Tries Problems

Here are 10 additional DSA questions covering various aspects of the Trie data structure, from basic to advanced, ensuring no repetition of the problems already completed:

Autocomplete System: Given a list of words and a prefix, implement a function that returns all the words in the list that start with the given prefix.

Delete a Word from a Trie: Implement a function to delete a word from a Trie. Ensure the Trie remains valid after deletion, removing unnecessary nodes.

Longest Common Prefix of a Set of Words: Given an array of words, find the longest common prefix that is shared by all the words.

Search for Words with Wildcards ('?'): Implement a function that searches for words in a Trie that match a given pattern containing wildcards. The '?' character can match any letter.

Count Words with Given Prefix: Given a list of words, count how many words have a specific prefix using Trie.

Palindrome Pair Check in Trie: Given a list of words, find if there exists a pair of words such that one word is the reverse of the other, using Trie to optimize the search.

Maximum XOR of Two Numbers in an Array: Use Trie to find the maximum XOR of two numbers in an array. This is a more advanced bit manipulation problem combined with Trie.

Shortest Unique Prefix for Every Word: For each word in a given list, find the shortest unique prefix that distinguishes it from all other words in the list.

Word Suggestions Based on Frequency: Design a system where, based on user input, the system suggests the most frequent words that match the input prefix using Trie with frequency tracking.

Concatenated Words: Given a list of words, find all the words that can be formed by concatenating two or more words from the list using a Trie.

explain tries problems with example that contains input and output in simple way and give optimized and simple java code with its time complexity

Memory updated

Here's an explanation and example for each Trie problem with Java code and time complexity analysis:

---

1. Autocomplete System

- **Problem**: Given a list of words and a prefix, return all words in the list that start with the given prefix.
- **Example**:
  - Input: `` `["dog", "deer", "deal"], prefix = "de"` ``
  - Output: `` `["deer", "deal"]` ``

Java Code:

```java
class TrieNode {
    TrieNode[] children = new TrieNode[26];
    boolean isEndOfWord;
}

class Trie {
    TrieNode root;

    public Trie() {
        root = new TrieNode();
    }

    public void insert(String word) {
        TrieNode node = root;
        for (char c : word.toCharArray()) {
            if (node.children[c - 'a'] == null) {
                node.children[c - 'a'] = new TrieNode();
            }
            node = node.children[c - 'a'];
        }
        node.isEndOfWord = true;
    }

    public List<String> autocomplete(String prefix) {
        List<String> result = new ArrayList<>();
        TrieNode node = root;
        for (char c : prefix.toCharArray()) {
            if (node.children[c - 'a'] == null) {
                return result;
            }
            node = node.children[c - 'a'];
        }
        dfs(node, prefix, result);
        return result;
    }

    private void dfs(TrieNode node, String prefix, List<String> result) {
        if (node.isEndOfWord) {
            result.add(prefix);
        }
        for (char c = 'a'; c <= 'z'; c++) {
            if (node.children[c - 'a'] != null) {
                dfs(node.children[c - 'a'], prefix + c, result);
            }
        }
    }
}
```

- **Time Complexity**:
  - Insertion: O(m), where m is the length of the word.
  - Autocomplete search: O(n + k), where n is the length of the prefix and k is the number of results.

---

2. Delete a Word from a Trie

- **Problem**: Implement a function to delete a word from a Trie, ensuring that the structure remains valid.
- **Example**:
  - Input: Insert `["apple", "app"], delete "app"`
  - Output: Trie will have only `"apple"`

