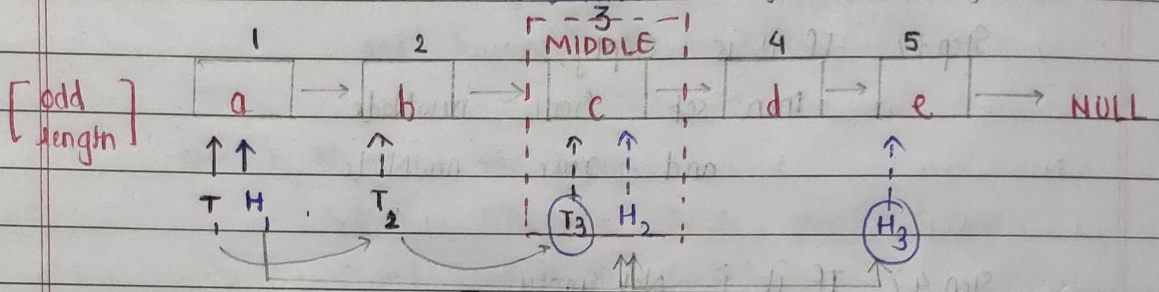
U19CS012

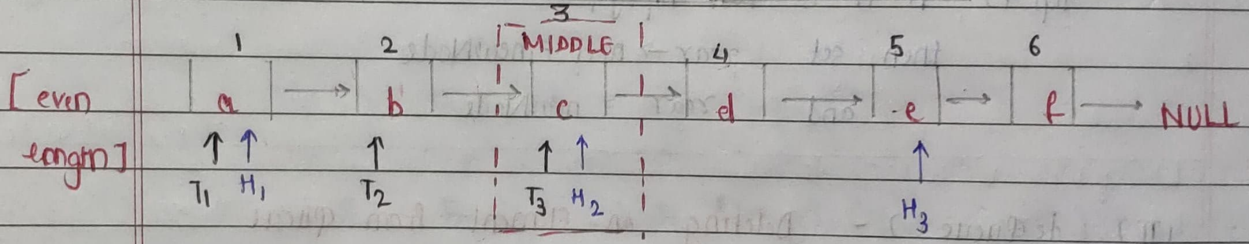
TUTORIAL - 11

CIRCULAR LINKED LIST

1.) Discuss Hare and Tortoise Algorithm to find the middle point of the list

1.) As name suggests (H) hare ← faster Speed (2X)
(T) tortoise ← slower (1X)





(A) Algorithm:

```
(1.) Intialise two pointers (hare and tortoise) both pointing
     to head of linked list
(2) Loop as long hare does not reach null
      (2.1) Set tortoise to next node
      (2.2) Set hare to next of next node
    End
(3.) After the loop ends, the node pointed by tortoise will
     be middle element of linked list as shown in above
     ~~dig~~ diagram.
```

(B) Implementation [C Code] :

```
node * Middle_Element_Using-hare-tortoise (node * head)
{
        node *hare, *tortoise;
        hare = head;                                      } step 1
        tortoise = head;
        while ( tortoise != NULL && hare != NULL )
            {
                  tortoise = tortoise -> next;                      step 2
                  hare = hare -> next -> next;
            }
        return tortoise; // Middle Element of LL              } step 3
}
```

2> Write Algorithm for:

a.) Traversal in Circular Linked List

Step1 > check whether list is Empty (head == NULL)

Step 2> If it is Empty, the display ' Empty List! Can't Traverse' and terminate the function

Step3 > If it is Not Empty, then define a Node pointer 'temp' and initialize with head.

Step4 > Keep displaying temp -> data with an arrow (->) until temp reaches to last node.

Step 5 > Finally display temp -> data with arrow pointing to head -> data.

U19CS012

(b) Circular Linked List Insertion

(b.1) Inserting at Beginning of the List

Step 1> Create a newNode with given value.

Step 2> Check whether list is Empty (head ==NULL)

Step 3> If it is Empty then, set head = Newnode and newnode→next = head;

Step 4> If it is Not Empty then, define a Node pointer 'temp' and initialize with 'head'

Step 5> Keep moving the 'temp' to its next node until it reaches to the last node (until temp→next == head)

Step 6> Set 'newNode→next = head', 'head = newNode' and 'temp→next = head',

(b.2) Inserting at End of List

Step 1, 2 & 3 are same as inserti at Beginning.

Step 4> If it is Not empty then, define a node pointer ('temp') and initialize with head.

