CSCI 572 : Assignment 1

*http.agent.name = CS572Crawler (All members).

Q1. a. Identify and make changes to Nutch configuration to deal with politeness

The following additions were made to conf/nutch-site.xml

A. fetcher.server.delay = 2.5 secs
B. Fetcher thread count & number of threads per queue was kept at default.
C. Number of requests made per second = 1

Q1. b. Identify and make changes to Nutch configuration to deal with URL filtering.

A. Removed MIME such as jpg, png, gif, ico, sit, zip, ppt, mpg, xls, jpeg, gz, wmf, bmp, mov.
B. Allowed characters like ?*!@= to occur in the url.
C. Restricted the crawl to the top level domains of the 3 repositories only.

Q1. c. Create team identification information and label your Nutch bot via Nutch configuration

The following additions were made to conf/nutch-site.xml

A. http.agent.name = CS572Crawler
B. http.robot.agents = CS572Crawler,*
C. http.agent.url = http://www-scf.usc.edu/~shantanb/cs572_bot.html
D. http.agent.email = shantanb@usc.edu

Q2. a. Develop a program or script to iterate through your Nutch crawl data and classify what MIME types you encounter.

- Please refer to getMime.py for the script.
  Note: Usage of the script: python getMime.py <Path_To_Crawldb_Folder>

Q2. b. Identify 15 different MIME types encountered while crawling.

- Please refer to 'MimeTypes.txt' placed in the 'REPORT' folder.

Q2. c. Provide a list of 100 URLs that were difficult to fetch and give reasons for the same.

- Please refer to the 'FAILED URLS WITHOUT TIKA AND NUTCH SELENIUM' section in 'FailedURLs.xlsx' placed in the 'REPORT' folder.
  ***Note**: .xlsx file was used to maintain formatting.

Q2. d. Provide crawl statistics from your crawls of each repository.

- Please refer to the 'STATISTICS WITHOUT TIKA' section in 'Stats.txt' placed in the 'REPORT' folder.

Q5. a. Provide a list of 100 URLs that were difficult to fetch and give reasons.

- Please refer to the 'FAILED URLS WITH TIKA AND NUTCH SELENIUM' section in 'FailedURLs.xlsx' section in placed in the 'REPORT' folder.
  *Note: .xlsx file was used to maintain formatting.

Q5. b. Are the URLs provided above present from Q2. d?

- Most of the URLs from Q2. d were present.

Q5. c. Did the enhanced TIKA parsing assist with that?

- Yes, enhanced TIKA parsing assisted with fetching some of the URLs that could not be fetched without TIKA.

Q5. d. Deliver updated crawl statistics from each repository of your crawl.

- Please refer to the 'STATISTICS WITH TIKA' section in 'Stats.txt' placed in the 'REPORT' folder.

Q6. a. Deduplication algorithm for exact match using extracted text

Algorithm:

A. Initialize an empty hash map whose 'keys' will be the hash of the url content and 'values' will be the url itself.
B. Initialise an empty string, say 's'.
C. Read the content from the file line by line.
D. For each line in the content,
    a. i = 0
    b. while i is less than the length of the line
        i. append line[i] to s
        ii. increment i by 5
    c. endwhile
E. endfor
F. compute the 'hash value' for s using md5 cryptographic hash function
G. If 'hash value' is present in hash map,
    a. mark the URL as a duplicate
H. Else,
    a. add the hash value and the url to the hash map
I. Endif

Note: 1. The string 's' will be composed of random characters from the content, i.e. characters at positions that are multiples of 5 in every line.
2. Please refer to the 'Running The Exact Duplicates Algorithm' section in readme.txt for instructions on how to run the code on the dump.

Q6. b. Deduplication algorithm for near duplicates using extracted metadata

Algorithm:

A. Initialize an empty hash map whose 'keys' will be the fingerprints (long int) of the url metadata and 'values' will be the url itself.
B. Initialise an integer array of size 32, say V.
C. Read the content from the file line by line.
D. For each word in each line,
    a. Calculate murmur hash value of the word
    b. For each bit i, in the hash value, if it is 1, then set V[i] += 1, else set V[i] -= 1.
E. Endfor
F. Initialise int 'fingerprint' = 0 and 'mask' = 1
G. i = 31
H. while i >= 0,
        i. if V[i] >= 1, fingerprint = fingerprint | mask
        ii. mask = mask << 1
        iii. decrement i by 1
I. endwhile

J. for each key in hash map,
    a. if hamming distance between key and fingerprint < 3 and >0,
        i. mark it as a near duplicate
    b. Else
        i. add fingerprint to the hash map
K. endfor
L. Endif

Note: 1. This algorithm runs on extracted text instead of extracted metadata, as discussed with Prof. Mattmann.
2. Please refer to the 'Running The Near Duplicates Algorithm' section in readme.txt for instructions on how to run the code on the dump.

Q7. a & b. Construct two URLFilter plugins for Nutch

The folder URLFiltersPlugin7a-b contains urlfilter-exactdedup for exact duplicates and urlfilter-neardedup for near duplicates. ***Note**- Please check the readme.txt for using these plugins.

Q7. c. Identify the number of duplicate URLs present from 6a

- AMD: 474, ACADIS: 3, ADE: 67.

Q7. d. Identify the number of duplicate URLs present from 6b

- AMD: 534, ACADIS: 250, ADE: 83.

