Ruchit Patel
CSE13S, Spring 2021
Prof. Darrell Long
25 May 2021

Assignment 7: The Great Firewall of Santa Cruz

This writeup contains analysis of several graphs with different inputs. The main categories covered in the writeup surround Linked Lists* (LLs), Bloom Filters (BFs), and Hash Tables (HTs). The subcategories are as follows:

1. Linked Lists
    a. LL seeks, links, and average seek length vs BF size
    b. LL seeks, links, and average seek length vs HT size
    c. Number of links with vs without move-to-front (mtf) rule
    d. LL lookups vs HT size

2. Bloom Filters
    a. BF size vs BF count
    b. BF size vs false positives
    c. BF size vs Average LL length

3. Hash Tables
    a. HT size vs HT count
    b. HT size vs Average LL length
    c. HT lookups vs BF size

Since one file might not give the overall picture, I chose three files** from the *corpora* directory provided in the resources folder. The files are as follows:

1. *artificial/random.txt* (100k bytes)
2. *calgary/news* (377k bytes)
3. *large/world192.txt* (2473k bytes)

*Note: The graphs for seeks and links would differ to my buffer implementation logic being different from the resources executable (see readme). There would be no difference in Hash Table and Bloom Filter stats, however. The graphs are made from this commit ID: b3430aa4fa754e...

**Other files are used when required and are explicitly mentioned in the description or directly above the corresponding graphs.

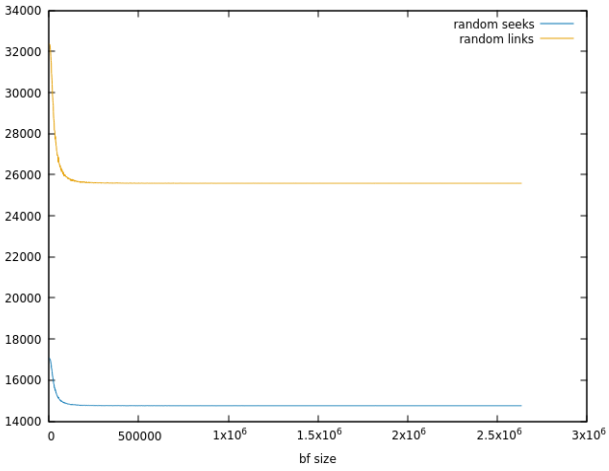
-> For all the graphs below, analysis is mentioned directly after the graph or after all graphs in a given subtopic.

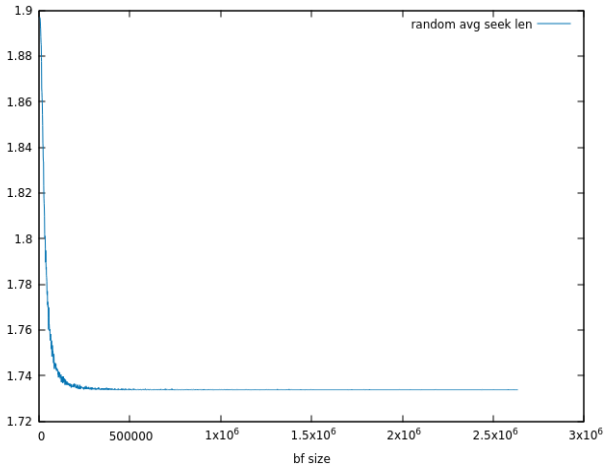
Linked Lists

LL seeks, links, and average seek length vs BF size

- Depending on the BloomFilter size, the number of seeks and links traversed in a linked list would vary. The reason being short BF sizes are prone to produce more false positives and thus we would have to lookup in the LL to verify and as a result the number of seeks and links increase. Just how much they change as the BF size increases is explored in the graphs below.

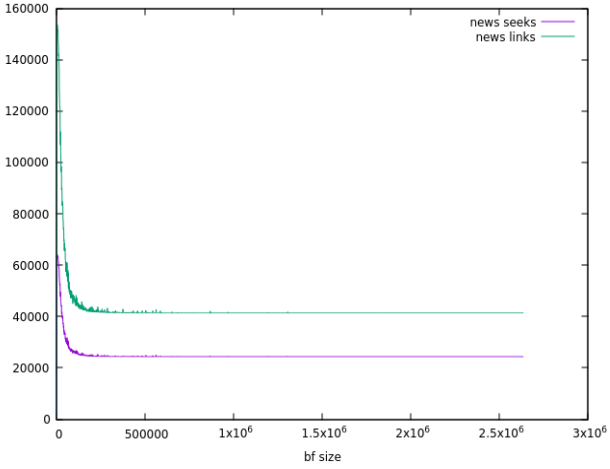
1. *artificial/random.txt*

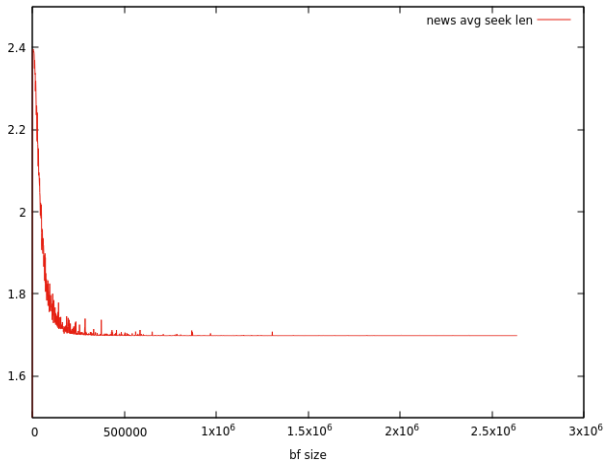




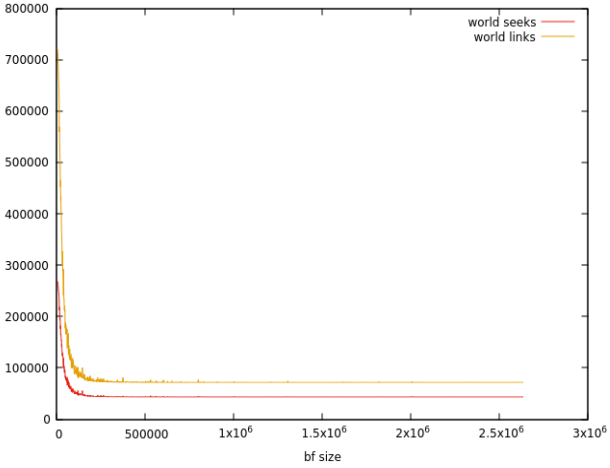
- It could already be seen that as the bf size increases, the number of seeks and links keep decreasing logarithmic-ishly until they reach a horizontal asymptote.
- One explanation for this is that for a file of a given size, if the bf size is lower the probability of bf probe returning false positives is higher. As a result, ht lookup would be called frequently to check false positives. Thus, the number of seeks and links are somewhat inversely proportional to the bf size.
- The average seek length given by $\frac{links}{seeks}$ would obviously follow a similar trend--starting higher and gradually decreasing as the bf size increases.
- At one point, however, the increase in size of the Bloom Filter has no effect on the number of seeks and links and the average seek length. The reason behind this--also repeated throughout the writeup (I won't mention it in detail here)--is that once all the words in a file have been added to the BF and because the hash returns the same index each time, there are no words that need to be added and as a result the LL statistics remain constant or the same as they were before.

