The algorithm Analysis of assignment two

In the assignment two. I used three structure to implement the application including Trie, HashMap and ArrayList.

# Trie:

In my Trie algorithm, there are 8 methods and a node structure.

**Node:**

Node defines the structure of each node.

Children: made by hashmap, it stores the character and pointes to next node.

isLeafNode: it justifies whether or not the character is the end of the word.

Count: it is used to count the number of each character occurred.

countWord: it is used to count the number of each end character occurred

col and row: they indicate the column and row that the word appear in the text. The values store in a Array List.

isLineEnd: it is used to ensure whether or not the word is then last word in a sentence.

**Function**

*insert(String str, int row, int col)*

when insert the document in the trie. The string will be stored like a branch of a tree. If two words have common prefix, they will share it. After reading each letter, the last character will note as the end of that word, and indicate the row and column of the first letter of that word, and countWord add one which means the word appear one time.

*isLineEnd(String str)*

isLineEnd method is used to judge whether or not the word is the end of a sentence. It is used to determine if the phrase occurred in the second line。

*getColumnElement(String str)/getRowElement(String str)*

These two methods are used to call the specific location of each word, and they will be used in the search related to determining the location of the word.

*searchCompleteWord(String str)*

This method is used to determine whether a word appears in an article and is used in an if statement to determine whether the method continues to execute.

*searchWord(String str)*

This method gets number of a complete word. It is used for count word.

*getPrefix(String str)*

(not complete)

This method is used to get all the words that start with a prefix. Used in associated queries.

*searchPartialWord(String str)*

This method is used to get the number of word that start with a prefix. It is used to prefixOccurrence.

Trie is a kind of search tree—an ordered tree data structure used to store a dynamic set or associative array where the keys are usually strings. Since this assignment is a search text in documents, Trie is a better choice on store and search text.

When reading a file, words may appear multiple times in a large text document. I can't store the same word in a new memory address each time. This will expense huge amounts of memory to store these words. However, I store each word once in Trie. At the same time, the same prefix word, for example, cooked and cooker, they will share the cooke, just use d and r as branches, and store related information, such as location, ending note and number of occurrences, in the last letter. After repeatedly reading the word, add information to update it. This will cost less memory addresses to store data.

When searching for words, such as cook, the algorithm searches for c-o-o-k in order and checks the end note in the k-letter node. If true, prove that cook appears in the article and get information such as location, number of occurrences, etc. If the end note is false, then the word cook is not present in the text. Compared to the method of searching for words from the beginning to the end of the article, Trie does not search word one by one, but has already preprocessed the relevant information when reading the file, and directly invokes the relevant information when searching. Speed ​​up search and save memory when searching.

# HashMap

The hashmap is the structure of the key-value pair, and the location of the data storage is determined by the hashcode. It is time efficiency since it uses hashcode to store and search data directly.

*hashCalculation(K key)*

calculate the store position by key and size of the hashmap

abs(key.hashCode()) % size;

*put(K newKey, V data)*

insert the key and value in map, if key is existed, the value will be replaced by the new one.

*V get(K key)*

Get value based on key. The position of the key is determined by **hashCalculation**.

*MyArrayList getKey()*

Get a list of all key in the map.

*int size()*

get the size of hashmap

In my assignment, hashmap is used to build trie node and store the index of the article.

**Trie Node:**

Since the trie is like a tree structure that each node points the next node. So, the key in node is the character and the value is used to point next node.

**Store index:**

One of the properties of hash map is that the key is unique. Normally, the index is unique in a book, thus, I store the title as key. The value indicates the section of a chapter of the book, for the assignment, value contains a pair which is the number of title row and the last row before next title. When we select title in key, it can be fast to find the corresponding section.

# ArrayList

For ArrayList, it takes the same amount of time to delete or add elements at the end of the collection, but the time it takes to add or remove parts in the middle of the list is greatly increased. But it is fast when looking for elements based on an index. However, the linked list is opposite.

In my application, the requirement of list is to add element one by one, I do not need to consider add element in the middle. Meanwhile, the list is mainly used to search. Based on this Arraylist is a better choice.