MSIS 2692: Data Structures and Algorithms

WORDLE

Lite

By

MEHAK SHRUTHI TANYA

What is

- This project introduces a word-guessing game implemented in Python, employing different data structures to enhance efficiency and memory management.
- The game challenges players to guess a hidden word within a limited number of attempts, providing feedback for each guess.
- Three variations of the game are presented, each utilizing a distinct data structure—lists, sets, and tries—to manage word validation and comparison.
- The first variation adopts a list-based approach for simplicity, utilizing linear search for validation. The second variation employs sets to expedite validation through constant-time set membership checks, trading off memory consumption.
- The third variation leverages tries data structures, offering efficient word searching with optimal memory usage.
- Each variation provides a user-friendly interface and visual feedback, guiding players through the guessing process.

Rules

Guess::

selfs

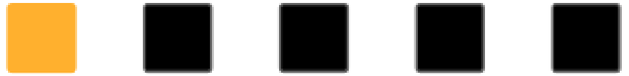
You won! It took you 6 guesses.

Correct

Denotes a correct letter in the exact position within the word.

Guess:

oasis



Wrong spot

Denotes a correct letter but positioned elsewhere in the word.

Guess:

apple

Not in Word

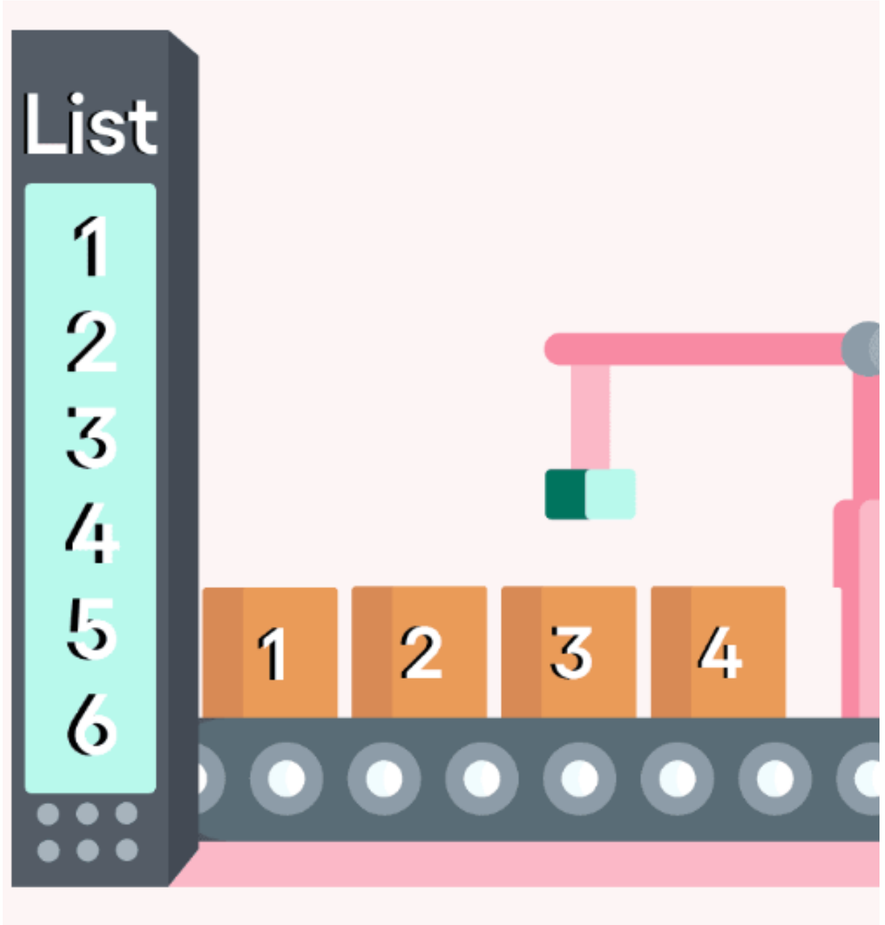
Denotes an incorrect letter that is not part of the word.

