Poetry Assistant Solution

Section 1: Essence of the Solution

The Poetry Assistant employs a Trie data structure and a modified Trie-based search algorithm to efficiently suggest rhymes. It leverages the Trie for quick word retrieval and identification of rhyming patterns. I made a change in my algorithm which analyzes accurate rhyming and delivers word pattern based on common suffixes.

Section 2: Explanation of Algorithms

The Trie-based search algorithm efficiently navigates the Trie structure, identifying words by matching common sound code. I first process all the words in the database and get their sound. Then insert the word and the corresponding sound into the Trie. Finally, when the user enters the word, it is converted into a sound, and the words with the same sound are searched in Tries.

Section 3: Pseudocode

```
Algorithm-InsertWord(rhyme, word)
    node = root
    for char in rhyme
        if char not in node.children
            node.children[char] = new TrieNode()
        node = node.children[char]
    node.isEndOfRhyme = true
    node.words.append(word)

Algorithm-SearchRhymes(rhyme)
    node = root
    for char in rhyme
        if char not in node.children
            return []
        node = node.children[char]
    return node.words
```

Section 4: Data Structures

Trie: Efficient for storing and retrieving words, aiding in quick rhyme suggestions.
TrieNode: Represents each node in the Trie, storing information about the rhymes, word and child nodes.

Section 5: JavaScript Implementation

```
//author: Swen Chan(Siwei, Chen)

//The soundex algorithm can transfer a word into its sound.
function Soundex(word) {
    let soundex_code = word[0];
    const mapping = { 'BFPV': '1', 'CGJKQSXZ': '2', 'DT': '3', 'L': '4',
'MN': '5', 'R': '6' };

    for (const [chars, digit] of Object.entries(mapping)) {
        for (const char of chars) {
            word = word.replace(new RegExp(char, 'g'), digit);
        }
    }

    word = word.slice(1);
    word = REMOVE_CONSECUTIVE_DUPLICATES(word);
    soundex_code += word.padEnd(3, '0').slice(0, 3);

    return soundex_code;
}

//remove the consecutive duplicates inside the sounds of each words
function REMOVE_CONSECUTIVE_DUPLICATES(word) {
    const result = [word[0]];
    for (let i = 1; i < word.length; i++) {
        if (word[i] !== word[i - 1]) {
            result.push(word[i]);
        }
    }
    return result.join('');
}

//The Trie construtive function.
class TrieNode {
```

```
    constructor() {
        this.children = {};
        this.isEndOfRhyme = false;
        this.words = [];
    }
}

//The Assistant which can get input parameter and output corresponding
rhymes.
class RhymingPoetryAssistant {
    constructor() {
        this.root = new TrieNode();
    }

    //Insert a word and its sound into the Trie
    insertWord(soundex_code, word) {
        let node = this.root;

        for (let char of soundex_code) {
            if (!node.children[char]) {
                node.children[char] = new TrieNode();
            }
            node = node.children[char];
        }

        node.isEndOfRhyme = true;
        node.words.push(word);
    }

    //Search the given doundex_code among the Trie
    searchRhymes(soundex_code) {
        let node = this.root;

        for (let char of soundex_code) {
            if (!node.children[char]) {
                return [];
            }
            node = node.children[char];
        }

        return node.words;
    }
}
```

```
// Define the URL of the wordlist, it's given by the document
const wordlistUrl =
'https://introcs.cs.princeton.edu/java/data/wordlist.txt';

// Fetch the wordlist using the Fetch API
fetch(wordlistUrl)
  .then(response => response.text())
  .then(data => {
    // Process the wordlist data as needed
    const wordArray = data.split('\n');

    const poetryAssistant = new RhymingPoetryAssistant();
    for (const word of wordArray) {
        // console.log('word:' + word);
        soundex_word = Soundex(word);
        poetryAssistant.insertWord(soundex_word, word);
    }

  //get the input word and implement poetryAssistant algorithm to
process with it.
  const userInput = 'poetry';
  const inputSoundex = Soundex(userInput);
  const words = poetryAssistant.searchRhymes(inputSoundex);

  //delete the original input word
  words = words.filter(fruit => fruit !== userInput);

  //output the rhyms
  console.log("Rhymes:", words);

  })

  // Handle errors
  .catch(error => {
    console.error('Error fetching wordlist:', error);
  });
```

Section 6: Defects and Remedies

The interactivity of this original algorithm needs to be improved. Currently, it needs to be run with node.js, and the words to be retrieved need to be entered in the code in advance. Next, you can create an html file, design a simple page, and encapsulate it into a web app to achieve better interactivity.