BAHRIA UNIVERSITY KARACHI CAMPUS

DEPARTMENT OF COMPUTER SCIENCE

Bahria University
Discovering Knowledge

Data Structures (CSC-221)

BSCS-3B

PROJECT NAME

"DIGITAL DICTIONARY (USING BST)"

Submitted by:

Name	Enrollment
1. Aliza Malik	02-134231-029
2. Ayesha	02-134231-026

Submitted To: Engr.

Rabia Amjad

Submitted On:

Date: 13/06/2024

Acknowledgment

This project on developing a dictionary management system using a binary search tree in C++ would not have been possible without the support and guidance of several individuals and organizations. I would like to extend my heartfelt gratitude to everyone who contributed to the successful completion of this project.

I am immensely grateful to Miss Rabia and Miss Lubna for their teachings and advice, which laid the foundation for my understanding and development of this project. Your dedication and passion for education inspired me greatly. Your invaluable expertise, guidance, and support throughout this project were instrumental. Your constructive feedback and encouragement helped shape this project into its final form.

Thank you all for your contributions and support.

Abstract

The objective of this project is to develop a dictionary management system using a binary search tree (BST) implemented in C++. This system facilitates efficient storage, retrieval, update, and deletion of dictionary entries, which include a word, its meaning, synonyms, antonyms, and various forms. By leveraging the properties of the BST, the system ensures optimized performance for operations such as insertion and search, thereby maintaining an organized and balanced structure. The project incorporates functionalities to read from and write to a text file, ensuring persistent storage. Users interact with the system via a consolebased menu, enabling them to insert new entries, search for existing words, update definitions, delete entries, and display all words in the dictionary.

Designed with essential features such as insertion, search, deletion, update, and display, the system effectively combines data structures and algorithms, showcasing the practical application of BSTs for real-world scenarios. This project demonstrates the seamless integration of file handling with advanced data structures, emphasizing the system's efficiency and robustness in managing dictionary entries. It serves as a practical example of utilizing data structures like BSTs to build efficient and reliable applications, laying a solid foundation for further exploration and enhancement in data management and algorithm development.

Table Of Content

Introduction

In today's digital world, having efficient ways to manage data is really important. This report talks about making a special system for managing dictionaries using Binary Search Tree (BST), which is a way to organize data. The main goal is to create a system that can store, find, change, and remove dictionary entries easily, including things like words, meanings, synonyms, antonyms, and parts of speech.

Dictionaries are super useful in lots of areas, like understanding languages, learning, and searching for things online. But regular dictionaries, whether they're books or basic computer

files, can struggle when there's a lot of information. By using BSTs, this project fixes those issues and makes a system that's really good at handling dictionaries. It also keeps the dictionary safe even when you turn off your computer and turn it back on. Plus, it's easy for people to use because it has a simple menu on the computer screen. This project shows how we can use smart ways of organizing data to make things work better, which can lead to even more improvements in how we manage information in the future.

Background/Literaure Review

Dictionaries play a vital role in various domains, serving as essential tools for linguistic analysis, educational purposes, and information retrieval. In traditional dictionary systems, managing extensive datasets efficiently poses significant challenges. To address these challenges, literature suggests leveraging Binary Search Trees (BSTs) for organizing and retrieving dictionary entries effectively. This review aims to explore the rationale behind employing BSTs in dictionary management systems and the existing literature supporting this approach.

Significance of Dictionaries

Dictionaries serve as invaluable resources in linguistic research, language learning, and information retrieval. They provide comprehensive information about words, including definitions, meanings, synonyms, antonyms, and various forms. In educational settings, dictionaries aid in vocabulary development and language comprehension. Moreover, in computational linguistics and natural language processing, dictionaries are essential for text analysis, language translation, and semantic interpretation.

Challenges in Traditional Dictionary Systems

Conventional dictionary systems encounter limitations in handling large datasets efficiently. These limitations include slow search and retrieval times, lack of scalability, and difficulties in managing updates and modifications. In printed dictionaries, accessing specific entries can be time-consuming, while digital dictionaries may face issues with indexing and searching large volumes of data.

Role of Binary Search Trees (BSTs) in Dictionary Management

BSTs offer an elegant solution to the challenges faced by traditional dictionary systems. As self-balancing binary trees, BSTs maintain an ordered hierarchy of dictionary entries, facilitating fast and efficient search operations. The inherent properties of BSTs, such as logarithmic search time and balanced tree structure, make them well-suited for organizing and managing dictionary data. By leveraging BSTs, dictionary management systems can achieve optimized performance, improved scalability, and enhanced usability.