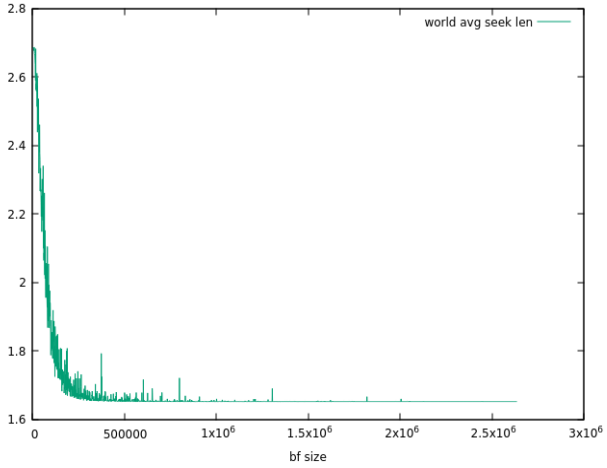
2. *calgary/news*



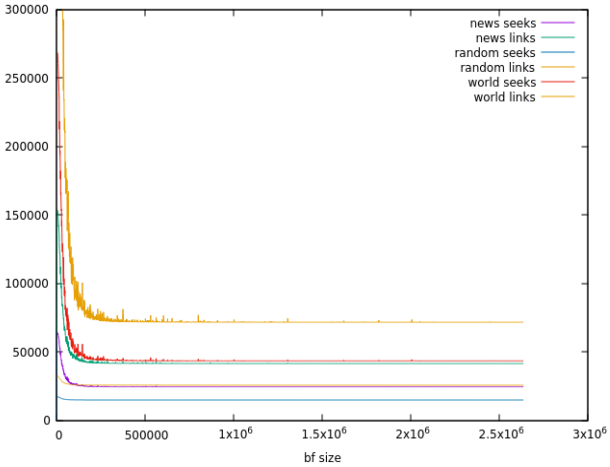


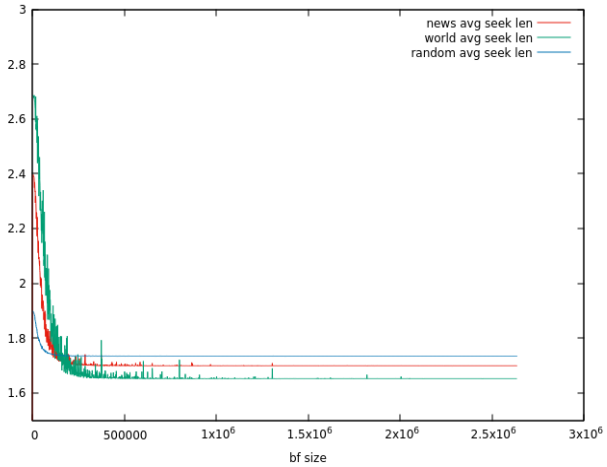
3. *large/world192.txt*





4. All files



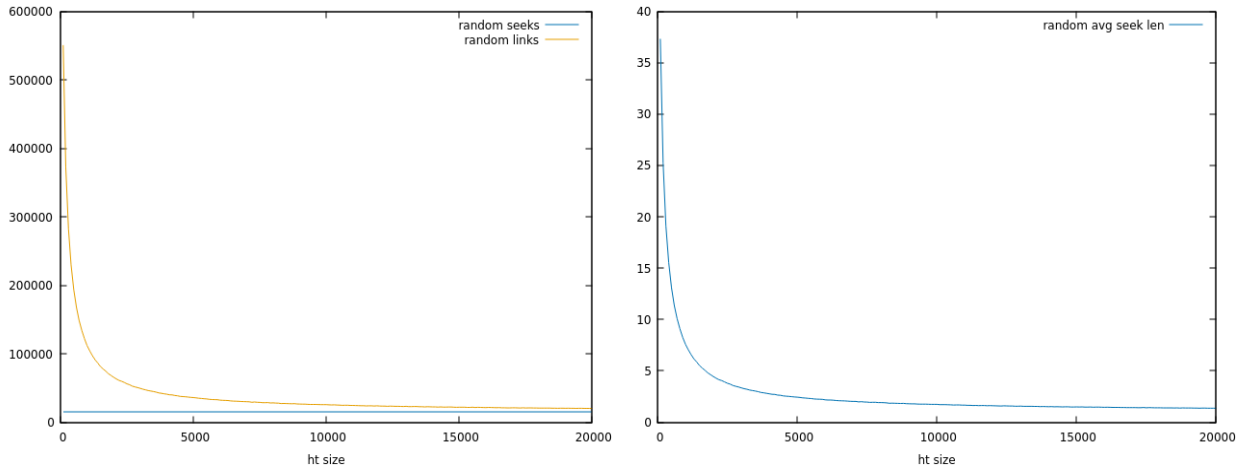


- It can be seen from the graphs above that the number of seeks and links decrease somewhat exponentially until a certain value which is going to be the absolute minimum.
- The initial values, ofcourse, depend on the size of the file. The higher the size of the file, the more the probability of its entropy being higher (more dissimilar words) and thus more words would be added to the HT and the LL. And since the bf size is fixed--also mentioned in depth in *BF size vs False positives* and *BF size vs HT Lookups*--we would have to call LL lookup more frequently to avoid adding false positives. Thus, that explains the nature of the graphs.
- One thing I am refraining from analyzing here is the spikes in the graph. I did the topic after all other ones and I want to avoid repeating the explanations and hence the analysis for the spikes is provided and is the same as the one in *BF size vs HT lookups*.

