel (0,0522) skim (0,0484) thomas (0,0460) experience (0,0460) tower (0,0458) great (0,0453) edison (0,0442) level (0,0438) patient (0,0437)
- Topic 3 the (0,0896) spectacular (0,0473) visited (0,0465) buy (0,0462) eiffel (0,0454) people (0,0451) elevator (0,0450) patient (0,0440) ticket (0,0439) edison (0,0438)
- Topic 4 the (0,0736) eiffel (0,0534) make (0,0482) tiny (0,0474) skim (0,0474) there (0,0469) apartment (0,0467) level (0,0463) buy (0,0455) edison (0,0444)
- Topic 5 the (0,0791) eiffel (0,0509) access (0,0487) level (0,0451) buy (0,0451) tiny (0,0451) great (0,0446) elevator (0,0444) visited (0,0443) ticket (0,0443)
- Topic 6 the (0,0707) eiffel (0,0495) thomas (0,0469) great (0,0457) spectacular (0,0457) ticket (0,0454) tiny (0,0448) make (0,0448) edison (0,0447) apartment (0,0443)
- Topic 7 the (0,0892) eiffel (0,0491) tower (0,0452) edison (0,0452) view (0,0450) tiny (0,0445) apartment (0,0441) there (0,0441) spectacular (0,0440) people (0,0438)
- Topic 8 the (0,0690) eiffel (0,0523) experience (0,0493) there (0,0464) tower (0,0457) great (0,0457) people (0,0455) view (0,0451) visited (0,0450) patient (0,0448)
- Topic 9 the (0,0919) eiffel (0,0515) view (0,0464) patient (0,0452) edison (0,0448) access (0,0446) thomas (0,0445) elevator (0,0445) people (0,0439) tower (0,0439)

D: Mapeie a distribuição temporal das revisões.

para esse item em específico, foi-se construído o algoritmo:

```
public static class MapperClass extends Mapper<Object, Text, Text, IntWritable> {
    public void map(Object key, Text data, Context context) throws IOException, InterruptedException {
        Review review = ReadJson.getReview(data.toString());
        context.write(new Text(review.getFormattedDate()), new IntWritable(1));
    }
}
```

Figure 20: Algoritmo montado para a resolução da questão 3-D mapper

```
public static class ReducerClass extends Reducer<Text, IntWritable, Text, NullWritable> {
    public void reduce(Text key, Iterable<IntWritable> values, Context context) throws IOException, InterruptedException {
        int count = 0;
        for (IntWritable val : values) {
            count += val.get();
        }
        context.write(new Text(key + "\t" + count), null);
    }
}
```

Figure 21: Algoritmo montado para a resolução da questão 3-D reducer

Por fim, o mapper e o reducer da questão 3 D ilustrados pelas figuras 20 e 21 respectivamente, realizam um mapeamento dos tweets de acordo com a data de seu registro, e os agrupam por quantidade, independente de sua repetição.

Uma pequena amostra da **resposta** do item D da questão 3 é: 01/01/2016 18

01/01/2017 4

01/02/2016 6

01/02/2017 2

01/03/2016 14

01/03/2017 12

01/04/2016 17

01/04/2017 11

01/05/2016 8

01/05/2017 6

01/06/2016 23

O restante encontra-se presente no link do github:https://github.com/davimota-dev/TP_Hadoop_UFC.git

References

[1] David M Blei, Andrew Y Ng, and Michael I Jordan. Latent dirichlet allocation. *Journal of machine Learning research*, 3(Jan):993–1022, 2003.

[2] Min Song and Tamy Chambers. Text mining with the stanford corenlp.
In *Measuring scholarly impact*, pages 215–234. Springer, 2014.