COMP38120

Documents, Services and Data on the Web

Laboratory 2

Alex-Radu Malan

9770386

Table of Contents

Introduction

The technologies used nowadays have the power to enrich the information available on the web which automatically makes everyone's work way easier in terms of seeking that information. The problem with this intel is that even though humans understand it and can make links inside their heads, machines are not able to have this ability since they have to be provided with step-by-step instructions. So the structure and the link between data will be a problem for humans as well since we get the information through machines; if machines do not have any idea of what we want, then we will not get to that wished information. In order to create a solution for this, there are different tools that were introduced, such as the Resource Description Framework (RDF). The aim of this framework is to provide relationships between different types of data so that machines will be also able to have a deeper understanding of our search. This framework is based on a subject-predicate-object form which is also known as triples. The subject is denoting the resource, then the predicate denotes an attribute of the resource which will lastly take us to the value of the attribute also known as the object. The main idea of the laboratory was to implement a data indexing program that would index documents that are making use of the framework described above.

Functionality

First of all, the input that is being used in the program is in form of text/literals which we know from the previous laboratory that in order to be indexed, it has to be tokenised and filtered from any different symbols except literals. The RDF has Literals which are used for strings, numbers and dates. The Literals we are checking in the program are divided into two types: plain literals and typed literals. Plain literals refer to a string that is combined optionally with a language tag which has a fixed meaning. Compared to the plain literals, the typed literals refer to a string combined with a datatype URI (<aaa: boolean, "true"> and <aaa: boolean, 1> points to the boolean TRUE value). Each of these datatypes is made of different components such as value and lexical form. The values are also known as data values which represent the set of all strings related to a datatype. The lexical form is linked to those data values and it can be of two types as well: empty string or having a language tag in it. This lexical form can be lowercase or case insensitive as well.

In order to create the indexing program, first of all, we will create a literal object and then by using it we will retrieve the lexical form and the language used. First of all, the text will be tokenised then each token will be either case sensitive or insensitive, so we will have two options that can be used to label the tokens. After indexing the plain literals, we have to also check for typed literals which do not have to be labelled in terms of case sensitivity since they are different from the plain literals, but we will label them as being typed literals.

Improvements

There are multiple ways the indexing can be improved since the one created is taking into consideration a limited number of variables. The RDF can be combined with flexible querying since the results of the current one is either a match to the query or not, so the query should be more permissive than restrictive for the search. There can also be many results for queries so that is why ranking is another option that can be implemented in order to retrieve more relevant information.

Conclusion

In conclusion, the Resource Description Framework is a tool that is helping the data on the web to have a better structure and also to provide machines with a better understanding of what a certain user would like to have retrieved in a search request. It is important to use the

indexing in such a way that multiple variables are allowed besides the triple which is standard, in order to have a better link and relationship between the data. The human brain and thought process is taking into consideration variables that sometimes we do not even think about, so implementing a tool that will make the search act as a brain is very hard, but if we continue to improve it, we might achieve that goal.