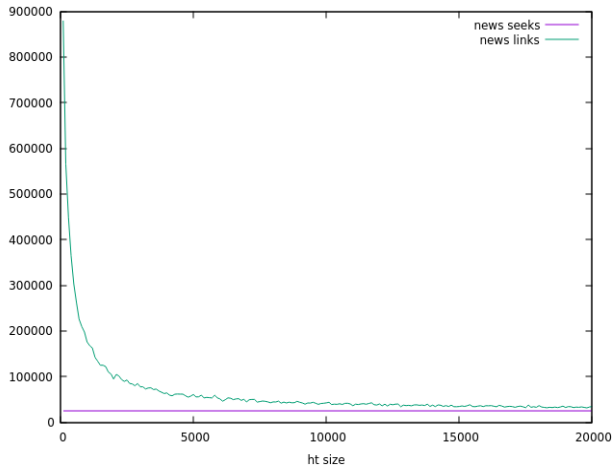
LL seeks, links, and average seek length vs HT size

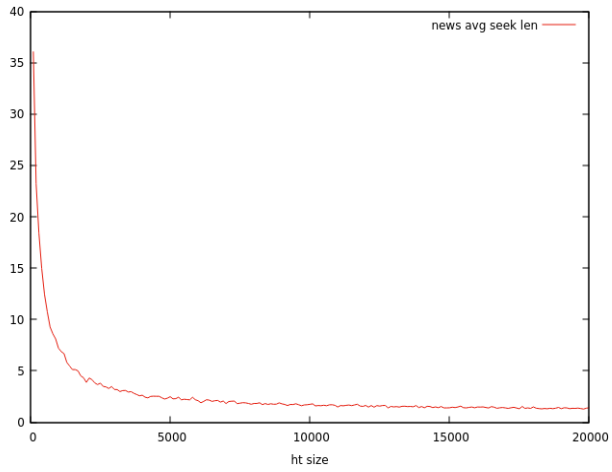
- Similar to the above test, it is easy to see that the number of LL links depends on the HT size. One reason for this is that the greater the HT size, the lower the chances of collision and as a result we would not have to traverse the list too far. On the other hand, my initial guess is that the number of seeks should not change since the number of times we lookup an entry in the HT has nothing to do with the size of the HashTable. The graphs and their analysis is as follows:

1. *artificial/random.txt*

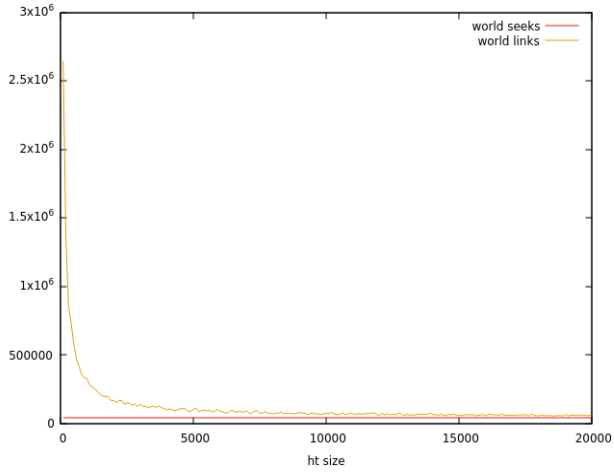


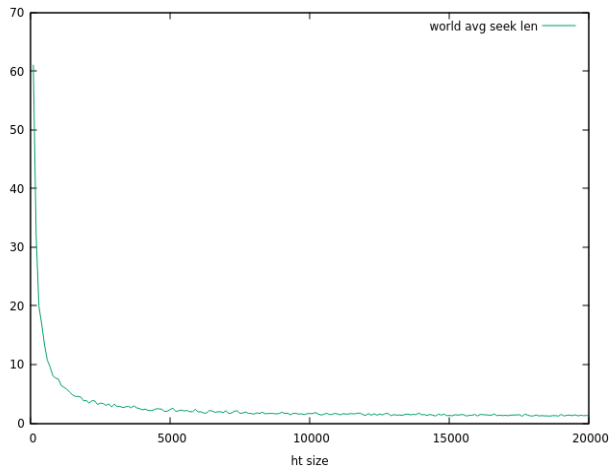
2. *calgary/news*



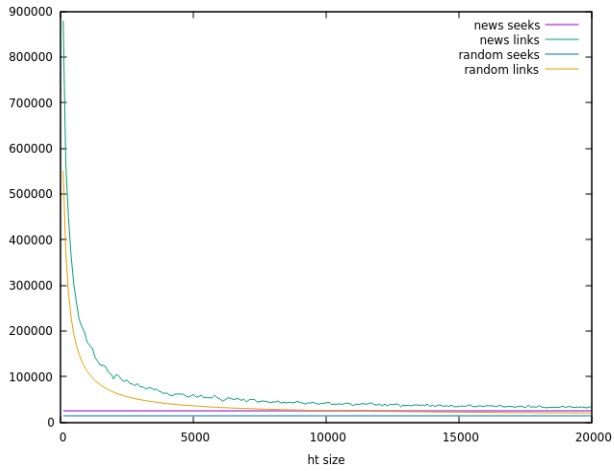


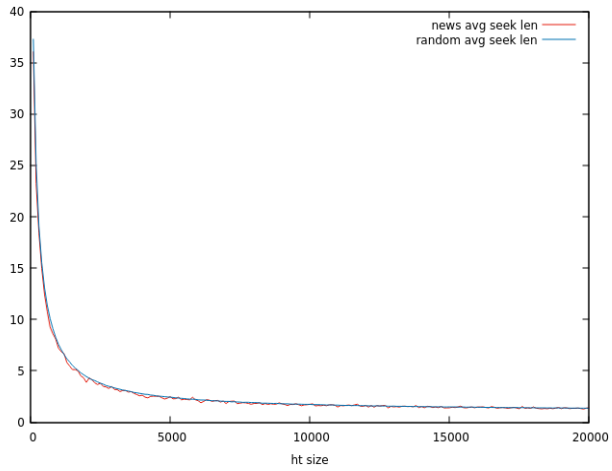
3. *large/world192.txt*





4. All files (lost world file)





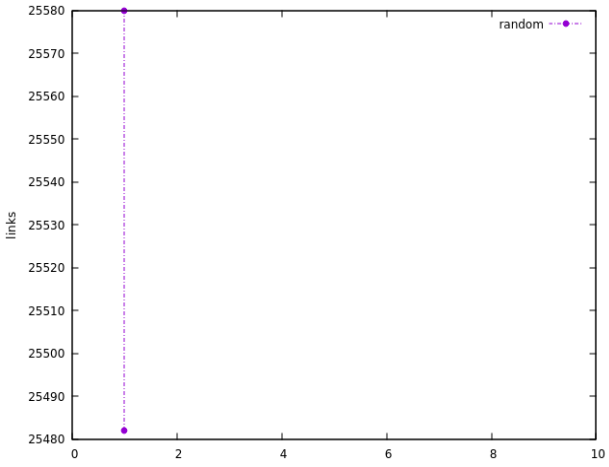
- The analysis of the above graphs is quite simple. As the ht size increases, the hash function has a higher range of index positions for a given word. Thus, as the size for the Hash Table increases, the words start to spread out over the hash table and the probability of hash collisions decreases and as a result the average length of a given Linked List also decreases. Hence, that explains the number of links graphs.
- The number of seeks, as it can be seen above, does not depend at all on the ht size. The explanation behind this is straightforward as well and is mentioned in detail in *LL lookups vs HT size* subtopic. For now, a short explanation that it has to do with checking false positives should be sufficient.
- Since, the number of seeks remains constant, we can derive as well as see that the average length would also have a similar graph as the number of seeks graph--only it would be factored down by a constant. That constant c is precisely $\frac{1}{seeks}$ and thus the graph should be $y = \frac{links}{seeks}$.