Existing Literature Supporting BSTs for Dictionary Management

Literature exploring the use of BSTs in dictionary management systems provides valuable insights into the efficacy of this approach. Research studies and academic publications highlight the advantages of BSTs in terms of search efficiency, memory utilization, and overall system performance. Additionally, code examples, tutorials, and practical implementations demonstrate the implementation of BST-based dictionary systems in programming languages like C++

Problem Statement

Traditional dictionary systems encounter inefficiencies in storage, retrieval, and updating of entries due to their inherent limitations. Challenges include slow search and retrieval times, lack of scalability, and difficulties in managing updates while maintaining data integrity and usability. To address these challenges, this project aims to develop a dictionary management system using Binary Search Trees (BST) in C++. The system will leverage BSTs to optimize storage, retrieval, and update operations, ensuring efficient management of dictionary entries while upholding data integrity and usability.

Objectives and Goals:

1. Create a dictionary management system using BST in C++: The main aim is to build a system that effectively manages a dictionary utilizing Binary Search Trees (BST) in C++. This system will handle storing, retrieving, updating, and deleting dictionary entries efficiently.

2. Implement efficient operations for storage, retrieval, update, and deletion: The system will utilize BST operations like insertion, search, deletion, and traversal to manage dictionary

entries efficiently. These operations will be optimized to ensure the system performs well and operates smoothly.

3. Design an intuitive user interface for seamless interaction: The system will have a userfriendly interface that allows users to interact with the dictionary effortlessly. It will provide easy-to-use functionalities for adding, searching, updating, deleting, and displaying dictionary entries.

4. Ensure reliability and robustness to handle various scenarios: The system will be designed to handle diverse scenarios effectively, ensuring reliability and robustness. Measures will be implemented to maintain data integrity, handle errors, and ensure stability under different conditions.

Project Scope

The project involves designing and implementing a dictionary management system with robust functionality to handle entries efficiently. The system will leverage BST data structure to optimize storage, retrieval, update, and deletion operations. Key aspects of the project scope include:

Dictionary Management System: Design and implement a dictionary management system using BST in C++.

Efficient Storage and Retrieval: Implement efficient storage and retrieval of dictionary entries using BST operations such as insertion, search, and traversal.

Update and Deletion: Enable seamless updating and deletion of dictionary entries while ensuring data integrity and consistency.

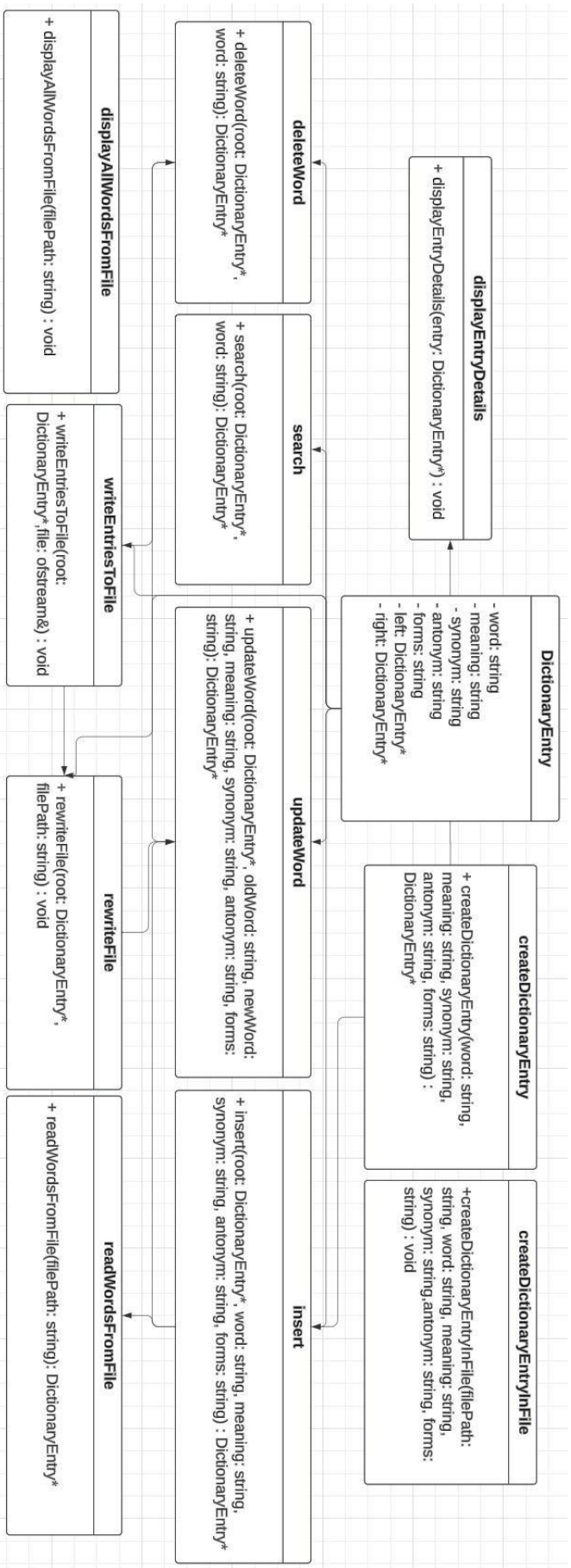
User Interface: Develop an intuitive user interface for easy interaction with the dictionary system, allowing users to perform operations such as insertion, search, update, deletion, and display of dictionary entries.

