Hammad Ahmad Cooper Baird

<u>**Overview, Architecture, and Implementation:**</u>

The overall purpose of this assignment was to become familiar with Hadoop/MapReduce on a cloud platform: either Amazon Web Services (AWS) or Google Cloud. Because we were already given Java code for a WordCount application that uses MapReduce, the actual coding portion of this assignment wasn't terribly involved or time consuming. We used the already existing code for MapReduce in WordCount and made alterations to create an inverted index (although the code for the mapper required more tweaking). The largest challenge of this assignment was properly setting up a cloud platform to run our code. We eventually found that Google Cloud gave us the least amount of troubles. In Google Cloud, we set up our bucket, which is essentially just a directory in Google Cloud where we can store our InvertedIndex Jar, input folders containing the necessary files, and output directories. Then, we set up our cluster in DataProc and submitted a job to run our Jar on our created cluster by specifying the job type as Hadoop, the main class, input and output directories, and the location of the Jar file in our bucket.

The actual code-base for this assignment is pretty small. The InvertedIndexMapper reads the files line by line, splitting each line at spaces to create an array of words. It then looks at each word in this array and calls the normalize method from the Normalizer class. This method removes the word's punctuation with a regular expression while retaining the word's meaning by using a HashMap to map some words that we could think of that would lose their meaning if you removed an apostrophe to their full form (ie. we're => we are). Then, we write the word mapped to its document as long as that word isn't in a list of predefined stopwords declared in the Normalizer class (we used the default stopwords for Lucene and Elasticsearch because if that list is good enough for a high-performance search algorithm, then it will certainly be good for our purposes of using MapReduce to generate an inverted index). The InvertedIndexReducer reduces the intermediate values outputted from InvertedIndexMapper by getting the iterable list of text values (document names) and concatenating them so that each word is mapped to a complete list of its appearances in the documents. We can then go to the output folder in out bucket to see the output inverted index files.

General Thoughts & Reflections/Answers to Questions:

- What are some challenges in searching web pages? How are these challenges similar/different from searching other corpuses/repositories?
  - The mapper essentially treats the documents as text files, splitting on a space delimiter to get individual words. While this implementation works in principle, it is not practical when dealing with web pages. This is because web pages have HTML, JavaScript, and CSS elements that have their own specific syntaxes, e.g. <tags/> in HTML, h1{ … } in CSS, and so on. We would have to add special handling for these elements so that the mapper does not map them. Similarly, there could be comments in the actual html file that makes up the web page--these comments would be mapped by the mapper as well. We would need to modify the mapper such that these comments are excluded from the MapReduce process.
  - Similarly, we observed that our output files contain several “unusual” words. We believe that this may be due to JavaScript interfering with some words before they are mapped.
  - Another challenge in searching web pages could be figuring out what results are actually relevant, and what is the order of relevance of the results list.
  - The challenges in searching webpages can be similar to searching other coded or formatted data in which there is programmed syntax or tags (ie. JSON files). Such files contain data structured around conventional syntax that needs to be parsed so that the data can be mapped properly. The challenges of searching web pages is different from searching other corpuses/repositories because of all the different types of code in the source. In gathering just the informational data (ignoring all the syntax), you have to account for CSS, Javascript, and HTML surrounding the information that we want.
- How did you handle punctuation? How did you define "punctuation"? What are the tradeoffs in your decision in terms of the resulting search/results?

- We defined punctuation as being anything that the Java Pattern class identifies to be punctuation: !"#$%&'()*+,-./:;<=>?@[\]^_`{|}~.
- We handled punctuation by checking if the removal of an apostrophe in the given word wouldn't change its meaning (only for a handful of words--to demonstrate the general idea behind stripping punctuation). If it didn't, then we simply call a replace all on the string, replacing the \\p{Punct} regular expression, which removes all of the above punctuation from the word, with an empty string.
- We used a map data structure in Java to map words like "we're" to "we are", so that stripping the punctuation does not change the meaning. That said, this is only dealing with a handful of words. There is a vast array of words that we have not dealt with, because it would have taken a long time and was beyond the scope of this project. Examples of such words would be words with hyphens, or possessive nouns turning into plural nouns when punctuation is stripped off, e.g. "car's" to "cars".
- We noticed that, even though we used the \\p{Punct} regex in the Pattern class to strip off the punctuation, several words in our output file contained the " character. We believe that this may be because of some special Unicode character that codes this specific symbol--which is beyond the scope of the \\p{Punct} regex, which only deals with the " symbol. Since this does not majorly interfere with our results, we decided to not attempt to fix this issue (and merely document it instead).
- The tradeoffs in our decision in terms of searching/results for our method of removing punctuation were speed, certainty, and simplicity. We could have manually declared a punctuation list to look for and replace with empty strings in a given word, and this would have given us a little bit more certainty. However, this method is way too verbose, and using a regular expression is a much more elegant solution. However, poorly written regular expressions can severely slow down code, so there is some danger there. Fortunately, there is a predefined regular expression in the Java Pattern class that removes all punctuation from a given string, so efficiency wasn't too much of a concern in using a regex.