Number of links with vs without move-to-front (mtf) rule

- For this experiment, I wanted to see whether the number of links increased or decreased before and after using the move-to-front rule. I speculate that the number of links would go down when using the mtf rule if there are repeated words in a file. Since using the mtf rule moves the nodes to the front, the more frequently the words are repeated, the lower the number of links should be. Since three files might not contain the variety of data needed for the graphs below, I included more manual files mentioned below.

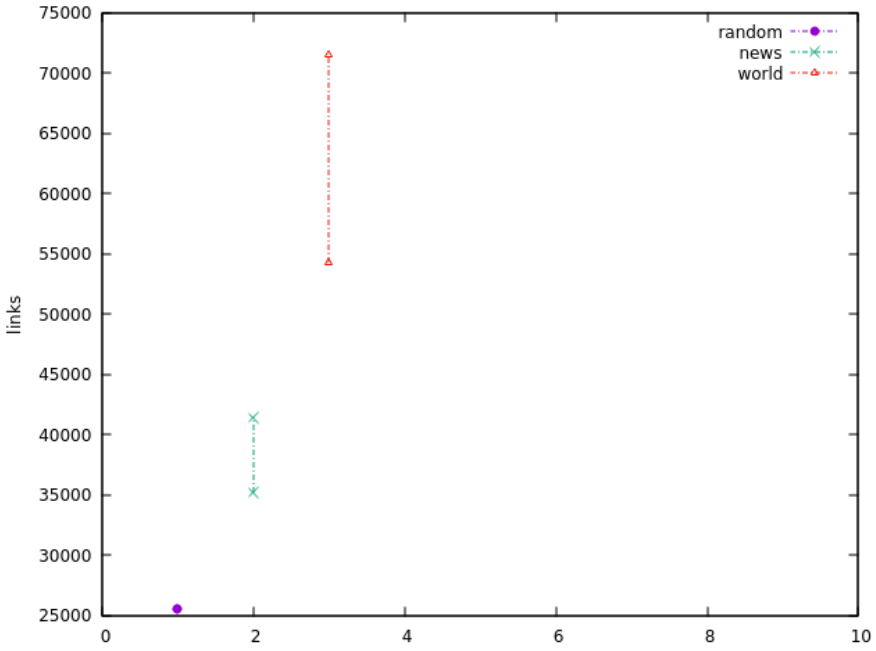
Patternless files (higher y value represents no mtf)

1. *artificial/random.txt*



*Note: Pattern and patternless here simply mean files with lower and higher entropy respectively.

All three:



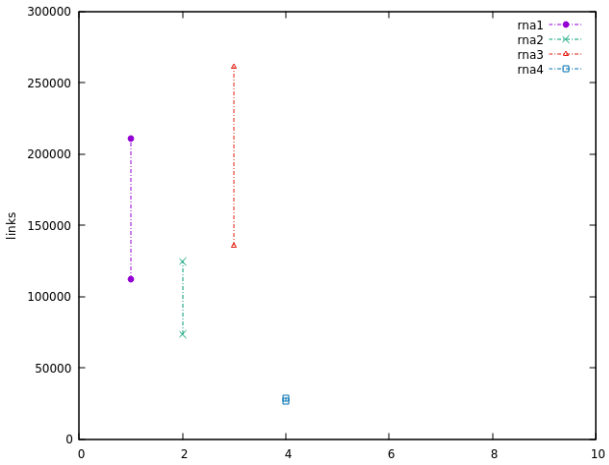
File	Links w/ mtf	Links w/o mtf	Difference
random.txt	25482	25580	92
news	35205	41397	6192
world129.txt	54312	71557	17245

- It could already be seen that using the mtf rule makes a huge difference and the difference only gets higher as the file size increases.
- The reason behind this is explained at the end of this subtopic (after all the graphs).

Repeating files

2. RNA sequences

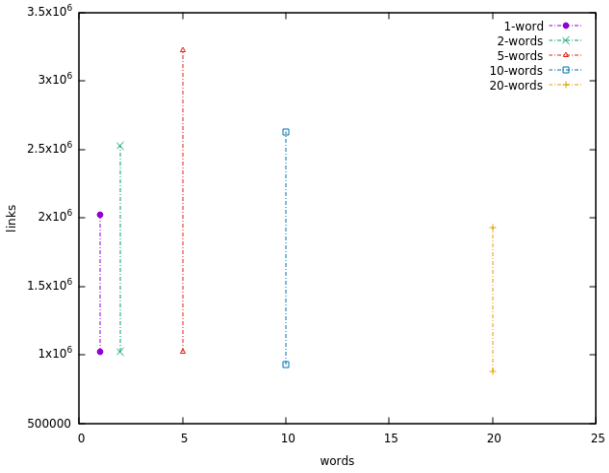
- The files were chosen randomly from this website.



- From the results of the previous graphs above and below and without analyzing the patterns in each *rna* file, we can conclude which file had the most patterns.
- We know that the greater the number of patterns, the lower the entropy is going to be and as a consequence the difference between links with mtf and without mtf should be higher.
- Thus, it could be said that the file with the highest number of differences has the most "patterns". In this case, it happens to be *rna3*. Hence, this is the important point of this experiment. There is an exception to this phenomenon which is stated in the next point.

3. *x* words repeated randomly

- These files were made by me in which I write *x* words (first *x* letters of the alphabet) to the corresponding files randomly. The PRNG used to make the files can be found here.



