Introduction to Linked List in Data Structures (With Notes)

codewithharry.com/videos/data-structures-and-algorithms-in-hindi-13

Linked lists are the new data structure we'll explore today. The study of linked lists will certainly be detailed, but first, I would like to inform you about one of the fundamental differences between linked lists and arrays.

Arrays demand a contiguous memory location. Lengthening an array is not possible. We would have to copy the whole array to some bigger memory location to lengthen its size. Similarity inserting or deleting an element causes the elements to shift right and left, respectively.

But linked lists are stored in a non-contiguous memory location. To add a new element, we just have to create a node somewhere in the memory and get it pointed by the previous element. And deleting an element is just as easy as that. We just have to skip pointing to that particular node. Lengthening a linked list is not a big deal.

Structure of a Linked List:

Every element in a linked list is called a node and consists of two parts, the data part, and the pointer part. The data part stores the value, while the pointer part stores the pointer pointing to the address of the next node.

Both of these structures (arrays and linked lists) are linear data structures.

Linked Lists VS Arrays:

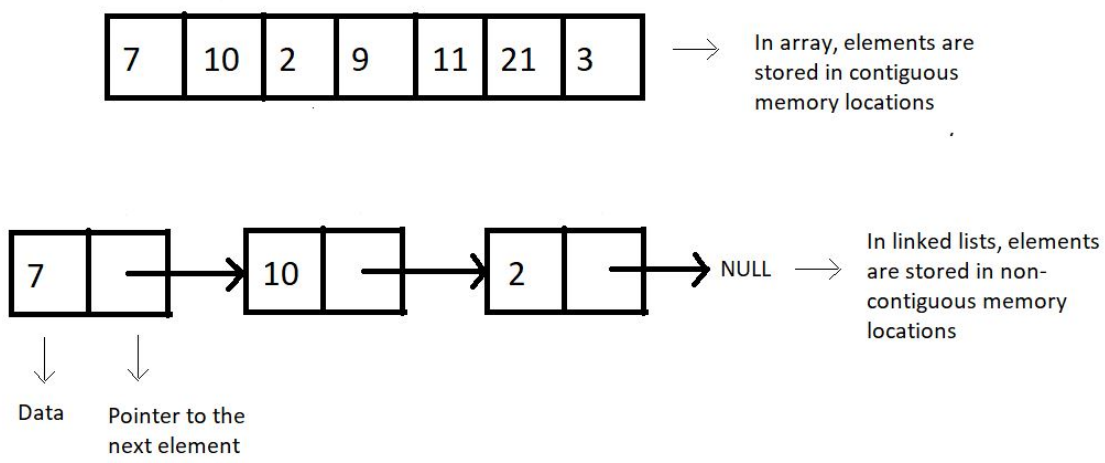

Figure 1: Arrays vs. Linked lists

Why Linked Lists?

Memory and the capacity of an array remain fixed, while in linked lists, we can keep adding and removing elements without any capacity constraint.

Drawbacks of Linked Lists:

- Extra memory space for pointers is required (for every node, extra space for a pointer is needed)
- Random access is not allowed as elements are not stored in contiguous memory locations.

Implementations

Linked lists are implemented in C language using a structure. You can refer to the snippet below.

Understanding the snippet below:

1. We construct a structure named *Node*.
2. Define two of its members, an integer *data,* which holds the node's data, and a structure pointer, *next,* which points to the address of the next structure node.

```
struct Node
{
    int data;
    struct Node *next; // Self referencing structure
};
```

Code Snippet 1: Implementation of a linked list

These were just the basics of the linked lists. We haven't delved into details yet. There's a lot to cover in this data structure. Just sit back and enjoy.

Thank you for being with me throughout. I hope you enjoyed the tutorial. If you appreciate my work, please let your friends know about this course too. Don't forget to download the notes from the link below. If you haven't checked out the whole playlist yet, move on to codewithharry.com or my YouTube channel to access it. See you all in the next tutorial, where we'll explore more of the functionalities of linked lists. Till then, keep learning.

Download Notes here