Mika Peer Shalem
Asgn 6
WRITEUP

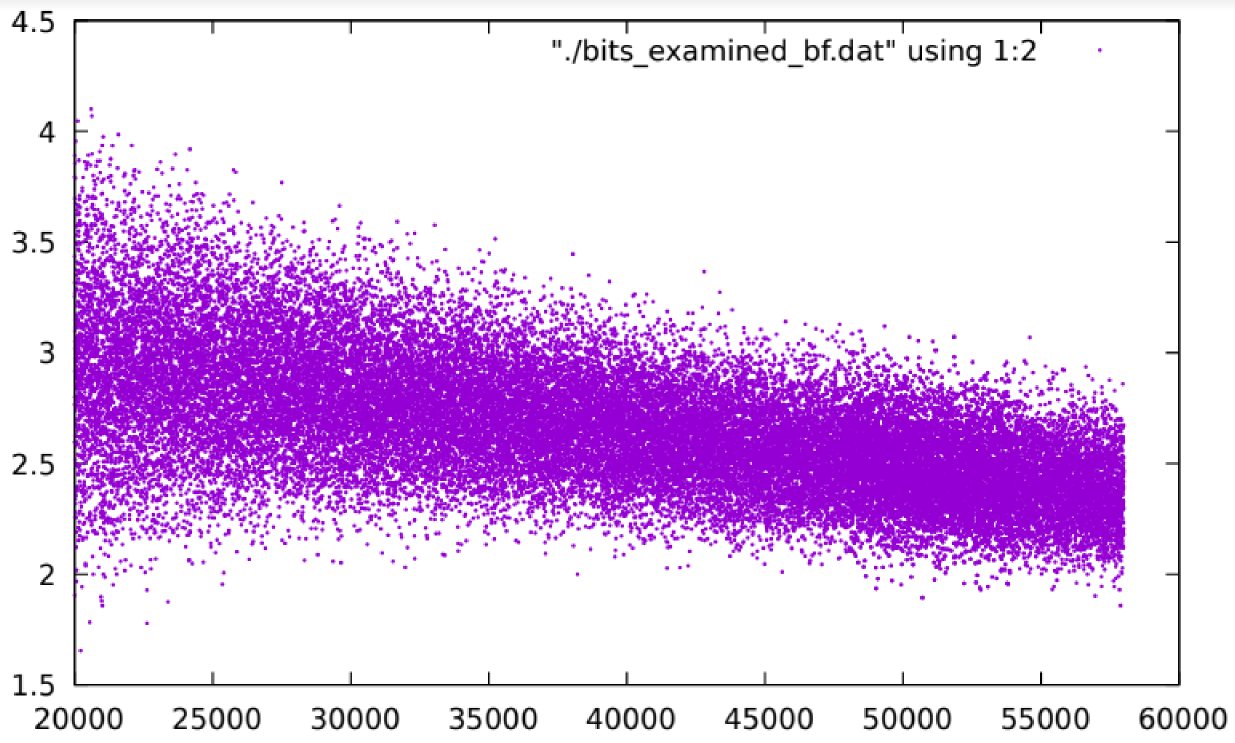
Questions

(a) How does the number of Bloom filter bits examined per miss vary (for the same input) as the Bloom filter varies in size?

The graph is in the orientation of the Bloom Filter size vs the number of Bloom Filter bits examined per miss. For the graph, I used a scale that goes from below the default size of the bloom filter (2^19) to above the default size. From the range of 20000 to 58000. The range of bits examined per miss is small and stays between 1.8 to 4.2. The graph illustrates that as the size of the bloom filter increases, the number of missed bits decreases.

The values at the end of the range go from 2 to 3.3, while at the beginning of the range, they go from 2.2 to 4.2. Therefore, I conclude that the size of the bloom filter has a negative correlation with the number of examined bits per miss.

