**CSCI 572 Report**

Poojan Jhaveri, Ruchi Patel, Puranjay Rajpal, Eisha Jain, Ritesh Chandana

**Nutch Politeness**

A good approach to start crawling the given websites is to ensure etiquette that is maintaining politeness by not disturbing a website's performance.

* Agent names used:

1. My Nutch Spider
2. Puranjay
3. RPRPE Agent CSCI 572

No. of request made per second = 10 (default value already set in nutch\_default)

No of threads fetched= 10

By changing the properties of nutch\_site.xml we handled nutch politeness.

We made following changes in nutch\_site.xml:

Property name: http.agent.name; Value : RPRPE Agent CSCI 572

Property name: http.agent.host ; Value : localhost

Property name: fetcher.threads.fetch ; Value : 10

Property name: fetcher.maxNum.threads ; Value : 25

**2.URL Filtering:**

The file conf/regex-urlfilter.txt provides regular expressions that allow nutch to filter and narrow the types of web resources to crawl and download.

Changes that were made in regex-url.txt to deal with URL Filtering are:

1. <http://gcmd.gsfc.nasa.gov/KeywordSearch/Home.do?Portal=amd&MetadataType=0>

**Filter**: +^http://([a-z0-9]\*\.)\*gcmd.gsfc.nasa.gov/KeywordSearch/

1. <http://nsidc.org/acadis/search/>

**Filter:** +^http://([a-z0-9]\*\.)\*nsidc.org/(data|acadis)/

1. <https://www.aoncadis.org/home.htm>

**Filter**: +^https://([a-z0-9]\*\.)\*aoncadis.org/

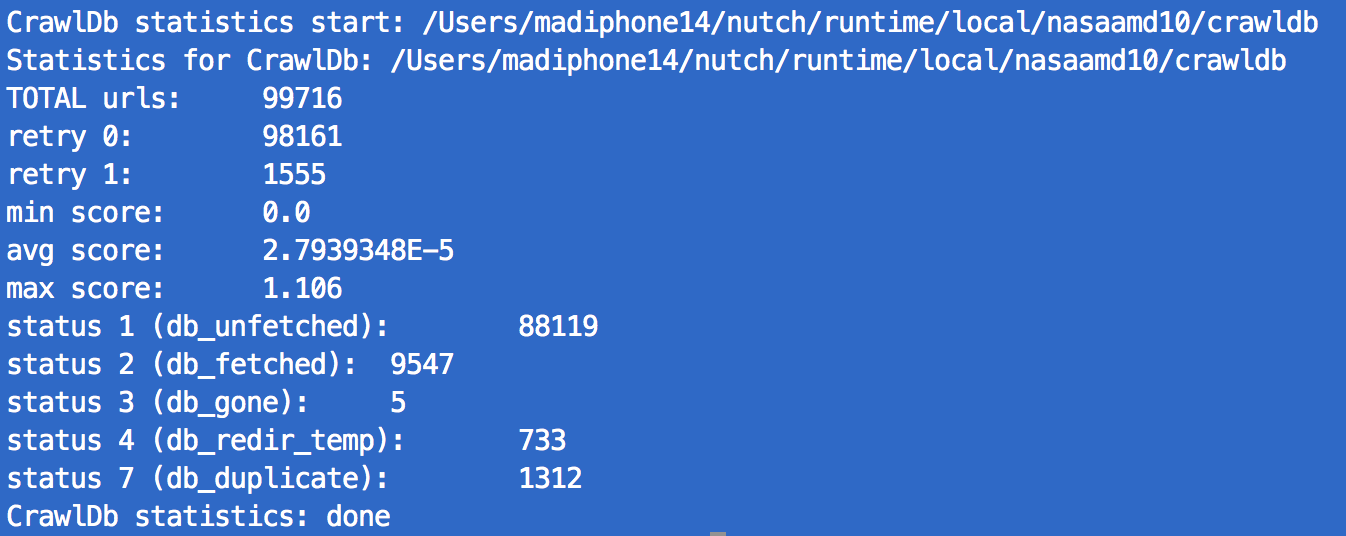
**Task 2:**

After successful crawling of all the three sites, we merged the segments and read the data to create dumps.

