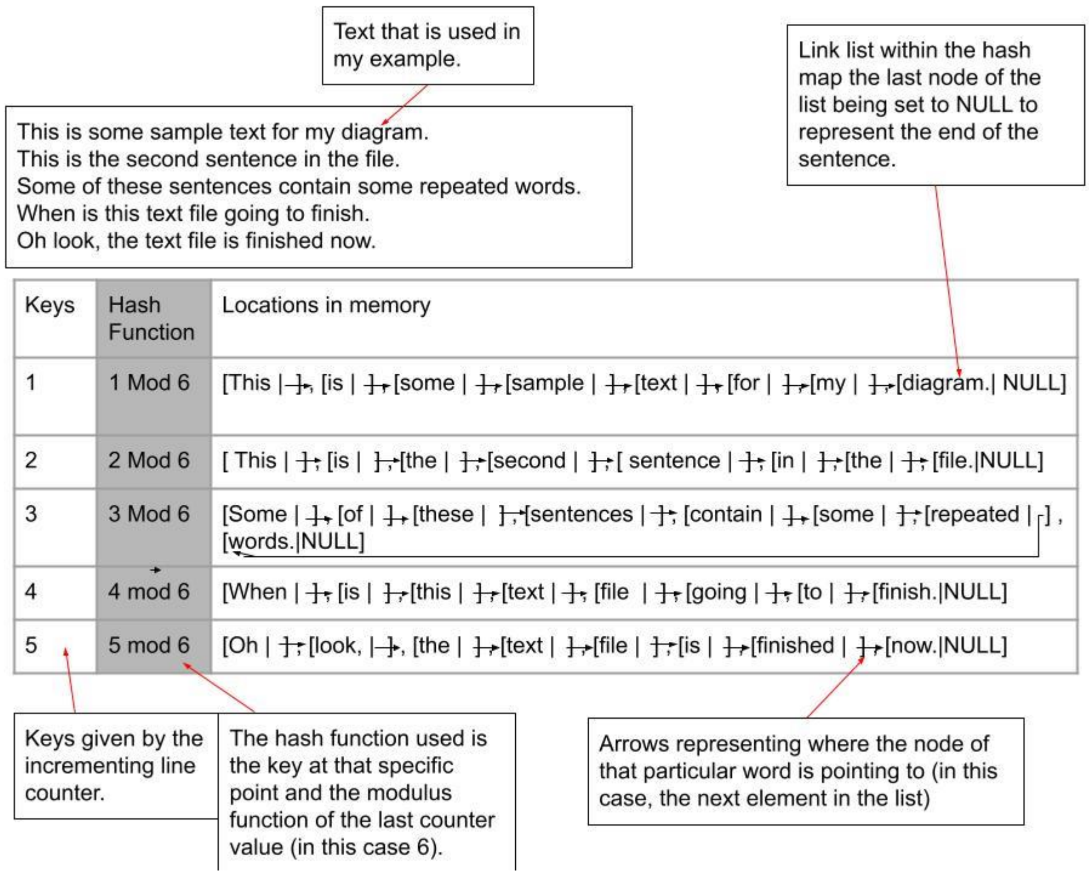
Part 1 – My Solution

The assumptions made in the assignment project description that we will acknowledge are: the text file being read one line at a time, that the line number is represented as a counter and is incremented after each line is read and once a line of text has been read, the line is broken up into a list of the individual words on that line. As well as these assumptions, some more to acknowledge would be the file fits into memory and that we are only looking for complete words and not partially written words. Finally, an assumption that we can make is that the data structure won't do any operations before completing the data structure first.

My solution to this problem would be to store the words in the text file within a link list and a hash map. When reading through the text file 1 sentence at a time, the sentence will be broken down and each word will be stored as separate node objects with the first and last word of each sentence being the head and tail node respectively. This will leave each node as a separate string which link together to recreate the full sentence within the text file. The hash map takes the counter (integer) stated above in part A and assigns it as the key for each sentence in the file. The sentence (which is now in a link list) is then stored as the value of that key. Once this value has been stored for this key, the text files counter increments and the process can start again. The hash function for this would be the Mod function. When the whole file is read through and stored in the hash map, the final counter value is incremented its last time. Since there are no lines left it won't increment again. This value is then used as the Mod value for the hash function. So, when calculating what key is associate with each sentence, say line 5, the operation would be 5 Mod X where X is the last incremented counter.