Scalability and Performance: Ensure scalability of the system to accommodate substantial data volumes while maintaining optimal performance.

Usability and Reliability: Enhance usability and reliability of the system to provide a seamless and robust user experience.

Documentation and Testing: Provide comprehensive documentation and conduct thorough testing to ensure the correctness, reliability, and robustness of the dictionary management system.

UML


displayEntryDetails
+ displayEntryDetails(entry: DictionaryEntry*) : void
DictionaryEntry
- word: string
- meaning: string
- synonym: string
- antonym: string
- forms: string
- left: DictionaryEntry*
- right: DictionaryEntry*
createDictionaryEntry
+ createDictionaryEntry(word: string, meaning: string, synonym: string, antonym: string, forms: string) : DictionaryEntry*
createDictionaryEntryInFile
+createDictionaryEntryInFile(filePath: string, word: string, meaning: string, synonym: string,antonym: string, forms: string) : void
deleteWord
+ deleteWord(root: DictionaryEntry*, word: string): DictionaryEntry*
search
+ search(root: DictionaryEntry*, word: string): DictionaryEntry*
updateWord
+ updateWord(root: DictionaryEntry*, oldWord: string, newWord: string, meaning: string, synonym: string, antonym: string, forms: string): DictionaryEntry*
insert
+ insert(root: DictionaryEntry*, word: string, meaning: string, synonym: string, antonym: string, forms: string) : DictionaryEntry*
displayAllWordsFromFile
+ displayAllWordsFromFile(filePath: string) : void
writeEntriesToFile
+ writeEntriesToFile(root: DictionaryEntry*,file: ofstream&) : void
rewriteFile
+ rewriteFile(root: DictionaryEntry*, filePath: string) : void
readWordsFromFile
+ readWordsFromFile(filePath: string): DictionaryEntry*

Overview Of Project

The project is centered on creating a dictionary management system that simplifies the tasks of organizing, accessing, and editing dictionary entries. Key functionalities such as addition, search, update, and deletion of entries are incorporated into the system. These features are seamlessly integrated into a unified system that leverages the efficiency and structure of a binary search tree (BST) data structure. By utilizing BST, the system ensures efficient organization and management of dictionary entries, enhancing overall usability and effectiveness.

Tools And Technologies:

1. C++ Programming Language : The primary language used for implementing the dictionary management system. C++ provides powerful features for low-level programming, making it suitable for efficient data structure implementations.

2. File Handling in C++ : C++ file handling libraries (<fstream>) are employed to read from and write to files. This allows for persistent storage and retrieval of dictionary entries.

3. Binary Search Tree (BST) : The project leverages the BST data structure to efficiently organize and manage dictionary entries. BST provides logarithmic time complexity for key operations such as search, insertion, and deletion, ensuring efficient manipulation of dictionary data.

4. Visual Studio C++ IDE :Visual Studio Integrated Development Environment (IDE) for coding, debugging, and building the C++ application. Visual Studio offers a range of features such as IntelliSense, debugging tools, and project management capabilities, enhancing the development process.

These tools and technologies collectively enable the creation of a robust and efficient dictionary management system in C++.

Project features

Functional Requirements:

1. Insert Word:

- Description: Users should be able to add a new word to the dictionary by providing its word, meaning, synonym, antonym, and forms.
- Input: User inputs word, meaning, synonym, antonym, and forms.
- Output: Confirmation message indicating successful addition.
- Action: The system creates a new entry with the provided details, inserts it into the binary search tree (BST), and adds it to the dictionary file.

