**KWAME NKRUMAH UNIVERSITY OF SCIENCE AND TECHNOLOGY, KUMASI**

COLLEGE OF ENGINEERING

DEPARTMENT OF COMPUTER ENGINEERING

DATA STRUCTURES AND ALGORITHMS (COE 363)

PROJECT REPORT

BY

GROUP 21

MARCH, 2024

|  |  |  |
| --- | --- | --- |
| INDEX NO | NAME | CONTRIBUTED |
| 7100921 | Keborni Evans Attakuma | YES |
| 7100621 | Incoom Stella Ama | YES |
| 7103121 | Mensah Kwegyir David | **NO** |
| 7100321 | Idan Edwin Richard | YES |
| 7110921 | Yawlui Sharon Enam | YES |
| 7096921 | Darko Kwadwo Opoku | YES |
| 7111621 | Dzotsi Daniel | YES |
| 7107921 | Owusu-Ampiaw Nana Kwame | YES |
| 7101921 | Lamptey Alexander Nii Lantey | **NO** |
| 7101121 | Klufio David-Andrew | YES |

GROUP MEMBERS

**CONTRIBUTIONS**

1. **Algorithm Implementation and Telegram Bot Interface** by Yawlui Sharon Enam   
2. **Project Management** by Keborni Evans Attakuma

3. **Testing** by Idan Edwin Richard and Incoom Stella Ama

4. **Algorithm Analysis** by Daniel Dzotsi and Darko Kwadwo Opoku

5. **Research** by Owusu-Ampiaw Nana Kwame and Klufio David-Andrew

**INTRODUCTION**

We were tasked with developing a simple dictionary program that allows users to find word definitions and perform operations like **searching**, **addition**, **amendment**, and **deletion** of words. We chose Python as the programming language and implemented a Telegram bot for user interaction. The data structure used for storing words and their definitions was a dictionary, and we utilized a JSON file for persistent storage.

**METHODOLOGY**

Our project involved operations like searching, adding and deletion words so the best data structure we thought of was the python dictionary. In Python, a dictionary is a built-in data structure used to store key-value pairs. It works by using a technique called hashing, which provides constant-time complexity for basic operations such as insertion, deletion, and lookup, on average.

Here's how a Python dictionary works with regards to algorithm analysis:

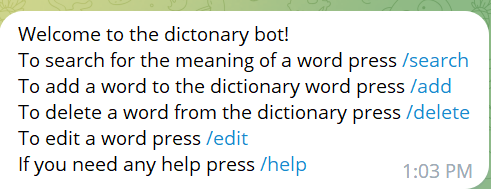
1. **Hashing**: When you insert a key-value pair into a dictionary or look up a value associated with a key, Python internally calculates a hash value for the key. This hash value is then used to determine the index where the key-value pair should be stored or retrieved in the underlying data structure.
2. **Hash Table**: Internally, Python dictionaries are implemented using a hash table, which is an array-like data structure where each element is a bucket that can store multiple key-value pairs. The hash value of the key determines which bucket the key-value pair should be stored in.
3. **Collision Handling**: Since multiple keys may have the same hash value (known as a collision), Python employs techniques such as chaining or open addressing to handle collisions. Chaining involves storing multiple key-value pairs with the same hash value in a linked list within the bucket, while open addressing involves finding an alternative location within the hash table for the collided key-value pair.
4. **Time Complexity**:
   * Insertion (Adding a key-value pair): O(1) average case, O(n) worst case due to potential collisions.
   * Deletion (Removing a key-value pair): O(1) average case, O(n) worst case due to potential collisions.
   * Lookup (Retrieving a value associated with a key): O(1) average case, O(n) worst case due to potential collisions.
5. **Space Complexity**: The space complexity of a dictionary is O(n), where n is the number of key-value pairs stored in the dictionary. However, it can vary based on factors such as the load factor and collision resolution strategy.
6. **Dynamic Resizing**: Python dictionaries dynamically resize themselves to accommodate an arbitrary number of key-value pairs. When the dictionary exceeds a certain load factor (typically around 2/3 full), it automatically resizes and rehashes its contents to maintain efficient performance.

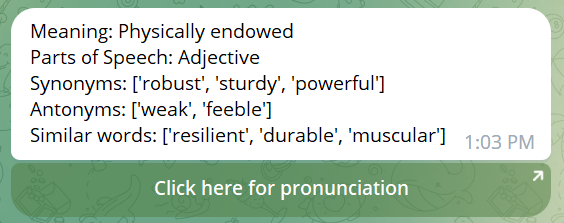
We agreed on a **telegram bot interface** due to its conversational experience. Bots on telegram provide a conversational interface, allowing users to interact using natural language commands. This makes the user experience more intuitive and engaging.

WHY THE TELEGRAM BOT

* Cross-platform compactibility
* User- Engagement
* Instant updates of new features
* Cloud -Based Architecture
* Security and API
* Seamless Integration with messaging platform

**IMPLEMENTATION**

 The Telegram bot interface was utilized for implementation, while the Python dictionary served as the underlying data structure. Word details, including meanings, parts of speech, synonyms, and antonyms, were stored in a JSON file. Commands such as '/search', '/add', '/delete', 'edit', and 'help' were employed for interaction with the bot.

Additionally, the bot includes a functionality that provides users with access to the pronunciation of a word.  
  
We included a

**RESULTS**

The first test run was successful. The challenge we encountered was that users had to enter a separate command for each word they wanted to add or search for the meaning of. This proved to be quite cumbersome, requiring multiple interactions for each task. To address this issue, we worked on implementing a feature that would keep the bot in a specific command mode. After some experimentation and refinement, we successfully integrated this functionality into the bot.

**CONCLUSION**

The project was deemed successful, leveraging a Telegram bot interface in conjunction with a Python dictionary known for its efficient insertion, deletion, and search capabilities. Word data, including meanings, parts of speech, synonyms, and antonyms, was stored in a JSON file. While all primary objectives were met, the feature for suggesting similar words during searches was not implemented.