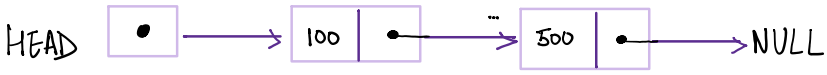
Linked List

19 March 2024 00:17

LINKED LISTS

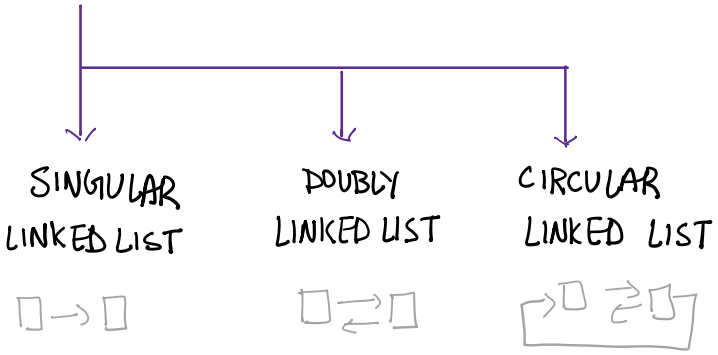
↳ collection of objects (NODE) each containing DATA and REFRENCE to at least one other node.



-> A linked list is a series of connected nodes
-> Each node contains — Data & Pointer to the next node in list

-> It can grow & shrink in size during execution of a program
-> Doesn't waste any memory space
-> Can be made just as long as required

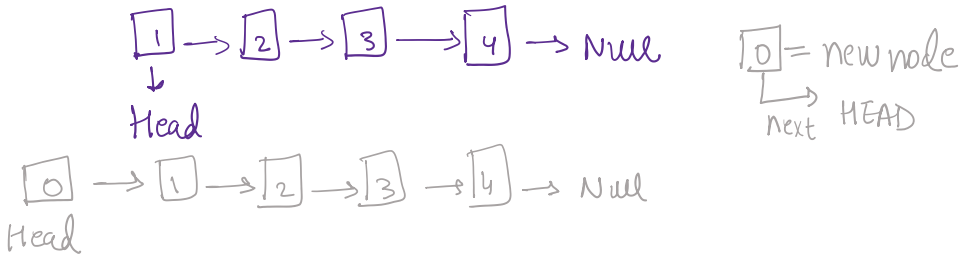
TYPES OF LIST



OPERATIONS OF LIST

IsEmpty	Determine whether the list is empty or not
InsertNode	Insert new node at particular position
FindNode	Find a new node with a given value
DeleteNode	Delete a node with a given value
DisplayList	Print all the nodes in the list

① Inserting a Node at the Beginning of a List



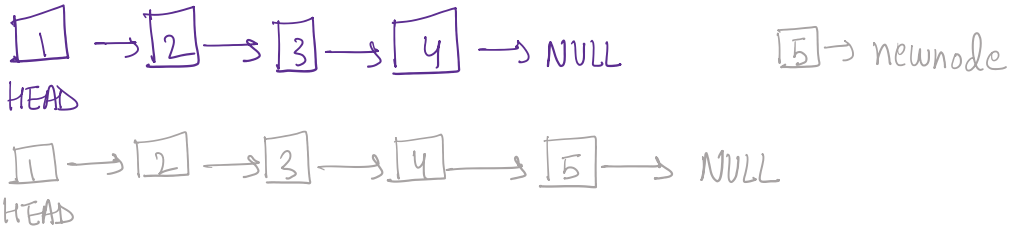
(i) Create Newnode

(ii) If list is empty the head will point to the newnode, Else:

(iii) newnode → next = head

(iv) head = newnode

② Insert a Node at the End of a List



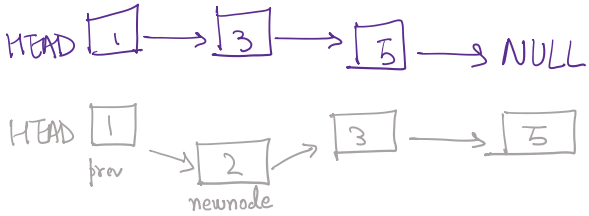
(i) Create newnode

(ii) Traverse the list starting from HEAD until the node where next pointer is NULL

(iii) Add newnode at the next pointer of last node

(iv) Set next pointer of newnode as NULL.