Another interesting fact about this measurement is that the number of bits examined per miss stay at 0 until the size of the bloom filter is around 10,000.

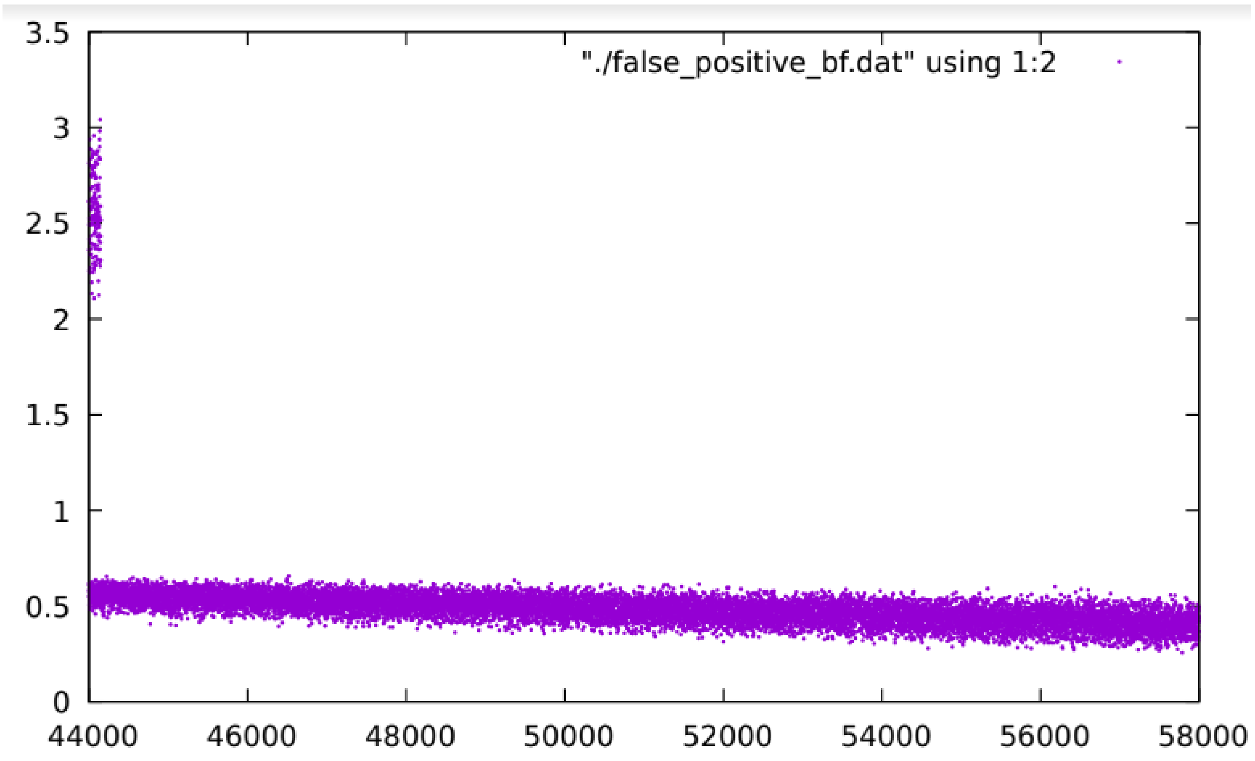


(b) How does changing the Bloom filter size affect the number of lookups performed in the hash table? This is related to the false positive rate of the Bloom filter; we discussed this in the lecture that discussed Bloom filters.

