Information Retrieval Course
CSE 435/535 - Fall 2011
Project 1
Parser & Indexer in C++ for Wikipedia

Project Description: An IR Engine should include the following major components: Parser, Indexer and Retriever. Your first project is to build the parser and indexer which will be used by subsequent projects. It must be coded in C++. One of the purposes of this project is to let you get familiar with C++ STL (Standard Template Library) and also process documents that have Wikipedia style markups. In this project we will be indexing a collection of roughly 100,000 documents belonging to the news and Wikipedia category.

Due Date **Online submission through UBLearn by** 26^{th} **September, 2011 11:59 P.M.**

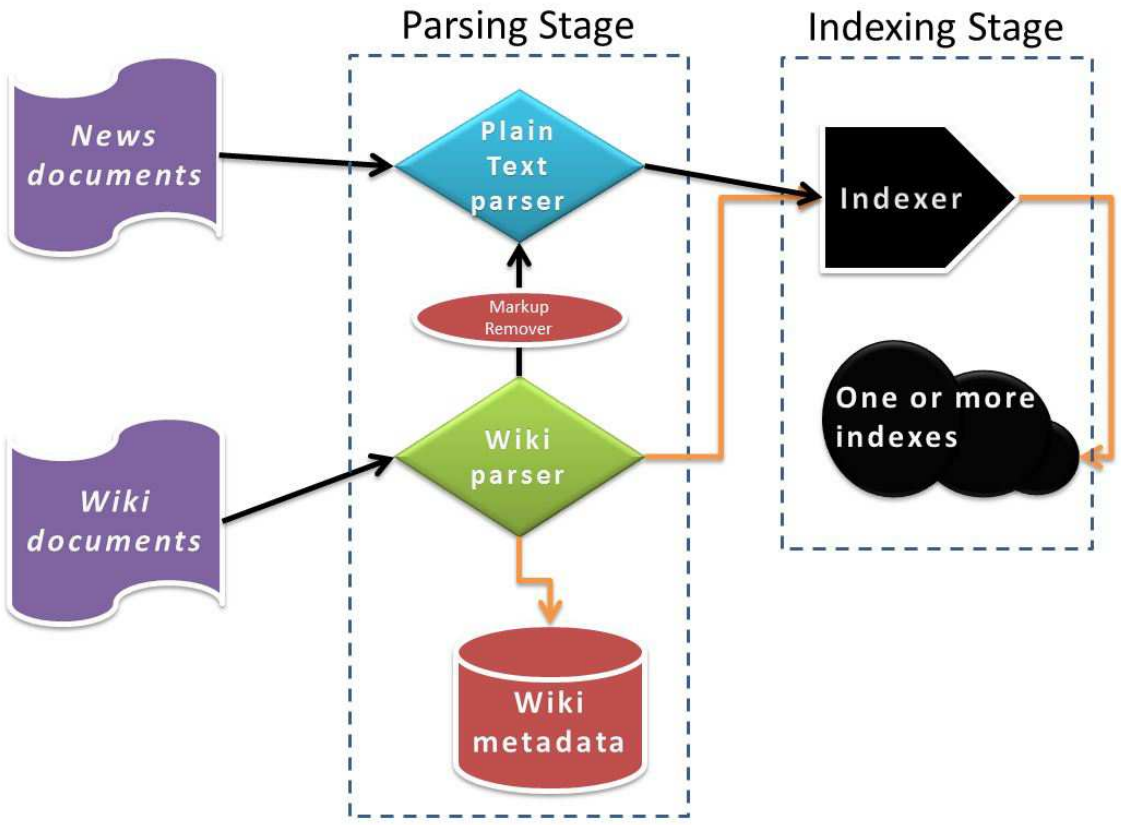

Figure 1: Schematic diagram of project 1

1 Parsing Stage

1.1 Plain Text parser (expected to finish by 12^{th} September)

TODO

Tokenization: Read documents into memory, tokenize to separate words/tokens in the documents, process the tokens and store the "terms" in a suitable STL container. Please read (3.2) for dictionary creation. (Tokenization rules will be discussed in class and recitation)

Document preprocessing steps typically include:

Tokenization to handle numbers, hyphens, punctuation marks, the case of letters (upper/lower), Removal of Wiki markups, Elimination of stopwords, Stemming of the remaining words

1.2 Wiki parser (expected to finish by 16^{th} September)

In this step the Wiki markup files will be parsed to grab some document metadata and the original text will then be fed into the tokenizer and the dictionary generator that you have created in the previous step. (cf. Figure 1).