We developed a java program, which was run on crawled data to classify MIME types. Hadoop log was used to get 100 URLs, which were difficult to fetch. We crawled at level 10 in order to get access to dark web as well as find more specific mime types.

The crawl statistics for the three repositories are as follows:

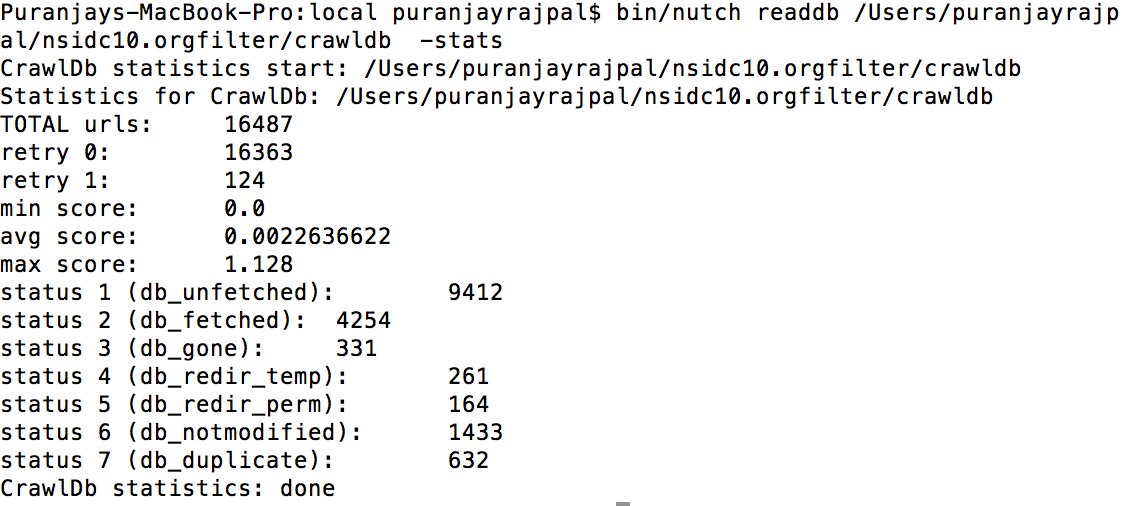
NASA AMD:



ACADIS:



NSIDC:



100 URLs and MIME Types are attached in separate files.

**Task 3,4,5**

We were unable to install tika and selenium. We have explained the reason for this in the readme file. The main reason was it took us as a while to get the right URL filters and once finished that, we got stuck up in installing selenium and updating tika. We used up all the available sources: emailed in nutch mailing list, contacted TAs and professors but still we were not bale to install selenium. Finally when we installed selenium after getting patch for mac and degrading firefox to version 29, professor instructed everyone to stop crawling. Hence, we were not able to get any data for the second crawl.

**De-duplication Algorithms**

Since we were not allowed to crawl again anymore. We developed dedup algorithms and ran it on our crawl 1 data. So for this reason, we did not try and create a url filter plugin because crawling wasnt allowed. But we have made our algo so that it can be easily ported to url filter plugin. We have tried making the url filter plugin by adding the dependencies for the plugin in plugin.xml,build.xml,ivy.xml.We edited the nutch-site.xml as well in order for nutch to use our plugin. We also had to edit the src/plugin/build.xml to add the <ant dir="[plugin-name]" target="deploy" />

This is added in order to make sure that ant deploys and compiles our plugin.

**Exact Duplicates Algorithm:**

//Using simhash and fnv-1a-64

for each line in the parsed-content(parseText and parseData each (one at a time)):

//Calculate SimHash.

1. for each word in the string

1.find FVN-1a-64 hash for each of the words.

2.change the bit of V according to the hash obtained using findFvn1a64(word).

