CSE2ALG
19167530
Junyi Su

*Task 1 programs are in A2 directory

Task 2 - Report

Section 1

Class: WordMatch
Attributes:

- StringBuffer wordsBuffer: Append valid words into String[] words
- String[] words: Store all valid words
- String inputFileName1: Input file name(in1.txt)
- String outputFileName1: Output file name(out1.txt) for build lexicon
- String inputFileName2: Input file name(in2.txt) for patterns
- String outputFileName2: Output file name(out1.txt) for matched words
- long startTime: Time when start current method
- long endTime: Time when end current method
- List<LenSameWordNode> lenSameWordNodes: Store words with same length
- List<String> patternList: Store all patterns in in2.txt
- List<Task2Result> result: Store matched words for each pattern

Methods:

- main: State attributes, calculate time cost for each stage, call required methods in order
- loadData: Scan file contents line by line, clear punctuations, split each line by spaces as separate words, store words into wordsBuffer
- readIn1: Scan file contents line by line, get file name in each line, call loadData in a loop to read files
- loadPattern: Scan file contents line by line, get pattern in each line, store patterns into patternList
- creatWordList: When counting frequencies of words, put all words with same length in a list, put words' length into the other list. (Divide-and-conquer algorithm). Firstly add the first LenSameWordNode, its length = words[0].length, then loop String[] words to check if current word.length is already exists in lenSameWordNodes. If it does not exist, add a new LenSameWordNode, its length = wlen.length. Add each word into separate lists sorted by length.
- isEmpty: Determine if the word is empty
- isContainNumber: Determine if the word contains numbers
- writeData(String fileName, List<WordNode> wordList): Write results into out1.txt to build lexicon, frequency and neighbors
- writeData(String fileName,List<Task2Result> result,String pattern): Write results into out2.txt to show all words match each pattern in in2.txt
- createOrderedWordsList: sort matched words in order for each pattern
- swap: swap position for two words

CSE2ALG
19167530
Junyi Su

Class: WordNodeList
Attributes: WordNode wordsHead;
Methods:

- ceateNeighbors: Construct neighbors. As all words are ordered in wordList, so loop the list to find neighbors. Find the words list required includes all words with same length, then find neighbors in this list for current word.
- isNeighbors: Determine whether the two strings are neighbors. Change both prefix to *, check if the remainings are the same. (boolean prefix) Change both suffix to *, check if the remainings are the same. (boolean suffix) If one of the boolean is true, those two words are neighbors.

Class: HeapSortUtil
Attributes:

- T arr[]
- int maxIndex
- int parentNodeIndex
- int leftIndex
- int rightIndex

Methods:

- heatSort:
    1. Initialize the maximum heap
    2. Swap the first element with the last element and adjust the heap length of i=N-1 as the max heap. Continue minimize sort range until there's no unsorted element
       2.1. Swap the first element with the last element
       2.2. Minimize unsorted range as i-1, adjust it to max heap
- initMaxHeap: Construct max heap(unsorted). Start from the last non-leaf node in Complete Binary Tree, if the index of root node in array is 0, the child nodes at index n is 2n+1 and 2n+2, its parent node index is (n-1)/2
- adjustMaxHeap: Adjust B-tree with required parent node as max heap. (If there’s only one element at top, no need adjusts). The node with index i has child nodes with indexs 2i+1 and 2i+2.
  1.1. If leftIndex > maxIndex, this parent node has no child node.
  1.2. Else if rightIndex > maxIndex, resultIndex = leftIndex
  1.3. Else Compare left and right node to get the max as resultIndex
  2.1. Compare parent node with the larger child node
  2.2. If paernt node is not the largest node, swap them. Adjust new sub-tree as it may not a max heap
- max: Get array indexes with larger number. When they are equal, set left node as the max
- swap: swap position for two elements in array

CSE2ALG
19167530
Junyi Su

Class: Helper
  Attributes:
- List<Task2Result> result
- String s
- String pattern

