Text Processing with MapReduce

Mahtab EzzatiKarami

Here are some explanations of the files submitted and the report for part 4

Part 1: Determining the number of Starbucks in a City

There are five items in this folder:

- Input1.csv is the csv files given to us which is a dataset consisting of information about each Starbucks location.
- mapper.py and reducer.py which are my mapper and reducer for this question.
- outputAssignment1 which is the final output of map-reduce job. It is the input file for query1.py as well.
- query.py is the python file which can be used to return the number of Starbucks in a given city. City name is hardcoded. So, in order to check any city, in the code, you need to change the query in line 14 and then run it. The output will be the number of Starbucks in the given city.

Part 2: Inverted Index

There are four items in this folder:

- movies.txt is the text file given to us.
- index.py is the python code which returns the inverted index and writes its output (inverted index) in a file named: ”output.txt”
- output.txt is the inverted index outputted by index.py.
- query2.py is the python file which can be used to find the list of movies for genre/genres.
  Both single word queries and boolean search queries are supported and it’s not case sensitive. So you can use both lowercase and uppercase letters.
  In order to use a different query, you need to change line 27 in which the query is defined. The result will be the list of movies for that genre/genres.

Part 2: Inverted Index

There are five items in this folder:

- movies.txt is the text file given to us.
- mapper3.py and reducer3.py which are my mapper and reducer for this question.
- 3out.txt which is the final output of map-reduce job. It is the input file for query3.py as well.
- query3.py is the python file which can be used to find the list of movies for genre/genres.

  Both single word queries and Boolean search queries are supported and it's not case sensitive. So, you can use both lowercase and uppercase letters.

  Query is hardcoded. So, in order to use a different query, you need to change the query in line 22 and then run it. The result will be the list of movies for that genre/genres.

Part 4: Some Questions

- **Describe the design you used for Part 1 and 3. This should include the keys and values you used.**
-

  Part1:

  Mapper

  Input: input1.csv file (CVS file of Starbucks information)

  Output: pairs of (city, 1)

  Reducer

  Input: pairs of (city, 1)

  Output: pairs of (city, Total number of Starbucks in that city)

  Explanation: As mentioned above, the input file for our map-reduce job is the csv file which includes all of the information of each Starbucks in each city. Each line in this CSV file belongs to one Starbucks. The mapper gets each of these lines one by one, processes it and use column[10] of each row which is the city name. the pair it produces is: (city, 1).

In another words, the key will be city name and the value will be: 1. Because each row shows 1 Starbucks in that city. Then, these pairs will be the input of the reducer. In the sort phase all of the pairs of each city will be beside each other. So, the reducer will start processing each one of them. For each row of the same city, it will add 1 to the total count of Starbucks for that city. At the end, the output of reducer will be pairs of city name and total number of starbucks in that city. In another words the key will be: city name. And the value will be: total count of starbucks in that city.

Part2:

Mapper

Input: movies.txt file (lines of movies)

Output: pairs of (genre, movie name of that genre)

Reducer

Input: pairs of (genre, movie name of that genre)

Output: pairs of (genre, [list of all movies of that genre])

Explanation:

As mentioned above, the input is movies.txt file. Each line in this file, belongs to a movie and the genres of that movie. So mapper will have movie name with its genres as the input. Because the index should be based on genres not movie names, the output of mapper will be pairs of (genre, movie name). So, for example if there is a movie called Titanic which has two genres: family and documentary; the output of mapper for this row will be: (family, titanic), (documentary, titanic). Then Hadoop will execute sort phase. The result of sort will be the input of reducer. So, reducer will add each movie with that genre name to the list of all movie names of that genre. The output of reducer will be pairs of (genre, list of movie names for that genre). In another words, the key will be genre and the value will be the list of movies names of that genre.

- **What else would you have done in the inverted index implementation, given more time, energy, resources, etc.?**

  In this inverted index we implemented, we are only supporting either one "and" or one "or" Boolean search. We can implement it the way it supports all other kinds of Boolean searches such as "not" or different combinations of "and", "or", "not" and etc. So, adding the ability of 1- multiple genres 2- in different combinations using different Boolean expressions can be added to the inverted index.

  Also, some combinations are more common that the others. For example, there are a lot of movies that are considered both "short" and "horror". Knowing this (by checking the number of movies for these combinations or knowing what are the most frequent search movie genre combinations), we can have another separate index for them. So, for the mentioned example, we will have a separate index of "short and horror" movies. Although it will need more memory, it can make the search to be done faster.

- **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?**

  - My answer depends on which implementation we are talking about. Implementing inverted index using map reduce, was ok because after implementing one map reduce job (part 1), we can understand the process easily. And the structure is pretty much the same.

    Implementing inverted index using python, was not difficult. Because, we are using a dictionary and it is easy to understand and easy to implement as well. However, we have to pay attention to the input file and be careful of appending the related movie names to the specific genre. Other than that, it was pretty straightforward.
  - By another task, we can assume two things: 1- map-reduce task, 2- inverted index

    As mentioned above, implementing another map-reduce job would be easy, because it takes one task to implement to completely understand the structure of map-reduce job.

About, inverted index, I believe it will not be difficult either. Because whether we are implementing in python or using map-reduce process, we understand the way it works, We understand the concept of key-values and how to give a nice format to them so that we can implement queries efficiently.

- The structure of the task will be straightforward. By structure, I mean the concept of (key, values) and the way map reduce works. Inverted index is not an exception either.

- Considering different file formats as input or output would need more attention and a little more time. Because, in order to have nice formatted inverted index to use for searching for queries, we should pay attention to the format of the input file, extracting related content and formatting the output as well. I believe it is not difficult, but it needs more attention.