

Adrian Gellert

Professor Maximillian Bender

CS231 Lecture A, Lab D

22 November 2022

Adrian's Project 6: Reddit Comments Analysis (2008-2015)

ABSTRACT

This project built upon project 6. That is, we first implemented a hash map to determine the total word count, number of unique words, and processing time of each Reddit comment data file as in project 6. In contrast to project 6, rather than including analyses on the Reddit files, we instead included a comparative analysis of the differences in processing times between hash maps and BST maps. The end goal for this project was to determine what process of mapping words from Reddit files is more efficient. In addition, we also wanted to find out how variation in numbers of unique words affects the number of resultant collisions when trying to map them into a hash map data structure. Through analysis, we find that on average, mapping via a binary search tree is less efficient than mapping via hash table.

If a word of the same length as other words within a linked list of key-value pairs at a certain hash code location (bucket) is unique to that particular linked list, adding that new word causes a collision. Adding a word of different length than other words, results in the creation of a new linked list at a different hash code. Incrementing the value associated with a key within a linked list does not cause a collision.

RESULTS

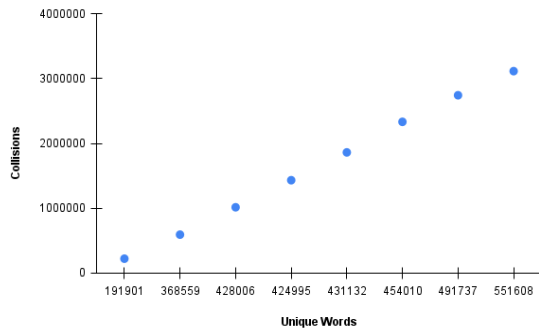


Figure 1: Correlation between unique word count and number of collisions

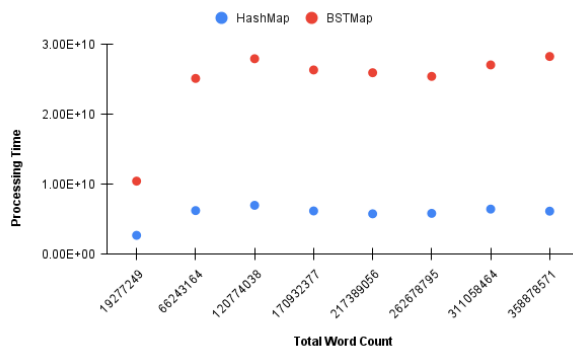


Figure 2: Relationship between processing time and total word count for HashMap and BSTMap data structures

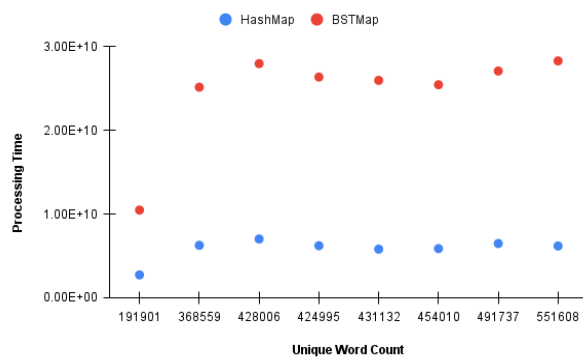


Figure 3: Relationship between processing time and unique word count for HashMap and BSTMap data structures

RESULTS ANALYSIS

- Do your plots make sense? (If not, figure out why not and fix your code). If there are any trends, describe them. Which data structure appears to be faster on this data set? Include the graphs in your report. Why do you think one is faster than the other?

Figure 1 makes sense. It is expected that as the number of unique words increases, the number of collisions will also increase. It also makes sense that the increase is linear because collisions occur when a unique word is placed into a linked list that contains words. Figure 2 and 3 also makes sense and show that a hash table method of mapping is on average faster than a binary search tree mapping method. By comparing the put methods, we can understand why the hash table data structure is faster. Within the binary search tree mapping method, we have to iterate through half of the map every time a key-value pair is added (whether as a replacement or as a new node). However, in the hash table mapping method, there is a limit to each bucket's size. This bucket size is very small compared to the binary search tree as a whole. Therefore, the iteration aspect of putting key-value pairs into a certain bucket of the hash table basically runs in constant time ($O(1)$), and all other components of the put method are also $O(1)$. Whereas, in the binary search tree method, all puts run in $O(\log n)$ if the tree was not previously empty.