2. Search Word:

- Description: Users should be able to search for a word in the dictionary.
- Input: User provides the word to search for.
- Output: Display of the word's details (meaning, synonym, antonym, and forms) if found, or a message indicating that the word is not found.
- Action: The system traverses the BST to find the word and retrieves its details if it exists.

3. Delete Word:

- Description: Users should be able to delete a word from the dictionary.
- Input: User inputs the word to be deleted.
- Output: Confirmation message indicating successful deletion.
- Action: The system removes the specified word from the BST and updates the dictionary file accordingly.

4. Update Word:

- Description: Users should be able to update an existing word in the dictionary by modifying its meaning, synonym, antonym, or forms.

- Input: User inputs the old word to be updated and provides the new word, along with its updated meaning, synonym, antonym, and forms.

- Output: Confirmation message indicating successful update.

- Action: The system deletes the old entry from the BST, inserts the updated entry, and updates the dictionary file with the changes.

5. Display All Words:

- Description: Users should be able to view all words present in the dictionary along with their details.

- Input: No user input required.

- Output: Display of all words along with their meanings, synonyms, antonyms, and forms.

- Action: The system reads entries from the dictionary file and displays them to the user.

6. Exit:

- Description: Users should be able to exit the program.

- Input: No user input required.

- Output: Program exits.

- Action: The system terminates the program execution.

Non-Functional Requirements:

1. Performance:

- The system should efficiently handle dictionary files of varying sizes.

- Insertion, deletion, and search operations should have reasonable execution times even for large dictionaries.

2. User-Friendliness:

- The user interface should have clear and intuitive prompts and instructions for each operation.

- Error messages should be informative and guide the user in case of incorrect inputs or failures.

3. Reliability:

- The system should handle file I/O errors gracefully and provide appropriate error messages.

- It should be robust against unexpected inputs or system failures, ensuring that the data integrity is maintained.

4. Scalability:

- The system should be designed in a modular and extensible manner to accommodate future enhancements or changes.

- It should support additional features or optimizations without requiring significant modifications to the existing codebase.

5. Security:

- The system should implement access controls to prevent unauthorized access to sensitive data stored in the dictionary file.

- Measures should be taken to ensure that the dictionary file is not tampered with or modified by unauthorized users.

6. Portability:

- The system should be platform-independent and runnable on different operating systems without needing modifications.

- It should use standard libraries and practices to ensure compatibility across different environments.

DSA Concepts

Binary Search Tree (BST):

Usage: The core data structure used in this project is a binary search tree, which organizes dictionary entries based on the alphabetical order of words.

Code Reference: The DictionaryEntry struct serves as the nodes of the binary search tree. Each node contains pointers to its left and right children, forming the hierarchical structure of the BST.

Functions: Functions like insert, search, deleteWord, updateWord, and readWordsFromFile operate on the BST to perform operations like insertion, search, deletion, and updating of dictionary entries.

Tree Traversal:

Usage: Tree traversal is essential for searching, displaying, and writing entries to files in a sorted order.

Code Reference: The writeEntriesToFile function recursively traverses the BST in an inorder manner to write dictionary entries to a file. Similarly, the displayAllWordsFromFile function traverses the BST to display all words along with their details.

File Input/Output (I/O):

Usage: Reading words from a file, writing entries to a file, and updating the dictionary file after modifications.

Code Reference: Functions like readWordsFromFile, writeEntriesToFile, createDictionaryEntryInFile, and rewriteFile handle file I/O operations. They read dictionary entries from a file, write entries to a file, and update the file after insertions, deletions, or updates.

Recursion:

Usage: Recursion is employed in various functions for traversing the BST and performing operations on its nodes.

Code Reference: Recursive functions like insert, search, deleteWord, and writeEntriesToFile are called recursively to traverse the BST and perform respective operations on nodes.

Additional Data Structures and Concepts:

Structures:

- The DictionaryEntry struct holds the data for each entry in the dictionary, including the word, meaning, synonym, antonym, and forms.

- This struct is utilized to create nodes for the BST and store dictionary entries.

Strings:

- Standard string data type from the C++ standard library (std::string) is used to store word, meaning, synonym, antonym, and forms for each dictionary entry.

- These string variables hold the textual data associated with each dictionary entry.

File Streams:

- The <fstream> header is used to work with file streams for reading from and writing to files.

