RUMDb Search Engine – 100 course points

The purpose of this assignment is to practice your understanding of the hash table data structure.

Start your assignment early! You need time to understand the assignment and to answer the many questions that will arise as you read the description and the code provided.

Refer to our Programming Assignments FAQ for instructions on how to install VSCode, how to use the command line and how to submit your assignments.

Overview

The goal of this assignment is to implement a separate-chaining hash table that stores movie's titles and descriptions that allows fast search of words from movie descriptions.

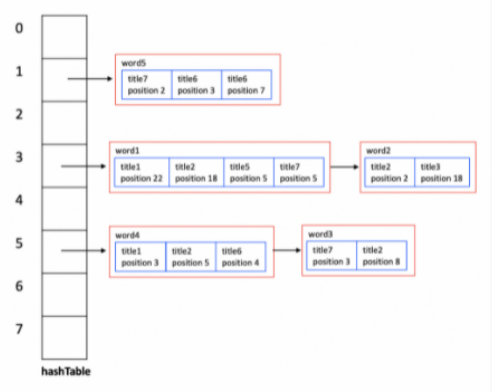
We will use the hash table to search for words in the description of movies. The search returns the movie titles where both words appear in the description. For example, using the given input data file, if the searched words are "tragic" and "love" the search returns the following 4 movie titles: *Anders als die andern*, *Namus*, *La rosa sulle rotaie*, *Sangue gitano*.

Implementation

We start with an input file that contains movies' titles and short descriptions. The title as well as the description of a movie may contain one or more words.

The **keys** into the hash table are the words from the movies description. The **values** are word occurrences (each movie where the word appears and its respective location within the description).

The **hash table uses chaining to resolve collisions**, meaning that each hash table (array) index is the beginning of a linked list of word occurrences.



In this example, the hash table size is 8 and the it contains 5 words. Each table index holds the front to a linked list of WordOccurrences.

A **WordOccurrence** (**red box**) holds the word and an ArrayList of Locations. For example, the WordOccurrence containing Word5 tells us that word5 is present at the descriptions of movies with title 7 at position 2, title 6 at positions 3 and 7.

A **Location** (**blue box**) holds the movie title where the word is present and its position within the description. For example, word2 is present at movies title 2 and title 3. In movie title 2 the word is at position 2 (the second word in the description). In movie title 3 the word is at position 18 (the 18th word in the description).

Overview of files

- **Location** class holds the position of one word in a movie's description. A movie's description first word has position 1, the second word has position 2, and so on.
- **WordOccurrence** class represents all the occurrences of a word in the entire movie database.
- **MovieSearchResult** class represents the result of the search for two words. The searched words will belong to the same file.
- **RUMdbSearchEngine** class contains some provided methods in addition to annotated method signatures for all the methods you are expected to fill in. You will write your solutions in this file, and it is the file which will be submitted for grading.
- **Driver** class, which you can run to test any of your methods. Feel free to edit this class, as it is provided only to help you test. It is not submitted and it is not used to grade your code.
- **StdIn** and **StdOut**, which are used by the driver, provided methods, and some of your implemented methods as well. **Do not edit these classes.**
- **data.txt**, contains input data.
- **dataSample.txt**, contains a small sample of data.txt. Used on the examples below.
- **noisewords.txt**, contains a list of "noise" words, one per line. Noise words are commonplace words (such as "the") that must be ignored by the search engine. You will use this file (and this file ONLY) to filter out noise words from the documents you read, when gathering keywords.

Implementation Notes

- YOU MAY only update the seven methods listed under "Methods to be implemented by you" section.
- **DO NOT** add any import statements to the RUMDbSearchEngine class.
- **DO NOT** remove the package statement from the RUMDbSearchEngine class.
- **DO NOT** add any instance variables to the RUMDbSearchEngine class.
- **DO NOT** add any public methods to the RUMDbSearchEngine class.
- **DO NOT** use System.exit() in your code.
- YOU MAY add private methods to the RUMDbSearchEngine class.

RUMDbSearchEngine.java

RUMDbSearchEngine.java

Methods provided to you:

1. **constructor**, that initializes instance variables and inputs noise words from file.
2. **hashFunction**, that is used to map a word to an index into the hash table.
3. **getLoadFactor**, that computes the hash table load factor.
4. **readInputFile**, that reads movies title and description from the input data file.
   - The method returns an ArrayList of ArrayList of Strings where each inner ArrayList represents a movie: the first index contains the title, the remaining indices contain the movie's description words (one word per index).
5. **isWord**, which given a word returns the word stripped of any trailing punctuation or null if the word is a noise word.
6. **print**, which prints the entire hash table.

