Linked-List

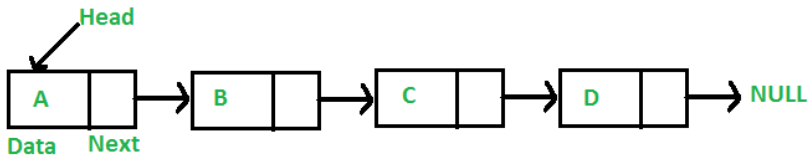
What is Linked List ?

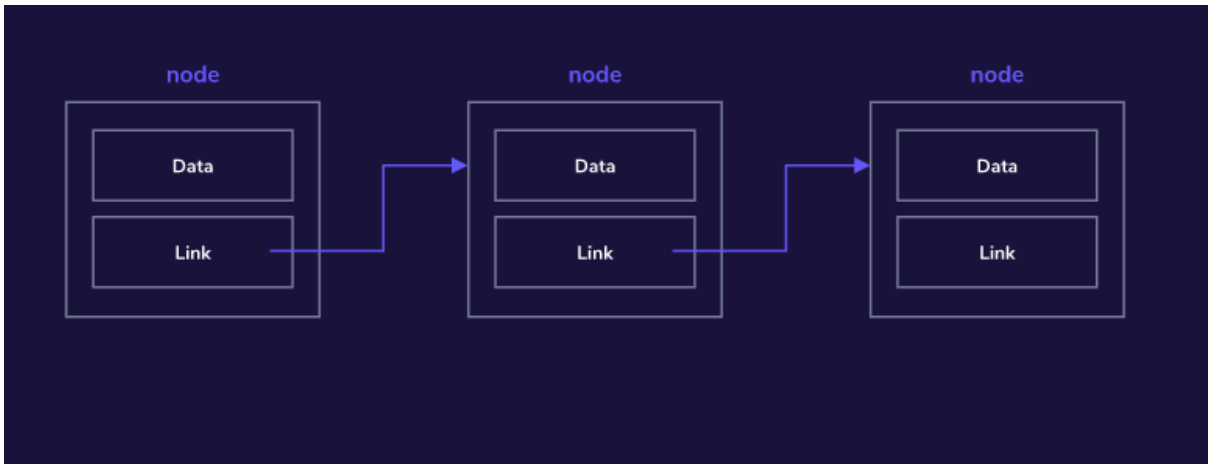
A linked list is a linear data structure, in which the elements are not stored at contiguous memory locations.

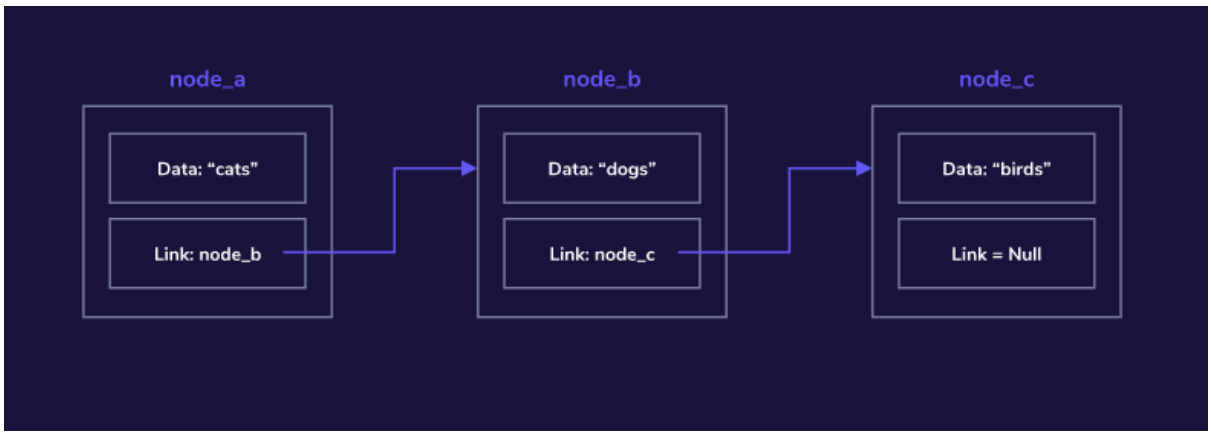
A Linked list is like a chain made of nodes and the links are pointers.

