Linked List

What is linked list?

A linked list is a data structures, with a sequence of links which contains items. Each link contains a connection to another link. Linked list is the second most-used data structure after array. It is a dynamic data structure whose length can be increased or decreased at run time.

Why Linked list?

Arrays can be used to store linear data of similar types, but arrays have the following limitations.

- The size of the arrays is fixed: So, we must know the upper limit on the number of elements in advance. Also, generally, the allocated memory is equal to the upper limit irrespective of the usage.
- Inserting a new element in an array of elements is expensive because the room has to be created for the new elements and to create room existing elements have to be shifted.

For example, in a system, if we maintain a sorted list of IDs in an array id [ ].
id [] = [1000, 1010, 1050, 2000, 2040].

And if we want to insert a new ID 1005, then to maintain the sorted order, we have to move all the elements after 1000 (excluding 1000).
Deletion is also expensive with arrays until unless some special techniques are used. For example, to delete 1010 in id [], everything after 1010 has to be moved.

Advantages of Linked List over arrays

- Dynamic size
- Ease of insertion/deletion

Disadvantages:

- Random access is not allowed. We have to access elements sequentially starting from the first node. So, we cannot do binary search with linked lists efficiently with its default implementation. Read about it here.
- Extra memory space for a pointer is required with each element of the list.

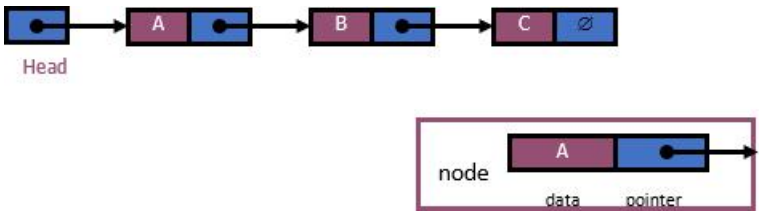
- Not cache friendly. Since array elements are contiguous locations, there is locality of reference which is not there in case of linked lists.

How linked lists are arranged in memory?

Linked list basically consists of memory blocks that are located at random memory locations. Now, one would ask how are they connected or how they can be traversed? Well, they are connected through pointers. Usually, a block

in a linked list is represented through a structure.

Linked List Representation



A **linked list** is a linear data structure where each element is a separate object.

Each element (*we will call it a **node***) of a list is comprising of two items - **the data** and a **pointer** to the next node.

The last node has a reference to **null.**

The entry point into a linked list is called the **head** of the list. It should be noted that head is not a separate node, but the reference to the first node.

If the list is empty then the head is a **null** reference.

Types of Linked List

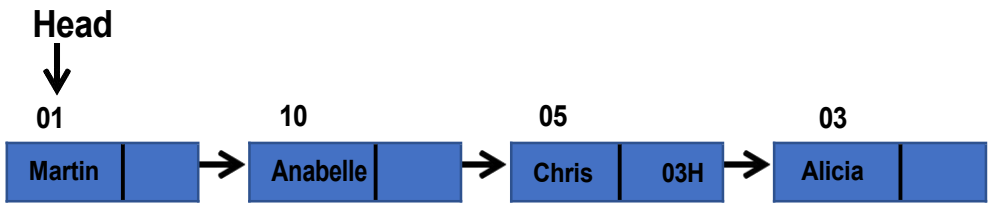
- **Singly-linked List**

Doubly-linked List Singly Linked List

In singly Linked List each element is represented as a structure called
Node. Each Node is divided into two parts:

- **Data Field**- data field is used to contain the value of the element.
- **Pointer Field**. The pointer field, link or simply **reference** contains the address of the next node in the list of a computer memory which is a number that refers to an object.

The next node in the list is called **SUCCESSOR.** If a node is the last in the list, meaning it has no successor, the pointer field would contain the value NULL**.**



The figure is an example of a singly-linked list containing four nodes. The node in the list is called the HEAD.

NOTE: *that there is no limit to the size of a singly-linked list. Adding a node to a linked-list is simply a matter of:*

- Creating a new node.
- Setting the data field of the new node to the value to be inserted into the list.
- Assigning the address of the new node to the pointer field of the current last node in the list.

Setting the pointer field of the new node to NULL.

Operation in Singly Linked-List

- **Insert Node** – for every insertion of a node, it is usually inserted at the beginning of the list. This is the simplest approach though it's also possible to insert nodes anywhere in the list.
- **Search Node** – when searching a value existing in a list, it moves along the list then prompts you the address of the node. However, when not found an error will return prompting you that the item can't be found.
- **Delete Node** – specify a value to delete in the list and it performs first the searching before deleting the value. Again, it moves along the list and when found, algorithm then deletes the value. After deleting the specified value, it then connects the arrow from the previous link straight across to the following link. This is how links are removed: the reference to the preceding link is changed to point to the following link.

Inserting A Node in A Singly-linked List General

Procedure:

- Create a new node for the element.
- Set the data field of the new node to the value to be inserted.
- Insert the node.

Three Locations in Inserting a New Node:

- Insert the node at the start of the list (i=1)
- Insert the node at the end of the list.(i>length of the list)
- Insert the node at the position i, where l<i<length of the list

Deleting A Node in A Singly-linked List

There are basically two ways to identify a node to be deleted in a singly- linked list:

- **By Position** – the position of the node with respect to the start of the list is specified.
- **By Value** – the contents of the data field of the node are identified.

This method may only be used if the nodes in the list are unique.

General Procedure:

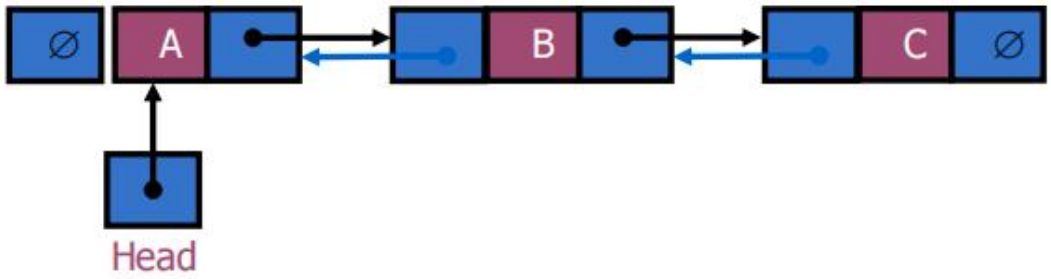
- Locate the node.
- Delete the node.
- Release the node from the memory.

Doubly-linked Lists

A **Doubly Linked List** (DLL) contains an extra pointer, typically called ***previous pointer***, together with next pointer and data which are there in singly linked list.

- Each node points to not only successor but the predecessor
- There are two NULLS: at the first and last nodes in the list

Advantage: given a node, it is easy to visit its predecessor. Convenient to traverse lists ***backwards.***



A Doubly Linked List allows you to traverse backward as well as forward through the list. Behind this concept is that each link has two references to other links instead of one. Each node is divided into three:

- left pointer field
- data field
- right pointer field.

Inserting A Node in A Doubly-linked List General

Procedure:

- Create a new node for the element.
- Set the data field of the new node to the value to be inserted.
- Insert the node in ***three locations:***
    - Insert the node at the start of the list (i=1)
    - Insert the node at the end of the list.(i>length of the list)
    - Insert the node at the position i, where l<i<length of the list

Deleting A Node in a Doubly-linked List General

Procedure

- Locate the node.
- Delete the node.
- Release the node from the memory