  Methods:
- regex2:
  1. Check if pattern is exactly match(only contains ?)
  1.1 When exactly match, group(0) will only catch chars which matches pattern, the whole word cannot be grouped, so add a *
  2. Check if pattern starts with ? or *
  2.1. startWithString = pattern.charAt(0) + "";
  2.2. add * before pattern
  3. Check if pattern starts with ?
  3.1. add * before pattern
  4. replace ? to [a-zA-Z]{1} ; replace * to [a-zA-Z]{0,}
  5. When a word is added to group(0), check if it matches the condition to add to result
  6. Add elements to result.

Class: WordNode
  Attributes:
- String word
- int frequency
- List<String> neighbors
- WordNode next

  Methods:
- WordNode: Constructor. Set word and frequency for new WordNode
- setWord: set existed word to newWord given
- increaseFrequency: frequency+1
- getNeighbors: WordNodeList.ceateNeighbors(wordList) contructs neighbors, this function outputs neighbors in required form.

Class: Task2Result
  Attributes:
- String key
- int count

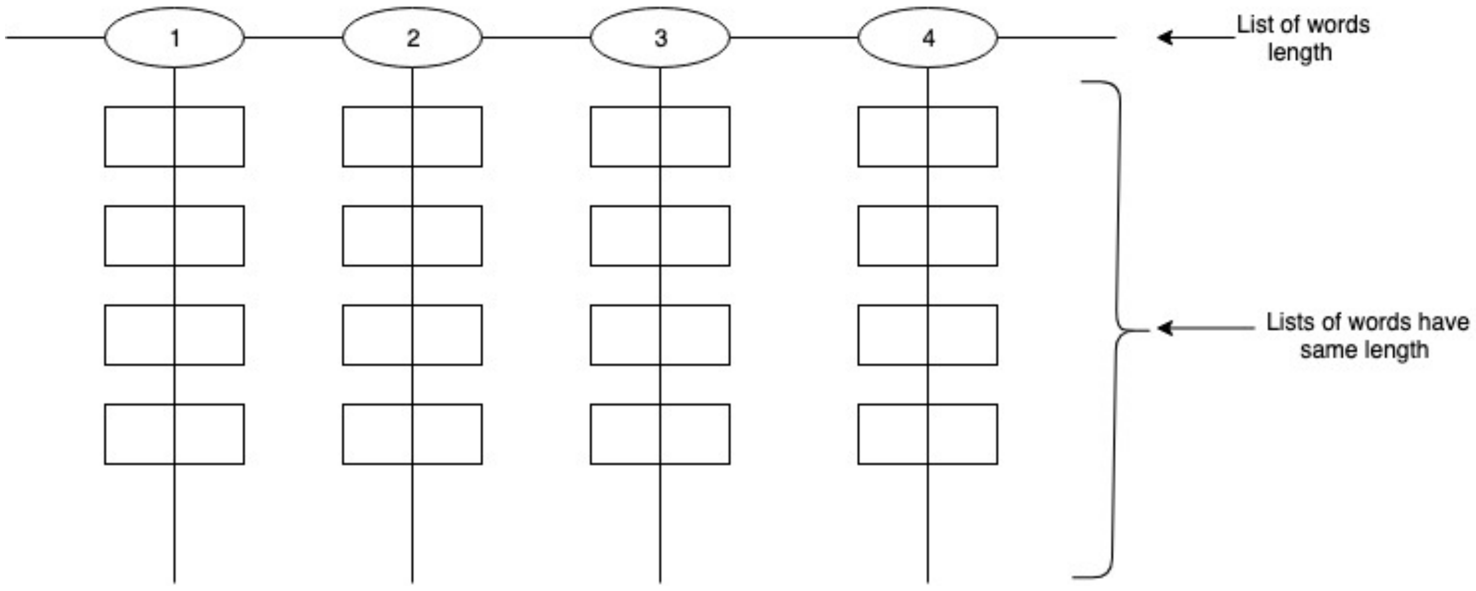
  Methods:
- Task2Result: Constructor. Set key and count for new Task2Result

