```
/**
* Código desenvolvido por: Gilvan Oliveira.
*/

package br.edu.ufam.gilvanoliveira7;

import java.io.BufferedReader;
import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.io.IOException;
import java.io.InputStreamReader;
import java.io.LineNumberReader;
import java.io.ByteArrayOutputStream;
import java.io.DataOutputStream;
import java.util.ArrayList;
import java.util.Arrays;
import java.util.HashMap;
import java.util.HashSet;
import java.util.Iterator;
import java.util.List;
import java.util.Map;
import java.util.Set;
import java.util.StringTokenizer;
import java.util.TreeSet;
import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.conf.Configured;
import org.apache.hadoop.fs.FSDataInputStream;
import org.apache.hadoop.fs.FileSystem;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.BytesWritable;
import org.apache.hadoop.io.DoubleWritable;
```

```
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.io.WritableUtils;
import org.apache.hadoop.io.VIntWritable;
import org.apache.hadoop.mapred.JobConf;
import org.apache.hadoop.mapreduce.Job;
import org.apache.hadoop.mapreduce.Mapper;
import org.apache.hadoop.mapreduce.Reducer;
import org.apache.hadoop.mapreduce.Partitioner;
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.mapreduce.lib.output.MapFileOutputFormat;
import org.apache.hadoop.util.Tool;
import org.apache.hadoop.util.ToolRunner;
import org.apache.log4j.Logger;
import org.apache.commons.cli.CommandLine;
import org.apache.commons.cli.CommandLineParser;
import org.apache.commons.cli.GnuParser;
import org.apache.commons.cli.HelpFormatter;
import org.apache.commons.cli.OptionBuilder;
import org.apache.commons.cli.Options;
import org.apache.commons.cli.ParseException;
import org.apache.hadoop.hbase.HBaseConfiguration;
import org.apache.hadoop.hbase.HColumnDescriptor;
import org.apache.hadoop.hbase.HTableDescriptor;
import org.apache.hadoop.hbase.TableName;
import org.apache.hadoop.hbase.client.HBaseAdmin;
import org.apache.hadoop.hbase.client.Put;
import org.apache.hadoop.hbase.io.ImmutableBytesWritable;
```

```
import org.apache.hadoop.hbase.mapreduce.TableMapReduceUtil;
import org.apache.hadoop.hbase.mapreduce.TableReducer;
import org.apache.hadoop.hbase.util.Bytes;

public class BuildInvertedIndexHbase extends Configured implements Tool {
   private static final Logger LOG = Logger.getLogger(BuildInvertedIndexCompressed.class);

   public static final String[] FAMILIES = { "p" };
   public static final byte[] CF = FAMILIES[0].getBytes();
   public static final byte[] FREQ = "freq".getBytes();

   private static final class InvertedIndexMapper extends Mapper<LongWritable, Text,
TextIntWritablePairComparable, IntWritable> {

      private static Map<String,Integer> inMapperHash;
      private static Text WORD;
      private static IntWritable IW;
      private static final TextIntWritablePairComparable pair = new
TextIntWritablePairComparable();

      @Override
      public void setup(Context context) throws IOException, InterruptedException {
         inMapperHash = new HashMap<String,Integer>();
         WORD = new Text();
         IW = new IntWritable();
      }

      @Override
      public void map(LongWritable docno, Text values, Context context) throws IOException,
InterruptedException {
         String line = values.toString();
```

```
//line normalization
line = line.replaceAll("[\'\",.:;=$#@%\\*!\\?\\[\\]\\(\\)\\{\\}<>&]","");
line = line.toLowerCase();
//System.out.println("line: " + line);

//line tokenization
StringTokenizer tok = new StringTokenizer(line);
String token = "";

while (tok.hasMoreTokens()) {
   token = tok.nextToken();

   while((token.startsWith("'")) || (token.startsWith("-"))){
      token = token.substring(1,token.length());
   }

   if(token.endsWith("-")){
      token = token.substring(0,token.length()-1);
   }

   if(token.length() == 0){
      continue;
   }

   Integer gotCount = inMapperHash.get(token);

   if (gotCount != null) {
      gotCount++;
   } else {
      gotCount = 1;
   }
```

```
            if(inMapperHash.containsKey(token+",1")){
               inMapperHash.put(token + "," + "1", inMapperHash.get(token+",1") + gotCount);
            }else{
               inMapperHash.put(token + "," + "1", gotCount);
            }
         }

      }

      @Override
      public void cleanup(Context context) throws IOException, InterruptedException {
         for (String key : inMapperHash.keySet()) {
            String[] term_docid = key.split(",");
            pair.set(new Text(term_docid[0]), new IntWritable(Integer.valueOf(term_docid[1])) );
            IW.set(inMapperHash.get(key));
            context.write(pair,IW);
         }
      }

   }

   private static final class InvertedIndexPartitioner extends