- Input file streams (ifstream) are used to read dictionary entries from a file, and output file streams (ofstream) are used to write entries to a file.

Pointers:

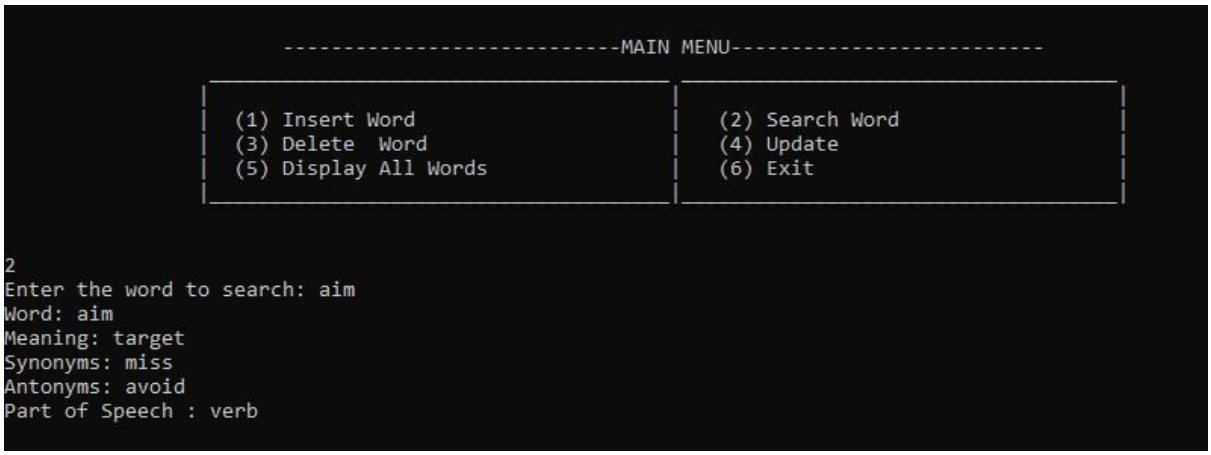
- Pointers to DictionaryEntry objects are used to link nodes in the binary search tree.

- Pointers are used to traverse the tree, insert new nodes, search for nodes, delete nodes, and update nodes.