Pointers are the connections that hold the pieces of linked structures together.
Pointers represent the address of a location in memory.

The elements in a linked list are linked using pointers as shown in the below image







Why Linked List ?

- They can be used to implement other common abstract data types, including lists, stacks, queues
- preferred data structures over arrays because of elements can be easily inserted or removed without reallocation or reorganisation of the entire structure

Pros & Cons: Linked List

Pros	Cons
Dynamic Size (there is no need to ever resize a linked list)	Very Slow when it comes to Access and Search we have to iterate over each element
Quick Insertion/ Deletion of Nodes because you just change the pointers of each node to insert/delete	Extra space for pointer

Time Complexity

Operation	Big O	Explanation
Insertion	O(1)	Change the pointer of tail or head to address of newly added node
Deleting a Node	O(1)	Change pointer
Accessing an Particular Node	O(n)	We have to iterate over each node from first node to the target node
Searching	O(n)	We have to iterate over each node from starting to the target node similar to Accessing an Particular Node

Code Implementation

Implementing a linked list

```
#include <bits/stdc++.h>
using namespace std;


// Creating Node Class
class Node
{
public:
  int data;
  Node *next;
  Node* head = NULL;
// Creating Constructor for Node Class
  Node (int data)
  {
    this -> data = data;
    next  = NULL;
```

```
  }
};


int main() {
  // Creating 1st Node Dynamically
  Node *n1 = new Node(5);

  // Making Head Pointing to 1st node
  Node *head = n1;

  // Creating 2nd Node Dynamically
  Node *n2 = new Node(6);

  //Linking 1st Node to 2nd one
  n1->next = n2;

  //Printing the data stored or values in node

  cout<<"Data in n1: "<<n1->data<<endl;
  cout<<"Data in n2: "<<n2->data<<endl;

}
```

Printing The whole Linked List

```
// Function To Print Linked List
void printNode(Node* head)
{

  // Creating A Temp Pointer so that we may not disturb head,
  // and head can be used in future
  Node* temp = head;

  // If temp==NULL It would be eND oF Linked List
  while(temp!=NULL)
  {
    // Printing Out data Stored in Node
    cout<<temp->data<<" ";

    //Changing The temp pointer to next node
    temp= temp -> next;
  }
}
```

Taking Linked List as Input

`inputLinkedList` function takes Liked List as input and -1 is considered ad terminating condition.

- This is for inserting new Node at Tail

```


Node* inputLinkedList()
{
  // Take Input
  int data; cin>>data;
  //Initlize Head & Tail pointing towards NULL
  Node *head = NULL;
```

```
  Node *tail = NULL;

  // In This Case -1 is considered as terminating int of Linked List
  while(data != (-1))
  {
    // Create a New Node
    Node *n = new Node(data);

    // If head = NULL Then Its First Node
    if(head == NULL)
    {
      // If First Node Then Head and Tail are same
      head = n;
      tail = n;
    }
    else
    {
      // If not First Node Then Insert New Node at Tail
      tail->next = n;
      tail = n;
    }

    // Again Take input to Keep while loop going until input  is -1
    cin>>data;
  }

  // When while Loop terminates return the head of linked list
  return head;
}
```