Java Code:

```java
class Trie {
    TrieNode root;

    public Trie() {
        root = new TrieNode();
    }

    public void insert(String word) {
        TrieNode node = root;
        for (char c : word.toCharArray()) {
            if (node.children[c - 'a'] == null) {
                node.children[c - 'a'] = new TrieNode();
            }
            node = node.children[c - 'a'];
        }
        node.isEndOfWord = true;
    }

    public boolean delete(String word) {
        return delete(root, word, 0);
    }

    private boolean delete(TrieNode node, String word, int index) {
        if (index == word.length()) {
            if (!node.isEndOfWord) return false;
            node.isEndOfWord = false;
            return node.children.length == 0;
        }
        char c = word.charAt(index);
        TrieNode nextNode = node.children[c - 'a'];
        if (nextNode == null) return false;

        boolean shouldDelete = delete(nextNode, word, index + 1);
        if (shouldDelete) {
            node.children[c - 'a'] = null;
            return node.children.length == 0 && !node.isEndOfWord;
        }
        return false;
    }
}
```

- **Time Complexity**: O(m), where m is the length of the word being deleted.

---

3. Longest Common Prefix of a Set of Words

- **Problem**: Find the longest common prefix shared by all words in an array.
- **Example**:
  - Input: `["flower", "flow", "flight"]`
  - Output: `"fl"`

Java Code:

```java
public String longestCommonPrefix(String[] words) {
    if (words == null || words.length == 0) return "";
    Trie trie = new Trie();
    for (String word : words) {
        trie.insert(word);
    }
    return trie.findLongestCommonPrefix();
}

public String findLongestCommonPrefix() {
    TrieNode node = root;
    StringBuilder prefix = new StringBuilder();
    while (countChildren(node) == 1 && !node.isEndOfWord) {
        for (char c = 'a'; c <= 'z'; c++) {
            if (node.children[c - 'a'] != null) {
                prefix.append(c);
                node = node.children[c - 'a'];
                break;
            }
        }
    }
    return prefix.toString();
}

private int countChildren(TrieNode node) {
    int count = 0;
    for (int i = 0; i < 26; i++) {
        if (node.children[i] != null) count++;
    }
    return count;
}
```

- **Time Complexity**: O(n * m), where n is the number of words and m is the length of the shortest word.

I'll continue with the next set of problems shortly!

give remaining

Continuing with the remaining Trie problems:

---

4. Search for Words with Wildcards ('?')

- **Problem**: Search for words in a Trie that match a pattern containing wildcards. The '?' character can match any letter.
- **Example**:
  - Input: `` `["bad", "dad", "mad"], pattern = "b?d"` ``
  - Output: `` `["bad"]` ``

Java Code:

java

```java
class TrieNode {
    TrieNode[] children = new TrieNode[26];
    boolean isEndOfWord;
}

class Trie {
    TrieNode root;

    public Trie() {
        root = new TrieNode();
    }

    public void insert(String word) {
        TrieNode node = root;
        for (char c : word.toCharArray()) {
            if (node.children[c - 'a'] == null) {
                node.children[c - 'a'] = new TrieNode();
            }
            node = node.children[c - 'a'];
        }
        node.isEndOfWord = true;
    }

    public List<String> searchWithWildcards(String pattern) {
        List<String> result = new ArrayList<>();
        searchWithWildcards(root, pattern, 0, new StringBuilder(), result);
        return result;
    }

    private void searchWithWildcards(TrieNode node, String pattern, int index,
StringBuilder current, List<String> result) {
        if (index == pattern.length()) {
            if (node.isEndOfWord) {
                result.add(current.toString());
            }
            return;
        }

        char c = pattern.charAt(index);
        if (c == '?') {
            for (char i = 'a'; i <= 'z'; i++) {
                TrieNode child = node.children[i - 'a'];
                if (child != null) {
                    current.append(i);
                    searchWithWildcards(child, pattern, index + 1, current, result);
                    current.deleteCharAt(current.length() - 1);
                }
            }
        } else {
            TrieNode child = node.children[c - 'a'];
            if (child != null) {
                current.append(c);
                searchWithWildcards(child, pattern, index + 1, current, result);
                current.deleteCharAt(current.length() - 1);
            }
        }
    }
}
```

- **Time Complexity**: O(m * 26^w), where m is the number of words and w is the length of the pattern.

---

5. Count Words with Given Prefix

- **Problem**: Count how many words have a specific prefix.

- **Example**:
  - Input: `` `["apple", "app", "appl"], prefix = "app"` ``
  - Output: `` `3` ``

Java Code:

```java
class TrieNode {
    TrieNode[] children = new TrieNode[26];
    boolean isEndOfWord;
    int wordCount; // Stores number of words passing through this node
}

class Trie {
    TrieNode root;

    public Trie() {
        root = new TrieNode();
    }

    public void insert(String word) {
        TrieNode node = root;
        for (char c : word.toCharArray()) {
            if (node.children[c - 'a'] == null) {
                node.children[c - 'a'] = new TrieNode();
            }
            node = node.children[c - 'a'];
            node.wordCount++;
        }
        node.isEndOfWord = true;
    }

    public int countWordsWithPrefix(String prefix) {
        TrieNode node = root;
        for (char c : prefix.toCharArray()) {
            if (node.children[c - 'a'] == null) {
                return 0;
            }
            node = node.children[c - 'a'];
        }
        return node.wordCount;
    }
}
```

- **Time Complexity**: O(m), where m is the length of the prefix.

---

6. Palindrome Pair Check in Trie

- **Problem**: Find if there exists a pair of words such that one word is the reverse of the other.
- **Example**:
  - Input: `` `["abcd", "dcba", "lls", "s", "sssll"]` ``
  - Output: `` `true` `` (since "abcd" and "dcba" are a pair)

Java Code:

```java
class TrieNode {
    TrieNode[] children = new TrieNode[26];
    boolean isEndOfWord;
    int index; // Store index of word for palindrome pair check
```

```
}

class Trie {
    TrieNode root;

    public Trie() {
        root = new TrieNode();
    }

    public void insert(String word, int index) {
        TrieNode node = root;
        for (char c : word.toCharArray()) {
            if (node.children[c - 'a'] == null) {
                node.children[c - 'a'] = new TrieNode();
            }
            node = node.children[c - 'a'];
        }
        node.isEndOfWord = true;
        node.index = index;
    }

    public boolean checkPalindromePairs(String[] words) {
        for (int i = 0; i < words.length; i++) {
            insert(words[i], i);
        }
        for (int i = 0; i < words.length; i++) {
            if (isPalindromePair(words[i], i)) return true;
        }
        return false;
    }

    private boolean isPalindromePair(String word, int index) {
        TrieNode node = root;
        for (int j = 0; j < word.length(); j++) {
            if (node.isEndOfWord && isPalindrome(word, j, word.length() - 1)) {
                if (node.index != index) return true;
            }
            node = node.children[word.charAt(j) - 'a'];
            if (node == null) return false;
        }
        if (node.isEndOfWord && node.index != index) return true;
        return false;
    }

    private boolean isPalindrome(String word, int start, int end) {
        while (start < end) {
            if (word.charAt(start) != word.charAt(end)) return false;
            start++;
            end--;
        }
        return true;
    }
}
```

- **Time Complexity**: O(n * m^2), where n is the number of words and m is the length of the longest word.

---

7. Maximum XOR of Two Numbers in an Array

- **Problem**: Find the maximum XOR of two numbers in an array using Trie.
- **Example**:
  - Input: `[3, 10, 5, 25, 2, 8]`
  - Output: `28` (since 5 ^ 25 = 28)

Java Code:

java

```java
class TrieNode {
    TrieNode[] children = new TrieNode[2]; // 0 or 1
}

class Trie {
    TrieNode root;

    public Trie() {
        root = new TrieNode();
    }

    public void insert(int num) {
        TrieNode node = root;
        for (int i = 31; i >= 0; i--) {
            int bit = (num >> i) & 1;
            if (node.children[bit] == null) {
                node.children[bit] = new TrieNode();
            }
            node = node.children[bit];
        }
    }

    public int findMaxXor(int num) {
        TrieNode node = root;
        int maxXor = 0;
        for (int i = 31; i >= 0; i--) {
            int bit = (num >> i) & 1;
            int toggledBit = 1 - bit;
            if (node.children[toggledBit] != null) {
                maxXor = (maxXor << 1) | 1;
                node = node.children[toggledBit];
            } else {
                maxXor = maxXor << 1;
                node = node.children[bit];
            }
        }
        return maxXor;
    }
}

public int findMaximumXOR(int[] nums) {
    Trie trie = new Trie();
    int maxXor = 0;
    for (int num : nums) {
        trie.insert(num);
        maxXor = Math.max(maxXor, trie.findMaxXor(num));
    }
    return maxXor;
}
```

- **Time Complexity**: O(n * m), where n is the number of elements and m is the number of bits (32 for integers).

---

8. Shortest Unique Prefix for Every Word

- **Problem**: For each word, find the shortest unique prefix that distinguishes it from all other words.
- **Example**:
  - Input: `` `["apple", "app", "appl"]` ``
  - Output: `` `["apple", "app", "appl"]` ``

Java Code:

```java
class TrieNode {
    TrieNode[] children = new TrieNode[26];
    boolean isEndOfWord;
    int prefixCount; // Count how many times this prefix appears
}

class Trie {
    TrieNode root;

    public Trie() {
        root = new TrieNode();
    }

    public void insert(String word) {
        TrieNode node = root;
        for (char c : word.toCharArray()) {
            if (node.children[c - 'a'] == null) {
                node.children[c - 'a'] = new TrieNode();
            }
            node = node.children[c - 'a'];
            node.prefixCount++;
        }
        node.isEndOfWord = true;
```

8. Shortest Unique Prefix for Every Word

give remaining from next

Certainly! Continuing from where we left off:

---

8. Shortest Unique Prefix for Every Word (Continued)

- **Problem**: For each word, find the shortest unique prefix that distinguishes it from all other words.
- **Example**:
  - Input: `` `["apple", "app", "appl"]` ``
  - Output: `` `["apple", "app", "appl"]` ``

Java Code (Continued):

```java
class TrieNode {
    TrieNode[] children = new TrieNode[26];
    boolean isEndOfWord;
    int prefixCount; // Count how many times this prefix appears
}

class Trie {
```

```
    TrieNode root;

    public Trie() {
        root = new TrieNode();
    }

    public void insert(String word) {
        TrieNode node = root;
        for (char c : word.toCharArray()) {
            if (node.children[c - 'a'] == null) {
                node.children[c - 'a'] = new TrieNode();
            }
            node = node.children[c - 'a'];
            node.prefixCount++;
        }
        node.isEndOfWord = true;
    }

    public String[] findShortestUniquePrefix(String[] words) {
        String[] result = new String[words.length];
        Trie trie = new Trie();
        for (int i = 0; i < words.length; i++) {
            trie.insert(words[i]);
        }
        for (int i = 0; i < words.length; i++) {
            result[i] = findPrefix(trie.root, words[i], new StringBuilder());
        }
        return result;
    }

    private String findPrefix(TrieNode node, String word, StringBuilder prefix) {
        for (int i = 0; i < word.length(); i++) {
            char c = word.charAt(i);
            prefix.append(c);
            if (node.children[c - 'a'].prefixCount == 1) {
                return prefix.toString();
            }
            node = node.children[c - 'a'];
        }
        return word; // Fallback case, shouldn't reach here
    }
}
```

- **Time Complexity**: O(n * m), where n is the number of words and m is the length of the longest word.

---

9. Word Suggestions Based on Frequency (Continued)

- **Problem**: Design a system that suggests the most frequent words based on user input prefix using Trie with frequency tracking.
- **Example**:
  - Input: `` `["apple", "apricot", "banana", "apartment", "application"]` ``, User input = `` `"ap"` ``
  - Output: `` `["apple", "apricot", "apartment", "application"]` ``

Java Code (Continued):

```
java
```

```
class TrieNode {
    TrieNode[] children = new TrieNode[26];
    boolean isEndOfWord;
    Map<String, Integer> frequencyMap = new HashMap<>(); // Store word frequency
}

class Trie {
```

```
    TrieNode root;

    public Trie() {
        root = new TrieNode();
    }

    public void insert(String word) {
        TrieNode node = root;
        for (char c : word.toCharArray()) {
            if (node.children[c - 'a'] == null) {
                node.children[c - 'a'] = new TrieNode();
            }
            node = node.children[c - 'a'];
            node.frequencyMap.put(word, node.frequencyMap.getOrDefault(word, 0) + 1);
        }
        node.isEndOfWord = true;
    }

    public List<String> suggestWords(String prefix) {
        TrieNode node = root;
        for (char c : prefix.toCharArray()) {
            if (node.children[c - 'a'] == null) {
                return new ArrayList<>();
            }
            node = node.children[c - 'a'];
        }
        List<Map.Entry<String, Integer>> list = new ArrayList<>
(node.frequencyMap.entrySet());
        list.sort((a, b) -> b.getValue() - a.getValue()); // Sort by frequency
        List<String> result = new ArrayList<>();
        for (int i = 0; i < Math.min(3, list.size()); i++) { // Get top 3 suggestions
            result.add(list.get(i).getKey());
        }
        return result;
    }
}
```

- **Time Complexity**: O(n * m), where n is the number of words and m is the average length of the words.

---

10. Concatenated Words

- **Problem**: Find all words in a list that can be formed by concatenating two or more words from the list using a Trie.
- **Example**:
  - Input: `` `["cat", "cats", "catsdogcats", "dog", "dogcatsdog", "hippopotamuses", "rat", "ratcatdogcat"]` ``
  - Output: `` `["catsdogcats", "dogcatsdog", "ratcatdogcat"]` ``

Java Code:

```
java
```

```
class TrieNode {
    TrieNode[] children = new TrieNode[26];
    boolean isEndOfWord;
}

class Trie {
    TrieNode root;
    Set<String> wordsSet;

    public Trie() {
        root = new TrieNode();
```

```
        wordsSet = new HashSet<>();
    }

    public void insert(String word) {
        TrieNode node = root;
        for (char c : word.toCharArray()) {
            if (node.children[c - 'a'] == null) {
                node.children[c - 'a'] = new TrieNode();
            }
            node = node.children[c - 'a'];
        }
        node.isEndOfWord = true;
        wordsSet.add(word);
    }

    public List<String> findConcatenatedWords(String[] words) {
        List<String> result = new ArrayList<>();
        for (String word : words) {
            if (word.isEmpty()) continue;
            wordsSet.remove(word); // Remove current word to avoid self-concatenation
            if (canForm(word)) {
                result.add(word);
            }
            wordsSet.add(word); // Add word back to set
        }
        return result;
    }

    private boolean canForm(String word) {
        int n = word.length();
        if (n == 0) return true;

        boolean[] dp = new boolean[n + 1];
        dp[0] = true;

        for (int i = 1; i <= n; i++) {
            for (int j = 0; j < i; j++) {
                if (dp[j] && wordsSet.contains(word.substring(j, i))) {
                    dp[i] = true;
                    break;
                }
            }
        }
        return dp[n];
    }
}
```

- **Time Complexity**:
  - Insertion into Trie: O(m * n), where m is the average length of words and n is the number of words.
  - Checking concatenation: O(n * m^2), where n is the number of words and m is the average length of words.

These implementations cover a range of Trie-related problems from basic to advanced, providing optimized Java code along with explanations of input, output, and time complexity.