model real-world objects such as a line of train carts. \n\nTerminology concerning linked lists is also discussed, including the head and tail of the list, the nodes, and the pointers or references which point to the next node. There are two types of linked lists, singly linked and doubly linked. Singly linked lists only contain a pointer to the next node, while doubly linked lists contain a pointer to the previous node as well. The pros and cons of using singly and doubly linked lists are discussed, including the fact that singly linked lists use less memory but cannot access previous elements, while doubly linked lists can access previous elements but use more memory.
Doubly linked lists allow for easier traversal backwards and removal in constant time, but use twice as much memory as singly linked lists. Inserting and removing elements involve seeking to the position in the list and changing the appropriate pointers. Singly linked lists require two pointers to remove an element, while doubly linked lists only require one. Searching for an element in a linked list takes linear time in the worst case. Adding or removing elements at the head or tail is done in constant time, but removing from the tail of a singly linked list takes linear time. The article also includes source code for a doubly linked list implementation in Java.
To remove the head or tail, the data is extracted and the head/tail pointers are moved accordingly, and memory deallocation is performed if needed. To remove an arbitrary node, the adjacent pointers are adjusted to skip over the node, and memory cleanup is performed. It's also possible to remove a node at a particular index or with a particular value using a linear search. The text then introduces stacks as a data structure that models real-world stacks, where elements are added and removed from the top. Stacks have two primary operations, push and pop, and are used in various applications such as text editors, compilers, and graph traversals. Stacks have constant time complexity for pushing, popping, and peeking, but linear time complexity for searching. The text concludes with an example problem that uses stacks, which is to determine whether a string with brackets is properly matched. The solution involves using a stack to keep track of left brackets and checking if right brackets match the top element of the stack.