③ Insert a Node after a given Node



(i) Create newnode

(ii) Declare node after which the newnode is to be Inserted as 'prev'

(iii) newnode → next = prev → next

(iv) prev → next = newnode

④ Print the Contents of Each Node of the List

(i) Start

(ii) Set Ptr = Start

(iii) Repeat (iv) & (v) until Ptr ≠ NULL

(iv) print Ptr → Data

(v) set ptr = ptr → Link

(vi) Stop

⑤ Search for an Item

(i) Set Ptr = Start

(ii) While Ptr ≠ NULL, repeat Step 3 –

(iii) if item == Ptr → Data, then

(iv) set Loc = Ptr & Exit

(v) else

(vi) set ptr = ptr → Link

(vii) Set Loc = NULL

(viii) Stop

Algorithms – (i) Print/Display all the elements of list
(ii) Insert Node at - Beginning, End, Middle
(iii) Search for a Node
(iv) Delete node from Beginning, End & Middle

SINGLE LINKED LIST

(i) Algorithm to Insert a node at the beginning of a Single Linked list

SINGLE LINKED LIST

(i) Algorithm to Insert a node at the beginning of a Single Linked list

Step 1 : Start

Step 2 : Create newnode and store the desired data in newnode

Step 3 : If list is empty -

3.1 : Set the head of the list as new node

head -> newnode

Step 4 : If list is not empty -

4.1 : Set the newnode to point to the current head of the list

newnode -> head

4.2 : Set newnode as the new head of the list

newnode = head

Step 5 : Stop / End

(ii) Algorithm to Insert a node at the End of a Single Linked list

Step 1 : Start

Step 2 : Create a newnode and store the desired data in newnode

Step 3 : Traverse the list until the lastnode is reached

Step 4 : Set lastnode to point to the newnode

lastnode -> newnode

Step 5 : Set newnode to point to NULL

newnode -> NULL

Step 6 : Stop / End

(iii) Algorithm to Insert a node in the middle of a Single linked list

Step 1 : Start

Step 2 : Create newnode and store the desired data in newnode

Step 3 : Traverse the list to find the two nodes - prevnode and nextnode; between where the newnode is to be Inserted

Step 4 : Set prevnode to point to newnode

prevnode -> newnode

Step 5 : Set newnode to point to nextnode.

prevnode -> newnode

Step 5: Set newnode to point to nextnode
newnode -> nextnode

Step 6: Stop/End

(iv) Algorithm to print all items of a Single linked list

Step 1: Start

Step 2: Set head as currentnode.

Step 3: While currentnode is NOT NULL:
3.1: Print the data of currentnode
3.2: Move to the nextnode
3.3: Set nextnode as currentnode

Step 4: Stop/End

(v) Algorithm to search for a particular node of a single linked list

Step 1: Start

Step 2: Set currentnode to the head of the list

Step 3: While the currentnode is NOT NULL
3.1: If the data of the currentnode matches the key
3.1.1: Return true
3.2: Move to the nextnode
3.2: Set currentnode to nextnode

Step 4: If the key is not found after traversing the entire list
4.1: Return False

Step 5: Stop/End

(vi) Algorithm to delete a node from the beginning of a single linked list

Step 1: Start

Step 2: If the list is empty
2.1: Return without performing any operation

Step 3: Traverse to the nextnode; node that head points next to.

Step 4: Set nextnode as head

Step 3: Traverse to the nextnode; node that head points next to.

Step 4: Set nextnode as head

Step 5: Delete the previous head

5.1: Release the memory allocated to the previous head

Step 6: Stop/End

(vii) Algorithm to delete a node from the end of a Single linked list

Step 1: Start

Step 2: If the list is empty

2.1: Return without performing any operation

Step 3: Traverse to the End of the list

3.1: Set the second-lastnode to point to NULL

Step 4: Delete the lastnode

4.1: Release the memory allocated to the previous lastnode

Step 5: Stop/End

(viii) Algorithm to delete a node from the middle of a Single linked list

Step 1: Start

Step 2: If the list is empty

2.1: Return without performing any operation

Step 3: Traverse to the node that is to be deleted; say currentnode.