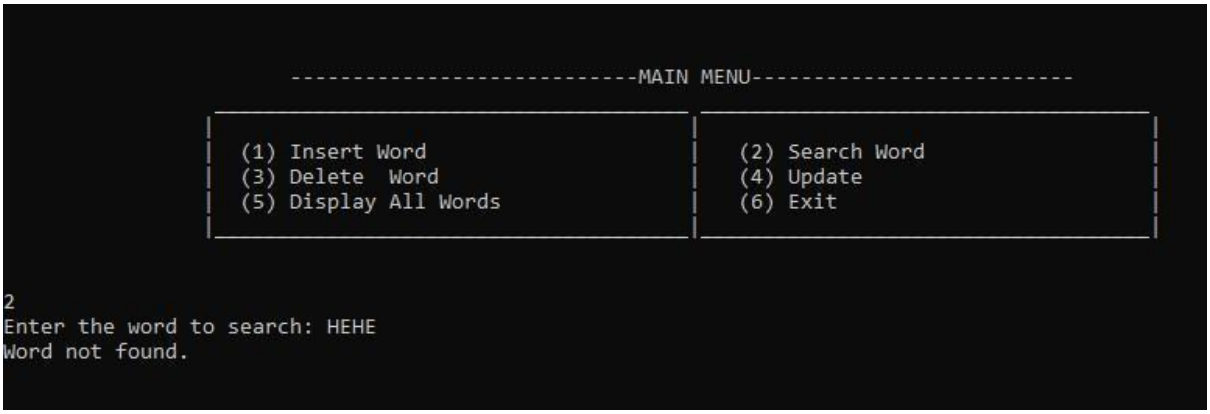
Output

INSERT

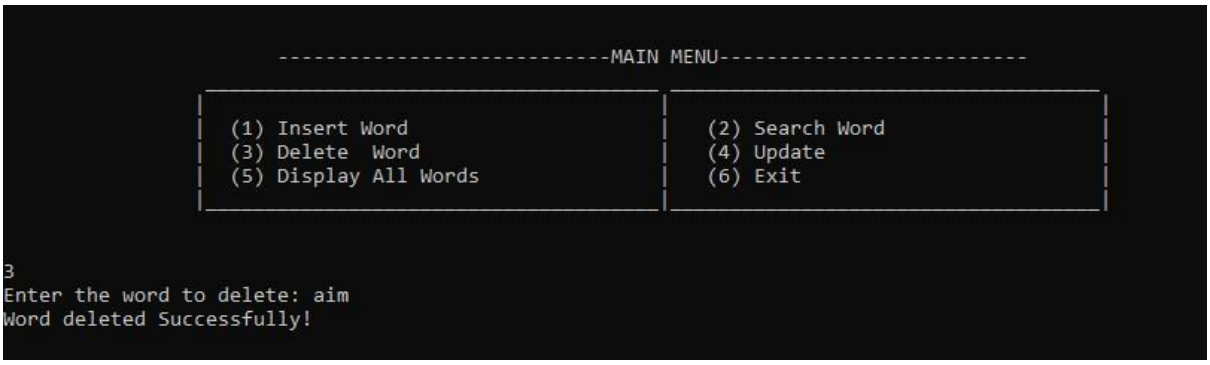
SEARCH



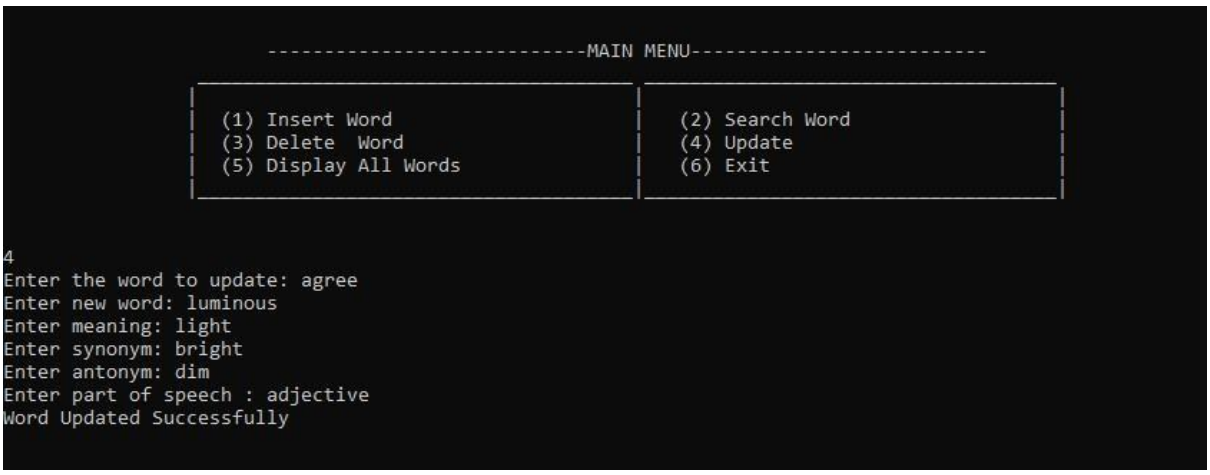
IF SEARCHED WORD NOT FOUND



DELETE



UPDATE



DISPLAY All WORDS

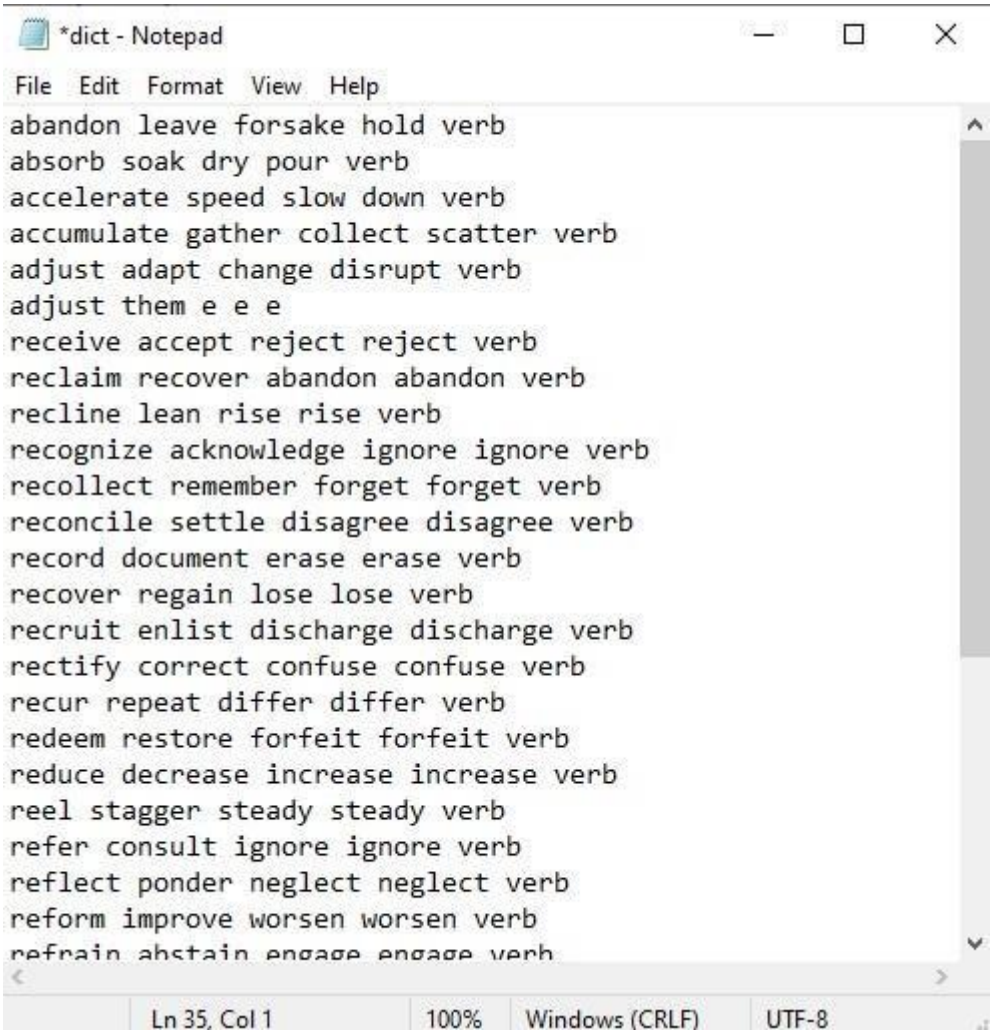
```
5
Word: abandon
Meaning: leave
Synonyms: forsake
Antonyms: hold
Part of speech : verb
----------------------
Word: abbreviate
Meaning: lengthen
Synonyms: lengthen
Antonyms: verb
Part of speech : shoulder
----------------------
Word: absorb
Meaning: soak
Synonyms: dry
Antonyms: pour
Part of speech : verb
----------------------
Word: accelerate
Meaning: speed
Synonyms: slow
Antonyms: down
Part of speech : verb
----------------------
Word: accumulate
Meaning: gather
Synonyms: collect
Antonyms: scatter
Part of speech : verb
----------------------
Word: addition
Meaning: annex
Synonyms: noun
Antonyms: ecosystem
Part of speech : environment
----------------------
Word: adhere
Meaning: detach
Synonyms: detach
----------------------
Word: attempt
Meaning: try
Synonyms: give
Antonyms: up
Part of speech : verb
----------------------
Word: attend
Meaning: go
Synonyms: skip
Antonyms: avoid
Part of speech : verb
----------------------
Word: attract
Meaning: draw
Synonyms: repel
Antonyms: push
Part of speech : verb
