Solución Práctica
Map Reduce

1. Cantidad de habitantes total del país agrupados de acuerdo a su edad en tres grupos :
    a. 0 - 14 años
    b. 15 - 64 años
    c. 65 años y más

    - **Map**: Se revisa la edad para ver en qué rango está. Se emite como clave el rango en el que está y como valor 1.
    - **Reduce**: se suman todos los valores emitidos y se emite la cantidad como valor y el rango etario como clave.

2. El promedio de habitantes por vivienda para cada tipo de vivienda.
    - **Map**: Se emite como clave el tipo tipo de vivienda y como valor el identificador de hogar.
    - **Reduce**: Se cuenta la cantidad de valores que se reciben (habitantes) y la cantidad de valores únicos (hogares). Se emite como clave el tipo de vivienda y como valor la cantidad de habitantes/cantidad de hogares

3. Los `n` departamentos con mayor índice de analfabetismo, donde el índice se calcula por el número total de habitantes analfabetos del departamento sobre el total de población del departamento, donde `n` provee el usuario.
    - **Map**: Se emite el nombre del departamento como clave y el valor de anafalbetismo como valor.
    - **Reduce**: Se llevan 2 contadores, uno de la cantidad de población y el otro solo cuenta si el índice de analfabetismo es igual a 2. Al finalizar se retorna la división de ambos contadores

    El orden y el límite no se pueden hacer por mapreduce, hay que procesar la respuesta para eso

4. Los departamentos de la provincia `prov` con una cantidad de habitantes menor a `tope`, donde `prov` y `tope` lo provee el usuario.
    - **Map**: Se emite el nombre del departamento como clave y 1 como valor solo si la provincia corresponde a la enviada.
    - **Reduce**: Se cuenta los valores recibidos para obtener el valor final que es la población.

    El límite por tope se calcula por fuera del proceso de mapreduce.

5. Los pares de departamentos que tienen la misma cantidad de cientos de habitantes. Para resolverlo se requieren dos proceso mapreduce:
    - **Map**: Se emite el nombre del departamento + provincia como clave y 1 como valor.
    - **Reduce**: Se cuenta los valores recibidos para obtener el valor final que es la población.
    - **Map**: Se emite como valor la población en miles y como valor el departamento-provincia
    - **Reduce**: Se emite por cada población en miles y como valor el listado de los departamentos que corresponden a esa población.

Map Reduce en Java.

Word count

Mapper

```
public class TokenizerMapper extends Mapper < LongWritable, Text, Text, IntWritable > {
 private final static IntWritable one = new IntWritable(1);
 private Text word = new Text();

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

  final String sanitizedLine = line.toString().replaceAll("\\p{Punct}|\\d", "").toLowerCase();
  final StringTokenizer itr = new StringTokenizer(sanitizedLine);
  while (itr.hasMoreTokens()) {
   word.set(itr.nextToken());
   context.write(word, one);
  }
 }
}
```

Reducer

```
public class IntSumReducer extends Reducer < Text, IntWritable, Text, IntWritable > {

 private IntWritable result = new IntWritable();

 @Override
 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);
}
}
```

Temperatura

Mapper

```
public class TemperatureMapper extends Mapper<LongWritable, Text, Text,
DoubleWritable> {

    @Override
    public void map(LongWritable key, Text line, Context context) throws IOException,
InterruptedException {
        final String[] values = line.toString().split(",");

        final String date = values[1];
        final String temp = values[8];

        try {
            context.write(new Text(date), new DoubleWritable(Double.parseDouble(temp)));
        } catch (NumberFormatException e) {
            // header line...
            // log for future references
        }
    }
}
```

Reducer

```
public class AveragerReducer extends Reducer<Text, DoubleWritable, Text,
DoubleWritable> {

   @Override
   public void reduce(Text key, Iterable<DoubleWritable> values, Context context) throws
IOException, InterruptedException {
      double sum = 0;
      int count = 0;

      for (DoubleWritable val : values) {
         sum += val.get();
         count += 1;
      }

      context.write(key, new DoubleWritable(sum / count));
   }
}
```