Pagelist Data Structure Justification

Introduction

In implementing the indexing program for Project 5: We the People, the Pagelist class stores up to four unique page numbers on which a specific word appears. Choosing the appropriate data structure for the Pagelist is crucial for ensuring efficient performance and meeting the project requirements. This write-up justifies using a fixed-size array for the Pagelist and discusses its advantages and disadvantages.

Data Structure Choice

Chosen Data Structure: Fixed-size array of integers.

Implementation Details:
Array Size: The array is initialized with a maximum capacity of 4, per the project requirement that each pagelist can hold up to four-page numbers.
Unique Entries: The pagelist ensures that duplicate page numbers are not added.
Order Preservation: Page numbers are stored in the order they are encountered, which aligns with the requirement to keep pages in the order they appear in the text.

Invariants:
The pages array contains up to four unique page numbers.
The count variable accurately reflects the number of pages stored (ranging from 0 to 4).
The pages are stored in the order they were added.

Advantages

Simplicity:
Ease of Implementation: Arrays are straightforward to implement and understand, making them suitable for simple storage needs.
Direct Access: Accessing elements by index is efficient (O(1) time complexity), although this benefit is minimal for small arrays.

Performance Efficiency:

Low Overhead: Arrays have minimal memory overhead compared to more complex data structures like linked lists or trees.
Predictable Memory Usage: The fixed size of the array (capacity of 4) ensures that memory allocation is known at compile time.

Meets Project Requirements:
Fixed Capacity: The project specifies that each pagelist can hold a maximum of four-page numbers. A fixed-size array perfectly fits this requirement without the need for dynamic resizing.
Order Preservation: Since the pages need to be stored in the order they are encountered, an array naturally maintains insertion order.

No External Dependencies:
Custom Implementation: By using an array, we avoid relying on Java's built-in collection classes, which is a project requirement (e.g., we must not use List, Vector, or other collection classes).

Disadvantages

Lack of Flexibility:
Fixed Size: If future requirements change to allow more than four pages per word, the array will not accommodate this without modification.
Inefficient for Larger Capacities: If the capacity were larger, arrays could become inadequate due to the need for resizing or increased memory consumption.

Manual Management Required:
No Built-in Bounds Checking: We must manually check for array bounds and manage the count variable to prevent IndexOutOfBoundsException.
Duplicate Handling: Additional code is required to ensure that duplicate page numbers are not added.

No Built-in Methods:
Lack of Utility Methods: Unlike Java's collection classes, arrays do not provide convenient methods for everyday operations (e.g., contains, add with automatic resizing), so these must be implemented manually.

Conclusion

Given the project's specific requirements, a fixed-size array is the most appropriate data structure for the Pagelist class. It provides a simple and efficient way to store a small, fixed number of unique page numbers while maintaining the order in which pages are added. Although there are some disadvantages, such as a lack of flexibility and the need for manual management, these are mitigated by the constraints of the project and the small size of the pagelist.

By choosing a fixed-size array, we ensure that the Pagelist is optimized for the given task without unnecessary complexity or overhead.

References

Project 5: We the People Assignment Description
Java Documentation: Oracle's Java SE Documentation for arrays and collections.