----------------------
Word: attribute
Meaning: noun
Synonyms: quotation
Antonyms: citation
Part of speech : excerpt
----------------------
Word: authenticate
Meaning: verify
Synonyms: falsify
Antonyms: forge
Part of speech : verb
----------------------
Word: authorize
Meaning: empower
Synonyms: forbid
Antonyms: prohibit
Part of speech : verb
----------------------
```

EXIT

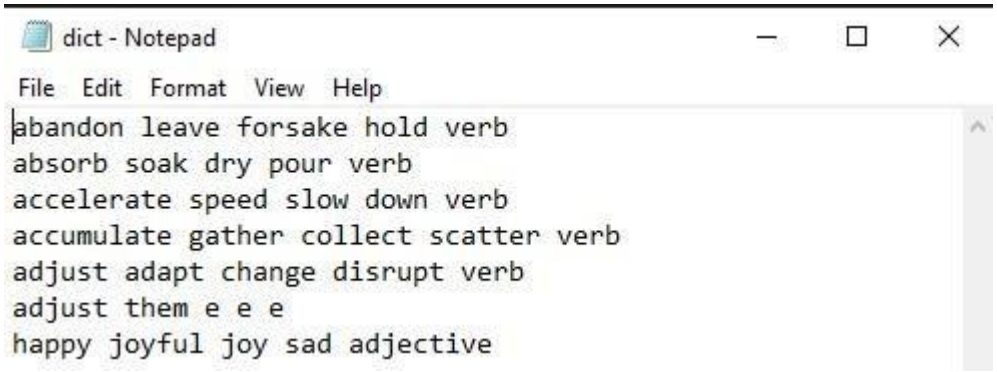
```
                ----------------------------MAIN MENU-------------------------
        ____________________________________ ____________________________________
       |                                    |                                    |
       |  (1) Insert Word                   |   (2) Search Word                  |
       |  (3) Delete  Word                  |   (4) Update                       |
       |  (5) Display All Words             |   (6) Exit                         |
       |____________________________________|____________________________________|


6
Exiting...
C:\Users\AALIYAN'Z COMPUTER\source\repos\pbl\pbl\x64\Debug\dictionary\x64\Debug\dictionary.exe (process
To automatically close the console when debugging stops, enable Tools->Options->Debugging->Automatically
Press any key to close this window . . .
```