Step 4: Set the node before the currentnode to point to the node after the currentnode

Step 5: Delete currentnode

5.1: Release the memory allocated to the deleted node

Step 6: Stop/End

DOUBLE LINKED LIST

i) Algorithm to insert a node at the beginning of a double linked list

Step 1: Start

Step1 : Start

Step2 : Create newnode and add the desired data to newnode

Step3 : If the list is empty

3.1 : Set the head and tail to newnode

Step4 : If the list is not empty

4.1 : Set the next pointer of newnode to point to the current head

newnode -> next = head

4.2 : Set the previous pointer of current head to point to newnode

head -> prev = newnode

4.3 : Set newnode as head

Step 5 : Stop/End

(ii) Algorithm to Insert a node at the end of a double linked list

Step1 : Start

Step2 : Create newnode and store desired data in newnode

Step3 : If the list is empty

3.1 : Set the head and tail to newnode

Step4 : If the list is not empty

4.1 : Traverse to the end of the list to lastnode

4.2 : Set lastnode next pointer to point to newnode

lastnode -> next = newnode

4.3 : Set newnode previous pointer to point to the lastnode

newnode -> prev = lastnode

4.4 : Set newnode next pointer to point to NULL

newnode -> next = NULL

Step5 : Stop/End

(iii) Algorithm to Insert a node in the middle of a double linked list

Step1 : Start

Step2 : Create newnode & store the desired data in newnode

Step3 : Traverse the list to the node where the newnode is to be inserted

Step 3: Traverse the list to the node where the newnode is to be inserted between; say prevnode and nextnode

3.1: Set prevnode next pointer to point to newnode
prevnode -> next = newnode

3.2: Set newnode previous pointer to point to prevnode
newnode -> prev = prevnode

3.3: Set newnode next pointer to point to nextnode
newnode -> next = nextnode

3.4: Set nextnode previous pointer to point to newnode
nextnode -> prev = newnode

Step 4: Stop/End

(iv) Algorithm to print all items of a doubly linked list

Step 1: Start

Step 2: Set the head to currentnode

Step 3: While currentnode is NOT NULL

3.1: Print all the data of the currentnode

3.2: Move to the nextnode

3.2: Set nextnode to currentnode

Step 4: Stop/End

(v) Algorithm to search for a node in double linked list

Step 1: Start

Step 2: Set head to currentnode

Step 3: while currentnode is NOT NULL

3.1: If the data of currentnode matches the key

3.1.1: Return true

3.2: Move to the nextnode

3.3: Set nextnode to currentnode

Step 4: If the key is not found while traversing the entire list

4.1: Return False

Step 5 : Stop / End

(vi) Algorithm to delete a node from the beginning of a Double Linked List

Step 1 : Start

Step 2 : If the list is empty

2.1 : Return without performing any operation

Step 3 : If the list is not empty

3.1 : Traverse to the second node of the list

3.2 : Set previous pointer of second node as NULL

secondnode → prev = NULL

3.3 : Set secondnode as head

3.4 : Delete the previous head

3.4.1 : Release the memory allocated for the deleted node

Step 4 : Stop / End

(vii) Algorithm to delete a node from the End of a Double Linked list

Step 1 : Start

Step 2 : If the list is empty

2.1 : Return without performing any operation

Step 3 : If the list is not empty

3.1 : Traverse to the end of the list to the secondlast node

3.2 : Set the secondlastnode next pointer to NULL

secondlastnode → next = NULL

3.3 : Delete the lastnode

3.3.1 : Release the memory allocated for the deleted node

Step 4 : End / Stop

(viii) Algorithm to delete a node from the middle of a Double Linked List

Step 1 : Start

Step 2 : If the list is empty

2.1 : Return without performing any operation

2.1: Return without performing any operation

Step3: If the list is not empty

3.1: Traverse the list to the node that is to be deleted; say currnode

3.2: Set the node previous to currnode, say prevnode to point next to the node after the currnode, say nextnode

prevnode -> next = nextnode

3.3: Set nextnode to point previous to prevnode

nextnode -> prev = prevnode

3.4: Delete currnode

3.4.1: Release the memory allocated to currnode

Step4: Stop/End

CIRCULAR SINGLY LINKED LIST

i) Algorithm to insert a node at the beginning of Circular Singly Linked List

Step1: Start

Step2: Create newnode & store the desired data in newnode

Step3: If the list is empty

3.1: Set next pointer of newnode to point to itself

newnode -> next = newnode

3.2: Set newnode as head

Step4: If the list is not empty

4.1: Set the next pointer of newnode to point to the current head

newnode -> next = head

4.2: Traverse the list and set the next pointer of lastnode to point to newnode

lastnode -> next = newnode

4.3: Set newnode as Head

Step5: End/Stop

ii) Algorithm to insert a node at the End of a Single Circular Linked List

Step 5 : End / Stop

(ii) Algorithm to Insert a node at the End of a Single Circular Linked List

Step 1 : Start

Step 2 : Create newnode & store the desired data in newnode

Step 3 : If list is empty

3.1 : Set the next pointer of new node to point to itself

newnode -> next = newnode

3.2 : Set newnode as head

Step 4 : If list is not empty

4.1 : Traverse to the end of the list and set the next pointer of lastnode to newnode