Methods to be implemented by you:

1. **rehash**
   - Rehashes the hash table to newHashSize.
   - Rehash happens when the load factor is greater than the threshold.
   - The load factor is the ratio wordCount : hashSize.

2. **getWordOccurrence**
   - Searches the hash table for the WordOccurrence object containing the target word.
   - Returns the WordOccurrence object containing the target word, or **null** otherwise.

3. **insertWordLocation**
   - This method inserts a Location object into the hash table.
   - Use getWordOccurrence() to check if the word is present in the hash table.
     - If the word is present, insert location into the matching WordOccurrence object.
     - If the word is not present create a new WordOccurrence object and insert location into it, then insert WordOccurrence into the hash table.
   - In the image above a **red box** is a WordOccurrence, a **blue box** is a word Location.
   - **Each hash table index holds the front of a linked list of WordOccurrence**.
   - Insert a new WordOccurence at the beginning of the linked list.
   - Rehash if the load factor becomes greater than threshold. When rehashing, double the length of the array.

4. **insertMoviesIntoHashTable**
   - Implement this method to read the input text file and insert its movies into the hash table.
   - Use the method **readInputFile**() to read the input file and use its output to insert the words into the hash table.
   - Use **isWord**() to discard noise words, remove trailing punctuation, and to transform the word into all lowercase character. The hash table will contain all lowercase words.
   - Use **insertWordLocation**() to insert the word into the hash table.
   - **This method must be implemented correctly for you to receive any credit in this assignment.**

Once the previous methods have been written you can test using the driver and *dataSample.txt* provided. The output is shown below.

Note that the word **madame** (at table index 3) is present in the descriptions whose titles are "madame dubarry" and "la ragazza con la cappelliera". At "madame dubarry" madame appears as the 4th word in the description and at "la ragazza con la cappelliera" the word appears twice as the 14th and the 16th words.

Also note the collisions, different words mapping into the same array index. For example, the words **score** and **restored** both map into array index 10.

The searched words "tragic" and "love" are not present in the database.

```
Table capacity: 20
Threshold: 8.0
Input file: dataSample.txt
Noise words file: noisewords.txt

[0]->[irne:la ragazza con la cappelliera(15)]->[war:independenta romaniei(6)]->[new:linferno(25)]->[gustav:linferno(13)]->[fabled:cleopatra(2)]
[1]->[mother:madame x(7)]->[time:madame dubarry(18)]->[antony:cleopatra(11)]->[roman:cleopatra(8)]->[ned:the story of the kelly gang(7)]
[2]->[moscow:la ragazza con la cappelliera(10)]->[queen:cleopatra(3)]->[australian:the story of the kelly gang(5)]
[3]->[madame:madame dubarry(4),la ragazza con la cappelliera(14),la ragazza con la cappelliera(16)]->[tangerine:linferno(28)]->[adapted:linferno(2)]
[4]->[movie:independenta romaniei(2)]->[film:linferno(18)]
[5]->[making:la ragazza con la cappelliera(11)]->[1877187B:independenta romaniei(9)]->[dubarry:madame dubarry(5)]->[1890s:miss jerry(9)]
[6]->[three:madame x(14)]->[rents:la ragazza con la cappelliera(27)]->[french:madame dubarry(21)]->[inspired:linferno(8)]->[ultimately:cleopatra(13)]->[notorious:the story of the kelly gang(4)]
[7]->[committee:la ragazza con la cappelliera(24)]->[cottage:la ragazza con la cappelliera(8)]->[dantes:linferno(4)]->[loosely:linferno(1)]
[8]->[natasha:la ragazza con la cappelliera(1),la ragazza con la cappelliera(26)]->[general:cleopatra(9)]->[egypts:cleopatra(5)]->[outlaw:the story of the kelly gang(6)]->[female:miss jerry(5)]
[9]->[disastrous:cleopatra(14)]->[marc:cleopatra(18)]->[adventures:miss jerry(2)]
[10]->[score:linferno(26)]->[restored:linferno(21)]
[11]->[romanian:independenta romaniei(5)]->[dream:linferno(29)]->[silent:linferno(17)]
[12]->[abandoned:madame x(9)]->[young:madame x(12)]->[hats:la ragazza con la cappelliera(12)]->[near:la ragazza con la cappelliera(9)]->[live:la ragazza con la cappelliera(5)]->[independence:independenta romaniei(8)]->[depicts:independenta romaniei(3)]
[13]->[defends:madame x(5)]->[housing:la ragazza con la cappelliera(23)]->[loves:madame dubarry(15)]->[france:madame dubarry(12)]->[original:linferno(16)]
[14]->
[15]->[room:la ragazza con la cappelliera(29)]->[told:la ragazza con la cappelliera(21)]->[dor:linferno(14)]->[illustrations:linferno(11)]->[comedy:linferno(6)]->[185580:the story of the kelly gang(9)]->[kelly:the story of the kelly gang(8)]
[16]->[lawyer:madame x(3)]->[grandfather:la ragazza con la cappelliera(4)]->[louis:madame dubarry(9)]->[mistress:madame dubarry(7)]
[17]->[divine:linferno(5)]->[story:the story of the kelly gang(2),madame dubarry(2)]
[18]->[av:madame dubarry(10)]->[true:the story of the kelly gang(1)]
[19]->[unknowingly:madame x(4)]->[husband:la ragazza con la cappelliera(19)]->[revolution:madame dubarry(22)]->[affair:cleopatra(6)]->[reporter:miss jerry(6)]

There are no movies with the words tragic and love at their description.
```

