Web Search Engine – final project report, g16

Knowledge Graph

* ***Motivation***

We have gone through all fundamental infrastructures of web search engine during the whole semester so far, now we are trying to get the useful information from the raw data and well organized them to be queried and utilized by the users. We want to structurally learn knowledge from the web context.

* ***Related Works***

1. Preview: Check the features and formats of Wikipedia corpus to see where we can take advantage with and how to get access to all Wikipedia pages.
2. Crawling: Based on some seed lists, parse articles within a certain distance.
3. Parsing: Parse the infobox and tables, if any, of individual Wikipedia articles and most of special list. For example the List of longest bridges in the world. All parsers get tons of structured “relations” to be indexed.
4. Indexing: Extract important and precise information from the relation as the index, which will be used when serving.
5. Serving: Process the query (more work if it is unstructured) and go through our index base to constitute response.

* ***Architecture***

1. Crawler: Crawler.java and some supporting classes
2. Parser: Two mina parts, List parser and Infobox parser and some supporting object.
3. Indexer: Indexer.java and some other help classes.
4. Nlp: Used when dealing with real sentence like list title and handling unstructured query.
5. Query: make instance of the query and handle its language structure.
6. Server: serve the user
7. Library and API: Stanford-corenlp (<http://nlp.stanford.edu/software/corenlp.shtml> ), jsoup (<http://jsoup.org/> ), MediaWiki (<https://www.mediawiki.org/wiki/MediaWiki> )

* ***Implementation***

1. For the parsing part, since most of our targets are formatted in xml/html, an open and a close tag/parenthesis are required. Based on this feature, we used stack to check if we have got an entity and if we reach the end of the content.
2. We tried to learn the pattern from the title of a list say “List of longest bridges” to know which column (in this case, length) we should extract. We assume the column, which is in order, is out target because of the superlative adjective here. Unfortunately, because of the relatively small size of our samples, we couldn’t be so sure about the effectiveness of this approach. But that’s how we implement.
3. Reverse relation gives us some precious information, so we have to utilize them carefully. For parsing infobox, we assign the infobox type as the property of every reverse relation. For example, the infobox type of Barack Obama is officeholder and he has a relation: Barack Obama (entity)->officeholder(type)->spouse(property)-> Michelle Obama(entity). Then the reverse relation here is Michelle Obama (entity) -> spouse(type)->officeholder(property)-> Barack Obama(entity). Basically, type and property is interchangeable in our model and even in Wikipedia. They don’t use verb or adjective to describe a relation.

For a list, such as the list of volcanoes in Spain, the idea is the same but we get type from entity’s infobox. For example, Fuerteventura (entity)->volcano (type)->country (property)-> Spain (entity), in which country is extracted from Spain’s infobox.

1. Indexer analyzes above relations and store information a map, which maps entity to their relations. When handling the query, we just check that posting-list-like record and build our response.

* Usage

1. Compile all .java files
2. Execute wseproject.server.KGServer
3. Structured query: <http://localhost:25816?entity=karachi&property=population>
4. To some degree, unstructured query: <http://localhost:25816?query=capital+of+venezuela>
5. Our engine supports union and intersection. You can achieve those by putting “or” and “and” in the query. For example, “movies of Matt Damon and Brad Pitt” will give you a list of movie stared by both of them.

* Evaluation
* Improvement

1. To allow better user experience, we applied loose matching when we are handling the query. That is, a query “population” will get not only “population” but also “population density” as the result, which lower the precision. Also we will have to do is applying strict matching. But in this case, we have to make sure we are doing good stemming, synonym handling and other query/index normalizing stuff.

* Group Member and Contribution

Syed Ali Ahmed: List Parsing, Table Parsing, Indexing, and Server Interface

Yen-Tin Liu: Crawling, Infobox Parsing, Project Report

Hao Xu: Evaluation, Consulting