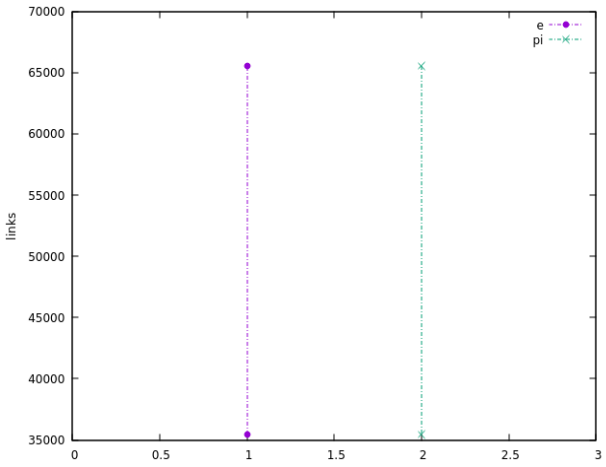
File	Links w/ mtf	Links w/o mtf	Difference
1-word	1025390	2025433	1000043
2-words	1025392	2525278	1499886

5-words	1025395	3226543	2201148
10-words	925630	2626626	1700996
20-words	875583	1926198	1050615

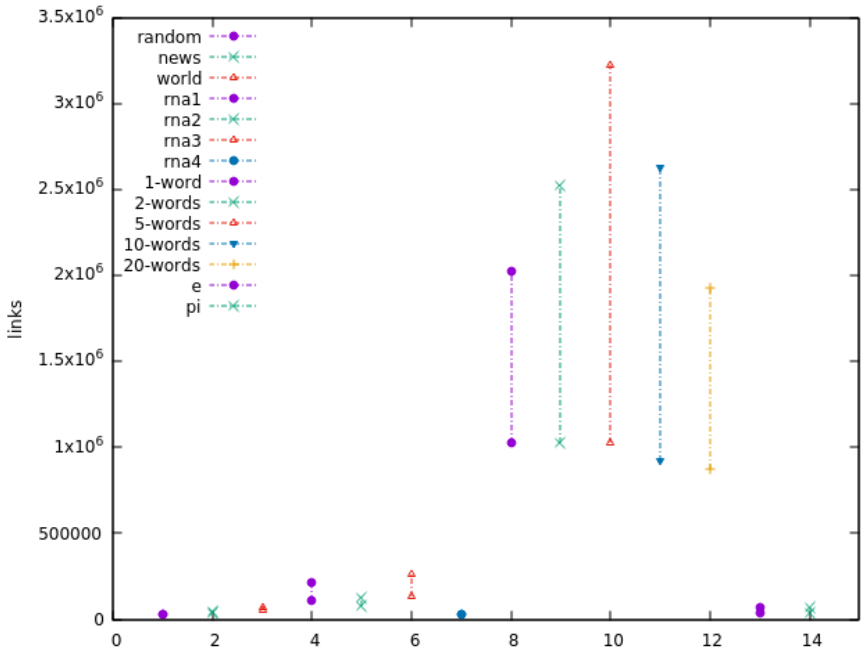
- To begin, note that the conclusion for differences relating to more patterns as listed in the previous point does not reflect for these graphs. The reason behind this is that the set of the input characters for these graphs is variable. For example, the previous graph only holds true if the set is fixed and it was true in the *rna* case: the files only had variations of 'a', 'g', 't', 'c'. Here, we add a new character every time and thus we get a bell-ish shape for the graph.
- I do not have much to add to this and the next point but to point out the obvious that using the mtf rule makes a tremendous difference.

4. First 100,000 digits of π and e

- These files could have been easily found on the internet but I wanted to understand the spigot algorithm so I tried it myself.
- The first file is made using the spigot algorithm whose implementation is provided in piazza post 26. The second file is made by me--also using the spigot algorithm. The description of the algorithm is described here and the implementation is done by myself.



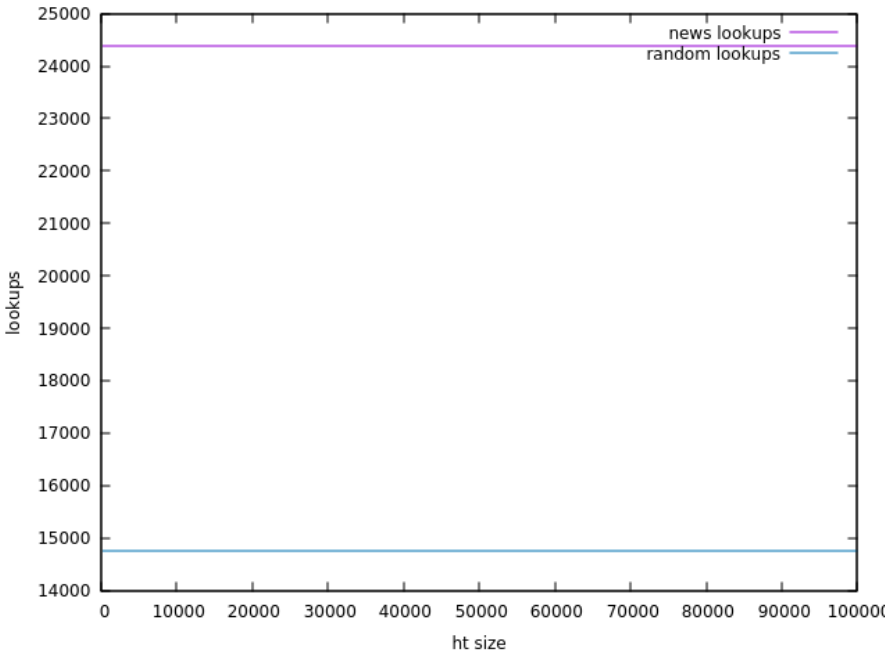
All files:



- As it can be seen from above, the difference between random and deterministic files is quite significant. This is self-explanatory. The “deterministic” files have more patterns and also lower entropy than the random files and thus the difference between using and not using *mtf* is not much compared to the random files.

LL lookups vs HT size

- In this somewhat trivial assessment, I wanted to see what shape do the graphs form if LL lookups are tested against HT size etc. In a sense, this is a subset of *LL seeks, links, and average seek length vs HT size* but more clear due to less compactness. (Lost World file).

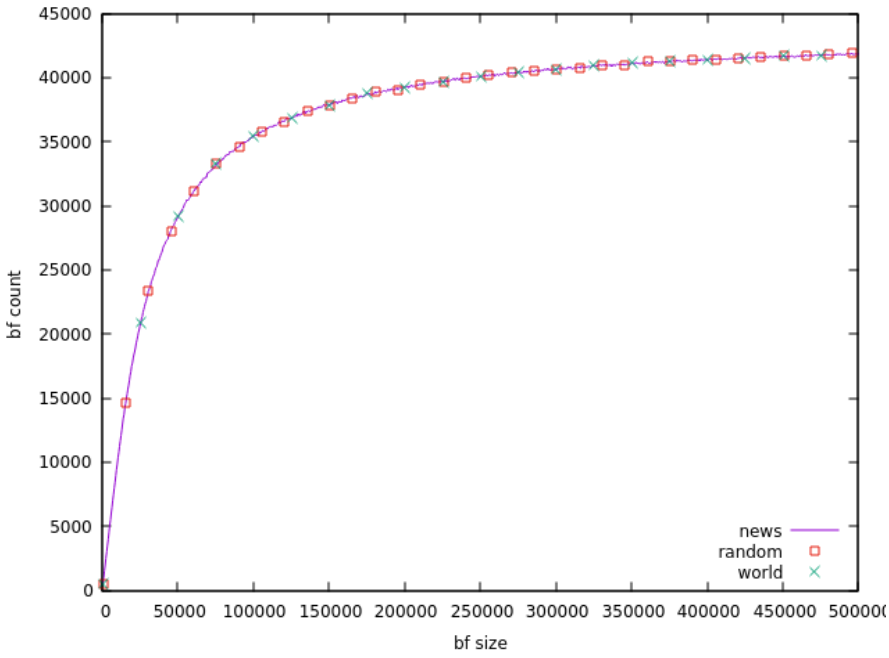


- This graph proves the claim that the number of lookups has nothing to do with size of the Hash Table.
- The reason for this is that ht lookup is called every time after using bf probe (to make sure there are no false positives). Thus, no matter the size of the hash table, the number of seeks performed on the linked lists should remain constant. The constant value then simply depends on the size of the file. The greater the size, the higher the entropy (generally speaking) and thus more words need to be looked up as a result for a given file.

