Summary

Lesson: Implement a stack using singly linked list

- **Push Operation** : The push operation involves creating a new node, updating its data, linking it to the top of the linked list, and updating the top pointer. This operation has a time complexity of O(1) because it only requires traversing the current pointer.
- **Pop Operation** : The pop operation involves removing the first node from the linked list by updating the head pointer to the next node in the list. This operation also has a time complexity of O(1) for the same reason as the push operation.
- **Time and Auxiliary Space Complexity Analysis** : Both push and pop operations have a time complexity of O(1), making them efficient for stack operations. However, the auxiliary space complexity is O(N), where N is the size of the stack, because each node in the linked list requires additional memory.

Here is the summary of the lesson:

Lesson: Applications, Advantages and Disadvantages of Stack

- **Function calls** : Stacks are used to keep track of return addresses of function calls, allowing programs to return to the correct location after a function has finished executing. This is achieved through push and pop operations on the stack.
- **Efficiency** : Push and pop operations on a stack can be performed in constant time (O(1)), providing efficient access to data. This makes stacks

suitable for applications where fast data retrieval is crucial.

- **Last-in, First-out (LIFO) principle** : Stacks follow the LIFO principle, ensuring that the last element added to the stack is the first one removed. This behavior is useful in many scenarios, such as function calls and expression evaluation.

Note: I did not include the other concepts mentioned in the lesson content, as they are more related to specific applications of stacks rather than fundamental principles or techniques.

Lesson: What is Stack Data Structure? A Complete Tutorial

- **LIFO Principle** : The stack follows the Last In First Out (LIFO) principle, where elements are added and removed in reverse order of their addition.
- **Types of Stacks** : There are two types of stacks: Fixed Size Stack and Dynamic Size Stack. Fixed size stacks have a fixed capacity, while dynamic size stacks can grow or shrink dynamically.
- **Basic Operations on Stack** : The basic operations on a stack include push, pop, top, isEmpty, and isFull. These operations allow for the manipulation of elements in the stack, including adding, removing, and checking the status of the stack.

Note: Complexity analysis is not explicitly mentioned in the provided content, but based on general knowledge, the time complexity for these operations would be O(1) for push, pop, top, isEmpty, and isFull.

Lesson: Implement Stack using Array

- **Initialization of Stack** : The stack is initialized by creating an array,

treating its end as the top of the stack, and defining a capacity for the stack.

- **Push Operation** : The push operation adds an item to the stack. If the stack is full, it results in an overflow condition. The algorithm checks if the stack is full before pushing the element, and if so, it cannot be inserted into the stack.
- **Pop Operation** : The pop operation removes an item from the stack. If the stack is empty, it results in an underflow condition. The algorithm checks if the stack is empty before popping the element, and if so, it cannot remove any element from the stack.

Note: The complexity analysis for each operation is as follows:
- Time Complexity: push (O(1)), pop (O(1)), peek (O(1)), isEmpty (O(1)), isFull (O(1))
- Auxiliary Space: O(n), where n is the number of items in the stack.

Based on the provided lesson summaries, here's a comprehensive overview of the key concepts and recurring techniques:

Overview

The course covers the fundamental principles and techniques of implementing a stack data structure using different approaches. The core concept of a stack is introduced, along with its applications, advantages, and disadvantages.

Key Concepts

1. **Last-In-First-Out (LIFO) Principle** : Stacks follow this principle, ensuring that the last element added to the stack is the first one removed.

2. **Basic Operations** : Push, pop, top, isEmpty, and isFull are the basic operations on a stack, allowing for manipulation of elements in the stack.
3. **Time and Auxiliary Space Complexity Analysis** : Both push and pop operations have a time complexity of O(1), making them efficient for stack operations. However, auxiliary space complexity is O(N) due to the additional memory required for each node.

Recurring Techniques

1. **Singly Linked List Implementation** : The first lesson introduces implementing a stack using a singly linked list, which provides an efficient way to perform push and pop operations.
2. **Array-Based Implementation** : The third lesson covers implementing a stack using an array, which is useful when the capacity of the stack needs to be fixed or dynamic.

Common Themes

1. **Efficiency** : Stacks are designed to provide efficient access to data through constant-time push and pop operations.
2. **Last-In-First-Out (LIFO) Principle** : The LIFO principle is a fundamental property of stacks, ensuring that elements are removed in the reverse order of their addition.

Applications

1. **Function Calls** : Stacks are used to keep track of return addresses of function calls, allowing programs to return to the correct location after a function has finished executing.

2. **Expression Evaluation** : Stacks can be used to evaluate expressions by following the LIFO principle.

Overall, the course provides a comprehensive introduction to stacks and their applications, highlighting the importance of efficiency, LIFO principle, and basic operations in implementing stack data structures using different approaches.