Although we will be parsing a smaller subset of the Wiki markup tags, for detailed descriptions of wiki markup tags, please visit `http://en.wikipedia.org/wiki/Help:Wiki_markup`. For our purposes, we will be interested in extracting "Author", "Infoboxes" and "Taxoboxes", Section and Subsection title strings and Internal Wiki links. Refer to Table (in the last page) for a quick view. The Wiki parser will help in creating text files corresponding to the original wiki files that stores the following meta information: Infoboxes or Taxoboxes, Catagory, Internal wiki links.

TODO

Simple Parsing: Generate a unique document ID (of type size_t) for each wiki document.
For each wiki document generate a text file named <wiki document ID>_semwiki_meta.txt that looks like

```
<#INFOBOX>
.... dump infobox contents here
<#SECTIONS>
heading_text_1 $ 20 words or less following the heading text 1
heading_text_2 $ 20 words or less following the heading text 2
....
heading_text_n $ 20 words or less following the heading text n
<#LINKS>
link_1 $ link_2 $ .... $ link_K
<#CATEGORY>
category_1 $ category_2 $ .... $ category_M
```

File Dictionary and store: Generate a unique document ID (of type size_t) for each semantic wiki Meta document and add it to the File dictionary (see 3.2). Create a folder named "SemWiki" and store each meta file into it with the filename given above.

→**Note:** *While parsing pay special attention to tags for Infoboxes and Taxoboxes and the tags mentioned in appendix.* Also ignore everything after the *last* "[[Category:xxx yyyy]]" string encountered in the Wiki markup file.

2 Indexing Stage

We need to build a forward index and an inverted index for a larger document collection. The data structures in this project should support the Vector Space Model; this means that your indexing module should be capable of calculating **TF-IDF** weights for all the terms in the collection. Another requirement is to distribute the forward index and the inverted index over a number of files instead of keeping it in one file.

TODO

2.1 Forward indices (expected to finish by 22^{th} September)

Store the classification label of the document in the forward index - 0 for news documents and 1 for wiki document.

1. If the document is a wiki document:

   a) Add the document ID of the smaller "semwiki" document. (This should indicate the actual filename of the semwiki document from where information must be read)

   b) Add the terms from the corresponding smaller "semwiki" document in the forward postings list of the wiki document

   c) This postings list will consist of term IDs, and each term ID should know from which part (i.e. Infobox, Title, Link, etc.) of the semwiki document it came from and its position (either byte wise or token wise)

2. If the document is a "news" document, do not store any postings list at this point.

Flush the forward index to disk whenever you have processed 10,000 files. The forward index barrels should be named as index.fwd.x where "x" is the barrel number $\in$ {000; 001; 002}. →**Note:** You should be able to maintain a range of document IDs that exist in a barrel and hence figuring out which barrel contains a particular document ID becomes easy.

2.2 Inverted indices (expected to finish by 24^{th} September)

Step 1 - Accumulation: Keep accumulating the termID and docID pairs in memory using an efficient inverted index data structure until the total length of the postings lists reach 50,000. Write this inverted index to the disk by splitting into separate files/barrels (e.g. elements of the index whose words start with alphabets "a-f" will be in one file, those starting with "g-k" will be in second file, and so on). File name is described as below:

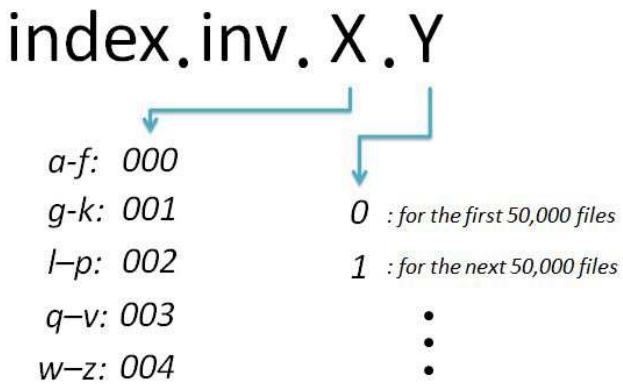

Figure 2: index name explanation

You need to have the "indexer configuration file" file in order to control the number of partitions. You are constrained to create not more than 100 barrels. (You could also have barrels like a→index.inv.001, b→index.inv.002 etc. However, the more barrels you have, the more fragmented your filesystem will be. [Hint: Some query terms have a longer postings list than others])

