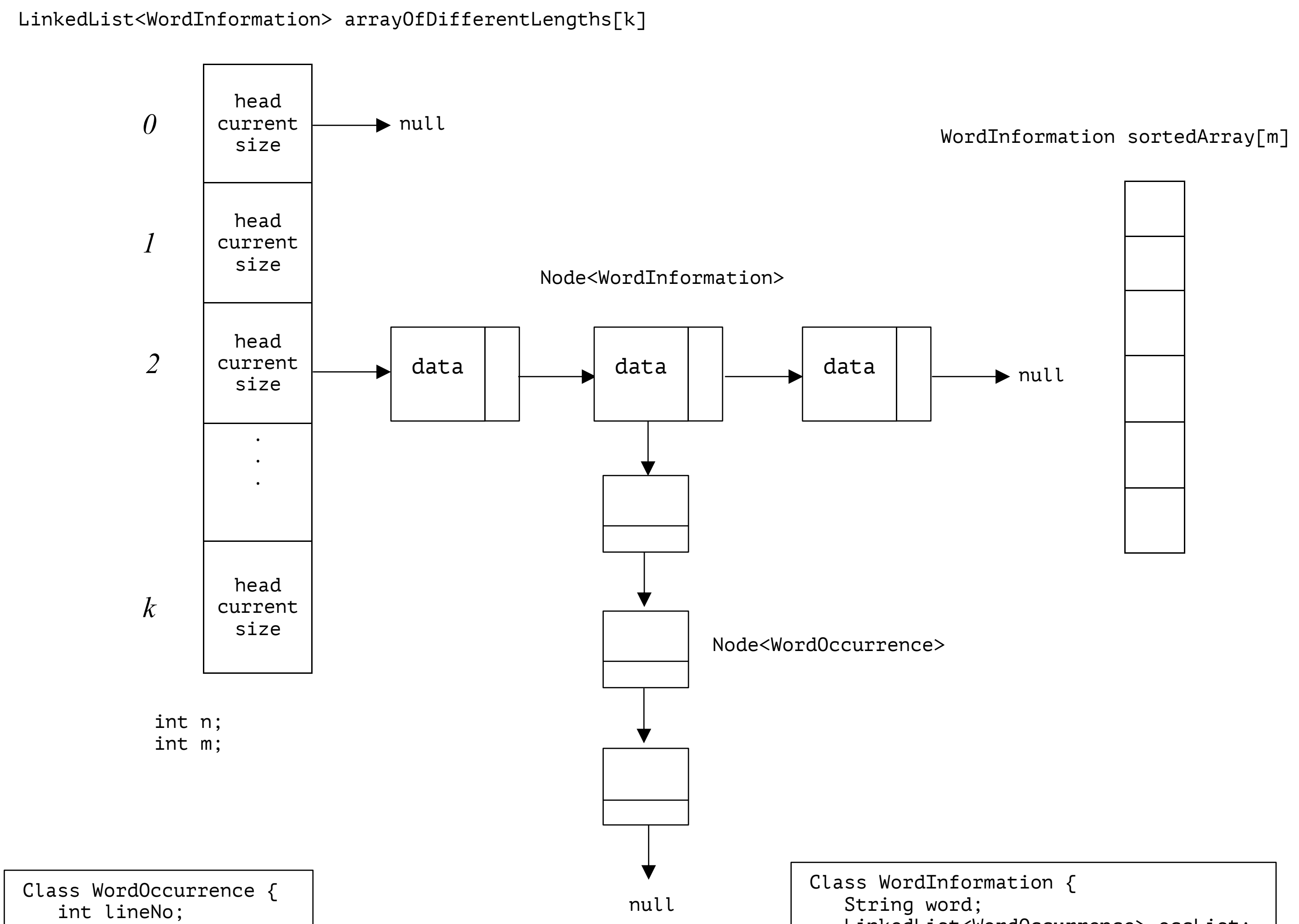
Data Structures (CSC 212)
Fall Trimester 2022
Course Project (Solution)

Phase 1 (10 Marks)

In the first phase of the project, you are asked to describe your suggested design of the ADT for the problem described above and perform the following tasks:

(a) Give a graphical representation of the ADT to show its structure. Make sure to label the diagram clearly.



```
Class WordOccurrence {
   int lineNo;
   int position;
}
```

```
Class WordInformation {
   String word;
   LinkedList<WordOccurrence> occList;
   int size;
}
```

(b) Write at least one paragraph describing your diagram from part (a). Make sure to clearly explain each component in your design. Also, discuss and justify the choices and the assumptions you make.