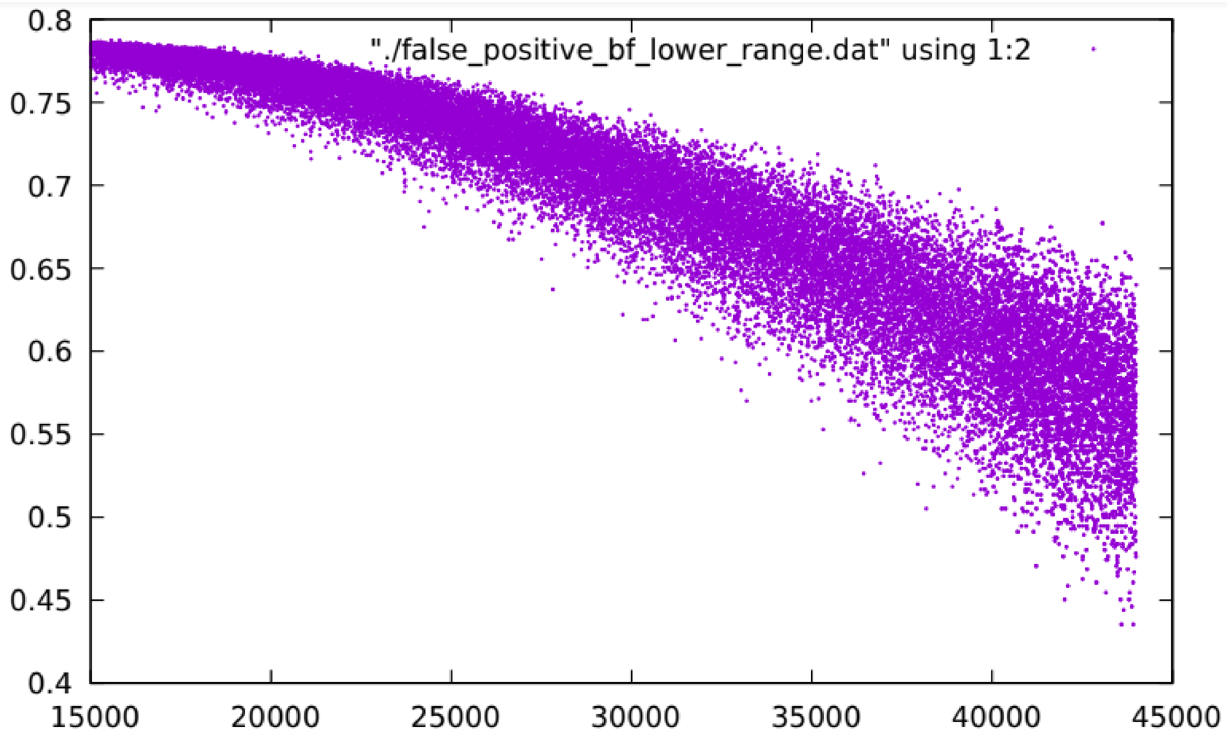
The false positive rate is calculated by dividing the number of missed bits in the hash table by the number of hits in the bloom filter. The bloom filter can make mistakes - it definitely states when a value is not in the filter, but only probably knows when a value is in the filter. Thus, some values will slip in and can be caught using the hash table.

As more bits are set in the bloom filter, the false positive rate increases. Therefore, for the same input sizes, larger bloom filters lead to a smaller amount of errors. This is because hashing a value that is not in the filter, might hit all of the set bits in the vector, and the bloom filter will think it was in the filter.

My first graph has a range of around the default value of the bloom filter. It is possible to see that at the beginning, the smallest size for the bloom filter, the rate of false positives was a lot higher.



