Architecture Report: KWIC Lexical Sort System

Team : CosmoSurf
Members : Pooja, Shreyas, Nikhil
Website: https://nickhegde.github.io/cosmosurf-team/

Introduction

This report describes the architecture and design of the “KWIC Lexical Sort System”. The system takes a given sentence, generates all possible circular shifts of the sentence, and then sorts them lexicographically. The implementation is based on a simple class design with encapsulated logic for generating shifts and sorting.

System Overview

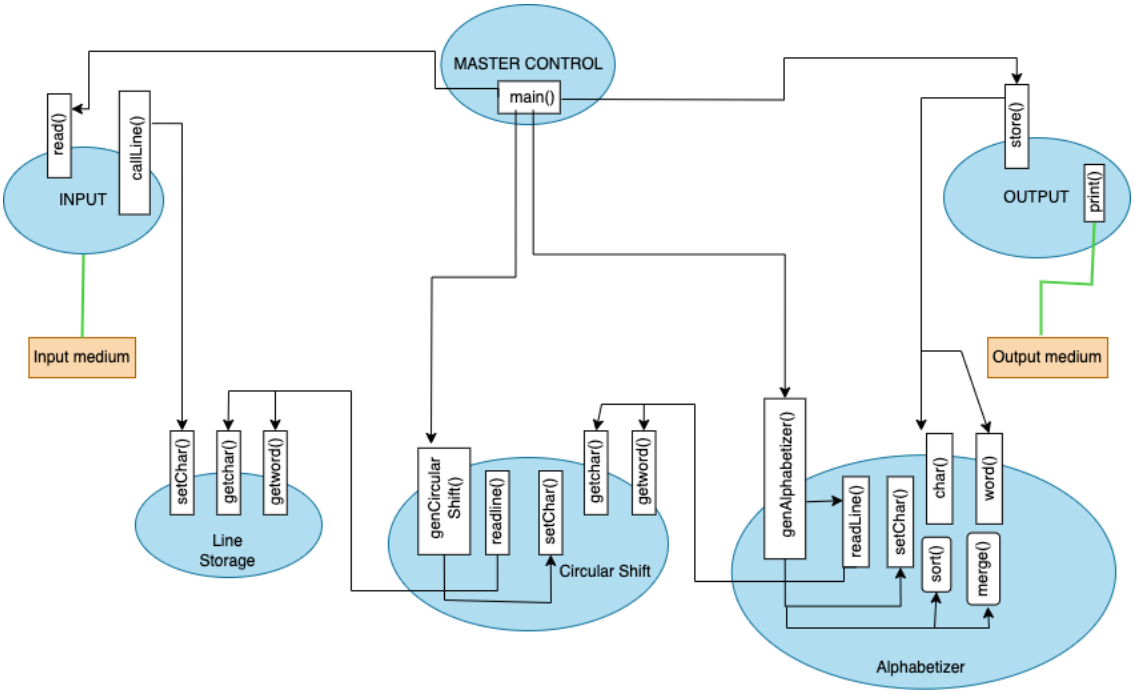
The system implements a ‘Key Word in Context (KWIC)’ system with a focus on circular shifts and lexicographical sorting. The circular shifts involve shifting words in the input sentence and generating all possible variations, where the first word moves to the end in each shift. After generating these shifts, the system sorts them in lexicographical order and displays the results.

Core Components

The system comprises the following core components:

1. KWICLexicalSort Class : The main class that encapsulates the logic for generating circular shifts and sorting.
2. KWICEntry Class : A nested class used to represent each circular shift as an object. This class implements `Comparable` to enable sorting based on lexicographical order.
3. Main Method: The entry point of the program, where an input sentence is provided, shifts are generated, sorted, and printed.

Architecture Diagram



Key Design Concepts

KWICEntry Class:

Encapsulation of Shifted Sentence : The `KWICEntry` class represents each shifted sentence. It encapsulates the string (`shiftedSentence`) representing the shift and overrides the `compareTo` method to allow lexicographical sorting.
Sorting Mechanism : By implementing the `Comparable<KWICEntry>` interface, the `compareTo` method compares the `shiftedSentence` of each `KWICEntry`, enabling natural ordering.

Circular Shift Generation :

Shifting Logic : The `generateCircularShifts` method takes a sentence, splits it into words, and systematically generates circular shifts by

reordering the words. It appends each circular shift to a list of `KWICEntry` objects.
Modular Arithmetic: The circular nature of the shifts is achieved using modular arithmetic (`j % words.length`) to loop back to the beginning of the word array when necessary.