5. **createMovieSearchResult**
   - Creates and returns an ArrayList of **MovieSearchResult** objects.
   - A MovieSearchResult refers to one movie. It contains all the locations of wordA and wordB within the movie's description.
   - **DO NOT** compute the minimum distance between two locations of wordA and wordB at this point. See next method.
   - Returns **null** if one or both words are not present in the hash table.

6. **calculateMinDistance**
   - Computes the minimum distance between the two words, wordA and wordB, in a MovieSearchResult object.
   - Updates the MovieSearchResult with the computed minimum distance.
   - Recall that a MovieSearchResult refers to ONE movie. Therefore, this method is computing how close these two words appear in the description.
   - See an example algorithm to compute the minimum distance in the Java file.

7. **topTenSearch**
   - This method's purpose is to search the movie database to find movies that contain two words (wordA and wordB) in their description.
   - The method searches and returns the top 10 movies where two words (wordA and wordB) appear closer together in the descriptions.
   - The search result is sorted by the words location distance in the movie description. The first movie will have the words closer together, and the last movie will have the words farther apart.

When the words "tragic" and "love" are searched in the hash table when the input file is "data.txt" the output movie titles are:

```
The shortest distance between tragic and love is located at:
anders als die andern   [10]
namus   [19]
la rosa sulle rotaie    [21]
sangue gitano   [23]
```

Note that these on the movie “Anders all die andern” the distance between these words is 10 (ten), love is at position 6 and tragic is at position 16.

VSCode Extensions

You can install VSCode extension packs for Java. Take a look at this tutorial. We suggest:

- Extension Pack for Java
- Project Manager for Java
- Debugger for Java

Importing VSCode Project

1. Download the zip file from Autolab Attachments.
2. Unzip the file by double clicking it.
3. Open VSCode
    - Import the folder RUMDbSearchEngine to a workspace through **File** > **Open Folder**

Executing and Debugging

- You can run your program through VSCode or you can use the Terminal to compile and execute. We suggest running through VSCode because it will give you the option to debug.
- How to debug your code
- If you choose the Terminal, from RUMDbSearchEngine directory/folder:
    - to compile: **javac -d bin src/searchengine/*.java**
    - to execute: **java -cp bin searchengine.Driver**

Before submission

Collaboration policy. Read our collaboration policy here.

Submitting the assignment. Submit *RUMDbSearchEngine.java* separately via the web submission system called Autolab. To do this, click the *Assignments* link from the course website; click the *Submit* link for that assignment.

Getting help

If anything is unclear, don’t hesitate to drop by office hours or post a question on Piazza. Find instructors office hours by clicking the *Staff* link from the course website. In addition to office hours we have the CAVE (Collaborative Academic Versatile Environment), a community space staffed with lab assistants which are undergraduate students further along the CS major to answer questions.

Problem by Haolin (Daniel) Jin and Ana Paula Centeno