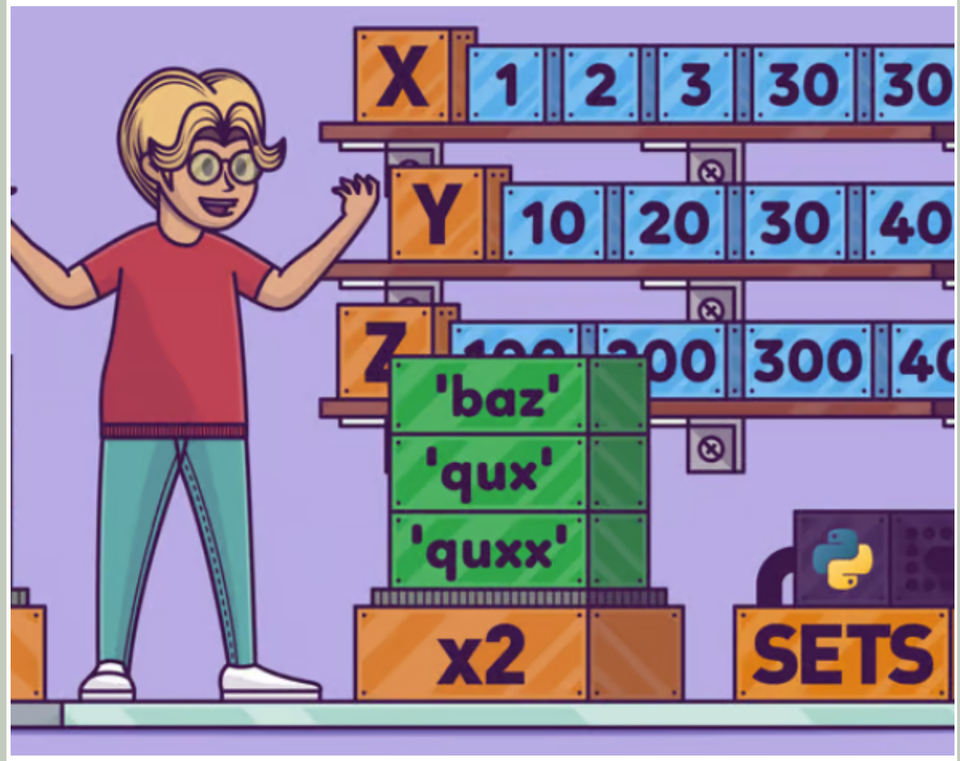
TIME FOR A

DEMO

Using Lists

- **Simplicity:** Lists are chosen for their simplicity, providing a straightforward way to manage words and game state.
- **Ease of Implementation:** Lists simplify tasks like word filtering, user input validation, and word comparison, making the code easy to understand and implement.
- **Readability:** Using lists enhances code readability, allowing for clear expression of logic related to word games and user interactions.
- **Effective for Smaller Datasets:** Lists are suitable for smaller word lists, where their linear search capabilities are sufficient for the game's requirements.Lists can be suitable when memory usage is not a significant concern, and the primary operations involve sequential access or iteration through elements.
- In the given code, if you have a small to moderate number of words in the wordlists (GAMEWORD_LIST_FNAME and GUESSWORD_LIST_FNAME) and the average length of words is not too large, lists could be used. However, lists may not be the best choice if fast searching or membership testing is required.
- **Cons:** Linear search for validation may be slow for larger word lists.