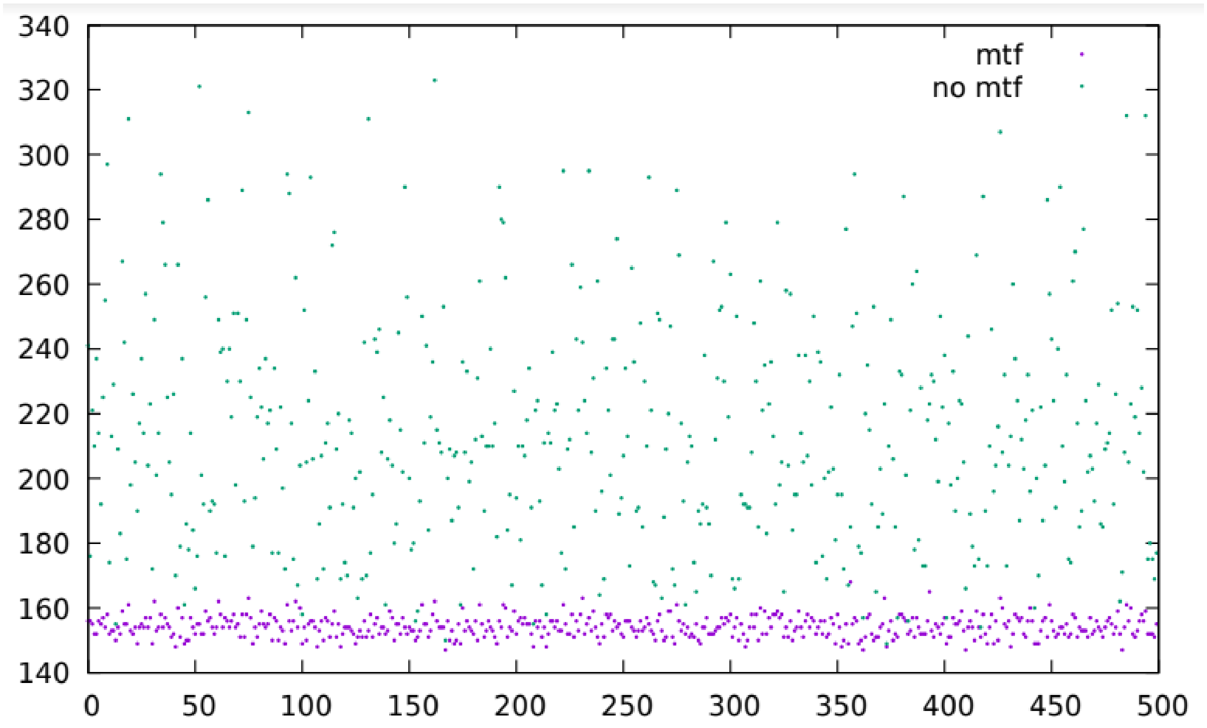
I changed the range of the values to see if there was a difference between smaller and larger values. The graph clearly shows that the rate of false positives decreases as the size of the bloom filter decreases. It started with around 0.8 errors and got down to 0.45. This improvement can help with efficiency issues in many programs.



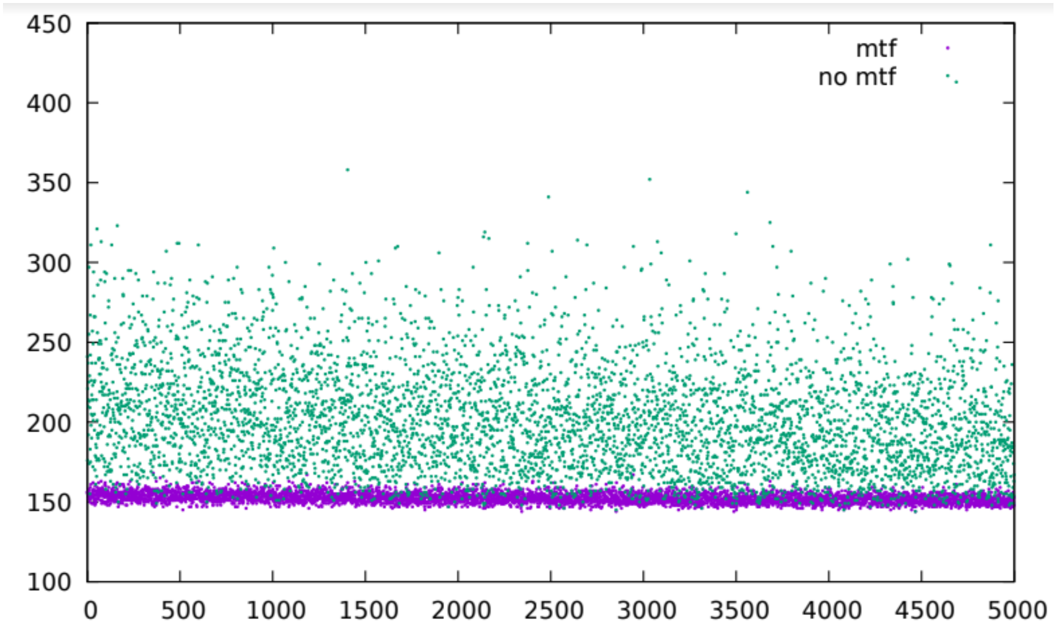
(c) How does the number of links followed without the move-to-front rule compare to the number followed with the move-to-front rule?