Step 5> keep moving the 'temp' to its next node until it reaches to the last node in the list (until temp→next == head)

Step 6 > ~~Ste~~ Set temp → next = newNode and
newNode → next = head

(6.3) Inserting At specific Location in List (After a Node)

Step 1, 2, 3 & 4 are Same as insert at Beginning.

Step 5 > keep moving the temp to its next node until it reaches to the node after which we want to insert the newNode (temp → data is equal to location)

Step 6 > Every time check whether temp is reached to last node or not. If it is reached to last node then display 'Given node is not found in the list! Insertion not possible!' and terminate, otherwise

Step 7 > If temp is reached to exact node after which we want to insert the newNode then check whether it is last node (temp → next == head)

Step 8 > If temp is last node, then set temp → next = newNode and newNode → next = head

Step 9 > If temp is not the last node, then set newNode → next = temp → next
[temp → next = NewNode]

U19CS012

(C) Deletion in Circular Linked List

(C.1) Deletion from Beginning of the List

Step 1> Check whether list is Empty (head == NULL)

Step 2> If it is Empty, then dispay " List is Empty! Deletion not Possible" and terminate this function

Step 3> If it is Not Empty, then define two Node pointers 'temp1' and 'temp 2' and intialize both 'temp1' and 'temp2' with head

Step 4> Check whether list is having only one node (temp1 → next == head)

Step 5> If it is TRUE, then set head = NULL and delete temp1 (Setting Empty list condition)

Step 6> if it is FALSE, move the temp 1 until it reaches to the last node (until temp1 → next == head)

Step 7> Then set head = temp2 → next temp1 → next = head and delete temp 2.

(C.2) Delete from End of the List

Step 1, 2, 3, 4, 5 are same from delete from Begin.

Step 6> if it is FALSE, then set 'temp2 = temp1' and move temp1 to its next node Repeat the same until temp1 reaches to the last node in the list. (until temp1 → next == head)

Step 7> Set temp2 → next = head and delete temp1.

(c.3) Deleting a Specific Node from list

Step 1, 2, 3 Same as Delete from Beginning

Step 4> Keep moving the temp1 until it reaches to the exact node to be deleted or to the last node. And ~~empty~~ every time set 'temp2 = temp1' before moving 'temp1' to its next node.

Step 5> If it is reached to the last node then display 'Given node not found in the list! Deletion not Possible' & (terminate)

Step 6> If it is reached to the exact node, which we want to delete, then check whether list is having only one node (temp1 → next == head)

Step 7> If list has only one node and that is the node to be deleted then set head = NULL and delete temp1 (free(temp1))

Step 8> If list contains multiple nodes then check whether temp1 is the first node in the list (temp1 == head)

Step 9> If temp1 is the first node then set temp2 = head and keep moving temp2 to its next node until temp2 reaches to last node. then set head = head → next, temp2 → next = head and delete temp1.

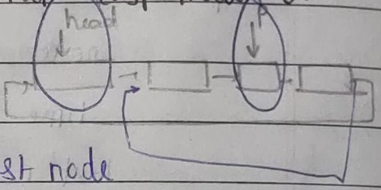
Step 10> If temp1 is not first node then check whether it is last node in the list (temp → next == head)

Step 11> If temp1 is last node then set temp2 → next == head and delete temp1 (free (temp1))

Step 12> If templ is not first node and not last node then set temp2→ next = templ → next and delete templ (free (templ))

3.>

(A) Write an Algorithm to exchange first and last node of circular Linked List



TASK 1 : Find pointer to previous of last node

Step 1> Intialize a pointer 'p' to head
Step 2> Loop / Iterate the list till it reaches to Node previous to Last Node ie (p→ next → next != NULL)

TASK 2 : To exchange first ~~head and~~ and last node using head and p.

```
Step 3 :> p→ next → next = head→ next;
          head → next = p→ next;
          p → next = head;
          head = head → next;
```
Link management

(B) Write an Algorithm to delete every alternate node of circular Linked List

Step 1.> Check if list is empty [or only one element] (head == NULL), terminate & return back

Step 2.> Intialize two Node pointers
* prev = head
* node = head → next

Step 3.> Iterate till (prev != NULL && node != NULL)

```
(3.1) Change next link of previous node
      ~~pt~~ prev→next = node → next

(3.2) Free that node
      free (node);

(3.3) Update prev and Node
      prev = prev → next;
      if ( prev != NULL)
            node = prev → next
```

(c) Split the Circular Linked List

Step 1> Store mid and last pointers of circular linked list using tortoise and hare algorithm.

Step 2.> Make second half ciraular

Step3 > Make first half circular

Step4> Set head (or start) pointers of two linked list

if no. of nodes are odd => [first list will have one extra node]

[Implementation in Code]

x