Output

1. (http://dbpedia.org/ontology/abstract, literal-plain-case-insensitive:underwent@en) [http://dbpedia.org/resource/A_Fish_Called_Selma]
2. (http://dbpedia.org/ontology/abstract, literal-plain-case-insensitive:unit@en) [http://dbpedia.org/resource/24_Minutes]
3. (http://dbpedia.org/ontology/abstract, literal-plain-case-insensitive:united@en) [http://dbpedia.org/resource/500_Keys, http://dbpedia.org/resource/A_Star_Is_Burns]
4. (http://dbpedia.org/ontology/abstract, literal-plain-case-insensitive:universe@en) [http://dbpedia.org/resource/A_Star_Is_Burns]
5. (http://dbpedia.org/ontology/abstract, literal-plain-case-insensitive:unsuccessful@en) [http://dbpedia.org/resource/A_Star_Is_Burns]
6. (http://dbpedia.org/ontology/abstract, literal-plain-case-insensitive:up@en) [http://dbpedia.org/resource/A_Star_Is_Burns]
7. (http://dbpedia.org/ontology/abstract, literal-plain-case-insensitive:violates@en) [http://dbpedia.org/resource/A_Star_Is_Burns]
8. (http://dbpedia.org/ontology/abstract, literal-plain-case-insensitive:visiting@en) [http://dbpedia.org/resource/A_Star_Is_Born_Again]
9. (http://dbpedia.org/ontology/abstract, literal-plain-case-insensitive:voice@en) [http://dbpedia.org/resource/A_Star_Is_Born_Again]
10. (http://dbpedia.org/ontology/abstract, literal-plain-case-insensitive:voting@en) [http://dbpedia.org/resource/A_Star_Is_Burns]
11. (http://dbpedia.org/ontology/abstract, literal-plain-case-insensitive:waiter@en) [http://dbpedia.org/resource/A_Hunka_Hunka_Burns_in_Love]
12. (http://dbpedia.org/ontology/abstract, literal-plain-case-insensitive:want@en) [http://dbpedia.org/resource/24_Minutes]
13. http://dbpedia.org/ontology/abstract, literal-plain-case-insensitive:sixteenth@en) [http://dbpedia.org/resource/A_Star_Is_Torn]
14. (http://dbpedia.org/ontology/abstract, literal-plain-case-insensitive:sixth@en) [http://dbpedia.org/resource/A_Milhouse_Divided, http://dbpedia.org/resource/A_Star_Is_Burns]
15. (http://dbpedia.org/ontology/abstract, literal-plain-case-insensitive:skinners@en) [http://dbpedia.org/resource/24_Minutes]
16. (http://dbpedia.org/ontology/abstract, literal-plain-case-insensitive:sloane@en) [http://dbpedia.org/resource/A_Star_Is_Born_Again]
17. (http://dbpedia.org/ontology/abstract, literal-plain-case-insensitive:slow@en) [http://dbpedia.org/resource/A_Fish_Called_Selma]
18. (http://dbpedia.org/ontology/abstract, literal-plain-case-insensitive:snake@en) [http://dbpedia.org/resource/A_Hunka_Hunka_Burns_in_Love]
19. (http://dbpedia.org/ontology/abstract, literal-plain-case-insensitive:sole@en) [http://dbpedia.org/resource/A_Milhouse_Divided]
20. (http://dbpedia.org/ontology/abstract, literal-plain-case-insensitive:song@en) [http://dbpedia.org/resource/A_Fish_Called_Selma]
21. (http://dbpedia.org/ontology/abstract, literal-plain-case-insensitive:special@en) [http://dbpedia.org/resource/A_Star_Is_Born_Again]
22. (http://dbpedia.org/ontology/abstract, literal-plain-case-insensitive:speech@en) [http://dbpedia.org/resource/A_Fish_Called_Selma]
23. (http://dbpedia.org/ontology/abstract, literal-plain-case-insensitive:speed@en) [http://dbpedia.org/resource/A_Fish_Called_Selma]
