Tiny Google

@author Anthony (Tony) Poerio @email adp59@pitt.edu

@author Andrew Masih @email anm226@pitt.edu

Design Specifications

TinyGoogle is a program comprised of three components.

1. User interface
2. Inverted Index MapReduce
3. Search Query MapReduce

We chose to use Python to program each of these components. The MapReduce jobs (2 and 3) both run using Hadoop Streaming.

The overall design is outlined below.

User Interface

The UI for TinyGoogle is a program invoked from the command line, with two available options.
1. **Index** a new file (must be a .txt)

```
python tinyGoogle.py -i /full/path/to/my/file
```

1. Perform a **Search** for some set of keywords
   `python tinyGoogle.py -s these are my keywords each separated by a space`

If action (a) **INDEX**] is invoked –> we index the the file by moving the target file into the `/books` directory. The `/books` directory contains all of the raw data files on which we can perform a search.

If action (b) **SEARCH** is invoked –> we store the keywords in an external file that each of our MapReduce jobs can easily access. Then, we perform two MapReduce jobs in series.

- First, we create an Inverted Index (MR Job #1), using all files contained in `/books` as our data source
- Second, we search this Inverted Index and its payloads to find the data relevant to our search, and display the results to use. (MR Job #2)
- After these jobs are completed, a cleanup script is invoked, removing the intermediate files created during both jobs, so that the user can re-run a new search, or index a new .txt file without complications.

Inverted Index (MapReduce)

The Inverted Index (II) is created via a MapReduce job that calls Hadoop Streaming. This MapReduce job works with the following steps.

Preprocessing

- Before any other actions are taken, we pre-process the data in two ways.

```
    1.  Append the `file name` to **each line** of
a given file
    2.  Append a `line number` to **each line** of
a given file
```

- The results of this preprocessing are stored in a new directory, named `/books_preprocess`
- This is done to make the MapReduce job simpler. We need BOTH the file name AND the line numbers in our payloads. And by performing this preprocessing step, that means that each line passed into the MR job has all of the information necessary to construct a payload, all with one lookup.
- The preprocessing itself is performed using `sed` and `nl`, and stored in a shell script named `preprocess.sh`

Mapper

- During the MAP phase, we construct a payload in this format:
  - **FORMAT:** `word\tbook_name:line_number`
  - **EXAMPLE:** `music	BeowulfbyJLesslieHall.txt:4`

Reducer

- The reducer then takes each of these lines and combines it into a format where we can get all data about any given word using this data structure:
  - **FORMAT:** `word\tbook_name:total_ocurrences:`

    `[line,numbers,list],book_name:total_ocurrences:[line,numbers,list]`
  - **EXAMPLE:** `'weather AdventuresOfHuckleberryFinnByMarkTwain.txt:12:[10444, 10125, 6579, 2958, 1103, 5460, 2975, 6592, 8653, 4621, 2953, 4219],DublinersbyJamesJoyce.txt:5:[3210, 604, 3473, 7339, 3713]'`
- This allows us to store the following format, each on one line of our Inverted Index:
  - Word (index)
  - Book Name
  - Count of occurrences in that book (need for ordering search results)
  - The lines on each each occurrence appeared (need for generating context
- This data is then stored as our Inverted Index
- Everything on the line after the the tab is called the `payload`

Search Query (MapReduce)

The Search Query MapReduce job picks up where the Inverted Index left off. This program searches through the Inverted Index that we've just constructed and:

1. Finds the most relevant results based on occurrence count –> higher count == more relevant
2. Indexes into the labeled files, using the line numbers from our `payload` to retrieve the word's context in the specified file. That is,

it uses these line numbers to display the lines surrounded the top word occurrence.

For each keyword sent into the search query, we display the **top three results**, the `three` books with the highest frequency of occurrence.

The output format looks like so:

```
>========================================
> SEARCH TERM: –> “you”
>========================================
>Result 1 for search term —> you:
>TITLE: AdventuresOfHuckleberryFinnByMarkTwain.txt >> Word Occurrences: 1555 <<
> “Don’t you worry–just depend on me.” Then he stooped down and begun
> to glide along the wall, just his shoulders showing over the people’s
> heads. So he glided along, and the powwow and racket getting more and
>
>Result 2 for search term —> you:
>TITLE: DublinersbyJamesJoyce.txt >> Word Occurrences: 526 <<
> “Miss Higgins, what for you?”
>
> “O, anything at all, Mr Conroy.”
>
>Result 3 for search term —> you:
>TITLE: AliceAdventuresinWonderlandbyLewisCarroll.txt >> Word Occurrences: 481 <<
> executed, whether you’re nervous or not.’
>
```

```
> ‘I’m a poor man, your Majesty,’ the Hatter began, in a trembling voice,
>
>=========================================
> SEARCH TERM: –> “yield”

>=========================================
>Result 1 for search term —> yield:
>TITLE: DublinersbyJamesJoyce.txt >> Word Occurrences: 1 <<
> She did not answer nor yield wholly to his arm. He said again, softly:
>
> “Tell me what it is, Gretta. I think I know what is the matter. Do I
>
>Result 2 for search term —> yield:
>TITLE: BeowulfbyJLesslieHall.txt >> Word Occurrences: 1 <<
> 55 Having the heap of hoard-gems, to yield my
> Life and the land-folk whom long I have governed."
>
>
>Result 3 for search term —> yield:
>TITLE: AdventuresOfHuckleberryFinnByMarkTwain.txt >> Word
Occurrences: 1 <<
> oppression. Misfortune has broken my once haughty spirit; I yield, I
> submit; ‘tis my fate. I am alone in the world–let me suffer; can bear
> it.”
```

Optimizations

The largest optimization at this point is found in storing the keywords in a external file so that the search query MapReduce job can ignore all lines streamed in which do NOT contain one of keywords. This is a tremendous speedup, as it allows us to reduce all processing which is unnecessary during the search lookup.