Partitioner<TextIntWritablePairComparable,IntWritable> {

      @Override
      public int getPartition(TextIntWritablePairComparable key,IntWritable value,int
numReducers) {
         return Integer.valueOf(key.getLeftElement().toString()) % numReducers;
      }

   }
```

```
   private static final class InvertedIndexReducer extends
Reducer<TextIntWritablePairComparable,IntWritable,Text,ArrayListWritable<PairOfWritables<
IntWritable,VIntWritable>>> {//BytesWritable> {

      private final static Text TERM = new Text();


      private static final ArrayListWritable<PairOfWritables<IntWritable,VIntWritable>>
index_postings = new ArrayListWritable<PairOfWritables<IntWritable,VIntWritable>>();

      private static PairOfWritables<IntWritable,VIntWritable> postings;

      private static IntWritable leftInt;

      private static VIntWritable rightInt;



      private static int lastDocno = 0;

      private static int thisDocno = 0;

      private static int dGapInt = 0;


      private static int docFreq = 0;

      private static int termFreq = 0;

      private static String currentTerm = null;

      private static String lastTerm = new String();

      private static String[] term_docid;


      @Override

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

         currentTerm = key.getLeftElement().toString();

         thisDocno = key.getRightElement().get();


         if (currentTerm != null && !lastTerm.equals(currentTerm)) {

            TERM.set(currentTerm);
```

```
            context.write(TERM,index_postings);

            index_postings.clear();

            //Reset counters
            lastDocno = 0;
            docFreq = 0;
            termFreq = 0;
            currentTerm = key.getLeftElement().toString();
            thisDocno = key.getRightElement().get();
            lastTerm = currentTerm;

        }

        Iterator<IntWritable> iter = value.iterator();

        while (iter.hasNext()) {
            termFreq += iter.next().get();
        }

        docFreq++;
        dGapInt = thisDocno - lastDocno;
        lastDocno = thisDocno;
        leftInt = new IntWritable(thisDocno);
        rightInt = new VIntWritable(termFreq);
        postings = new PairOfWritables<IntWritable,VIntWritable>();
        postings.set(leftInt,rightInt);
        index_postings.add(postings);
        docFreq++;
    }
```

```
    public void cleanup(Context context) throws IOException, InterruptedException {
      TERM.set(currentTerm);
      context.write(TERM,index_postings);
      index_postings.clear();
    }
  }

  public static class InvertedIndexTableReducer extends
TableReducer<Text,ArrayListWritable<PairOfWritables<IntWritable,VIntWritable>>,
ImmutableBytesWritable>  {
    public void reduce(Text key,
Iterable<ArrayListWritable<PairOfWritables<IntWritable,VIntWritable>>> values, Context
context) throws IOException, InterruptedException {
      int sum = 0;

      for (ArrayListWritable array:values){
        for(PairOfWritables<IntWritable,VIntWritable> pair:array){
          sum += pair.getLeftElement().get();
        }
      }
      Put put = new Put(Bytes.toBytes(key.toString()));
      put.add(CF, FREQ, Bytes.toBytes(sum));

      context.write(null, put);
    }
  }

  private BuildInvertedIndexHbase() {}

  private static final String INPUT = "input";
  private static final String OUTPUT = "output";
  private static final String NUM_REDUCERS = "numReducers";
```

```
  @SuppressWarnings({ "static-access" })
  @Override
  public int run(String[] args) throws Exception {
    Options options = new Options();

    options.addOption(OptionBuilder.withArgName("path").hasArg().withDescription("input path").create(INPUT));
    options.addOption(OptionBuilder.withArgName("table").hasArg().withDescription("HBase table name").create(OUTPUT));

options.addOption(OptionBuilder.withArgName("num").hasArg().withDescription("number of reducers").create(NUM_REDUCERS));

    CommandLine cmdline;
    CommandLineParser parser = new GnuParser();

    try {
      cmdline = parser.parse(options, args);
    } catch (ParseException exp) {
      System.err.println("Error parsing command line: " + exp.getMessage());
      return -1;
    }

    if (!cmdline.hasOption(INPUT) || !cmdline.hasOption(OUTPUT)) {
      System.out.println("args: " + Arrays.toString(args));
      HelpFormatter formatter = new HelpFormatter();
      formatter.setWidth(120);
      formatter.printHelp(this.getClass().getName(), options);
      ToolRunner.printGenericCommandUsage(System.out);
      return -1;
    }
```

```
    String inputPath = cmdline.getOptionValue(INPUT);
    String outputTable = cmdline.getOptionValue(OUTPUT);
    int reduceTasks = cmdline.hasOption(NUM_REDUCERS) ?