Step 2 - Optimization: Merge each block of records index.inv.X.Y for all values of Y and for each X $\in$ {000; 001; 002, ...} as follows:

1. For each "X", create a file named index.inv.x.pstngs
2. For each termID in $X.Y$, retrieve the postings lists, merge the postings lists and write the final sorted postings list into index.inv.X.pstngs.
   By doing this, the inverted index file index.inv.X will now contain <termID byte(or line)-offset-of-the-postings-list-in-index.inv.x.pstngs> (e.g. all the posting list of a-f will be merged into index.inv.000.pstngs)
   →**Note**: The postings list can be sorted by any kind of consistent document ordering (will be discussed in class)
3. Programmatically delete the intermediate index files like "index.inv.001.Y"

Addition: You need to do exactly the same procedure (step 1 and step 2) to build inverted indices for "Catagory" and "Author". (so, the corresponding indices should be: <catagoryID byte(or line)-offset-of-the-postings-list-in-index.inv.x.pstngs> and <authorID byte(or line)-offset-of-the-postings-list-in-index.inv.x.pstngs>). →**Note**: In order to structure files in a neat way, you need to create folders for each of them. (the directory structure should be like this: "/your_team_name/inv_index/term_inv_index", "/your_team_name/inv_index/catagory_inv_index" and "/your_team_name/inv_index/author_inv_index"). You need to have a flexible datastructure for the posting list in order to store more than just docIDs (e.g. authorID or catagoryID etc). Future expansion will be discussed during the recitation.

3 Miscellaneous

3.1 Data:

The data will be organized in two folders:

News: Documents in this folder have originated from news sources and had been used in competitions like TREC.

Wiki: Documents in this folder are extracted from Wikipedia. These documents have been/are being crawled and fetched beginning September 2, 2010 and onwards.

You will be given sample set from these two sources just to get you started on tokenization and dictionary creation. The full data set is also available to you. But please make sure that your program can pass the sample test set (you can get partial marks from demo if your program fails to run on the full dataset).

3.2 Dictionaries & Maps:

Three kinds of dictionaries and two kinds of maps are needed to be generated during the parsing stage (you need to create a folder named "Dictionary" to hold them). The dictionary can be a simple text file with each line formatted as <Key Value> without the '<' and the '>'. (For example, <Key Value> could be <TermID TermString>)

a) Term Dictionary: Build a "dictionary", which assigns each processed word/token to a numerical ID and keeps this correspondence information. Each unique processed token is assigned to the same unique numerical ID. The dictionary should be implemented such that element lookups are very fast. We will assume that the IDs here will be of data type size_t. (In real world scenarios, it makes sense to assign IDs that are represented as "Big Integers" possibly converted to hexadecimal strings)

b) File Dictionary: You need to keep a Dictionary to map each document name to a unique numerical ID also of type size_t. Write all filenames as a continuous '$' separated string in some text file (as the file path could be very long). Thus, the <fileID filename> pair is changed to <fileID byte-offset-of-the-first-alphabet-of-the-filename-in-file>

c) Term Count Dictionary: You also need to keep a Dictionary to map each unique numerical term ID to its count in the corpus.

d) Link Map: A link dictionary to map pairwised internal link. There will be some (or none) links for each wiki file. You can build the dictionary like <SOURCE, DESTINATION>, where "SOURCE" is actually the wiki file (e.g. A.txt) name without extension. And "DESTINATION" is the link you extracted from the corpus (of A.txt). Please refer to the "Links" section of the table on the last page. More details will be discussed during recitations.

e) Author Map: Do the same procedure with Link Map for "Author". →**Note**: Some wiki files have multiple authors which could be interpreted as many-to-many relationship. In this case, extracting all the one-to-one relationship within it will satisfy our requirement.

3.3 Command line interface:

$ > falcon [-i] <path_to_input_data_dir> <path_to_index_store_dir>
<path_to_dict_store_dir> <inverted_index_config_filename>
→**Note:** the "-i" is the switch that says the program should be in the indexing mode (later we will have "-r" for the retrieving mode). The name of the executable must be **falcon**

3.4 Development environment:

Linux + Eclipse CDT is highly recommended. Environment set up and configuration will be covered during recitation.