CSE2ALG
19167530
Junyi Su

Class: LenSameWordNode
Attributes:
- int len
- List<String> words

Section 2

(a). main issues: Sorting algorithm used, Divide-and-conquer algorithm for gathering neighbors
(b). Sorting algorithm: Heap sort
Divide-and-conquer algorithm: Get the length of word firstly, and find the index of this length in List<LenSameWordNode> lenSameWordNodes (List of words length). The words neighbors are all contained in List<String> words under the index (List of words have same length). It saves time for searching neighbors.



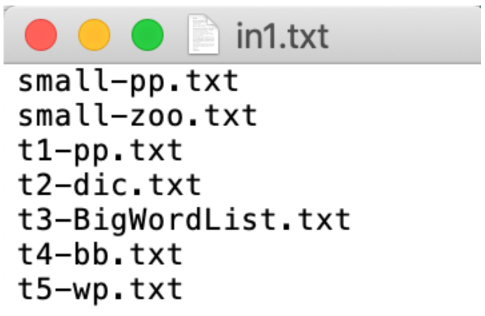
Section3

Test functional correctness by changing input factors. Including the size and format of input files, different patterns.

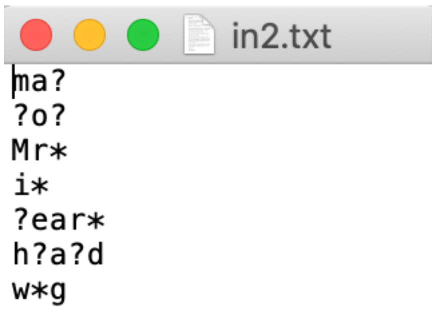
CSE2ALG
19167530
Junyi Su

Section 4

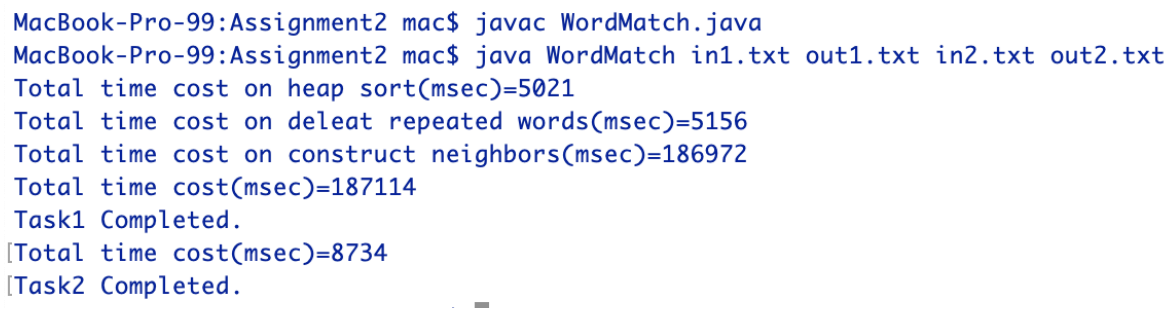
Test the efficiency by calculating time cost in each stage



```
small-pp.txt
small-zoo.txt
t1-pp.txt
t2-dic.txt
t3-BigWordList.txt
t4-bb.txt
t5-wp.txt
```



```
ma?
?o?
Mr*
i*
?ear*
h?a?d
w*g
```



```
MacBook-Pro-99:Assignment2 mac$ javac WordMatch.java
MacBook-Pro-99:Assignment2 mac$ java WordMatch in1.txt out1.txt in2.txt out2.txt
Total time cost on heap sort(msec)=5021
Total time cost on deleat repeated words(msec)=5156
Total time cost on construct neighbors(msec)=186972
Total time cost(msec)=187114
Task1 Completed.
Total time cost(msec)=8734
Task2 Completed.
```