Bloom Filters

BF size vs BF count

- How many times do the BF bits overlap? That is the core question that is going to be explored in this subpart. An easy guess is that the bit count should increase as the BF size increases since there is less chance of index positions overlapping after the hash function. The basic reason is that there are % BF size possible places for the index and as the size increases the group space should increase as well.



- As hypothesized, the bf count does increase as the bf size increases and, as it could be seen, it increases logarithmically.
- Like the graphs before, this graph also seems to approach a certain y-value after which the size of the Bloom Filter and no possible effect on the bf count. One reason for this could be that since a file has a finite number of bytes/words, once all of them are added the BF would have the bits as spread out as it possibly could be.
- For example, if there are no “hash collisions” the maximum number of bits that could be set in a BF is *number of words * 3* (three hash filters). After that, increasing the bf size could, theoretically and practically, have no effect on the bf count.
- If there are overlapping bits--which is most likely going to be the case--the bf counts would be lower.
- To my surprise, however, bf count follows an eerily similar pattern for all the files regardless of their size. A possible explanation would have to do with the restriction of the range of the hash function. Since the hashed value can only be within the range of bf size, the size of the file may not play a role in determining the bf count as we would have already achieved the maximum spread/set bits in the BF by that point.

BF size vs false positives

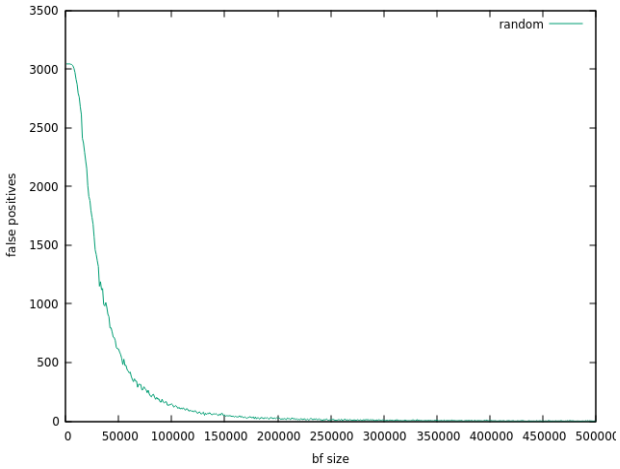
- For this experiment, I wanted to see how many false positives are produced as the BF size increases. My guess is that the graphs would have a decreasing slope; however, I am not sure whether the shape of the graphs would be logarithmic-ish or linear-ish.

- The methodology used for determining false positives is quite simple. Every time I call bf_probe, I also additionally call ht_lookup to confirm the bf_probe result. If a node is found, we know the lookup did not return false positives. Although computationally intensive, this method does guarantee that the graphs below are accurate.
- To even make false positives happen, I had to modify this test condition

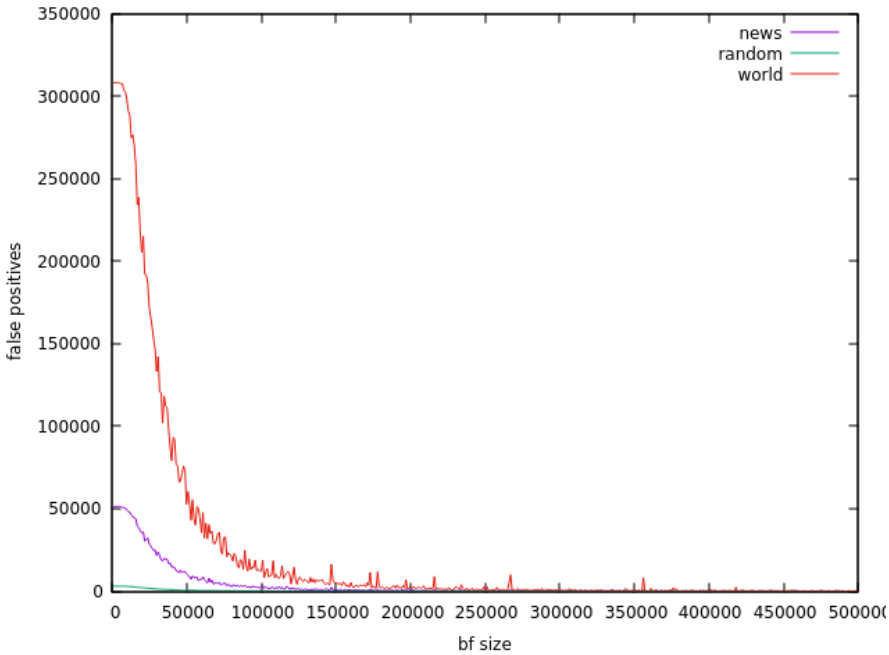
```
if (bf_probe(bf, word) && (temp = ht_lookup(ht, word)))
```

and remove the second condition so false positives are dependent on the bf size.