3.5 Submission:

Submit a zip file named <your-team-name_project1>.zip using in UBLearn. The contents of the compressed file are as follows:

a) falcon.zip (all source files and a README file) →**Note:** please make sure all your programs at least compile on at least one CSE machine, or scored 0 otherwise

b) report.pdf (must contain the following items)

1. Snapshots of the dictionaries and a sample snapshot of any two wiki meta file generated as outputs.
2. Some statistics found from the corpus that you have parsed and/or tokenized. This will contain information on:
   "How many terms were there in the corpus?",
   "How many raw files were there in the corpus all together?",
   "How many semantic wiki meta files were generated all together?",
   "What are the term counts of the terms that mark the top 95% and the bottom 5% of all the terms in the vocabulary w.r.t term counts?"
3. Contribution of each member (this can be done by showing a list of APIs indicating by whom they're designed and implemented)

Peer evaluation: To be responsible for your teammates, you must assign each team member (including yourself) a evaluation grade from 0 (for he/she contributes nothing) to 10 (very satisfied) with a brief explanation by sending TA email privately. (One email per person. The title must be: <CSE535_eval_yourTeamName_yourFullName>, or be filterd otherwise). This evaluation will play an important role for project scoring.

TAs will download your most recent tar files (up to the submission date) from UBLearns and run it.

A Word about Parsing and Indexing Stage

Spend enough time to read sample code and discuss with your teammates. Bad design will entangle your team for extra weeks.

Your code should be modifiable and extensible so that you can build on this codebase for the following phases.

In general, when a text document is read from disk, it will be stored internally as a list of space separated tokens (not terms which are processed tokens). Thus for a document that is not a wiki document, you can use plain text parser to tokenize. For a document that is a wiki document, it is first parsed to grab semantic metadata for the wikipedia article, and then its token list could be processed through another tokenizer that is derived from the base tokenizer to eliminate remaining wiki markups.

In real life, when a document is crawled, there is some mechanism to identify the major class of the document like whether it is a news document, a wiki article, a blog article, a page generated by submitting a query to a shopping website etc. For our projects, we have two types of documents named "News" and "Wiki". All files under the same directory contain files of the same category. Thus the directory name will give you enough hint as to what kind of document it is.

Although there are two different types of document here, there will be one term dictionary, one file dictionary and one term count dictionary for all the text files. Note that the semwiki files will not be indexed in the usual sense of inverted indexes but will be used in a clever fashion in the future.

For large scale engines, the index runs into GBs/TBs of documents, and hence their indexing process is both memory and disk based. Thus, the design of the Indexer should be easily extended for querying large data sets.

You don't need to invent a new STL container. If you don't like STL, feel free to use any open source data structures like Google Sparse hash tables `http://code.google.com/p/google-sparsehash/`

Infobox	{{Infobox Tennis player |playername= |nickname= ''Boom Boom'' ''The Lion of Leimen'' |country= {{flagicon|Germany}}[[West Germany]] (19831990) {{GER}} (from 1990) |residence= [[Schwyz]], {{CHE}} |datebirth= {{birth date and age|df=yes|1967|11|22}} |placebirth= [[Leimen (Baden)|Leimen]], {{flagicon|Germany}}[[West Germany]] |height= {{height|m=1.90}} |weight= {{convert|85|kg|lb st|abbr=on}} |turnedpro= 1984 |retired= June 30, 1999 |plays= right-handed; one-handed backhand |careerprizemoney= US $25,080,956 * [[ATP_Tour_records#Earnings|5th All-time leader in earnings]] | tennishofyear = 2003 | tennishofid = boris-becker |highestdoublesranking= 6 (22 September 1986) }}
Taxobox	{{Taxobox | name = Cucumber | regnum = [[Plant]]ae | divisio = [[Flowering plant|Magnoliophyta]] | classis = [[Magnoliopsida]] | ordo = [[Cucurbitales]] | familia = [[Cucurbitaceae]] | genus = ''[[Cucumis]]'' | species = '''''C. sativus''''' | binomial_authority = [[Carolus Linnaeus|L.]] }}
Sections and subsections	==Section headings== ''Headings'' organize your writing into sections. The Wiki software can automatically generate a table of contents from them. Start with 2 'equals' characters. ===Subsection=== Using more 'equals' characters creates a subsection. ====A smaller subsection==== Don't skip levels, like from two to four 'equals' characters.
Links	London has [[public transport]] New York also has [[public transport|public transportation]] [[kingdom (biology)|]] [[Wikipedia:Village pump|]] [[Wikipedia:Manual of Style (headings)|]] A [[micro-]]<nowiki>second</nowiki>

Table 1: Quick view of relevant wiki markups