DATA STRUCTURES ANALYSIS

- Analyze the performance of the BSTMap and Hashtable to see how close to ideal they perform using either total words or file size versus run-time. Start by stating what ideal means in terms of big-O notation. For the Hashtable, does the number of collisions affect performance? For the BSTMap, does the depth/height of the tree affect performance?

Include this analysis in your report.

In an ideal situation, each data structure implementation would not have any methods that include recursion/iteration because these methods alter the runtime from constant to with $O(\log n)$ or $O(n)$.

In comparison to an array based data structure, the BSTMap data structure is more ideal in terms of big-O notation. However, this method still uses recursion to create a map based on the file.

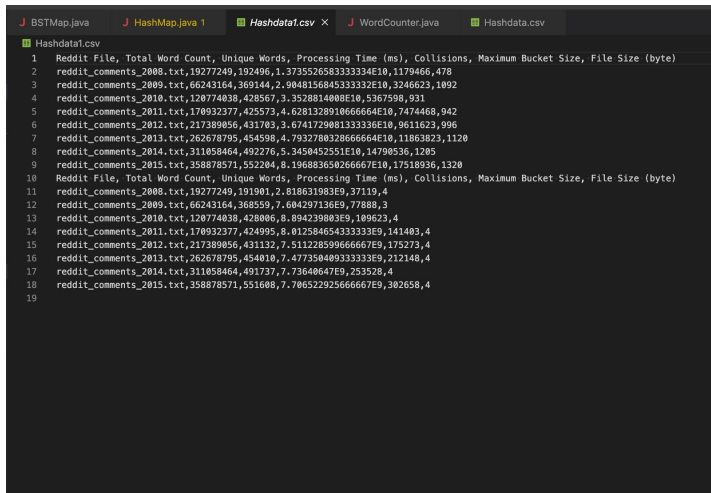
Specifically, it uses comparison to choose which node (left or right) to go down until an empty spot is located, or the key equals the key to be placed in the map. The method of recursion takes $O(\log n)$ time. Increasing the depth of the tree, would increase the number of comparisons needed before the key to be added or replaced is reached. Therefore, increasing the depth would have a negative impact on the performance of the binary search algorithm.

Hash tables do better in comparison to BSTMaps in terms of run time. In this case, unless the table needs to be increased in size to account for more hash code locations (buckets) each put runs in constant time. In the case that the hash table does need to be resized, a put method takes $O(n)$ time to run because we need to iterate over every key-value pair to add them all to a new map with a larger size. Otherwise, the only other time iteration is necessary is if we wish to add a key-value pair to a bucket already containing a linked list with key-value pairs. In this case, we need to iterate to check whether we can just replace the value associated with a certain key in the linked list. However, as stated previously, there is a limit to the size of each linked list that is so small that this iteration basically runs in constant time. Thus, a hash table runs in constant time ($O(1)$) overall with limited

numbers of puts that run in $O(n)$ time. Focusing specifically on collisions, collisions negatively impact this overall $O(1)$ runtime because collisions inform us that we are getting closer to the hash table's overall capacity. Therefore, with an increase in number of collisions, we will have an increase in number of times the resize method ($O(n)$ runtime) is called, making the overall time complexity closer to $O(n)$ rather than $O(1)$.

EXTENSIONS

I created an automated process for creating CSV files using the word counter class. An example of such a CSV file is shown below.