lastnode -> next = newnode

4.2 : Set the next pointer of new node to point to head

newnode -> next = head

Step 5 : Stop / End

(iii) Algorithm to Insert a node in the middle of a Single Circular linked list

-> Same as the Algorithm to Insert a node in the middle of a Single linked list

(iv) Algorithm to print all the data from a Single Circular Linked list

-> Same everytime

(v) Algorithm to search for a particular node in Single Circular Linked list

-> same as well

(vi) Algorithm to delete a node from the Beginning of a Single Circular Linked list

Step 1 : Start

Step 2 : If List is empty

2.1 : Return without performing any operation

Step 3 : If list is not empty

3.1 : Set the next pointer of the lastnode to the secondnode of the list

lastnode -> next = Secondnode

3.2 : Set second node as head

3.3 : Delete the [illegible]

3.2 : Set second node as head

3.3 : Delete the previous head

3.3.1 : Release the memory allocated to previous head

Step4 : Stop/End

(vii) Algorithm to delete a node from the End of the Single Circular Linked List

Step1 : Start

Step2 : If list is empty

2.1 : Return without performing any operation

Step3 : If list is not empty

3.1 : Set next pointer of the secondlastnode to head

Secondlastnode -> next = head

3.2 : Delete lastnode

3.2.1 : Release the memory allocated to lastnode

Step4 : End/Stop

DOUBLE CIRCULAR LINKED LIST

(i) Algorithm to Insert a Node at the Beginning of Double Circular Linked List

Step1 : Start

Step2 : Create newnode & insert desired data in newnode

Step3 : If the list is empty

3.1 : Set the next and previous pointer of newnode to point to itself

newnode -> next = newnode

newnode -> prev = newnode

3.2 : Set newnode as head

Step4 : If the list is not empty

4.1 : Set the next pointer of lastnode to newnode

lastnode -> next = newnode

4.2 : Set the previous pointer of newnode to lastnode

newnode -> prev = lastnode

4.2 : Set the next pointer of newnode to head

newnode -> next = head

4.2 : Set the next pointer of newnode to head
newnode -> next = head
4.4 : Set the previous pointer of head to newnode
head -> prev = newnode
4.5 : Set newnode as head

Step 5 : Stop / End

(ii) Algorithm to Insert a node at the End of Double Circular Linked List

Step 1 : Start
Step 2 : Create newnode & store the desired data in newnode
Step 3 : If the list is empty
3.1 : Set the next and previous pointer of newnode to point to itself
newnode -> next = newnode
newnode -> prev = newnode
3.2 : Set newnode as head
Step 4 : If the list is not Empty
4.1 : Set the next pointer of lastnode to newnode
lastnode -> next = newnode
4.2 : Set the previous pointer of newnode to lastnode
newnode -> prev = lastnode
4.3 : Set the next pointer of newnode to head
newnode -> next = head
4.4 : Set the previous pointer of head to newnode
head -> prev = newnode

Step 5 : Stop / End

(iii) Algorithm to delete a node from the Beginning of a Double Circular Linked list.

Step 1 : Start
Step 2 : If the list is empty
2.1 : Return without performing any operations
Step 3 : If the list is not empty
3.1 : Set nextpointer of lastnode to secondnode

3.1 : Set nextpointer of lastnode to Secondnode
lastnode -> next = Secondnode
3.2 : Set previous pointer of secondnode to lastnode
secondnode -> prev = lastnode
3.3 : Set secondnode as head
Step 4 : Delete previous head
4.1 : Release the memory allocated to the deleted node
Step 5 : Stop/End

(iv) Algorithm to delete a node from the End of a Double Circular Linked list
Step 1 : Start
Step 2 : If the list is empty
2.1 : Return without performing any operations
Step 3 : If the list is not empty
3.1 : Set the next pointer of secondlast node to head
secondlastnode -> next = head
3.2 : Set the previous pointer of head to secondlastnode
head -> prev = secondlastnode
Step 4 : Delete lastnode
4.1 : Release the memory allocated to the deleted node
Step 5 : Stop/End