FILE



HAPPY WORD ADDED



Conclusion

In conclusion, the development of the "Digital Dictionary using Binary Search Tree (BST)" project has successfully demonstrated the practical application of advanced data structures in managing and organizing dictionary entries efficiently. By leveraging the BST data structure, the system ensures optimized performance for essential operations such as insertion, search, update, and deletion of

dictionary entries. This project addresses the inherent limitations of traditional dictionary systems by providing a scalable, efficient, and user-friendly solution.

The system's design integrates critical functionalities like file handling, dynamic memory allocation, and recursive algorithms, all of which are essential in maintaining data integrity and facilitating seamless interaction with the dictionary. The user interface offers intuitive menu-driven options that allow users to effortlessly add, search, update, delete, and display dictionary entries, thereby enhancing usability and accessibility.

Furthermore, the project highlights the significance of BSTs in real-world applications, showcasing their role in optimizing data storage and retrieval. Through comprehensive testing and documentation, the system ensures reliability and robustness, capable of handling various scenarios and maintaining consistent performance across different data volumes.

Overall, this project serves as a foundational example of utilizing advanced data structures and algorithms to build efficient and reliable software applications. It underscores the importance of structured data management in enhancing information retrieval systems, paving the way for future advancements in data structure research and application development.

References

1. P. H. Hsiao, "Binary Search Tree Dictionary," OpenDSA, Virginia Tech, 2022. [Online]. Available:

https://opendsa.cs.vt.edu/ODSA/Books/CS3/html/BSTDict.html. [Accessed: May. 12, 2024].

- This source provided foundational insights into implementing dictionary functionality using a binary search tree (BST), focusing on efficient data retrieval and management.

2. N. Chandrasekaran, "Digital Dictionary Using Binary Search Algorithm," ResearchGate, Aug. 2019. [Online]. Available:

[https://www.researchgate.net/publication/335657221_Digital_Dictionary_Using_Binary_S

earch_Algorithm](https://www.researchgate.net/publication/335657221_Digital_Dictionary_Using_Binary_Search_Algorithm). [Accessed: May. 18, 2024].

- This research paper offered practical applications of BSTs in creating digital dictionary systems, emphasizing algorithmic approaches to optimizing search and update operations.

3. Wikipedia contributors, "Binary search tree," Wikipedia, The Free Encyclopedia. [Online]. Available: https://en.wikipedia.org/wiki/Binary_search_tree. [Accessed: May. 22, 2024].

- Wikipedia provided foundational knowledge and definitions related to binary search trees, offering a comprehensive overview of their structure, properties, and applications.

4. GeeksforGeeks, "Binary Search Tree Data Structure," GeeksforGeeks. [Online]. Available: [https://www.geeksforgeeks.org/binary-search-treedatastructure/](https://www.geeksforgeeks.org/binary-search-tree-data-structure/). [Accessed: Jun. 1, 2024].

- This source provided practical examples, code snippets, and explanations related to implementing binary search trees, aiding in understanding key operations such as insertion, deletion, and traversal.

5. H. Khosravi, "Binary Search Trees," Department of Computer Science, University of British Columbia, 2021. [Online]. Available: [https://hassan-khosravi.net/courses/CPSC221/handouts/hu/slides/lecture08-bst-1up.pdf](https://hassankhosravi.net/courses/CPSC221/handouts/hu/slides/lecture08-bst1up.pdf). [Accessed: Jun. 12, 2024].

- These lecture slides provided academic insights into the theoretical foundations of binary search trees, focusing on their properties, algorithms, and applications in computer science.

THANKYOU.