- For Inserting at Head

```
Node* inputLinkedList()
{
  // Take Input
  int data; cin>>data;
  //Initlize Head & Tail pointing towards NULL
  Node *head = NULL;
  Node *tail = NULL;

  // In This Case -1 is considered as terminating int of Linked List
  while(data != (-1))
  {
    // Create a New Node
    Node *n = new Node(data);

    // If head = NULL Then Its First Node
    if(head == NULL)
    {
      // If First Node Then Head and Tail are same
      head = n;
      tail = n;
    }
    else
    {
      // If not First Node Then Insert New Node at Head

      // Linking The newly formed node to the previous head
      n->next = head;

      // Updating Head to newly formed node
      head = n;

    }

    // Again Take input to Keep while loop going until input  is -1
```

```
    cin>>data;
  }

  // When while Loop terminates return the head of linked list
  return head;
}
```

Counting No of nodes in Linked list

```
int lengthOfLL(Node* head)
{
  Node* temp = head;
  int count=0;

  // If temp is NULL then its the end of linked list
  while(temp!=NULL)
  {
    count++;

    // Shifting temp to next node
    temp = temp -> next;
  }
  return count;
}
```

Printing ith Node of Linked List

```
int print_ith_node(Node *head,int n)
{
  Node *temp = head;

  // Itrating from 1st node to nth node
  for(int i=0;i<n;i++)
  {
    // Updating temp to next node
    temp = temp -> next;
  }

  // Returning Data stored in temp
  return temp -> data;

}
```

Inserting a Node at ith position

This Takes the assumption that the given position to insert is a valid one

- `i` is the position to insert the node and `data` is the value which we want to store in node
- Indexing starts form `0`

```
Node* insertNode(Node* head, int i, int data)
{
  // iF i = 0 then we have to insert at head
  if(i==0)
  {
    // make a new node for inserting
    Node *n = new Node(data);

    // Make link of new node to previous head
    n->next = head;

    // Change head to the newly formed node
    head = n;
    return head;
  }
  else
  {
    // make a new node for inserting
    Node *n = new Node(data);

    // make a new temp node for itrating over the List so that we may not disturb the orignal head
    Node *temp = head;

    // Iterating till i-1 node
    for(int j=0;j<i-1;j++)
    {
      temp = temp -> next;
    }

    // Set link feild of newly created node as link feild of i-1 th node
    n->next = temp-> next;

    // Change i-1 th node to newly created node
    temp -> next = n;
    return head;
  }
}
```

Delete Node at ith Position

```
Node * deleteNode(Node * head, int i)
{
  // make a new temp node for itrating over the List
  // so that we may not disturb the orignal head
  Node* temp1 = head;

  // If i=0 then we have to delete the head
  if(i==0)
  {
    // Change head feild to the head + 1 node
    head = temp1 -> next;

    // delete temp node to avoid memory leak
    delete temp1;
    return head;
  }
  else
  {

    // Itrate to i-1 th node
    for(int j=0;j<i-1;j++)
    {
```

```
      temp1 = temp1 -> next;
    }

    // Make a new node pointing to temp1 +1 node (which is ith node)
    Node* temp2 = temp1 -> next;
    temp1 -> next = temp2 -> next;
    delete temp2;
    return head;
  }
}
```

All Of The Above Functions Are Implemented In This

https://s3-us-west-2.amazonaws.com/secure.notion-static.com/ee286e80-44a4-4f30-bae6-57207d774b77/Linked-List.cpp

Algorithms or Techniques Used in Linked List

Find Mid of Linked List: Slow and Fast pointer Approach

Middle of the Linked List - LeetCode

Level up your coding skills and quickly land a job. This is the best place to expand your knowledge and get prepared for your next interview.

https://leetcode.com/problems/middle-of-the-linked-list

Given a non-empty, singly linked list with head node `head`, return a middle node of linked list.