1- `ArrayofDifferentLengths` will be used to store words such that each index *i* of the array will contain a list of words that have length of *i* characters.

2- The list of Nodes with `WordInformation` type will contain information about each word such as the word itself, a list of occurrences of that word (`LinkedList<WordOccurrence> occList`), and the size of the occurrences list.

3- `occList` will contain the different occurrences of the same word in terms of which line and position it occurs in.

(c) Give a specification of the operations (1), (2), (3), (4), (5), (6), and (7) as well as any other supporting operations you may need to read the text from a text file and store the results in the ADT (e.g., insert).

Specification ADT WordAnalysis

Elements: WordOccurrence, WordInformation, Node<T>, LinkList<T>

Structure: See the graphical representation.

Operations: (Important operations only)

1. Method readFileAndAnalyse (filename f)
   Input: filename f

   Requires: f should be an existing file with valid filename.

   Results: This operation will open file f as input, read the text in the file word by word and analyze the text as follows: When a word w of length x is read, the list for words of length x is searched for word w. If w is found, a new node is inserted for its occurrence, containing w's line number and word position in the line. If w is not found, a new node is inserted in the list for words of length x, containing the string for the word and a new node is inserted for in its occurrence list, containing w's line number and word position.

2. Method documentLength (int l)

   Results: The method will return the length of document (i.e., total number of all words).

   Output: l

3. Method uniqueWords (int u)

   Results: The method will return the number of unique words in the document.

   Output: u

4. Method totalWordsForLength (int l, int t)
   Input: l

Results: The method will return the size of the word list for words with length l.

Output: t

5. Method totalWordsForLength (int l, int t)
   Input: l

   Results: The method will return the size of the word list for words with length l.

   Output: t

6. Method displayUniqueWords ()

   Results: The method will display the unique words in the file sorted by the total occurrences of each word (from the most frequent to the least). If no sorting is performed during the readFileAndAnalyse operation, then this method shall perform sorting of the unique words.

7. Method occurrences (String w, LinkList<WordOccur> l)
   Input: w
   Results: The method will check the length of word w and then the word is searched in the word list of that length. If the word is not found, then null is returned otherwise the occurrence list for the word is returned as l.

   Output: l

8. Method checkAdjacent (String w1, String w2, boolean res)
   Input: w1, w2
   Results: The method will check if both word w1 and word w2 are adjacent to each other by retrieving the occurrence information for each word and verifying the positions of both words. If they are adjacent, then return true otherwise return false.

   Output: res

(d) Provide the time complexity (worst case analysis) for all the operations discussed above using Big O notation. For operations (3) and (4), consider two cases: the first case, when the words in the text file have lengths that are evenly distributed among different lengths (i.e., the words should have different lengths starting from 1 to the longest with k characters), and the second case, when the lengths of words are not evenly distributed. For all operations, assume that the length of the text file is $\boldsymbol{n}$, the number of unique words is $\boldsymbol{m}$, and the longest word in the file has a length of $\boldsymbol{k}$ characters.

(1) An operation to determine the total number of words in a text file (i.e., the length of the file). $O(1)$

(2) An operation to determine the total number of unique words in a text file. O(1)

(3) An operation to determine the total number of occurrences of a particular word. Case 1 = O(m/k), Case 2 = O(m) or O(n)

(4) An operation to determine the total number of words with a particular length occurring in the text file. Case 1= O(1), Case 2 = O(1)

(5) An operation to display the unique words in the file sorted by the total occurrences of each word (from the most frequent to the least). O(m)

(6) An operation to display the locations of the occurrences of a word starting from the top of the text file (i.e., as a list of line and word numbers). Note that every new-line character indicates end of a line. O(n)

(7) An operation to examine if two words are occurring adjacent to each other in a text file (at least one occurrence of both words is needed to satisfy this operation). O(n)