TASKS

3.a. How many job files were produced? (with our parser)

12,461,832 individual job files were produced with our parser.

4.a. How many job files were produced? (with ETLlib parser)

We have observed that the number of json job files produced by ETLlib (without unique -u parameter) is exactly the same as what we got int **3.a (with our parser)** i.e 12,461,832 individual job files.

5.a. Decide what job information can be used to deduce uniqueness

Algorithm for Detecting Duplicates

This task has been quite challenging given that the data is in a foreign language and the size of the data is enormous. We understand that these are the two factors we have to deal with a typical real world application. We tried to apply deduplication at multiple levels as described below

- We didn't apply our deduplication algorithm upstream for the following reasons.
    - We incorporated TSV Parsing capability into tika. So we had to call "java -jar tika-app-1.6.jar" from our crawler python script. The problem we encountered here is that there is no way to maintain a "global" data structure such as a hashtable to remember the unique job postings across tsv files. Although this seems to be a better approach, had we used our own main program to call the TSV parser. But in the later case, we are kind of bypassing Tika a bit and we wanted to learn and see how easy it is to integrate a new functionality to Tika. So we went with deduplicating data at the downstream approach.
    - Generate md5 for each json file. We learn that with any hashing technique such as md5, if the hash values are different, then the inputs have to be different but if the hash values are same, this doesn't necessarily mean the inputs will be same due to the pigeonhole principle. For this reason we have a second layer of filtering duplicates by comparing the fields: **url, contactPerson, postedDate.**
    - The other reason is to also identify near duplicates. We define near duplicates as recordA is a near duplicate to recordB if they have same value for the tuple: <url, contactPerson, postedDate> and differ in at least one other field.
- How much time the deduplication program took to complete.
  24 hours approximately with the following system configuration.

  on Mac OSX
  1. 8 GB RAM

2. 2.5Ghz
3. 256 SSD

Please note that the reason for taking such a long time is that we call our parser through tika and we were able to generate around 12 million json files and our deduplication algorithm runs through all the json files to find unique job postings through our deduplication algo as mentioned above

5b.Integrate deduplication into your programs from #3 and #4

276,892 unique job listings were found using ETLlib.

5c.How many job files were produced when deduplication is enabled, and why?

277,289 json job files were produced. We found that 97.78% of the data is duplicate. Based on the unique combination of three fields URL+contactPerson+postedDate assuming that the MD5 hash for individual json files is unique for each posting. The detailed steps for deduplication is mentioned above. Our assumption is that if the above mentioned fields are same but rest of the other fields are not identical then such records are considered as near duplicates and have been removed.

6. crawler run on full data-set of Employment jobs to produce individual JSON job files

a. How many job files were produced (no deduplication)?

We got 119,453,091 individual json files after running our tsv parser on the 40 GB data. This has been done in chunks on four laptops by distributing the tsv data.

b. How many job files were produced (with deduplication on)?

We ran our deduplication program on the JSON files obtained in the 6.a and found that there are 26,517,434 unique job postings.

<u>Why do you think there were duplicates?</u>

- Same data might have been crawled multiple times by the crawler at different points of time.
- There might have been multiple data sources for the crawler and same job posting might have been present in each of the data source.
- User might have mistakenly posted the same details multiple times thereby increasing duplication.
- Users might have shared a job posting on the other websites.
- The same data might have been present in various formats such as PDFs, Web documents etc when the crawler is run.

Description about simple crawlers.

The heavy lifting is being done by our TSVParser class and JSONContentHandler classs at the upstream and deduplication program at the downstream. Our crawler is just coordinating these three prices by iterating over all the tsv files. This task took about 7 hours time on the same Mac OS (configuration mentioned above).
We could have improved this further by distributing crawler processing part or the input files or both.

Takeaways from this assignment.

- Think really hard about the use cases and thorough testing on handful of input files before handling enormous data. Our experience: generating an extra comma in one of the fields in the JSON data. This has been identified since we ran our crawler on a test data first.
- Encoding as part of cleansing data so that JSON structure is not disrupted.For example there was data which contained some of the reserved words like “&” which when parsed by SAX parser produces exception as it has been reserved for entities.Hence some amount of cleaning was required.

What Worked?

- We could successfully implement a parser for processing TSV data and incorporate it in TIKA.
- We have also implemented the JsonContentHandler to convert XHTML produced to JSON.
- We could test run on the data set given and observe the results.
- We identified reasonable number of duplicates in the given dataset using our deduplication algorithm.

What didn’t work?

- We couldn’t find a better way to handle 40 GB data other than increasing the processing power by running our algorithm on a cluster and probably using MapReduce for our crawler
- We thought of inserting each job posting in a database to identify exact duplicates. We did think about approaches storing in database (disk, B+ trees at backend) vs hash table (in memory, collisions).

Apache Tika ETL Lib

- We have ETL lib without tika on Mac and ran this application on input size to produce individual json job
- From each TSV we have written each line into a temporary csv file and and passed this to tsvtojson command which gives individual json as output
- Our De-Duplication algorithm goes through all the individual json to find unique jsons
- We have also fixed a bug in ETL lib’s deduplication algorithm. Attaching the link to Merge

https://github.com/chrismattmann/etllib/pull/27

Project Details:

- The link for the code is located at:https://github.com/Monil200/csci572HW1
- The submitted zip file also has the source code. The instructions can be found in the Readme file in the zip file.