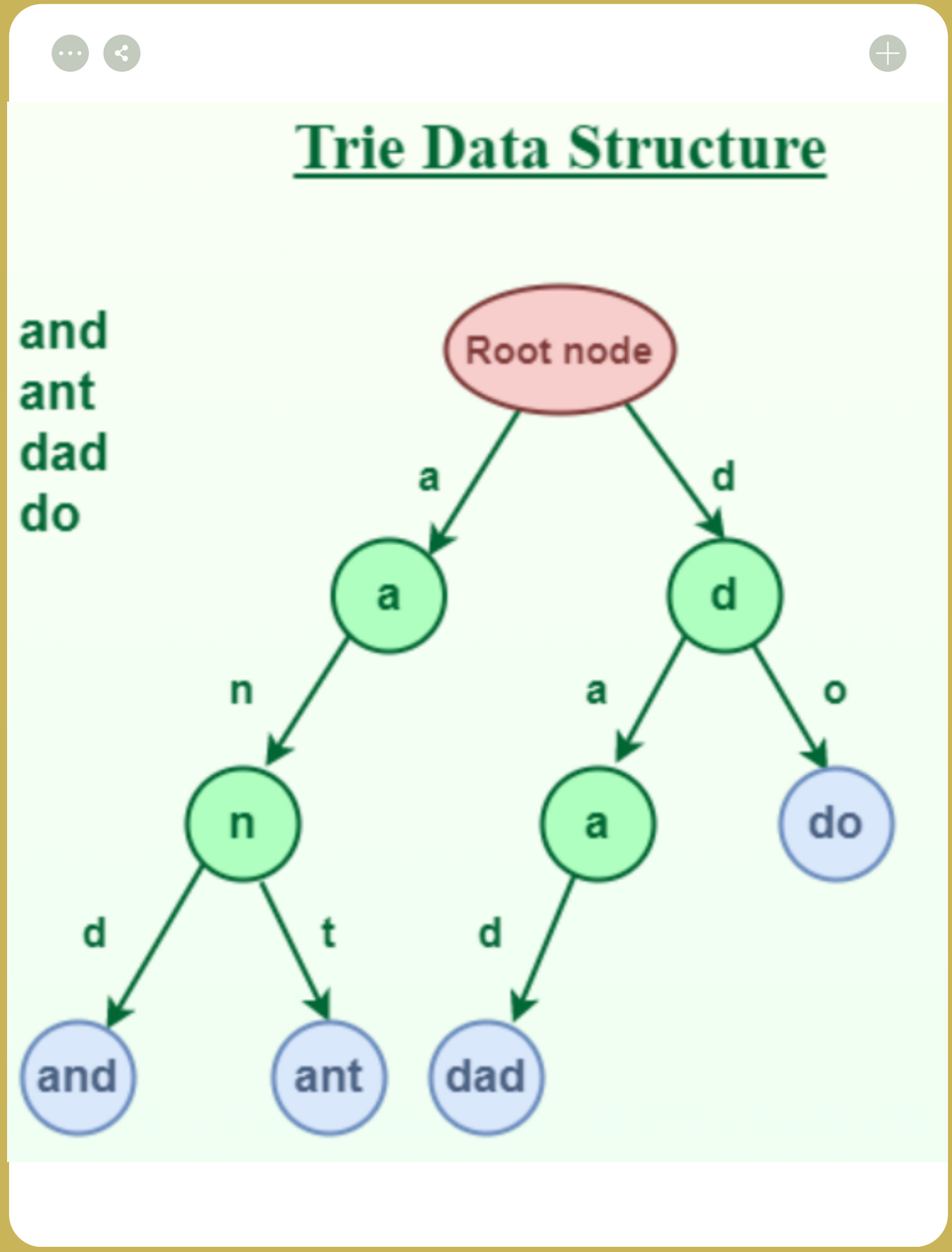


Using Sets

- **Uniqueness:** Sets only allow unique elements. When working with collections where uniqueness is important, using a set can simplify code and ensure that each element appears only once.
- **Set Operations:** Sets support various set operations such as union, intersection, and difference. These operations are useful for comparing and combining different sets of elements.
- **Mathematical Representation:** Sets are a natural representation for mathematical concepts such as sets in set theory. This can lead to more readable and expressive code when dealing with algorithms or problems that involve set-related operations.
- **Iterating Over Unique Elements:** When iterating over elements, a set guarantees that each element is encountered only once. This can be useful in scenarios where you want to process unique items.
- **Performance:** Sets are implemented for efficient access and lookup operations. For large datasets, using sets can lead to better performance compared to other data structures for certain types of operations.

Using Tries

- **Efficient Prefix Matching:**Tries excel at prefix matching, making them highly efficient for tasks such as autocomplete, spell-checking, and searching for words with common prefixes.
- **Dynamic Insertion and Deletion:**Tries support dynamic insertion and deletion of keys, allowing for efficient updates to the data structure without affecting the rest of the trie.
- **Ordered Iteration:** Tries can provide ordered iteration over keys, which is useful in applications where the order of keys matters. This makes tries suitable for tasks requiring sorted or ordered data.
- **Memory Overhead:**Tries can have a higher memory overhead due to the need for additional memory for pointers and metadata in each trie node. This can be a significant drawback, particularly for large datasets or when memory usage is a critical consideration.



Which is the best approach

TRIES

When prioritizing fast searching or membership testing with a non-critical constraint on memory usage, employing tries is advisable.

SETS

If the primary goal is to minimize memory usage while necessitating swift membership testing or filtering operations, sets would be a more suitable option.

LISTS

In scenarios where operations involve sequential access or iteration through elements, and memory usage is not a major concern, lists can be effectively utilized.

THANK YOU