**Documents and Data on the Web Coursework 2**

1. Introduction

This report presents the decisions taken while choosing a design for an indexing application for RDF data and analyze the advantages and disadvantages of the final result.

1. Design

People tend to search for something specific they remembered from a movie or for something they want to learn more about. In most cases, it will be impossible to search for a specific page to find the title of the article on that page. Also, people want to get additional information for the resource they are receiving. We know that in a RDF triple {subject, predicate, object}, the object is one of the many specified terms in the given source/subject and we know that this object can be found with a given predicate. That is why, I decided to create an inverted index (term -> resource), where terms are extracted from the object and the resource is given by the subject of the RDF and the predicate that describes the relationship between the term and the resource.

1. Terms

Several methods for extracting specific terms from objects are implemented:

* 1. **URIs containing additional information**

Few of the objects contain a date or other datatype that is concatenated with the URI of the datatype. One such example is 121^^http://www.w3.org/2001/XMLSchema#integer. In such cases, we are leaving only the term (removing the URI part), because when searching for the specific date, we rarely write where the datatypes (such as seconds) are stored. The term from the example above is 121.

* 1. **Normal URIs**

It is rarely seen for a user to search the URI of the page he is looking for, except if it is not one of the commonly used ones such as [www.google.com](http://www.google.com), because it is difficult to remember the whole URI of the page. In order to extract the term from the URI, we need to get the last part of the path of the URI, to convert all ‘\_’ symbols to blank spaces and to remove unnecessary endings such as ‘.jpg’. For example, <http://dbpedia.org/resource/Mark_Kirkland> results in the term “Mark Kirkland” and <http://upload.wikimedia.org/wikipedia/commons/d/d7/Al_Jean_by_Gage_Skidmore.jpg> results in the term “Al Jean by Gage Skidmore” . We do convert underscores because they represent blank spaces in the URIs and we do remove unnecessary image ending because we want to create terms that enable users to find additional resources such as images in pages where the specific token is not used but it is implied.

* 1. **Objects containing text**

When we are having a text as an object, we remove language tags, because the language is implied from the term. In addition, as stated in the course, Google search queries with length at most 5 words form approximately 90% of the searches. This is the reason why few operations are performed on a text that is longer than 5 words to reduce the length of terms:

* + 1. **Tokenization**

The text needs to be split into several tokens in order to enable users to find this resource by searching only part of the text. Single-word terms are used for splitting the text. Such splits may enable the index to have terms for someone’s name, as well as terms for just part of the name. For instance, in the Simpsons data set we have the tokens “Matt Groening” and “Matt”, where the second token is part of the first one. This operation enables users to search for a sub-term they remember (such as a surname) if they do not know the entire term as well as the full term.

* + 1. **Phrases**

The only multi-terms that are left are phrases surrounded with brackets. Such terms represent quotes that may be frequently searched. For example, “22 Short Films about Springfield” may be searched a lot, because is the title of one of the episodes of the Simpsons’ series.

* + 1. **Stopwords**

After tokenizing the long texts, the stopword list from the first exercise is used to remove stopwords such as ‘a’. We do this process to remove terms that may produce a large number of misleading results.

1. How we represent resources

Resources that are stored in the index are tuples. A tuple contains the URI link leading to the source page and the role of the term in this resource. Each index may have several tuples, representing the different usages of a given term. The URI is the subject of the RDF and the description is the predicate of the RDF with operation Normal URI(explained above in 3b) applied on it. Usually, users interpret text easier than a URI and for that reason, the operation from 3b was performed.

The described resources enable users to understand how the term is used in a given article. Having several tuples allow us to see the different purposes of the term that we are looking for. In addition, the description part of each resource enables a “second” layer of searching, if we are looking for a specific role of the term. In such a way, the resources can be reduced and only relevant URIs will be returned.

1. Search availability
   1. **What searches are supported**

The index enables a big range of searches. A user may search for dates, numbers, titles, image names, phrases and everything that is in text form. The search enables us to understand the different roles of a term. For example, in the Simpsons data set, we can observe that “Lance Kramer” is both a commentary and a director.

* 1. **Unsupported searches**

The index does not support URI searching. However, if a user wants to search for a URI, he or she can search only the last part of the URI path to get the information that he or she is looking for. Additionally, searching for specific images (with ‘.jpg’ included) will be unsuccessful. We can go around that search by removing the ‘.jpg’ ending.

Fragment of output:









