SMART DICTIONARY

VER 1.0

CS163 - Data structure

Võ Minh Nhân - 18125136

Nguyễn Lê Thanh Khiết - 18125088

Table of contents

1. INTRODUCTION

The purpose of this technical report is to provide an overview of the design and algorithms used in the dictionary application. The application allows users to search for words, add new words with their definitions, update word definitions, and delete words. We will discuss the design choices made for loading a dataset, searching, adding, updating, and deleting words. Additionally, we will evaluate the algorithmic complexity (O notation) of each action and report on their running times.

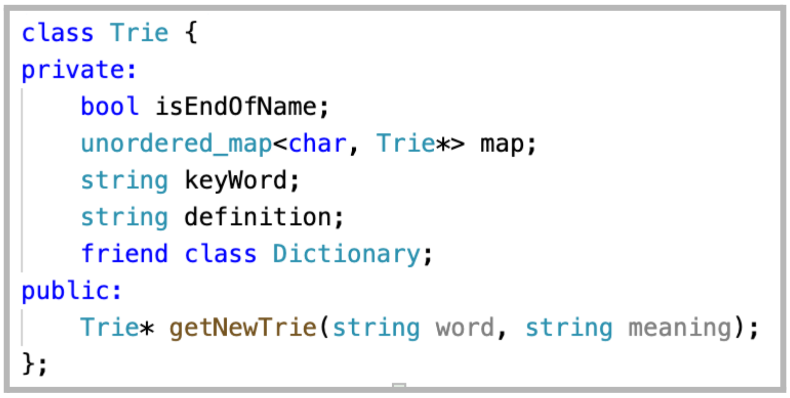
2. DATA STRUCTURE

In the design and implementation of the dictionary application, several key data structures were employed to efficiently manage the storage, retrieval, and manipulation of word data and their corresponding definitions. The primary data structure used in the application is the Trie, also known as a prefix tree. Let's provide an overview and analysis of the data structures used:

2.1. Trie

The Trie is the central data structure used in the dictionary application. It is particularly well-suited for tasks involving string data, such as storing and searching for words efficiently. Trie is a type of k-ary search tree used for storing and searching a specific key from a set. Using Trie, search complexities can be brought to optimal limit (key length).

2.1.1 Structure



```
class Trie {
private:
    bool isEndOfName;
    unordered_map<char, Trie*> map;
    string keyWord;
    string definition;
    friend class Dictionary;
public:
    Trie* getNewTrie(string word, string meaning);
};
```

private Members:

- **isEndOfName**: A boolean flag that indicates whether the current Trie node represents the end of a valid keyword or not.
- **map**: An unordered map that maps characters to child Trie nodes. This represents the branching structure of the Trie.

- **keyWord**: A string that stores the keyword of Dictionary associated with the current Trie node.
- **definition**: A string that stores the definition associated with the keyword. This would be populated only if isEndOfName is true.

2.1.2 Advantages

- **Efficient Searching**: Tries excel in searching for words and prefixes. The depth of the tree is determined by the length of the longest word, leading to fast retrieval times.
- **Space Optimization**: Tries optimize memory usage by sharing common prefixes among words.
- **Auto-Completion**: Tries can provide auto-completion suggestions based on partially entered words.

2.1.3 Analysis

- **Insertion**: The insertion of a word into the Trie takes O(m) time, where m is the length of the word. This is because the algorithm involves traversing down the tree, creating nodes as needed for each character.
- **Searching**: The search operation for a word takes O(m) time, where m is the length of the queried word. The algorithm traverses the Trie character by character, either finding the word or determining its absence.
- **Space Complexity**: The space complexity of a Trie is O(N * L), where N is the number of words and L is the average length of words. While this is efficient for most use cases, it might lead to increased memory consumption for extremely large datasets with long words.

.

2.2 Loading a Dataset

2.2.1 Algorithm

The dictionary application supports loading words and their definitions from external files. The design employs a Trie data structure to efficiently store and retrieve words. When loading a dataset, the application reads each line of the file, splits it into word and definition, and inserts the word into the Trie.

2.2.2 Evaluation

Algorithm Complexity: O(n), where n is the number of words in the dataset.

2.3 Searching

2.3.1 Algorithm

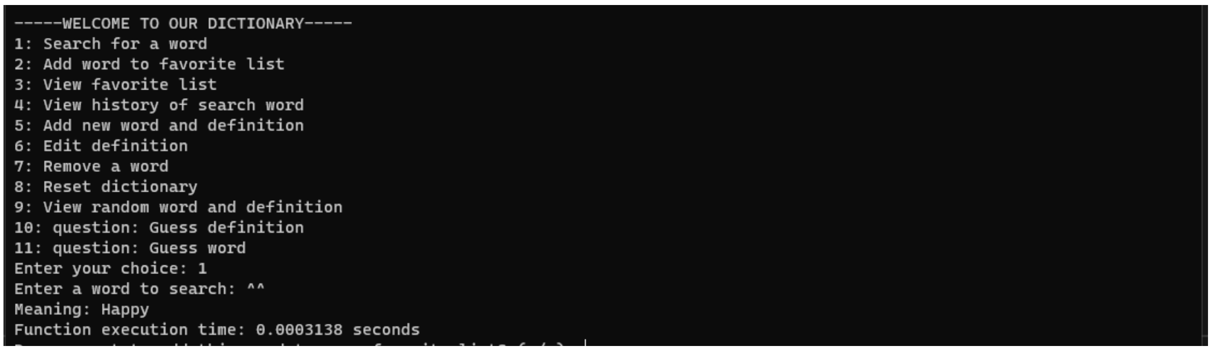
The search algorithm is based on Trie traversal and provides support for searching both by keywords and definitions. The search for words with the same definition involves traversing the Trie and has a time complexity proportional to the number of nodes in the Trie with the matching definition.

Search Flow:

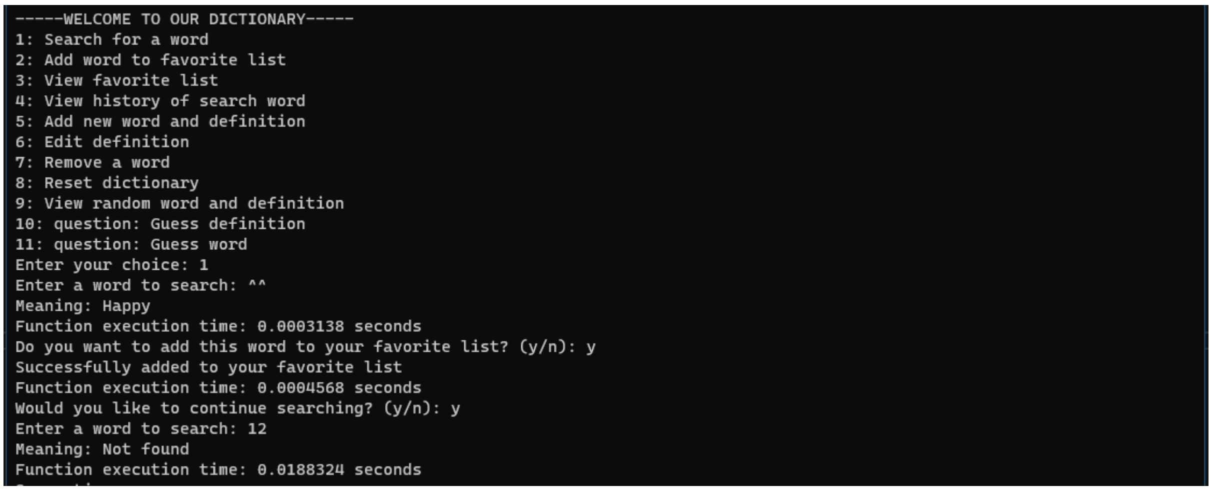
- Normalization: Convert the input query to lowercase for case-insensitive searching.
- Search by Keyword:
  - Traverse the Trie using each character of the normalized query.
  - If characters are not found during traversal, return "Not found."
  - If traversal completes and a keyword is found, return its definition.
- Search by Definition (Fallback):
  - If no exact keyword match is found, search for keywords with similar definitions.
  - Recursively traverse the Trie and collect keywords with matching definitions.
- Result Generation:
  - If an exact match was found (step 2), return the keyword's definition.
  - If only similar definitions are found, format and return matching keywords.

2.3.2 Evaluation

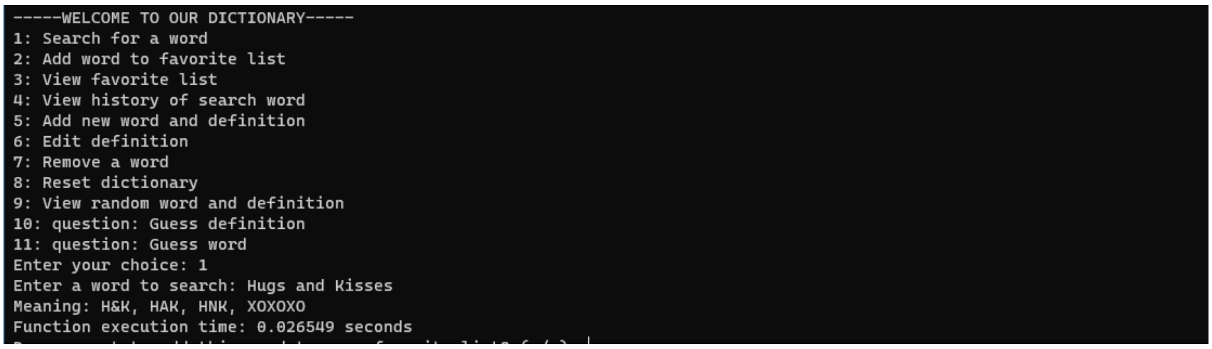
Algorithm Complexity: O(m), where m is the length of the query.



```
-----WELCOME TO OUR DICTIONARY-----
1: Search for a word
2: Add word to favorite list
3: View favorite list
4: View history of search word
5: Add new word and definition
6: Edit definition
7: Remove a word
8: Reset dictionary
9: View random word and definition
10: question: Guess definition
11: question: Guess word
Enter your choice: 1
Enter a word to search: ^^
Meaning: Happy
Function execution time: 0.0003138 seconds
```

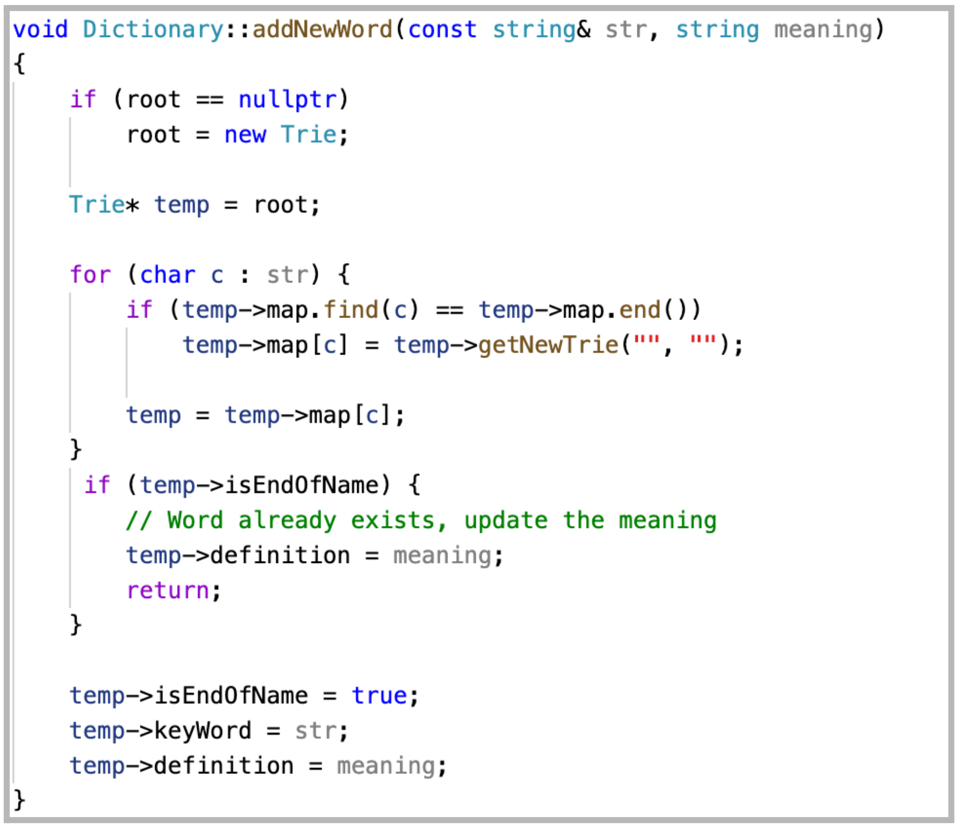


```
-----WELCOME TO OUR DICTIONARY-----
1: Search for a word
2: Add word to favorite list
3: View favorite list
4: View history of search word
5: Add new word and definition
6: Edit definition
7: Remove a word
8: Reset dictionary
9: View random word and definition
10: question: Guess definition
11: question: Guess word
Enter your choice: 1
Enter a word to search: ^^
Meaning: Happy
Function execution time: 0.0003138 seconds
Do you want to add this word to your favorite list? (y/n): y
Successfully added to your favorite list
Function execution time: 0.0004568 seconds
Would you like to continue searching? (y/n): y
Enter a word to search: 12
Meaning: Not found
Function execution time: 0.0188324 seconds
```



```
-----WELCOME TO OUR DICTIONARY-----
1: Search for a word
2: Add word to favorite list
3: View favorite list
4: View history of search word
5: Add new word and definition
6: Edit definition
7: Remove a word
8: Reset dictionary
9: View random word and definition
10: question: Guess definition
11: question: Guess word
Enter your choice: 1
Enter a word to search: Hugs and Kisses
Meaning: H&K, HAK, HNK, XOXOXO
Function execution time: 0.026549 seconds
```

2.4 Adding a Word



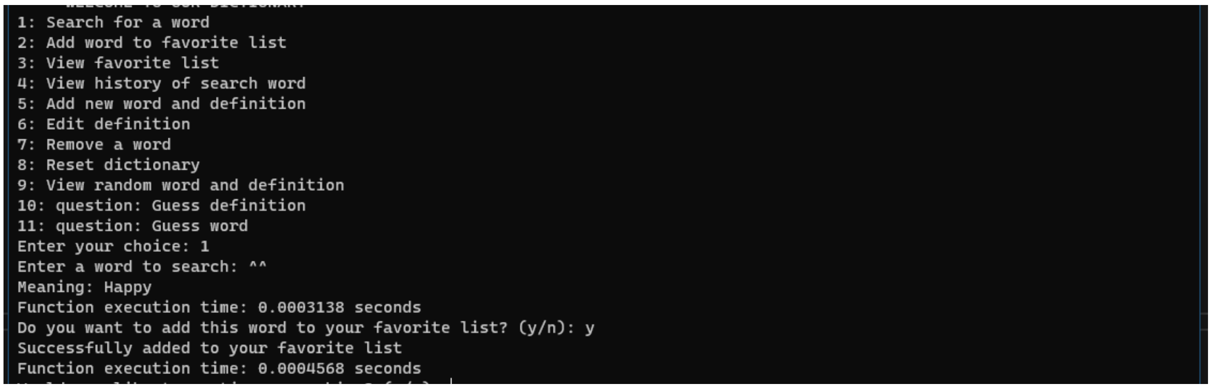
```cpp
void Dictionary::addNewWord(const string& str, string meaning)
{
    if (root == nullptr)
        root = new Trie;

    Trie* temp = root;

    for (char c : str) {
        if (temp->map.find(c) == temp->map.end())
            temp->map[c] = temp->getNewTrie("", "");

        temp = temp->map[c];
    }
     if (temp->isEndOfName) {
        // Word already exists, update the meaning
        temp->definition = meaning;
        return;
    }

    temp->isEndOfName = true;
    temp->keyWord = str;
    temp->definition = meaning;
}
```

2.4.1 Algorithm

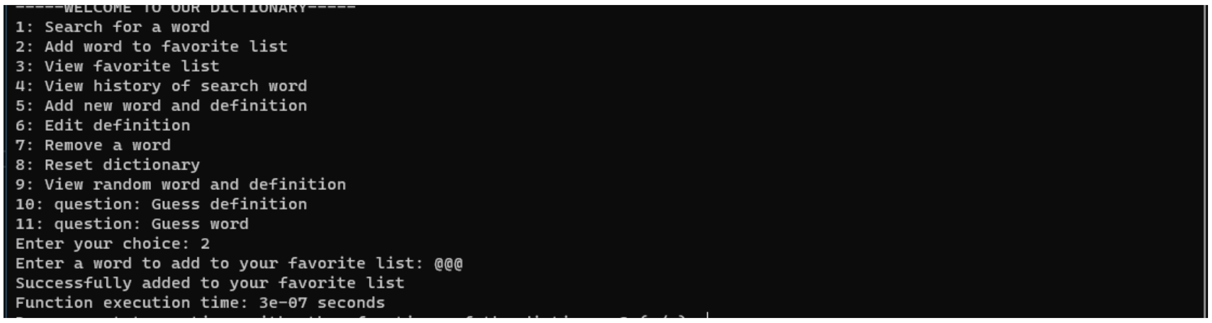
When a user adds a new word, the application iterates through the characters of the word, navigating the Trie to find the appropriate position to insert the word. If the word already exists, the algorithm updates its definition.

2.4.2 Evaluation

Algorithm Complexity: O(m), where m is the length of the word.

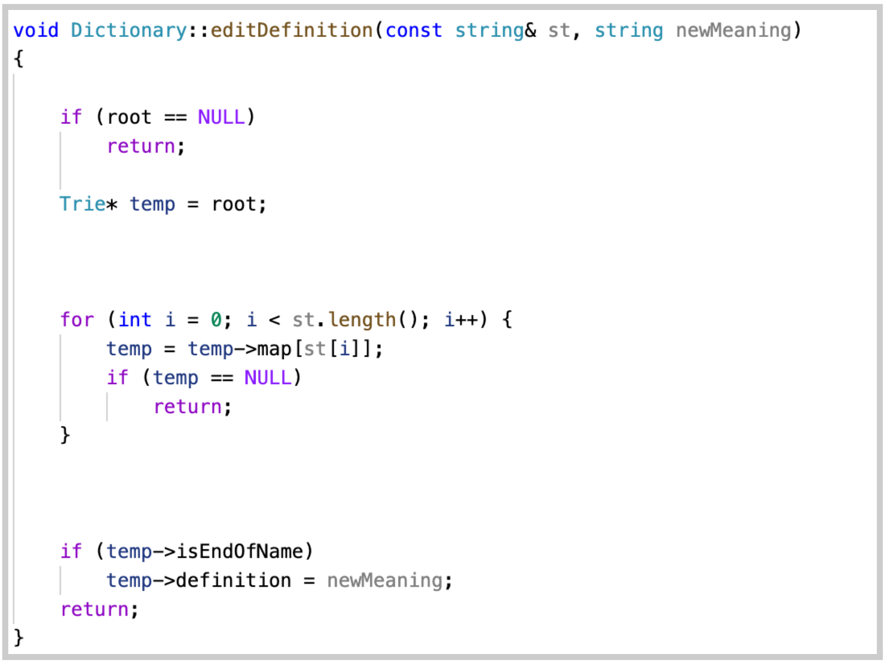


```
1: Search for a word
2: Add word to favorite list
3: View favorite list
4: View history of search word
5: Add new word and definition
6: Edit definition
7: Remove a word
8: Reset dictionary
9: View random word and definition
10: question: Guess definition
11: question: Guess word
Enter your choice: 1
Enter a word to search: ^^
Meaning: Happy
Function execution time: 0.0003138 seconds
Do you want to add this word to your favorite list? (y/n): y
Successfully added to your favorite list
Function execution time: 0.0004568 seconds
```



```
-----WELCOME TO OUR DICTIONARY-----
1: Search for a word
2: Add word to favorite list
3: View favorite list
4: View history of search word
5: Add new word and definition
6: Edit definition
7: Remove a word
8: Reset dictionary
9: View random word and definition
10: question: Guess definition
11: question: Guess word
Enter your choice: 2
Enter a word to add to your favorite list: @@@
Successfully added to your favorite list
Function execution time: 3e-07 seconds
```

2.5 Updating a Word



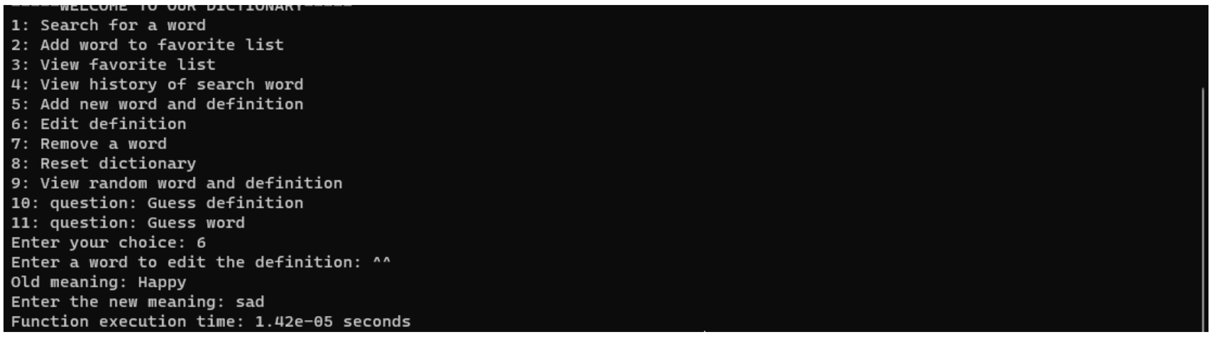
```
void Dictionary::editDefinition(const string& st, string newMeaning)
{

    if (root == NULL)
        return;

    Trie* temp = root;



    for (int i = 0; i < st.length(); i++) {
        temp = temp->map[st[i]];
        if (temp == NULL)
            return;
    }



    if (temp->isEndOfName)
        temp->definition = newMeaning;
    return;
}
```

2.5.1 Algorithm

For updating a word's definition, the algorithm searches for the word in the Trie and updates its associated definition.

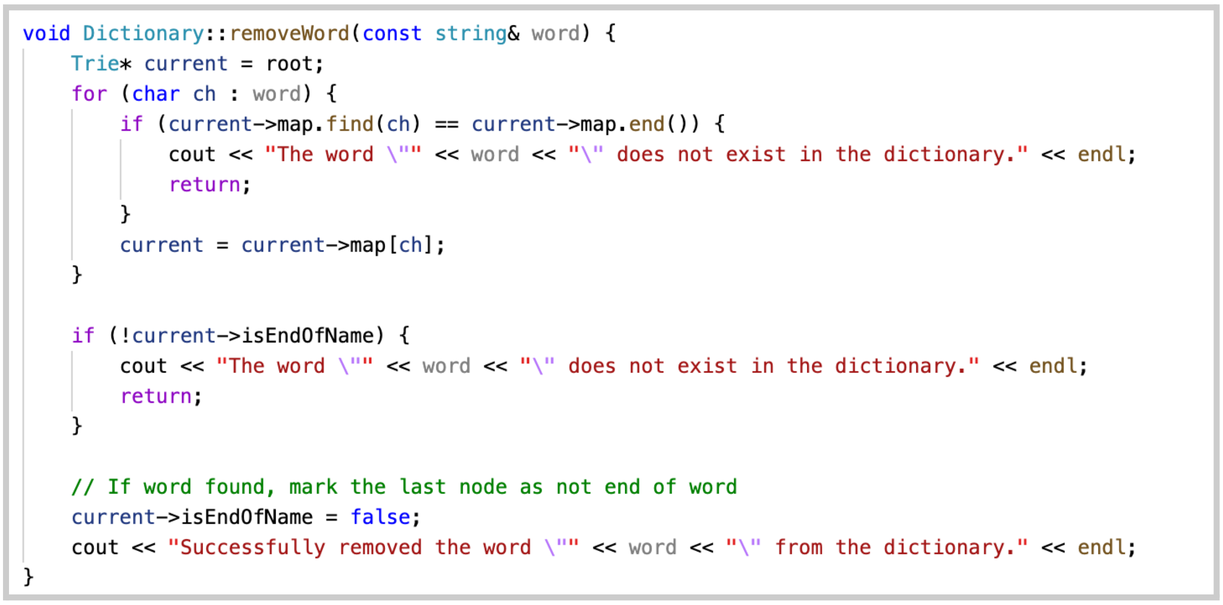
2.5.2 Evaluation

Algorithm Complexity: O(m), where m is the length of the word.



```
=====WELCOME TO OUR DICTIONARY=====
1: Search for a word
2: Add word to favorite list
3: View favorite list
4: View history of search word
5: Add new word and definition
6: Edit definition
7: Remove a word
8: Reset dictionary
9: View random word and definition
10: question: Guess definition
11: question: Guess word
Enter your choice: 6
Enter a word to edit the definition: ^^
Old meaning: Happy
Enter the new meaning: sad
Function execution time: 1.42e-05 seconds
```