For this graph, I used the number of bits examined from the HashTable for the value of the number of links. I had one color to represent the move-to-front (purple) and another (green) to represent the values without the move-to-front rule. In order to see the difference between them over a large set of values, I changed the size of the HashTable. I used the same input for both graphs, a Sherlock Holmes text with 700 words.

My first graph looks at a small range of values for the size of the HashTable, from 10000 to 10500, to make it easier to see the data.



My second graph looks at a larger range of values for the size of the HashTable, from 10000 to 15000.

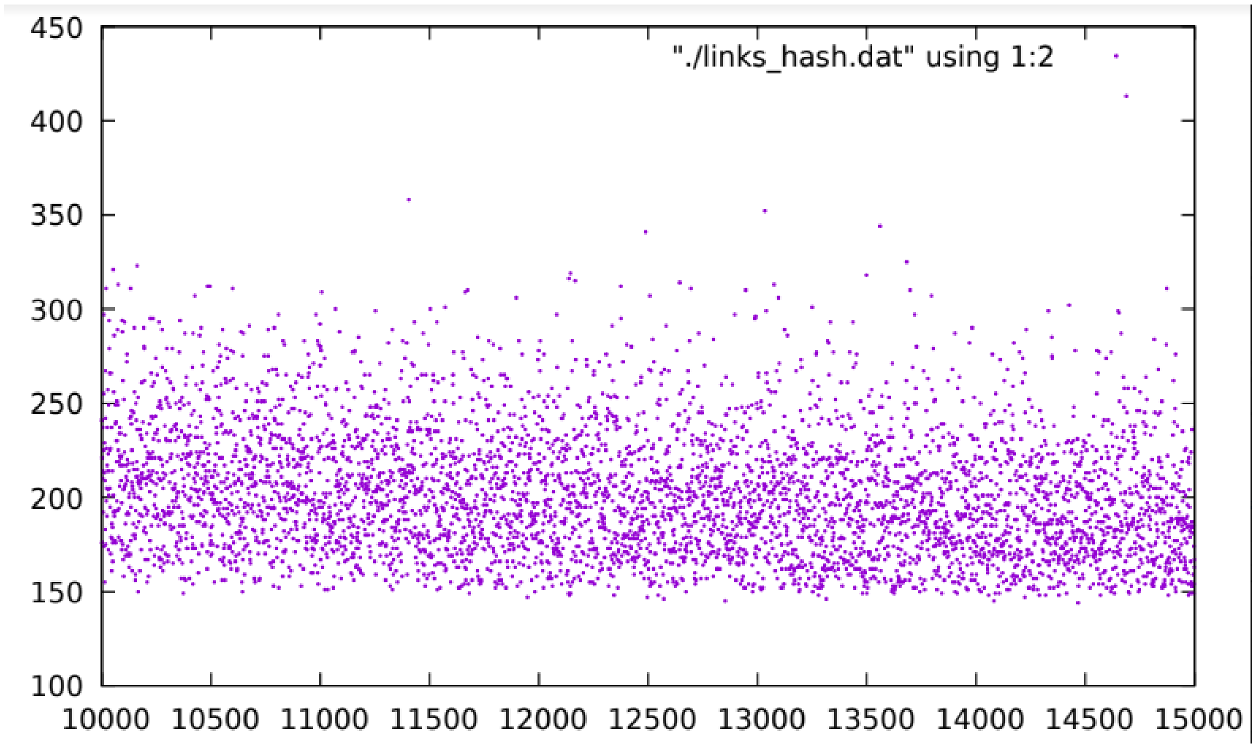