    Integer.parseInt(cmdline.getOptionValue(NUM_REDUCERS)) : 1;

    Configuration conf = getConf();
    conf.addResource(new Path("/etc/hbase/conf/hbase-site.xml"));

    Configuration hbaseConfig = HBaseConfiguration.create(conf);
    HBaseAdmin admin = new HBaseAdmin(hbaseConfig);

    if (admin.tableExists(outputTable)) {
      LOG.info(String.format("Table '%s' exists: dropping table and recreating.",
outputTable));
      LOG.info(String.format("Disabling table '%s'", outputTable));
      admin.disableTable(outputTable);
      LOG.info(String.format("Droppping table '%s'", outputTable));
      admin.deleteTable(outputTable);
    }

    HTableDescriptor tableDesc = new HTableDescriptor(TableName.valueOf(outputTable));
    for (int i = 0; i < FAMILIES.length; i++) {
      HColumnDescriptor hColumnDesc = new HColumnDescriptor(FAMILIES[i]);
      tableDesc.addFamily(hColumnDesc);
    }
    admin.createTable(tableDesc);
    LOG.info(String.format("Successfully created table '%s'", outputTable));

    admin.close();
```

```
    LOG.info("Tool: " + BuildInvertedIndexHbase.class.getSimpleName());
    LOG.info(" - input path: " + inputPath);
    LOG.info(" - output path: " + outputTable);
    LOG.info(" - number of reducers: " + reduceTasks);

    Job job = Job.getInstance(conf);
    job.setJobName(BuildInvertedIndexHbase.class.getSimpleName());
    job.setJarByClass(BuildInvertedIndexHbase.class);

    job.setMapOutputKeyClass(TextIntWritablePairComparable.class);
    job.setMapOutputValueClass(IntWritable.class);

    job.setMapperClass(InvertedIndexMapper.class);
    job.setPartitionerClass(InvertedIndexPartitioner.class);
    job.setReducerClass(InvertedIndexReducer.class);
    job.setNumReduceTasks(reduceTasks);

    FileInputFormat.setInputPaths(job, new Path(inputPath));
    TableMapReduceUtil.initTableReducerJob(outputTable, InvertedIndexTableReducer.class,
job);

    long startTime = System.currentTimeMillis();
    job.waitForCompletion(true);
    LOG.info("Job Finished in " + (System.currentTimeMillis() - startTime) / 1000.0 + "
seconds");

    return 0;
  }

  /**
  * Dispatches command-line arguments to the tool via the {@code ToolRunner}.
  */
```

```
    public static void main(String[] args) throws Exception {
        ToolRunner.run(new BuildInvertedIndexHbase(), args);
    }

}
```

```
/**
* Código desenvolvido por: Gilvan Oliveira.