24. (http://dbpedia.org/ontology/abstract, literal-plain-case-insensitive:spinoff@en) [http://dbpedia.org/resource/22_Short_Films_About_Springfield]
25. (http://dbpedia.org/ontology/abstract, literal-plain-case-insensitive:spoof@en) [http://dbpedia.org/resource/24_Minutes]
26. (http://dbpedia.org/property/airdate, literal-typed:1995-03-05^^http://www.w3.org/2001/XMLSchema#date) [http://dbpedia.org/resource/A_Star_Is_Burns]
27. (http://dbpedia.org/property/airdate, literal-typed:1996-03-24^^http://www.w3.org/2001/XMLSchema#date) [http://dbpedia.org/resource/A_Fish_Called_Selma]
28. (http://dbpedia.org/property/airdate, literal-typed:1996-04-14^^http://www.w3.org/2001/XMLSchema#date) [http://dbpedia.org/resource/22_Short_Films_About_Springfield]
29. (http://dbpedia.org/property/airdate, literal-typed:1996-12-01^^http://www.w3.org/2001/XMLSchema#date) [http://dbpedia.org/resource/A_Milhouse_Divided]
30. (http://dbpedia.org/property/airdate, literal-typed:2001-12-02^^http://www.w3.org/2001/XMLSchema#date) [http://dbpedia.org/resource/A_Hunka_Hunka_Burns_in_Love]
31. (http://dbpedia.org/property/airdate, literal-typed:2003-03-02^^http://www.w3.org/2001/XMLSchema#date) [http://dbpedia.org/resource/A_Star_Is_Born_Again]
32. (http://dbpedia.org/property/airdate, literal-typed:2005-05-08^^http://www.w3.org/2001/XMLSchema#date) [http://dbpedia.org/resource/A_Star_Is_Torn]
33. (http://dbpedia.org/property/airdate, literal-typed:2007-05-20^^http://www.w3.org/2001/XMLSchema#date) [http://dbpedia.org/resource/24_Minutes]
34. (http://dbpedia.org/property/airdate, literal-typed:2011-05-15^^http://www.w3.org/2001/XMLSchema#date) [http://dbpedia.org/resource/500_Keys]
35. (http://dbpedia.org/property/commentary, literal-plain-case-sensitive:Cohen@en) [http://dbpedia.org/resource/22_Short_Films_About_Springfield]
36. (http://dbpedia.org/property/commentary, literal-plain-case-sensitive:David@en) [http://dbpedia.org/resource/22_Short_Films_About_Springfield]
37. (http://dbpedia.org/property/commentary, literal-plain-case-sensitive:Dean@en) [http://dbpedia.org/resource/A_Milhouse_Divided]
38. (http://dbpedia.org/property/commentary, literal-plain-case-sensitive:Goldblum@en) [http://dbpedia.org/resource/A_Fish_Called_Selma]
39. (http://dbpedia.org/property/commentary, literal-plain-case-sensitive:Groening@en) [http://dbpedia.org/resource/22_Short_Films_About_Springfield]
40. (http://dbpedia.org/property/commentary, literal-plain-case-sensitive:GroeningBill@en) [http://dbpedia.org/resource/A_Milhouse_Divided]
41. (http://dbpedia.org/property/commentary, literal-plain-case-sensitive:James@en) [http://dbpedia.org/resource/22_Short_Films_About_Springfield]
42. (http://dbpedia.org/property/commentary, literal-plain-case-sensitive:Jean@en) [http://dbpedia.org/resource/A_Star_Is_Burns]
43. (http://dbpedia.org/property/commentary, literal-plain-case-sensitive:Jeff@en) [http://dbpedia.org/resource/A_Fish_Called_Selma]
44. (http://dbpedia.org/property/commentary, literal-plain-case-sensitive:Jon@en) [http://dbpedia.org/resource/A_Star_Is_Burns]