In both of them, it is possible to see that the move-to-front rule minimizes the number of links used. It seems to stay around the numbers 140-160. On the other hand, without using move-to-front, the values almost double in size. Therefore, move-to-front leads to a decrease of half of the calls for links in the program. This is because it moves repetitive values, or words, to the beginning of the list. Thus, when looking for them, the program does not need to look through all the lists, just at the first few, which decreases the number of links.

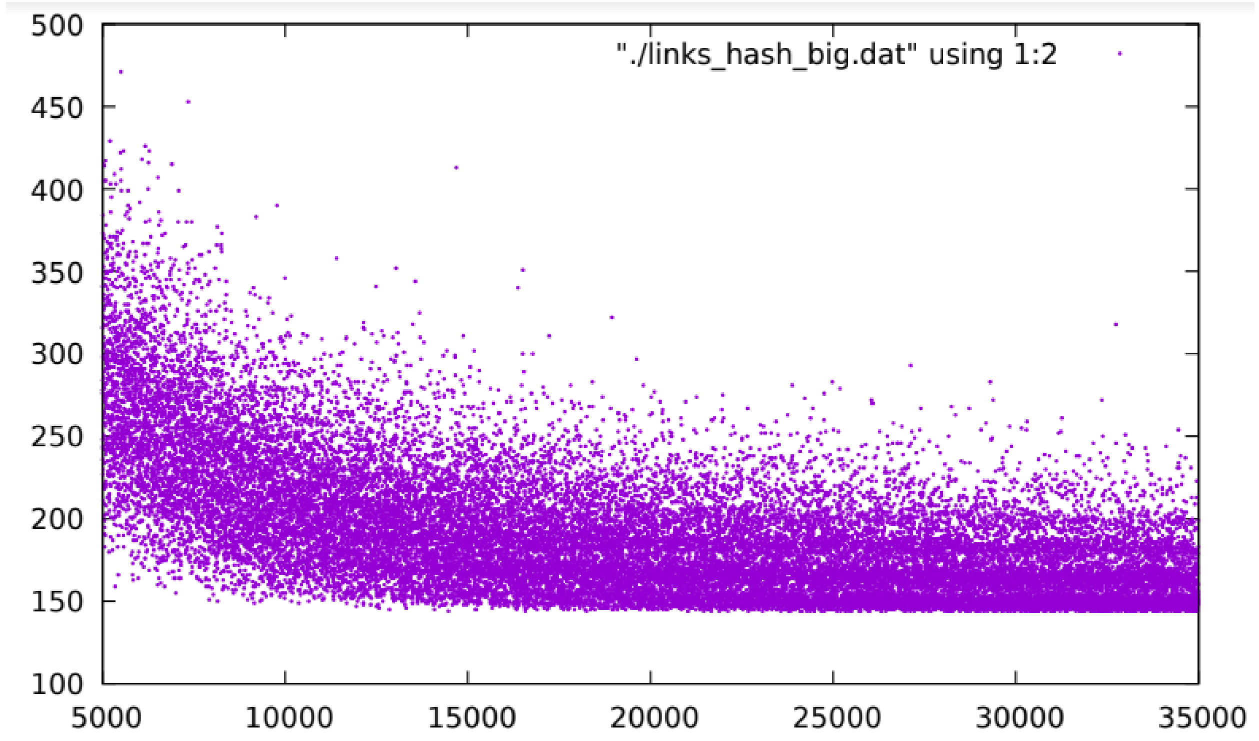
(d) How does the number of links examined vary as the size of the hash table varies? What does this say about setting the size of the hash table when using a chained hash table?

