Linked Lists

INTRODUCTION

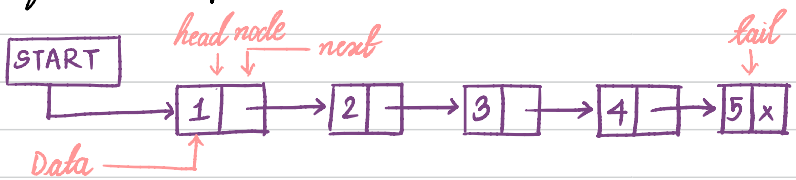
- While declaring arrays we have to specify the size of the array, which would restrict the number of elements that the array can store.
- For example, int age [15] can store a maximum number of 15 data elements, not more than that. But what if we are not sure of the number of elements in advance ?
- Moreover, to make efficient use of memory, the elements must be stored randomly at any location rather than in consecutive locations.
- Linked List is a data structure that is free from such aforementioned restrictions.
- In Linked Lists, elements aren't stored in consecutive memory locations and the user can add any number of elements to it.
- However, unlike arrays, linked lists do not allow random access of data.
- Elements in Linked Lists can only be accessed in a sequential manner.
- But like arrays, insertions and deletions can be done at any point in the list in a constant time.

ARRAY VS LINKED LIST

ARRAYS	LINKED LISTS
Size of an array is fixed.	Size of a list is not fixed.
Array supports random access. Hence, accessing elements in an array is fast with a constant time.	Linked lists supports sequential access. To access nth element of a linked list, time complexity is O(n).
In an array, elements are stored in contiguous memory location or consecutive manner in the memory.	In a linked list, new elements can be stored anywhere in the memory.
In array, Insertion and Deletion operation takes more time, as the memory locations are consecutive and fixed. Inserting new elements at the front is potentially expensive because existing elements need to be shifted over to make room.	In case of Linked list, a new element is stored at the first free and available memory location, with only a single overhead step of storing the address of memory location in the previous node of linked list. Therefore, Insertion & Deletion are fast.

LINKED LISTS

- A linked list, in simple terms is linear collection of data elements. These data elements are called nodes.
- Linked list is a data structure which in turn can be used to implement other data structures.
- Thus, it acts as a building block to implement data structures such as stacks, queues and their variations.
- A linked list can be perceived as a train or a sequence of nodes in which each node contains one or more data fields and a pointer to the next node.



Simple Linked List

- In a linked list, every node contains two parts, left part containg the data and right part containing pointer to the next node (or address of the next node in sequence).
- The data in the left part of the node may include a simple data type, an array, or a structure.
- The last node will have no next node connected to it, so in the right part, it will store a special value called NULL represented as x. NULL pointer denotes the end of list.
  While programming, we usually define NULL as -1.
- Linked list is also called as 'self-referential data type' as every node contains a pointer to another node which is of the same type.
- Linked list also conlain a pointer variable START that stores the address of the first node in the list.
- The NEXT part of the node stores the address of its succeeding node.
- If START == NULL, the linked list is empty and contains no nodes.

In C++, we can initialize a simple linked list as:

```
struct node {
        int data;
        struct node *next;
};
```

MEMORY ALLOCATION AND DE-ALLOCATION

- If we want to add a node to an already existing linked list in the memory and then use it to store the information.
- The question is which part of the memory is available and which part is occupied?
- The computer maintains a list of all free memory cells. This list of available space is called free pool.
- For the free pool, we have a pointer variable AVAIL which stores the address of the first free space.
- whenever, we enter information into this free space, the next available free space's address will be stored in AVAIL. This was all about inserting a new node.
- When we delete a particulare node from an existing linked list, the space occupied by it must be given back to the free pool, so that the memory can be reused by some other program that needs memory space.
- The operating system (OS) does this task of adding the freed memory to the free pool. OS will perform this operation whenever it finds CPU idle or whenever the programs are falling short of memory space.
- The OS scans through all the memory cells and marks those cells that are being used by some program. Then it collects all the cells that are not being used and adds their address to the free pool, so these cells can be reused by other programs. This process is called garbage collection.

END