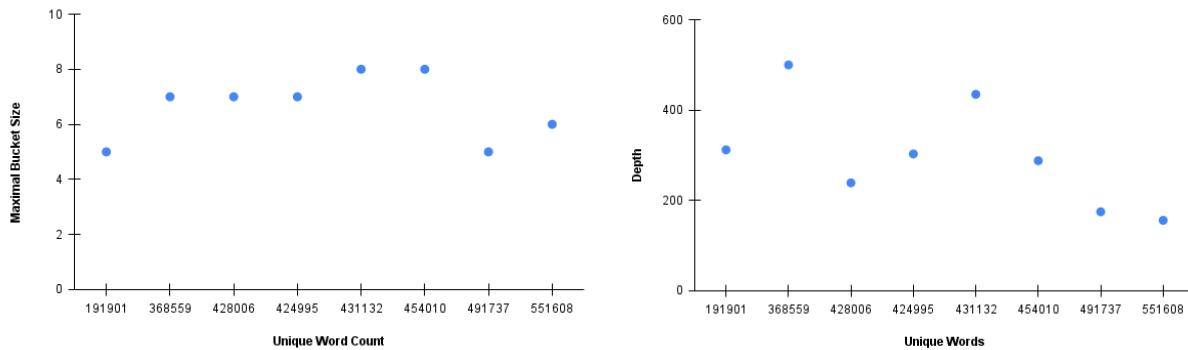


```

1 Reddit File, Total Word Count, Unique Words, Processing Time (ms), Collisions, Maximum Bucket Size, File Size (byte)
2 reddit_comments_2008.txt,19277249,192496,1.3735263833333334E10,1179466,478
3 reddit_comments_2009.txt,66243164,369144,2.9848156843333332E10,3246623,1092
4 reddit_comments_2010.txt,129774038,428567,3.3528814088E10,5367598,931
5 reddit_comments_2011.txt,170932377,425573,4.6281328910666664E10,7474468,942
6 reddit_comments_2012.txt,217389856,431703,3.6741729081333336E10,9611623,996
7 reddit_comments_2013.txt,262678795,454598,4.7932780328666664E10,11863823,1120
8 reddit_comments_2014.txt,311858464,492276,5.3458452551E10,14798936,1205
9 reddit_comments_2015.txt,358878571,552204,6.198883658266667E10,17518930,1320
10 Reddit File, Total Word Count, Unique Words, Processing Time (ms), Collisions, Maximum Bucket Size, File Size (byte)
11 reddit_comments_2008.txt,19277249,191901,2.818631983E9,37119,4
12 reddit_comments_2009.txt,66243164,368559,7.684297136E9,77888,3
13 reddit_comments_2010.txt,129774038,428806,8.894239883E9,109623,4
14 reddit_comments_2011.txt,170932377,424995,8.012584654333333E9,141403,4
15 reddit_comments_2012.txt,217389856,431132,7.511228599666667E9,175272,4
16 reddit_comments_2013.txt,262678795,454018,7.477358409333333E9,212148,4
17 reddit_comments_2014.txt,311858464,491737,7.73640647E9,253528,4
18 reddit_comments_2015.txt,358878571,551608,7.706522925666667E9,302658,4
19

```

The project also stated that “While not explicitly required, a more interesting graph would be the maximal bucket size versus unique word count. So the x-axis would be the number of unique words, and the y-axis would be the maximal bucket size in your hash map for the reddit file of that many unique words. This you could compare to the depth of the BST on the same file.”



The figure on the left shows maximal bucket size in comparison to unique word count in the file. The figure on the right shows the depth in comparison to unique words. As we can, the depth is much greater than the bucket size on a factor of a hundred more. By comparison, we can see clearly why putting keys into the BSTMap takes longer than putting the same keys into a hash map.

I tried to implement two different methods of creating a hash code for keys. One method that worked was an implementation of what is called a folding method:

```
private int hashCode(K key) {
    char ch[];
    String holder = getKey(key);
    ch = holder.toCharArray();

    int sum = 0;
    for (char i : ch) {
        sum += i;
    }

    if (size > 0) {
        return sum % size;
    }
    return size;
}
```

I also tried to implement a multiplication method:

```
private int hashCode(K key){
    Double constant = 0.5;
    char ch[];
    String holder = getKey(key);
    ch = holder.toCharArray();

    int sum = 0;
    for (char i : ch){
        sum += i;
    }
    double fraction = constant * sum;

    Double value = fraction - (int) fraction;
    if (size > 0){
        Double timesTable = value * size;
        Double floor = Math.floor(timesTable);
        int code = floor.intValue();
        return code;
    }
    return size;
}
```

Currently, the implementation of this method does not seem to work because the runtime is really long, and I could not complete processing even a single file.

REFLECTION

We used a hash map in this project because the put and removal methods for a hash map on average run in constant time, which is faster than any other data structure we have looked at thus far. As we can see given the figures shown above, the hash map implementation takes a shorter amount of time than the binary search tree.

COLLABORATION

I received help from Professor Naser to understand hash tables more fully. I also worked with Jaime to debug each other's codes.

I searched on geeksforgeeks and wikipedia for different methods of hashing.