A chained hash table has many linked lists, each corresponding to a different hash index. All values that correspond to the same hash index go to the same linked list. This means that as the size of the hash table increases, we have more possible hash indexes to use, so the number of linked lists increases. This decreases the number of links since each linked list will hold fewer values. The smaller size of a hash table will lead to more collisions since more values will go to the same list, so there are more links to go through in one list.

The first graph looks at a smaller range of values, only sizes 10000 to 15000 of the hash table. In this range, there is not a lot of variance in the data. It starts to have a linear decreasing line, but it is a bit difficult to see.



It is easier to see in my second graph. It goes below the default value of the hash table to above the default value. From 5000 to 35000. It is easier to see that the number of links goes from around 400 links per run to 150 links. It is interesting to see that the value slowly approaches 150. This is likely because the text that I used to create the graph has around 150 unique words.



* Note: in order to create the graphs, I changed the print lines of banhammer to only print the statistics needed for the graph. For example, for the first graph I printed "bloom_filter_size number_of_bits_examined_per_miss". Then, using the bash script, I set the x-axis to be the first column of the text file, and the y-axis as the second column.

What I learned

The assignment taught me about bloom filters, linked lists, and hash tables. It combined the topics we have learned in the past few weeks of class into one program. It was interesting to understand when we are supposed to use each structure and the difference in functionality between them, each with its own advantages and disadvantages. Hash tables are helpful in finding a specific value from a large number of values, as it has O(1) time complexity on average and O(n) on worst-case scenarios. On the other hand, the linked list's average time is O(n). During class, the professor talked about the difference between a good programmer and a great one. He said that the big difference is knowing when to use each data structure. This assignment helped me understand when to use each one.

I understand more about the debugging process. I had to use Valgrind and scan-build to understand my mistakes. I was able to find problems with my memory allocation, and in the process learned more about malloc and calloc. I realized the differences between using string pointers and string arrays. In the parser, when I defined the next word as *word, I had malloc issues when I tried to use the string operations on the variable that led to the use of uninitialized values. However, when I started to use word[], I was able to successfully allocate dynamic memory.

Again, since the assignment had many components, I learned how to order the files that I need to write. I started with the Makefile, because without it, my program would not compile. Afterward, I wrote my BitVector, since it is needed for the Bloom Filter function which I wrote later. Every file depended on different files. So, I understand how to test each idea or file by itself before using it in different areas of code. I also learned how to reproduce the string library functions, which I took for granted before.

Overall, I am confident that I can employ the ideas from the Hash Table and Linked Lists and use them in future projects.

Citations

1)) Understand error message -
https://stackoverflow.com/questions/27636306/valgrind-address-is-0-bytes-after-a-block-of-size-8-allocd

2)) Use Valgrind to find uninitialized vars -
https://stackoverflow.com/questions/2612447/pinpointing-conditional-jump-or-move-depends-on-uninitialized-values-valgrin

3)) Understand fgets() usage-
https://www.ibm.com/docs/en/i/7.4?topic=functions-fgets-read-string

4)) convert char to an int (didn’t use at the end) -
https://www.tutorialspoint.com/how-do-i-convert-a-char-to-an-int-in-c-and-cplusplus

5)) More Valgrind commands -
https://stackoverflow.com/questions/5134891/how-do-i-use-valgrind-to-find-memory-leaks

6)) Malloc error -
https://stackoverflow.com/questions/2987207/why-do-i-get-a-c-malloc-assertion-failure

7)) Understanding diff - https://www.geeksforgeeks.org/diff-command-linux-examples/