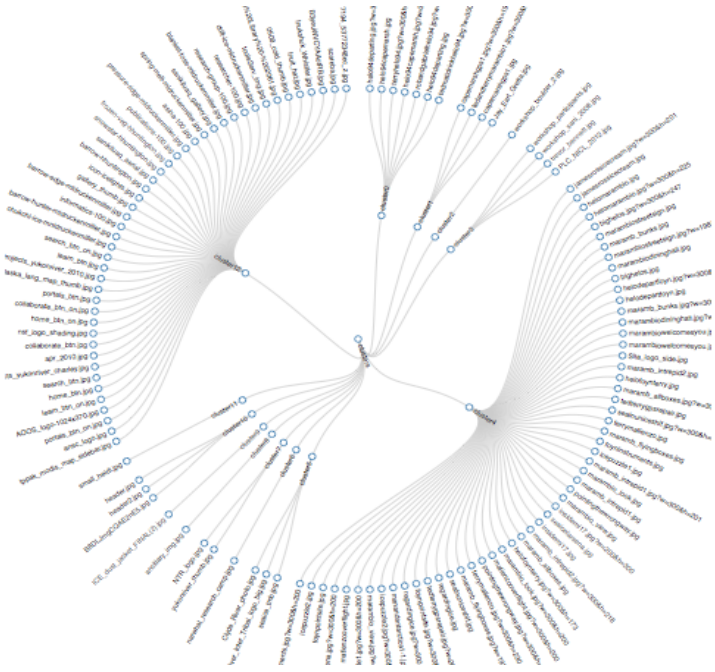
Q7. e. Deliver your updated crawl statistics from each repository that you crawl

We could not crawl after developing the Nutch plugins because :

1. the websites went down by the time we developed the plugins.
2. we were asked not to crawl from this point in time.

Q8. b. Are there any interesting clusters?

- Yes, we have created a Flare Dendrogram with 12 clusters.



Q8. c. Can you explain the clusters that you see?

- The clusters in the Flare Dendrogram consist of images that are clustered based on how similar they are to each other. For example:
    a. Cluster 0 consists of seven images (.jpg), all of which are related to helicopters.
    b. Cluster 1 consists of four images that are related to a particular location (Cape Marsh).

Q8. d. Develop a pull request for the Github repo to generalize it into

- We established a pull request, with the ID: **pranshu1992,** to the Etllib Github repo.

Why do you think there were duplicates?

- There were duplicates probably because the content of these pages was quite important and referred to from a lot of other pages.
- We also observed that few websites were using web pages with similar content even though they belonged to different departments in the organisation.

Were they easy to detect?

- Exact duplicates were easier to detect as they had the exact same hash values.
- Near duplicates were not that easy to detect as we had to modify and revise our algorithm several times to end up with a perfect match of the hash function used and the threshold value (hamming distance). It became a lot tougher with lesser content.

Describe your algorithms for deduplication.

a. **For exact matches -** As explained in Q6. a.
b. **For near duplicates -** As explained in Q6. b.

How did you arrive at the algorithms?

- For the algorithm to detect near duplicates we were inspired by the following two papers: 'Detecting Near-Duplicates For Web Crawling by Gurmeet Singh Manku, Arvind Jain and Anish Das Sharma' and 'Probabilistic Near-Duplicate Detection Using Simhash by Sadhan Sood and Dmitri Loguinov'.
- For the algorithm to detect exact duplicates we referred to the lecture slides.

What worked about the algorithms? What did not work about the algorithms?

- The algorithm to detect near duplicates involved a hash function, tokenization of the text into words and a stopping condition (Hamming Distance threshold) that worked for us. However, classifying smaller contents as duplicates was difficult.
- The algorithm to detect exact duplicates involved a simple comparison of hash values which worked well for us.

Describe the URLs that worked and why?

- URLs such as 'nsidc.org' and 'nsidc.org/cryosphere' worked as they had neglible difference in the content.

What MIME types did you retrieve from these websites? Were they mostly web pages?

- We retrieved MIME types such as: text/html, text/x-csrc, etc. (complete list in 'MimeTypes.txt' in the 'REPORT' folder) most of which were web pages.

Did you get science data?(e.g. HDF and NetCDF and GRIB files) If not, then why?

- No, we did not get science data as we did not crawl for the appropriate number of rounds (approx. 25 as suggested by Prof. Mattmann).

Did the Selenium plugin help?

- The Selenium plugin could not reach the dark web with the number of rounds in our crawl, i.e. 5.

Did you get more data after installing the Tika updates and recrawling?

- Yes, we got more data after installing Tika updates and recrawling.

Do you think you achieved good coverage of the 3 repositories?

- No, we did not achieve a good coverage of the repositories due to less number of rounds in our crawl.
- Also, we were later instructed to stop our crawls.

Your thoughts about using Apache Nutch and Apache Tika - what was easy about using them? What wasn't?

- Apache Nutch and Tika are great tools for crawling the web and analyzing the content retrieved.
- The plugin model of nutch helped us in developing and customizing our own filters.
- Configuring the crawl properties did not require a rebuild of nutch.
- All in all, using nutch gave us a deeper insight in crawling web pages.

- **What was easy:**
  - Installing nutch
  - Configuring it
  - Setting up the crawls
  - Easy to use command line features

- **What wasn't easy:**
  - Integrating nutch with other tools such as Tesseract, GDAL, Solr and Selenium
  - Developing the URLFilterplugins
  - Using Nutchpy