- What else would you have done in the inverted index implementation, given more time, energy, resources, etc.?
    - Given more time to work on the project, we would have definitely tried to implement not mapping any word that looks like <tag> or <tag/>, because part of the idea behind this project was getting an idea about how inverted indices are used in web search engines.
    - Our job with the final test data took slightly longer than 2 hours to complete with a relatively small cluster. Given more time, we would have liked to see how this time changes as the size of the cluster/number of worker nodes increases (and how this affects the cost of the job).
- How difficult was it to implement the inverted index? How difficult would it be to implement another task, given this experience? What would be straightforward? What would take more time?
    - Implementing the inverted index was relatively straightforward. We used the WordCount example as a guide for how to structure our code. The concept behind generating the inverted index was a relatively simple one.
    - One of the biggest difficulties that we had to overcome was figuring out the name of the file that contains the words, since the default file format did not provide us with the name from within our map function.
    - There was a big learning curve for actually getting our accounts set up on Google Cloud, and figuring out how to use Dataproc--including creating clusters and starting jobs. We believe that this took longer than actually writing up the code for the inverted index, just because of the sheer number of unexpected complexities involved.
    - Now that we’ve had this experience, we feel like implementing another task would be relatively easier. It would definitely be easier to actually run the jobs on the cluster. Writing code for the reducer and the main class would be simpler as well, since we noticed that reducers and the main classes tend to have similar looking code (which can be modified to fit your application). Implementing the mapper might take longer, because mappers are specific to the task that they are intended for. For example,

our InvertedIndexMapper was pretty different from TokenizerMapper, while the reducers were somewhat similar.

- If you used Google Cloud Platform, when did the job start reducing, i.e., at what level of mapping did reducing start?
    - The job started reducing when the mapping process was 95% complete.
    - We also noticed that the mapping process took around 2 hours, while the reducing process took merely a couple of minutes.
- What did you think of using AWS/Google Cloud? Tell me the positives and the negatives.
    - We ended up using Google Cloud, so we really didn't get to see the positives of AWS. The interface was very buggy and not very intuitive. There was no kind of warning before you hit the end lab button, and we accidentally ended our student account by clicking this button. Perhaps a positive for both AWS and Google Cloud is the free initial credit for students to use.
    - For Google Cloud, it was a bit easier to locate everything and get our cluster and bucket set up. The interface was not super unintuitive. The only negative that we found to using Google Cloud was that it was easy to lose your allotted funds. If you didn't delete the cluster after your job was done, then the service will continue to charge you because of the resources that you allocated. We found this out the hard way because Hammad's account ran out of funds.

A snippet of the output:

```
kemenceacutes       www.wlu.edu_special-programs_wandl-traveller_travel-programs_budapest_budapest-itinerary
kemerli   www.wlu.edu_mudd-center_faculty-fellows
kemnade   www.wlu.edu_computer-science_after-graduation_computer-science-alumni
kemper    www.wlu.edu_shepherd-program_internships_2017-interns
kendyll
www.wlu.edu_shepherd-program_community-engagement_campus-kitchen-at-washington-and-lee_student-leadership
kennedy   www.wlu.edu_williams-school_departments-and-programs_politics_major-requirements
kennedyembr         www.wlu.edu_lifelong-learning_live-streaming-events
kennon    www.wlu.edu_alumni-affairs_campus-events_affinity-reunions_black-alumni-weekend-2017
kentuckypwww.wlu.edu_support_contact-us_corporate-and-foundation-relations_profileIDx14296
kenya     www.wlu.edu_shepherd-program_about_faculty-and-staff_profileIDx1340
kenyaspan
www.wlu.edu_center-for-international-education_international-students_prospective-students_international-students-persp
ectives_waringa-kamau-kenya
kept      www.wlu.edu_general-counsel_code-of-policies_disability-accommodation_use-of-service-animals-on-campus
kerbyfultonem       www.wlu.edu_womens-gender-and-sexuality-studies-program_faculty-and-staff_profileIDx213
kerinp    www.wlu.edu_support_gifts-and-pledges_the-annual-fund_student-philanthropy_senior-class-agents_julia-gsell
kerinz
www.wlu.edu_provost_resources-for-faculty_academic-advising_advising-advice_2017-2018-fall-term-advising-advice
kernbr    www.wlu.edu_university-registrar_policies-and-procedures_university-honors-board_valedictorian
kerper
www.wlu.edu_student-life_first-year-experience_arrivals-and-orientation_first-year-orientation-committee_fyoc-leadership
kerr      www.wlu.edu_williams-school_honors-and-awards_awards_glynn-family-scholarship
kerri
www.wlu.edu_general-counsel_answer-center_travel-domestic-and-internationalinternational-visitors_protocol-for-faculty-
and-staff-traveling-abroad-under-university-auspices
kester
www.wlu.edu_provost_resources-for-faculty_faculty-recognitions_endowed-professorships_the-williams-school_the-mami
e-fox-twyman-martel-professorship
kevin     www.wlu.edu_classics-department_resources-for-students
keya      www.wlu.edu_university-registrar_my-resources_parents-and-families
keyboard  www.wlu.edu_the-college_for-students_who-signs-what
keyboardali         www.wlu.edu_disability-accommodations_web-accessibility_evaluation-tools
keybr     www.wlu.edu_student-life_residential-life_housing-agreement
keycare   www.wlu.edu_human-resources_benefits_health-dental-and-voluntary-vision-benefits_health-benefits
keynotea  www.wlu.edu_ssa_t-shirt-pickup
keyp      www.wlu.edu_disability-accommodations_web-accessibility_evaluation-tools
```