UNIT–4 : Linked List Algorithm

1. Insertion at Beginning

In this operation, we are adding an element at the beginning of the list.

Algorithm

```
1. START
2. Create a node to store the data
3. Check if the list is empty
4. If the list is empty, add the data to the node and
   assign the head pointer to it.
5. If the list is not empty, add the data to a node and link to the
   current head. Assign the head to the newly added node.
6. END
```

2. Insertion at Ending

In this operation, we are adding an element at the ending of the list.

Algorithm

```
1. START
2. Create a new node and assign the data
3. Find the last node
4. Point the last node to new node
5. END
```

3. Insertion at a Given Position

In this operation, we are adding an element at any position within the list.

Algorithm

```
1. START
2. Create a new node and assign data to it
3. Iterate until the node at position is found
4. Point first to new first node
5. END
```

4. Deletion at Beginning

In this deletion operation of the linked, we are deleting an element from the beginning of the list. For this, we point the head to the second node.

Algorithm

```
1. START
2. Assign the head pointer to the next node in the list
3. END
```

5. Deletion at Ending

In this deletion operation of the linked, we are deleting an element from the ending of the list.

Algorithm

```
1. START
2. Iterate until you find the second last element in the list.
3. Assign NULL to the second last element in the list.
4. END
```

6. Deletion at a Given Position

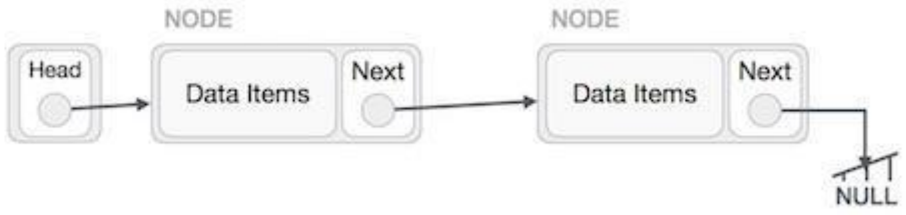
In this deletion operation of the linked, we are deleting an element at any position of the list.

Algorithm

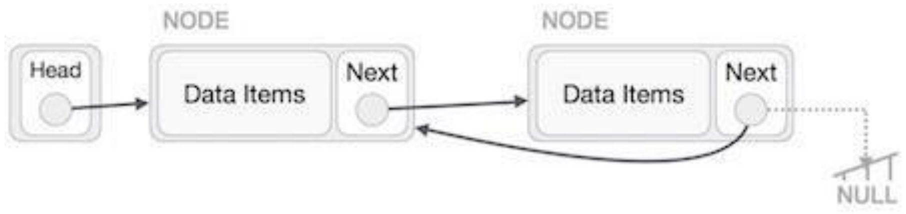
```
1. START
2. Iterate until find the current node at position in the list.
3. Assign the adjacent node of current node in the list
   to its previous node.
4. END
```

7. Linked List - Reversal Operation

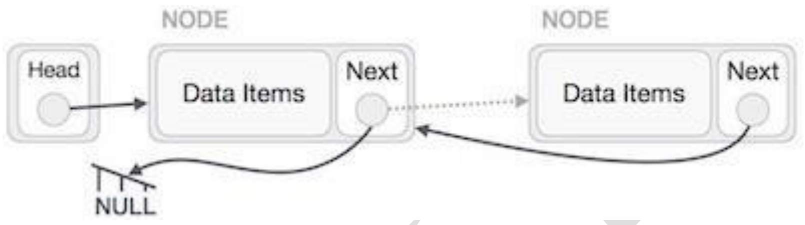
This operation is a thorough one. We need to make the last node to be pointed by the head node and reverse the whole linked list.



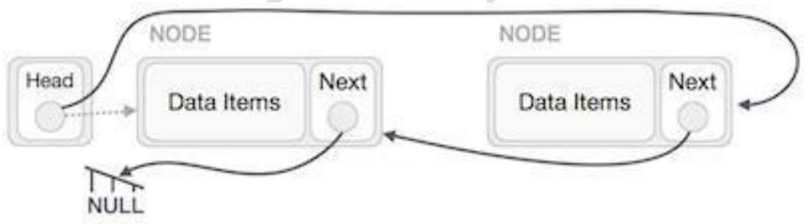
First, we traverse to the end of the list. It should be pointing to NULL. Now, we shall make it point to its previous node –



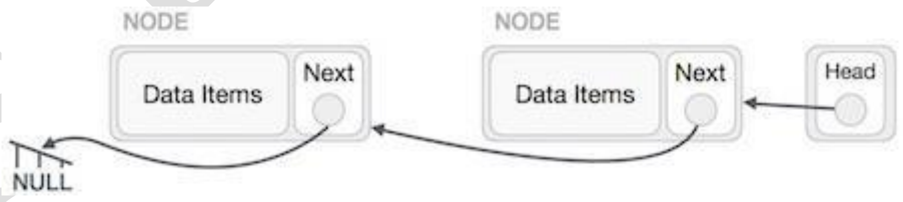
We have to make sure that the last node is not the last node. So we'll have some temp node, which looks like the head node pointing to the last node. Now, we shall make all left side nodes point to their previous nodes one by one.



Except the node (first node) pointed by the head node, all nodes should point to their predecessor, making them their new successor. The first node will point to NULL.



We'll make the head node point to the new first node by using the temp node.



Algorithm

Step by step process to reverse a linked list is as follows −

```
1. START
2. We use three pointers to perform the reversing:
   prev, next, head.
```

```
3. Point the current node to head and assign its next value to
   the prev node.
4. Iteratively repeat the step 3 for all the nodes in the list.
5. Assign head to the prev node.
```

8. Linked List - Search Operation

Searching for an element in the list using a key element(data). This operation is done in the same way as array search; comparing every element in the list with the key element given.

Algorithm

```
1 START
2 If the list is not empty, iteratively check if the list
  contains the key
3 If the key element is not present in the list, unsuccessful
  search
4 END
```

9. Linked List - Traversal Operation

The traversal operation walks through all the elements of the list in an order and displays the elements in that order.

Algorithm

```
1. START
2. While the list is not empty and did not reach the end of the list,
   print the data in each node
3. END
```