*/

package br.edu.ufam.gilvanoliveira7;

import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;
import java.util.ArrayList;
import java.util.Arrays;
import java.util.Set;
import java.util.Stack;
import java.util.TreeSet;

import org.apache.commons.cli.CommandLine;
import org.apache.commons.cli.CommandLineParser;
import org.apache.commons.cli.GnuParser;
import org.apache.commons.cli.HelpFormatter;
```

```
import org.apache.commons.cli.OptionBuilder;
import org.apache.commons.cli.Options;
import org.apache.commons.cli.ParseException;
import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.conf.Configured;
import org.apache.hadoop.hbase.HBaseConfiguration;
import org.apache.hadoop.hbase.client.Get;
import org.apache.hadoop.hbase.client.HConnection;
import org.apache.hadoop.hbase.client.HConnectionManager;
import org.apache.hadoop.hbase.client.HTableInterface;
import org.apache.hadoop.hbase.client.Result;
import org.apache.hadoop.hbase.util.Bytes;
import org.apache.hadoop.fs.FSDataInputStream;
import org.apache.hadoop.fs.FileSystem;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.BytesWritable;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.MapFile;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.util.Tool;
import org.apache.hadoop.util.ToolRunner;

public class BooleanRetrievalHbase extends Configured implements Tool {
  private HTableInterface table;
  private FSDataInputStream collection;
  private Stack<Set<Integer>> stack;

  private BooleanRetrievalHbase() {}

  private void runQuery(String q) throws IOException {
    String[] terms = q.split("\\s+");
```

```
  for (String t : terms) {
    if (t.equals("AND")) {
      performAND();
    } else if (t.equals("OR")) {
      performOR();
    } else {
      pushTerm(t);
    }
  }

  Set<Integer> set = stack.pop();

  for (Integer i : set) {
    String line = fetchLine(i);
    System.out.println(i + "\t" + line);
  }
}

private void pushTerm(String term) throws IOException {
  stack.push(fetchDocumentSet(term));
}

private void performAND() {
  Set<Integer> s1 = stack.pop();
  Set<Integer> s2 = stack.pop();

  Set<Integer> sn = new TreeSet<Integer>();

  for (int n : s1) {
    if (s2.contains(n)) {
```

```
      sn.add(n);
    }
  }

  stack.push(sn);
}

private void performOR() {
  Set<Integer> s1 = stack.pop();
  Set<Integer> s2 = stack.pop();

  Set<Integer> sn = new TreeSet<Integer>();

  for (int n : s1) {
    sn.add(n);
  }

  for (int n : s2) {
    sn.add(n);
  }

  stack.push(sn);
}

private Set<Integer> fetchDocumentSet(String term) throws IOException {
  Set<Integer> set = new TreeSet<Integer>();

  for (PairOfInts pair : fetchPostings(term)) {
    set.add(pair.getLeftElement());
  }
```

```
    return set;
  }

  private ArrayList<PairOfInts> fetchPostings(String term) throws IOException {
    Get get = new Get(Bytes.toBytes(term));
    Result result = table.get(get);

    PairOfWritables<IntWritable, ArrayListWritable<PairOfInts>> postings =
result.getValue(BuildInvertedIndexHbase.CF, BuildInvertedIndexHbase.FREQ);

    return postings.getRightElement();
  }


  private String fetchLine(long offset) throws IOException {
    collection.seek(offset);
    BufferedReader reader = new BufferedReader(new InputStreamReader(collection));

    return reader.readLine();
  }

  private static final String TABLE = "table";
  private static final String WORD = "word";

  /**
   * Runs this tool.
   */
  @SuppressWarnings({ "static-access" })
  public int run(String[] args) throws Exception {
    Options options = new Options();
```

```
   options.addOption(OptionBuilder.withArgName("table").hasArg().withDescription("HBase
table name").create(TABLE));
   options.addOption(OptionBuilder.withArgName("word").hasArg().withDescription("word to
look up").create(WORD));

   CommandLine cmdline = null;
   CommandLineParser parser = new GnuParser();

   try {
    cmdline = parser.parse(options, args);
   } catch (ParseException exp) {
    System.err.println("Error parsing command line: " + exp.getMessage());
    System.exit(-1);
   }

   if (!cmdline.hasOption(TABLE) || !cmdline.hasOption(WORD)) {
      System.out.println("args: " + Arrays.toString(args));
      HelpFormatter formatter = new HelpFormatter();
      formatter.setWidth(120);
      formatter.printHelp(this.getClass().getName(), options);
      ToolRunner.printGenericCommandUsage(System.out);
      return -1;
   }

   String tableName = cmdline.getOptionValue(TABLE);
   String word = cmdline.getOptionValue(WORD);

   Configuration conf = getConf();
   conf.addResource(new Path("/etc/hbase/conf/hbase-site.xml"));

   Configuration hbaseConfig = HBaseConfiguration.create(conf);
   HConnection hbaseConnection = HConnectionManager.createConnection(hbaseConfig);
```

```
    table = hbaseConnection.getTable(tableName);


    String[] queries = { "outrageous fortune AND", "white rose AND", "means deceit AND",
        "white red OR rose AND pluck AND", "unhappy outrageous OR good your AND OR fortune
AND" };


    for (String q : queries) {
      System.out.println("Query: " + q);


      runQuery(q);
      System.out.println("");
    }


    return 1;
  }


  /**
   * Dispatches command-line arguments to the tool via the {@code ToolRunner}.
   */
  public static void main(String[] args) throws Exception {
    ToolRunner.run(new BooleanRetrievalHbase(), args);
  }
}
```