This diagram is a representation of what the finished data structure would look like.



Text that is used in my example.

This is some sample text for my diagram.
This is the second sentence in the file.
Some of these sentences contain some repeated words.
When is this text file going to finish.
Oh look, the text file is finished now.

Link list within the hash map the last node of the list being set to NULL to represent the end of the sentence.

Keys	Hash Function	Locations in memory
1	1 Mod 6	[This |→ [is |→ [some |→ [sample |→ [text |→ [for |→ [my |→ [diagram.| NULL]
2	2 Mod 6	[This |→ [is |→ [the |→ [second |→ [sentence |→ [in |→ [the |→ [file.|NULL]
3	3 Mod 6	[Some |→ [of |→ [these |→ [sentences |→ [contain |→ [some |→ [repeated |→] , [words.|NULL]
4	4 mod 6	[When |→ [is |→ [this |→ [text |→ [file |→ [going |→ [to |→ [finish.|NULL]
5	5 mod 6	[Oh |→ [look, |→, [the |→ [text |→ [file |→ [is |→ [finished |→ [now.|NULL]

Keys given by the incrementing line counter.

The hash function used is the key at that specific point and the modulus function of the last counter value (in this case 6).

Arrows representing where the node of that particular word is pointing to (in this case, the next element in the list)

Part 2 – My Justification

The reason why I chose a hash map combined with a link list as my data structure is because I wanted to be able to make the worst-case time complexity be as low as possible for multiple tasks. However, the file has multiple ways of increasing this time complexity as well as creating errors. For instance, the worst-case time complexity (in big O notation) whilst in the build phase is O (n^2). This is because as both the length of the sentences increases and the number of lines increases, the more time the data structure will take to finish. If both the sentence length and lines in the file have their maximum respective storage, then they would have a quadratic incrementation. Other operations however like insertion and deletion of a word would have a worst-case time complexity of O(N) as its worst and searching for words would have the complexity of O(N^2) since you must search the entire text file at worst. The data structure isn't without fault. Because we don't know how many keys there are whilst creating it, when searching through the hash map, collisions may occur. The incrementing counter variable is a way that limits this possibility. However, a collision may still occur when let's say the user wanted to add another line. Some other data structures like an array list instead of a link list would have better access to each word in a specific line and can call upon them when necessary but both the insertion and deletion of data would have a tome complexity of O(n^2).

Part 3 – My Operations

1)Print all the line numbers on which a given word occurs
This operation searches methodically through every word in the link lists and then increments the counter, to change the current key in the hash map to get to the next line. They continue this method until it has gone through the entire text file. When it finds the certain word, the counter variable is appended to an empty list which Is then printed off at the end of the operation. This operation would have a worst-case time complexity of O(n^2) since the operation needs to go through every word in the text file.

2) Print true/false depending upon whether a given word was in the file or isn't

Again, the file would search through the entire text file like in the operation above. However instead of adding the counter variable to a list, it changes a Boolean variable to true and that variable is then printed off later. The worst-case time complexity of this operation is O (N^2). They use the searching operation for

3) Print the total number of times a given word occurs on a specific line (may be zero)
This operation gives us a line number for the file. This line number is then used as the key for the hash map in order to locate the sentence. Once found, the program again searches through this line 1 object at a time. If it finds the word that it is looking for, a separate variable increments itself to show that they have found the word they were looking for. This operation has a worst-case time complexity of O(N) since it only needs to go through 1 line and not all of them

4) Print all the words that occurred on a given line
Again, like the operation above, the line is already specified. Due to this, all the operation does is when looking at a word (object) it prints the word out. When you go through the entire sentence, the operation then stops. This operation also has a worst-case time complexity of O(N).