1. *artificial/random.txt*



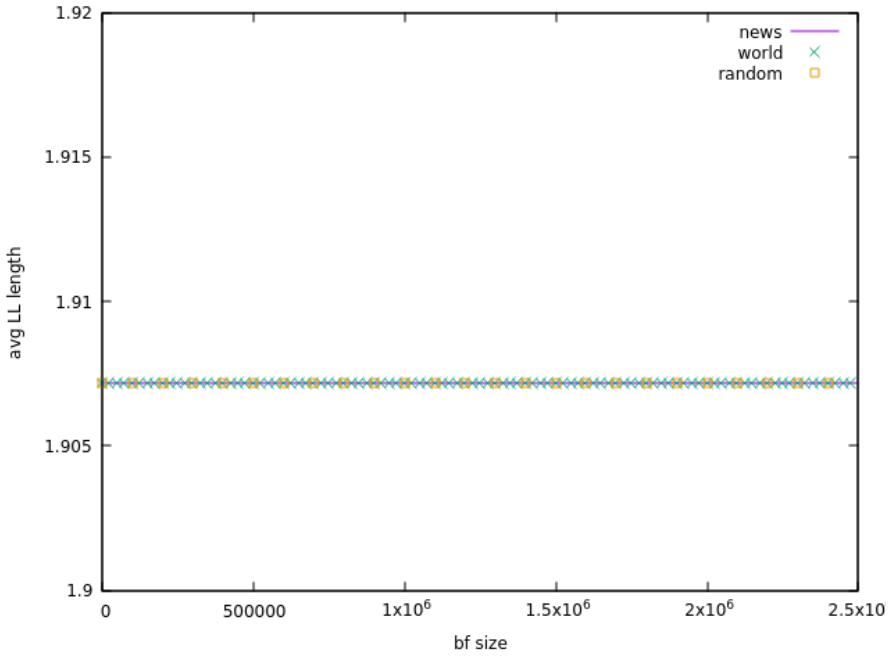
2. All files



- Well the graphs clearly show a logarithmic-ish decrease in the false positives as the bf size increases. The explanation, as mentioned before, is quite simple and logical. The lower the number of bf size, the faster all the bits are prone to get set. Thus, producing more false positives when we do have to check for the presence of the word.
- For smaller values of bf size, the function is almost y = c for some constant c. A basic reason is because, regardless of the bf size, all the bits get quickly set as there are simply more non-repeating words in the files. For each file, however, this stops occurring at a certain value for the bf size. This should happen roughly when, assuming all words are non-repeating--the bf size is greater than the file size. This is since the index to get the bit is %ed with the bf size and the greater the bf size the more bits to be set and thus the number of false positives reduce with each successive increase in the bf size.
- It could also be seen that false positives approach zero after a certain bf size. This happens around half a million in this case but it is completely dependent on the file. Speaking of which, the higher the value of the file, the more false positives it is going to produce. This is, quite simply, because it has more words in it.

BF size vs Average LL length

- This is done for the sole purpose of clearing some of my doubts. At first glance, I found the correlation between BF size and LL length to be a bit blurry. After all, I never use bf_probe before or after calling ll_insert. Thus, I wanted to see if my doubts were true and that BF size should have no impact on the average LL length. In other words, the graphs should be in the form y = c where c is a constant/average length. (1.907817 to be exact)

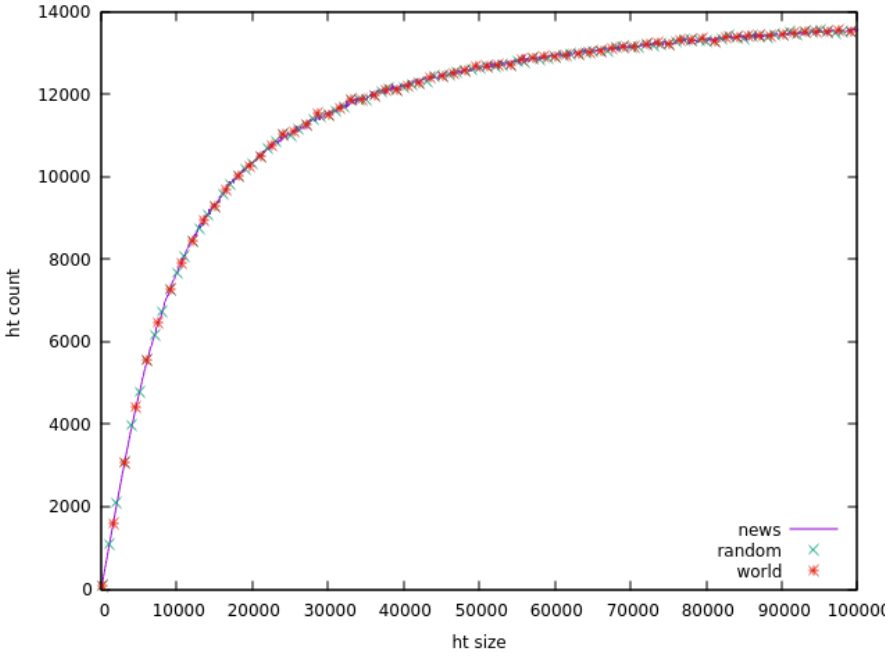


- The graph speaks for itself. It is indeed true that changing the bf size should have no impact on the average length of the LL since there are no chances of false positives with ht lookup and ll lookup guards. Surprisingly, the file size does not matter here as well. I reckon if the file sizes were lower than the average length might have varied but in this case the words in the linked list are again as spread out as possible. This can also be seen in the subtopic below where ht count shows the above mentioned phenomena.

Hash Tables

HT size vs HT count

- The number of unique linked lists pointers in a given hash table are completely dependent on the size of the hash table. The working and the explanation are similar to the *BF size vs BF count* experiment. The relation and ratio between the HT size and the HT count is graphed below.