II. Find value of simHash from updated V by checking the sign of V

III. Return the simHash for the current URL content

IV. Put the URL,simHash of the content of that URL in a map

//m is the map of URL,simHash of parseData

//m1 is the map of URL,simHash of parseText

//dupurls: set of duplicate URLs

/\*Taking parseData and parseText together and getting only one simHash gave rise to more number of collisions and false exact duplicates. Thus calculating them separately and AND'ing them.\*/

V. for each key in the map m,m1

for each nextKey in the map m,m1

int rslt = findHammingDistance(m.get(key), m.get(nextKey));

int rslt1 = findHammingDistance(m1.get(key), m1.get(nextKey));

//if both parseData **AND** parseText match

if(!(rslt & rslt1)){

dupurls.add(key);

}

IV. End

Calculating the Fvn-1a-64 of each word:

long h = 0xCBF29CE484222325L;

long prime = 0x100000001B3L;

for (int i = 0; i < s.length(); i++ )

h = ( h ^ s.charAt(i) ) \* prime;

return h;

File attached: FindExactDuplicates.java

To execute the code:

javac FindExactDuplicates.java

java FindExactDuplicates <path of parseData dump file> <path of parseText dump file>

References:

http://www.isthe.com/chongo/tech/comp/fnv/

http://www.titouangalopin.com/blog/2014-05-29-simhash

<http://stackoverflow.com/questions/10587506/creating-a-hash-from-several-java-string-objects>

Results: (only on crawl1, crawl2 and crawl3 data not available)

In crawl1 data, only text is parsed, thus different pages with same text but different images or pdf or any other mime type are considered duplicates.

Also the parseText that we get and the actual content on the web page are different, resulting in more number of false positives.

Number of exact duplicates on crawl 1 data(with false positives)(NSIDC): 2173

Number of Exact duplicates with false positives:(AONCADIS) is : 332

Number of Exact duplicates with false positives:(NASA AMD) is : 190

**Near Duplicate Algorithm**

For Near Duplicates the above algorithm is run where, the simHash values within the range of 5% are accepted. The exact duplicates are subtracted from these.

File attached: NearDuplicates.java

To execute the code:

javac NearDuplicates.java

java NearDuplicates <path of dump file dat contains only URLs and parsedata> <path of the file that contains only URLs and parsetext>

Results for crawl1 for near duplicates are as follows:

Number of Near duplicates with false positives(NSIDC) is : 4295

Number of Near duplicates with false positives(AONCADIS) is : 345

Number of Near duplicates with false positives(NASA AMD) is : 199

**Extra Credit**

We tried doing the extra credit. We were able to regenerate the original image files from dump by running :

Bin/nutch dump

We even tried installing tika python. But unfortunately, we got this error :

ld: internal error: atom not found in symbolIndex(\_\_ZN7JNIEnv\_13CallIntMethodEP8\_jobjectP10\_jmethodIDz) for architecture x86\_64

clang: error: linker command failed with exit code 1 (use -v to see invocation)

error: command 'g++' failed with exit status 1

**Other Answers:**

Since, we were not able to install tika and selenium to nutch, we are unable to say anything about the data after empowering nutch.

We have noted the MIME types that we obtained from nutch crawl. Since, we got the data after crawling at depth 10 and more, we were able to get science data. We know this because, we got mime types :Iso9139,Google earth file type, geological spatial data, rss atom data, octet-stream.

Hence we are satisfied with the crawl data and mime types we got as those were relevant to the arctic subject. However, we wish we had enough time to crawl at even deeper depths.

Yes, we feel we have achieved good coverage of the three repositories as we have covered and fetch more than 15k urls successfully and more than 100k urls in total.

What we like the most about Nutch is that it is an open source software and is readily available to all the users. It is very easy to install. We just have to write few commands to install it.

However, there is no proper updated documentation of nutch available.

As beginners, we faced lot of problems to understand the functionality of nutch as there was not much information available online.

Tika was difficult to install. No proper instruction was present anywhere on the internet about different versions and OSes.

Overall it was educative and challenging assignment. We learnt a lot about how the crawl and crawl bot works.