If there are two middle nodes, return the second middle node.

```
Node* Mid(Node * head)
{

  // Make a slow Pointer pointing to head of LL
  Node* slow_ptr = head;

  // Make a fast pointer pointing to next node of head of LL
  Node* fast_ptr = head-> next;

  // We Need to traverse the List until either of fast pointer
  // of the Node next to fast pointer is NULL (NULL means End of LL)
  while(fast_ptr and fast_ptr->next)
  {

    // Slow Ptr takes 1 jump
    slow_ptr = slow_ptr -> next;

    // Fast ptr takes 2 jump
    fast_ptr = fast_ptr -> next -> next;
```

```
  }

  // Condition for Even length of LL
  if(fast_ptr!=NULL){return slow_ptr->next;}

  // Condition for Odd length of LL
  else{
    return slow_ptr;
  }
}
```

Reverse a Linked List

Reverse Linked List - LeetCode

Given the head of a singly linked list, reverse the list, and return the reversed list. Example 1: Input: head = [1,2,3,4,5]Output: [5,4,3,2,1] Example 2: Example 3:

https://leetcode.com/problems/reverse-linked-list

Given the head of a singly linked list, reverse the list, and return the reversed list.

```
Node* reverseLL(Node* head)
{
  Node* current = head;

  // Initally previous will be NULL coz current starts from head
  Node* previous = NULL;
  while(current != NULL)
  {
    // Create a new node n so that we may not lose head while reversing the list
    Node* n = current->next;

    // Break the link btwn current and end and reverse it
    // Before: previous->current->n
    // After: previous <- current
    current -> next = previous;

    // Move the pointers one node ahead
    previous = current;
    current =n;
  }

  //At the end prev will be pointing to the head of reversed linked list bcz current will be pointing to NULL
  return previous;
}
```

Find Kth Node from the End

Nth node from end of linked list | Practice | GeeksforGeeks

Given a linked list consisting of L nodes and given a number N. The task is to find the Nth node from the end of the linked list. Example 1: Input: N = 2 LinkedList: 1->2->3->4->5-

https://practice.geeksforgeeks.org/problems/nth-node-from-end-of-linked-list/1

There are two ways to do this

- In 2 traversal

Firstly we need to find the number of nodes in the list and then

$$KthNodeFromEnd = (n - k + 1)thNodefromFront$$

Where, n = Number of Node

$$TimeComplexity = O(N)$$

- In 1 traversal

```
Node* NodeFromEnd(Node* head,int k)
{
  // This is for only Valid value of K

  // Create 2 pointer pointing towards head
  Node* one =head;
  Node* two = head;
  int temp = k;

  // Make th pointer one do k jumps so that the diffrence
 //btwn ptr one & two is k
  while(temp--)
    {
      one = one -> next;
    }

  // Now Both will do the same no of jumps until one is equal to NULL
  // When one is eqal to Null its the end of LL
  while(one!=NULL)
  {
    one = one -> next;
    two = two -> next;
  }
  // when one reaches the end of LL two
  // will be at Kth position from end as the no of elements in btwn them is K
  return two;
}
```

Delete Kth Node from End

Given the head of a linked list, remove the nth node from the end of the list and return its head.

Similar to the above one, Instead of finding we have to delete also

Remove Nth Node From End of List - LeetCode

Level up your coding skills and quickly land a job. This is the best place to expand your knowledge and get prepared for your next interview.

https://leetcode.com/problems/remove-nth-node-from-end-of-list/

In 1 traversal, Complexity is O(n)

```
Node* removeNthFromEnd(Node* head, int n) {
        Node* one = head;
        Node* two = head;
        while(n--)
        {
            one = one -> next;
        }
        // If we have to remove head from LL
        if(one == NULL){return two-> next;}

        while(one->next!=NULL)
        {
            one = one -> next;
            two = two -> next;
        }

          // Link two with its next to next node
        two->next = two->next->next;
        return head;
    }
```

Merge Two Sorted Linked List

Merge Two Sorted Lists - LeetCode

Merge two sorted linked lists and return it as a sorted list. The list should be made by splicing together the nodes of the first two lists. Example 1: Input: l1 = [1,2,4], l2 =

https://leetcode.com/problems/merge-two-sorted-lists/

- In-Place Approach: Iterative

$$TimeComplexity = O(M + N)$$

- Where M and N are the Number of elements present in the two LL

1. Create 4 pointers two named head and tail and two pointer named `temp1` and `temp2` pointing towards head of `LL1` and `LL2` respectively.
2. Find the element with smallest value which should be either if 1st element of `LL1` or `LL2`
3. Initially make both head and tail point towards that small element as if its the only element in LL