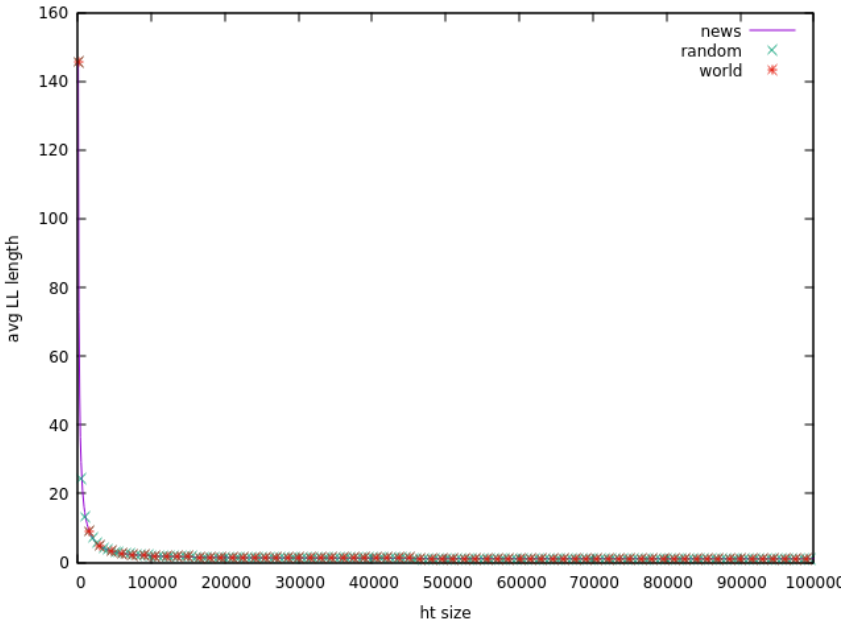


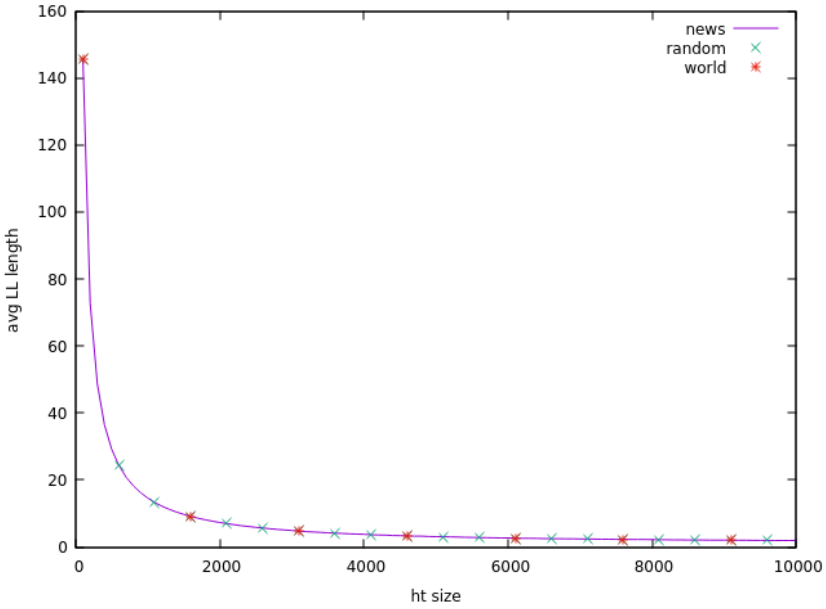
- As it can be seen from the above graph, the higher the value for HashTable size, the more indexes/entries available to fill up after hashing.
- The reasoning for the same pattern regardless of the file size is the same as mentioned in *BF size vs BF count* and *BF size vs False Positives*.
- In this case too, the graph approaches a horizontal asymptote. The reason behind this is because the file has a finite number of words and after a certain ht size, increasing it

would have no effect on the ht count since hash function always returns the same index for a given word.

HT size vs Average LL length

- Similar to the tests above, in this part I explore how the average length of a linked list changes depending on the HT size. I was confused on how the graph would change. I am guessing it would have a decreasing trend but whether that would be uneven or not is the main point of focus.





(the above graph is “zoomed in” version of the previous one)

- The average length approaches a horizontal extremely quickly as it could be seen in the graphs above. This has to do with the previous subtopic *HT size vs HT count*. In fact, the average LL length should be inversely proportional to the HT count since it calculated using this formula:

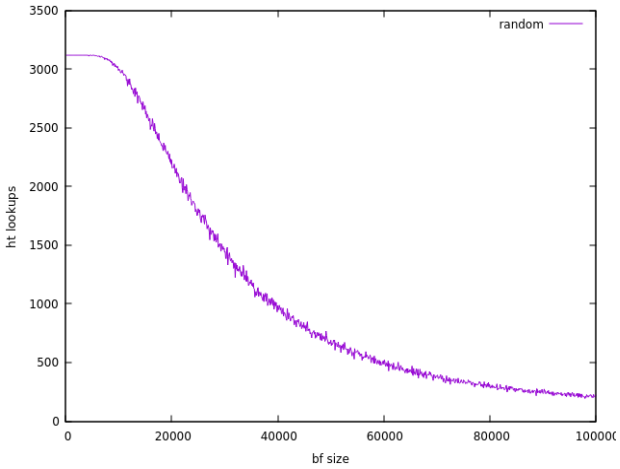
$$\frac{1}{ht\ count} \sum_{0}^{ht\ size} non\ null\ ll\ length$$

- Thus, it is easy to see that as ht size increases, ht count also increases as mentioned in *HT size vs HT count* and as a result the average length keeps on decreasing until ht count reaches the aforementioned horizontal asymptote.

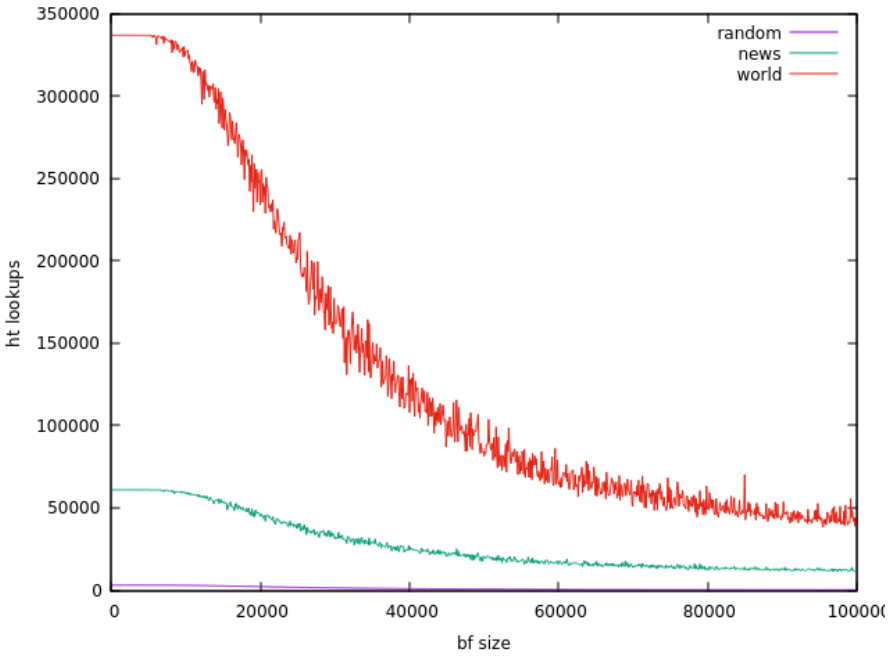
BF size vs HT lookups

- The purpose of these graphs is the same as *BF size vs Average LL length*'s one. I am unable to find a connection between HT lookups and the size of the BF so I tried to graph it to get a clearer picture. The results are as follows:

1. *artificial/random.txt*



2. All files



- After looking at the graph, I was confused at first, however, I soon realised that the relation between bf size and ht lookups derives from *BF size vs false positives* experiment.
- The lower the bf size, the more the false positives and as a result the number of ht lookups performed--to avoid false positives--would naturally be higher.
- The spikes in the graph are a result of, in my opinion, bf insert. Increasing the bf size by one or two or three might not necessarily have any impact on the bf count. In other words, it is not guaranteed that more bits would be set for a small change in the bf size. It is, however, given that as more and more bits would be set as the bf size increases over a certain amount. Hence, the increasing trend in *BF size v BF count* and decreasing trend in the graph above.
- The reasoning behind y = c nature and the graphs approaching a horizontal asymptote follows the explanation given in *BF size vs false positives*.
- It was quite amazing to learn how intricate and connected things were for this lab. This concludes the writeup.