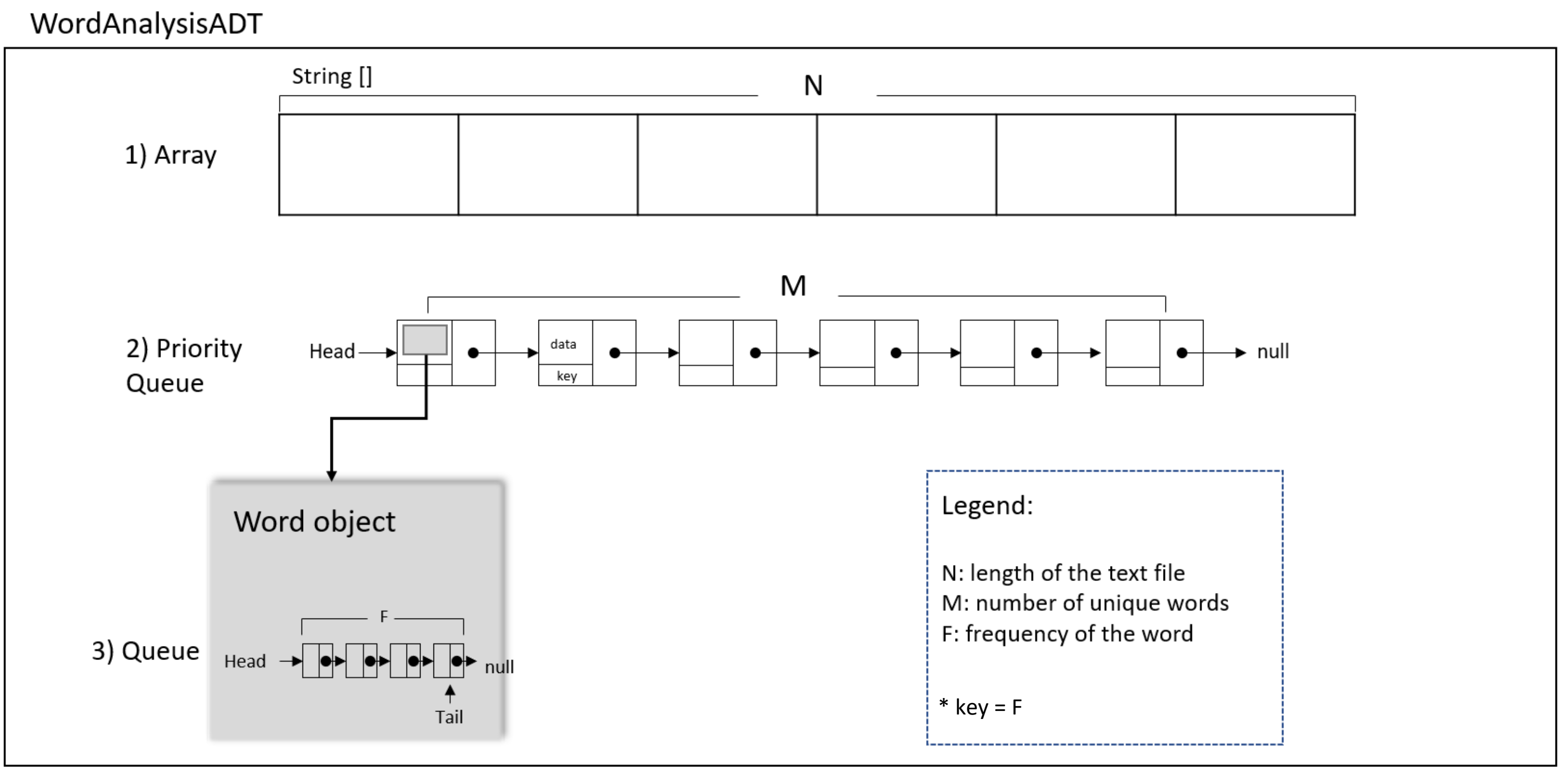
ADT Design Description

The main ADT is an array and a Priority Queue. The array stores all the words in the text file. Each unique word then is stored inside a Word object and enqueued to the Priority Queue. The purpose of using the Priority Queue is to help us store and sort all the unique words in the file by their frequencies. The Word object contains another Queue which stores all the positions of the word.



Operations

1. **Method** totalWords (int total_size)
   requires: list L is not empty.
   input: none
   results: display the total number of words in the text.
   output: total_size

2. **Method** totalUnique (int unique_size)
   requires: list L is not empty.
   input: none
   results: display the total number of unique words in the text.
   output: unique_size.

3. **Method** wordOccurrences (String word, int number)
   requires: list L is not empty.
   input: word
   results: display the total number of occurrences of a particular word.
   output: number

4. **Method** wordsOfLength (int length, int number)
   requires: list L is not empty.
   input: length
   results: display the number of words with this length.
   output: number

5. **Method** allOccurrences ()
   requires: list L is not empty.
   input: none.
   results: display unique words and their occurrences sorted by the total occurrences of each word (from the most frequent to the least).
   output: none.

6. **Method** wordLocations (String word)
   requires: list L is not empty.
   input: word
   results: display the locations of all occurrences of a particular word starting from the top of the text file.
   output: none.

7. **Method** adjacent (String word1, String word2, Boolean flag)
   requires: list L is not empty.
   input: word1 and word2
   results: check if two words are occurring next to each other in the file or not. Return true if the two words are found next to each other, and false otherwise.
   output: flag

8. **Method** InsertUnique(String word)
   requires: word is not null
   input: word.
   results: If the word doesn't exist in the unique priority Queue, a new node will be created then enqueued to the Priority Queue. Otherwise, the new position is added to the word's position queue.
   output: none.

9. **Method** readFile (String file_name)
   Input: file_name
   requires: file exists.
   results: Stores the file's contents in an array. Additionally, it will store all unique words in a Priority Queue along with two defined counters: lineCNT and wordCNT which keeps track of both the number of lines and the position of the word.
   output: none.

Time Complexity

Operation	
1	O (1)
2	O (1)
3	A. First case: O(m) B. Second case: O(m)
4	A. First case: O (1) (any word count gives right result) B. Second case: O(m)
5	O(m)
6	O(m)
7	O(n)
8	O(n)
9	O(n^2)