4. find the smallest element among `temp1` and `temp2` and append it at the tail of LL by making tail point towards that element so that it happens in-place.
5. Repeat step 4 until any one of LL ends
6. Now append all the elements of the LL which is left to the tail and return head.

```
Node* Merge2SortedLL(Node* LL1, Node* LL2)
{

  // If LL1 is Null that means there is no LL1 and same goes for LL2
  if(LL1 == NULL){return LL2;}
  if(LL2 == NULL){return LL1;}

  // Create a new head and tail pointer initially pointing to NULL
  Node* head = NULL;
  Node* tail = NULL;

  // Create new temp1 and temp2
  //pointing towards head of LL1 & LL2 respectively
  Node* temp1 = LL1;
  Node* temp2 = LL2;


  // Checking the smaller value among
  // the first element of temp1 and temp2
  if(temp1->data <= temp2->data)
  {

    // initially Making Both head and tail point
    // towards the small 1st element
    head = temp1;
    tail = temp1;
    temp1 = temp1 -> next;
  }
  else
  {
    // initially Making Both head and tail point towards the small 1st element
    head = temp2;
    tail = temp2;
    temp2 = temp2 -> next;
  }


  // If any of them is NULL that means of of the LL has been
  // completly traversed and ended
  while(temp1!=NULL and temp2!=NULL)
  {
    // Checking the smaller value the first element of of temp1 and temp2
    if(temp1 -> data <= temp2 -> data)
    {
      // Appending the smaller element among the two linked list to the tail
      tail->next = temp1;
      tail = temp1;
      temp1 = temp1 -> next;
    }
    else
    {
      tail -> next = temp2;
      tail = temp2;
      temp2 = temp2 -> next;
    }
  }
```

```
  // It means that LL2 has been completly traversed
  // and there still some elements left in LL1
  // so simply make tail point to those left elements and
  // append them all in one shot
  if(temp1!=NULL)
  {
    tail->next = temp1;
  }
  else
  {
    tail -> next = temp2;
  }

  // Returning the head of LL
  return head;
}
```

Merge Sort on Linked List: Slow and Fast Pointer

Sort List - LeetCode

Level up your coding skills and quickly land a job. This is the best place to expand your knowledge and get prepared for your next interview.

https://leetcode.com/problems/sort-list/

- Using recursion

1. Break the list into two LL from Mid.
2. Apply Sorting recursion on both the broken LL so that it is sorted.
3. Merge those two sorted Linked List using the function discussed above.

The base case for recursion is that if the head is NULL or there is only one element is the List then return head.

$TimeComplexity = O(nLogN)$

```
Node* MergeSort(Node* head)
{

  // Bse case of recursive function
  if(head == NULL or head->next == NULL){return head;}

  // Initilize two pointers to implement slow and fast pointer approach
  Node* slow_ptr =head;
  Node* fast_ptr =head->next;

  // If any of this Condition satisfies then its end of LL
  while(fast_ptr!=NULL and fast_ptr->next!=NULL)
  {
    // Slow makes one jump while fast make 2 jumps
    slow_ptr = slow_ptr->next;
    fast_ptr = fast_ptr->next->next;
  }

  // Here we are about to brak the LL into 2 LL from the mid
```

```
  // Store the head of 2nd part of LL
  Node* n = slow_ptr->next;

  // Assign Null to the end of 1st LL
  slow_ptr->next = NULL;

  // Again call the recursive function on both the parts of broken LL
  Node* a = MergeSort(head);
  Node* b = MergeSort(n);


  // Merge the two sorted List into one using the previously
  // implemented Merge2SortedLL function
  Node* final = Merge2SortedLL(a,b);
  return final;
}
```