RDFInvertedIndex.java

```
/**
 * Copyright (C) University of Manchester - All Rights Reserved
 * Unauthorised copying of this file, via any medium is strictly prohibited
 * Proprietary and confidential
 * Written by Kristian Epps <kristian@xepps.com>, August 28, 2013
 *
 * RDF Inverted Index
 *
 * This Map Reduce program should read in a set of RDF/XML documents and output
 * the data in the form:
 *
 * {predicate, object]}, [subject1, subject2, ...]
 *
 * @author Kristian Epps
 *
 */
package uk.ac.man.cs.comp38120.exercise;

import java.io.IOException;
import java.io.InputStream;
import java.util.Arrays;
import java.util.Collections;
import java.util.Iterator;
import java.util.StringTokenizer;

import org.apache.commons.cli.CommandLine;
import org.apache.commons.cli.CommandLineParser;
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.commons.io.IOUtils;
import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.conf.Configured;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Job;
import org.apache.hadoop.mapreduce.Mapper;
import org.apache.hadoop.mapreduce.Reducer;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;
import org.apache.hadoop.util.Tool;
import org.apache.hadoop.util.ToolRunner;
import org.apache.log4j.Logger;

import com.hp.hpl.jena.rdf.model.Model;
import com.hp.hpl.jena.rdf.model.ModelFactory;
import com.hp.hpl.jena.rdf.model.RDFNode;
import com.hp.hpl.jena.rdf.model.Resource;
import com.hp.hpl.jena.rdf.model.Literal;
import com.hp.hpl.jena.rdf.model.Statement;
import com.hp.hpl.jena.rdf.model.StmtIterator;
import com.hp.hpl.jena.rdf.model.Property;
```

```
import uk.ac.man.cs.comp38120.io.array.ArrayListWritable;
import uk.ac.man.cs.comp38120.io.pair.PairOfStrings;
import uk.ac.man.cs.comp38120.util.XParser;

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

  public static class Map extends
      Mapper<LongWritable, Text, PairOfStrings, Text>
  {

    protected Text document = new Text();
    protected PairOfStrings predobj = new PairOfStrings();
    protected Text subj = new Text();

    private void index(Context c, Resource  s, Property p,
          RDFNode o, String type) {
        String flag = type + ":";
        predobj.set(
           p.toString(),
           flag + o.toString()
         );
        subj.set(s.toString());
        try {
                        c.write(predobj, subj);
                } catch (IOException | InterruptedException e) {
                        e.printStackTrace();
                }
      }

    public void map(LongWritable key, Text value, Context context)
        throws IOException, InterruptedException
    {
      // This statement ensures we read a full rdf/xml document in
      // before we try to do anything else
      if(!value.toString().contains("</rdf:RDF>"))
      {
        document.set(document.toString() + value.toString());
        return;
      }

      // We have to convert the text to a UTF-8 string. This must be
      // enforced or errors will be thrown.
      String contents = document.toString() + value.toString();
      contents = new String(contents.getBytes(), "UTF-8");

      // The string must be cast to an inputstream for use with jena
      InputStream fullDocument = IOUtils.toInputStream(contents);
      document = new Text();

      // Create a model
      Model model = ModelFactory.createDefaultModel();

      try
      {
```

```
model.read(fullDocument, null);

StmtIterator iter = model.listStatements();

// Iterate over triples
while(iter.hasNext())
{
 Statement stmt      = iter.nextStatement();
 Resource  subject   = stmt.getSubject();
 Property  predicate = stmt.getPredicate();
 RDFNode   object    = stmt.getObject();

 if (object instanceof Literal)
 {
  Literal l = object.asLiteral();

  // Checking the literal type
  if (l.getDatatype() == null)
  {
   // Declaring and retrieving the lexical form
   String lexicalFormLiteral = l.getLexicalForm();

   // Declaring and retrieving the language
   String languageLiteral = l.getLanguage();

   // Declaring the tokenizer that will iterate through them
   StringTokenizer interating = new StringTokenizer(lexicalFormLiteral);

   while (interating.hasMoreTokens())
   {
   // Declaring and filtering the tokens so that only strings will remain
    String theToken = interating.nextToken().replaceAll("[^a-zA-Z0-9]", "");

    // Declaring and creating model for case insensitive tokens
    Literal insensitiveToken = model.createLiteral(theToken.toLowerCase(), languageLiteral);

    // Declaring and creating model for case sensitive tokens
    Literal sensitiveToken = model.createLiteral(theToken, languageLiteral);

    // Checking the token for the plain literal
    if (theToken.length() > 1 && theToken != null && !theToken.isEmpty())
    {
     // Indexing the plain literals that are case insensitive
     index(context, subject, predicate, insensitiveToken, "literal-plain-case-insensitive");

     // Indexing the plain literals that are case sensitive
     index(context, subject, predicate, sensitiveToken, "literal-plain-case-sensitive");
    } // if
   } // while
  } // if
  else
  {
   // Indexing the typed literals + flagging them
   index(context, subject, predicate, l, "literal-typed");
  } // else
 } // if
} // while
```

```
    } // try
    catch(Exception e)
    {
      LOG.error(e);
    } // catch
   } // map
  } // Map

  public static class Reduce extends Reducer<PairOfStrings, Text, PairOfStrings,
ArrayListWritable<Text>>
  {

   // This reducer turns an iterable into an ArrayListWritable, sorts it
   // and outputs it
   public void reduce(
      PairOfStrings key,
      Iterable<Text> values,
      Context context) throws IOException, InterruptedException
   {
     ArrayListWritable<Text> postings = new ArrayListWritable<Text>();

     Iterator<Text> iter = values.iterator();

     while(iter.hasNext()) {
       Text copy = new Text(iter.next());
       postings.add(copy);
     }

     Collections.sort(postings);

     context.write(key, postings);
   }
  }

  public RDFInvertedIndex()
  {
  }

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

  @SuppressWarnings({ "static-access" })
  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("path").hasArg()
      .withDescription("output path").create(OUTPUT));
   options.addOption(OptionBuilder.withArgName("num").hasArg()
      .withDescription("number of reducers").create(NUM_REDUCERS));

   CommandLine cmdline = null;
   CommandLineParser parser = new XParser(true);
```

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

    Job RDFIndex = new Job(new Configuration());

    RDFIndex.setJobName("Inverted Index 1");
    RDFIndex.setJarByClass(RDFInvertedIndex.class);
    RDFIndex.setMapperClass(Map.class);
    RDFIndex.setReducerClass(Reduce.class);
    RDFIndex.setMapOutputKeyClass(PairOfStrings.class);
    RDFIndex.setMapOutputValueClass(Text.class);
    RDFIndex.setOutputKeyClass(PairOfStrings.class);
    RDFIndex.setOutputValueClass(ArrayListWritable.class);
    FileInputFormat.setInputPaths(RDFIndex, new Path(inputPath));
    FileOutputFormat.setOutputPath(RDFIndex, new Path(outputPath));

    long startTime = System.currentTimeMillis();

    RDFIndex.waitForCompletion(true);
    if(RDFIndex.isSuccessful())
      LOG.info("Job successful!");
    else
      LOG.info("Job failed.");

    LOG.info("Job Finished in " + (System.currentTimeMillis() - startTime)
        / 1000.0 + " seconds");

    return 0;
  }

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