2.6 Deleting a Word:



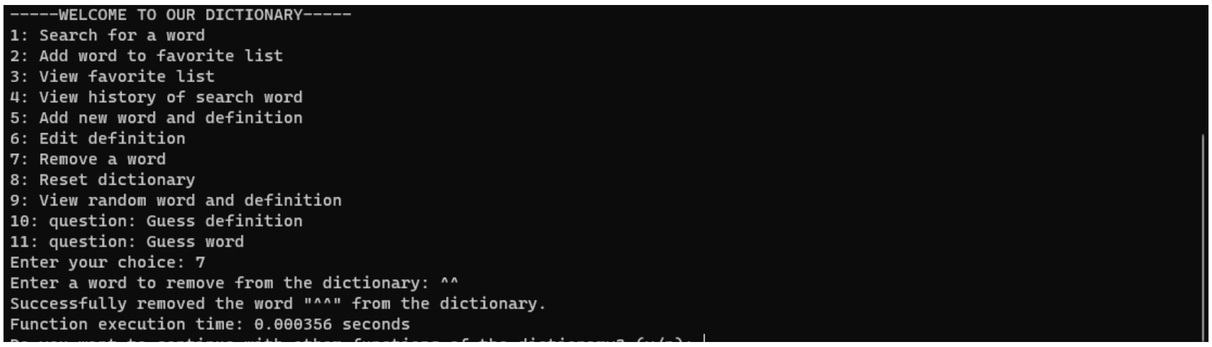
```cpp
void Dictionary::removeWord(const string& word) {
    Trie* current = root;
    for (char ch : word) {
        if (current->map.find(ch) == current->map.end()) {
            cout << "The word \"" << word << "\" does not exist in the dictionary." << endl;
            return;
        }
        current = current->map[ch];
    }

    if (!current->isEndOfName) {
        cout << "The word \"" << word << "\" does not exist in the dictionary." << endl;
        return;
    }

    // If word found, mark the last node as not end of word
    current->isEndOfName = false;
    cout << "Successfully removed the word \"" << word << "\" from the dictionary." << endl;
}
```

2.5.1 Algorithm

When a user requests to delete a word, the algorithm searches for the word in the Trie and marks its node as not the end of the word. If the node is a leaf, it might be pruned.

2.5.2 Evaluation

Algorithm Complexity: O(m), where m is the length of the word.



```
-----WELCOME TO OUR DICTIONARY-----
1: Search for a word
2: Add word to favorite list
3: View favorite list
4: View history of search word
5: Add new word and definition
6: Edit definition
7: Remove a word
8: Reset dictionary
9: View random word and definition
10: question: Guess definition
11: question: Guess word
Enter your choice: 7
Enter a word to remove from the dictionary: ^^
Successfully removed the word "^^" from the dictionary.
Function execution time: 0.000356 seconds
```

3. TIME COMPLEXITY

Action	Time Complexity	Explanation
Adding a New Word (`addNewWord`)	O(L), L is keyword length	Traverse and insert each character of the word into the trie.
Searching for a Keyword (`searchKeyword`)	O(L)	Traverse the trie to find the keyword.
Searching for Words by Definition (`searchWordsByDefinition`)	O(N)	Traverse through all words to find those with a specific definition.
Searching for a Query (`search`)	O(Q + D), Q is query length, D is definition length	Traverse the trie for the query; search for all words with the same definition if needed.

Editing a Definition (`editDefinition`)	O(L)	Traverse the trie to find the word and update its definition.
Adding to Favorite List (`addToFavoriteList`)	O(L)	Call `searchKeyword` to get the meaning of the word.
Viewing Favorite List (`viewFavoriteList`)	O(F), F is favorite list size	Read and display all words in the favorite list.
Removing from Favorite List (`removeFromFavoriteList`)	O(F)	Read and update the favorite list to remove a word.
Viewing History of Search Words (`viewHistoryOfSearchWord`)	O(H), H is history list size	Read and display all search history entries.
Removing a Word (`removeWord`)	O(L)	Traverse the trie to find and remove the word.
Resetting the Dictionary (`resetDictionary`)	O(1)	Delete the root of the trie.
Viewing Random Word	O(N), N is dictionary size	Traverse through all words to

(`viewRandomWord`)		choose a random one.
Getting a Random Word Guess Question (`getRandomWordGuessQuestion`)	O(N)	Traverse through all words to choose a random one; shuffle options.
Getting a Random Definition Guess Question (`getRandomDefGuessQuestion`)	O(N)	Traverse through all words to choose a random one; create options based on distinct definitions.
Autocompletion (`autocomplete`)	O(P+K), P is the length of the prefix and K is the number of words with that prefix.	Traverse to find the prefix; traverse the subtree to get suggestions with that prefix.