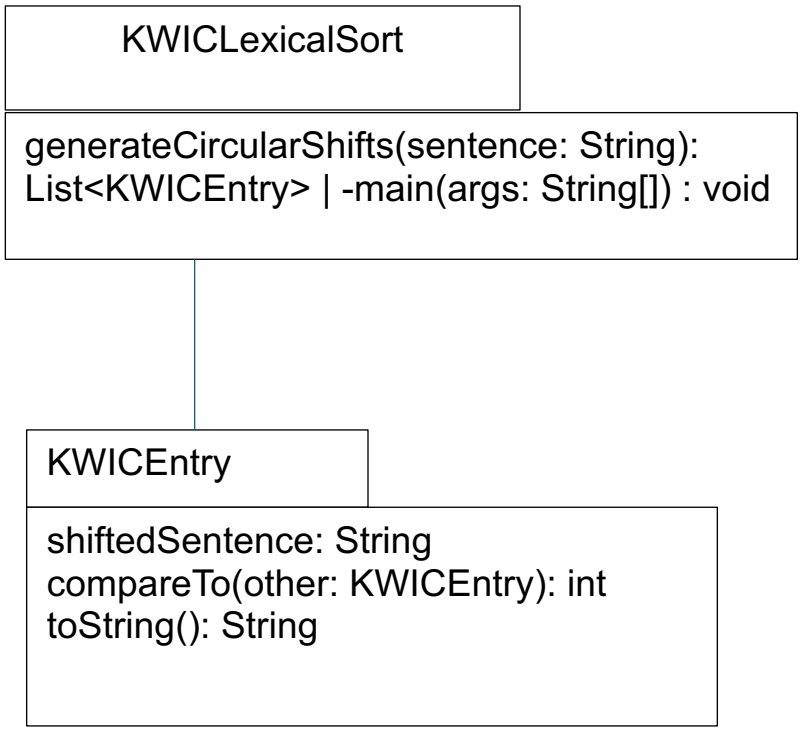
Lexicographical Sorting :

Sorting with Collections.sort() : Once all circular shifts are generated, the system uses `Collections.sort()` to sort the list of `KWICEntry` objects. Since `KWICEntry` implements `Comparable`, this sort is lexicographically based on the shifted sentences.

Sequence of Operations

1. The user inputs a sentence (`inputSentence`).
2. The `generateCircularShifts` method splits the sentence into words and generates all possible circular shifts.
3. Each shift is stored as a `KWICEntry` object in a list.
4. The system outputs all generated circular shifts.
5. The list is then sorted lexicographically using the `Collections.sort()` method.
6. The sorted list is displayed as the final output.

Class Diagram



Explanation of Class Diagram :

KWICLexicalSort Class :
1. Contains the `generateCircularShifts()` method that generates circular shifts from the input sentence.
2. The `main()` method serves as the entry point of the system.

KWICEntry Class :
1. Holds a single circularly shifted sentence (`shiftedSentence`).
2. Implements the `Comparable<KWICEntry>` interface to define the natural ordering for lexicographical sorting.
3. The `toString()` method is used to display each circular shift when printed.

Flow of Execution

1. Initialization : The input sentence "Web browsers display web pages" is passed to the `generateCircularShifts()` method.
2. Shift Generation : For each word in the sentence, a new circular shift is created by rotating the words in the sentence. This continues until all possible shifts are generated.
3. Sorting : The list of generated circular shifts is passed to `Collections.sort()`, which sorts the shifts based on their lexicographical order.
4. Output : The system outputs both the unsorted and sorted lists of circular shifts.

Potential Extensions

1. Support for Multiple Sentences : The system could be extended to handle multiple sentences, storing shifts for multiple input sentences.
2. Different Sorting Criteria : In the future, the system could be extended to allow different sorting criteria, such as sorting based on word length or frequency of words.
3. File Input/Output : The system could be adapted to read input sentences from a file and write the sorted shifts to a file.

Functional Requirements

ID	Description
FR001	The KWIC system shall accept the input entered by keyboard.
FR002	The KWIC system shall accept the ordered set of lines from the user.
FR003	The KWIC system shall accept input characters supported by ASCII character set.
FR004	The System shall circularly shift each word in a line until the original line produced.
FR005	The KWIC system shall be able to add additional modules without any issues.
FR006	The KWIC system shall output the circular lines within 10 seconds.

Non-Functional Requirements

ID	Description
NFR001	The KWIC system shall be user friendly.
NFR002	The KWIC system shall be responsive.
NFR003	The KWIC system shall be good performance.
NFR004	The KWIC system should be enhanceable.
NFR005	The KWIC system should be portable.

Limitations

1. No semantic understanding.
2. Limited scalability.
3. Inablity to capture cross-sentence or document context.
4. Inefficient for multi-keyword searches.
5. Not ideal for all languages.

Software

Conclusion

The KWIC Lexical Sort system is a modular and efficient implementation of generating circular shifts and sorting them lexicographically. By encapsulating the logic for each shift in the `KWICEntry` class and leveraging Java's built-in sorting mechanisms, the system ensures clean and maintainable code. Future extensions can be easily incorporated by modifying or adding functionality to the core methods.