Lecture :- Linked list 2

Agenda

- Middle element of LL
- Merge sort of LL
- Cycle detection

Qu1 find middle element of linked list.

$1 \longrightarrow 2 \longrightarrow 3 \longrightarrow 4 \longrightarrow 5$ ans = 3

$1 \longrightarrow 2 \longrightarrow 3 \longrightarrow 4.$ ans = 2.

Approach1

$\overset{0}{1} \longrightarrow \overset{1}{2} \longrightarrow \overset{2}{3} \longrightarrow \overset{3}{4} \longrightarrow \overset{4.}{5}$ $len = \frac{0+4}{2} = 2.$

Kth Element of LL [k=2]

$\overset{0}{1} \longrightarrow \overset{1}{2} \longrightarrow \overset{2}{3} \longrightarrow \overset{3}{4.}$ $len = \frac{0+3}{2} = 1$

Kth Element of LL [k=1].

Pseudocode

```
Node findMiddleNode( Node head) {

        if ( head == null) {
            return null;
        }

        Length of LL.
        int cnt = 0;
        temp = head;
        while( temp != null) {
            cnt ++;
            temp = temp.neat;
        }

        Go to $\frac{len}{2}$ - 1 element.

        middleIndex = $\frac{len - 1}{2}$.

        return kthElement (middleElement);
}
```

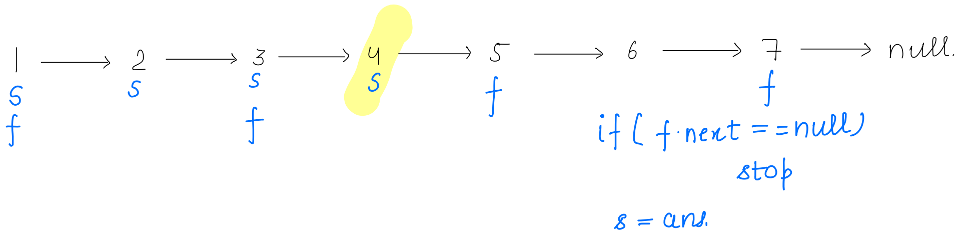
TC: O(n)

SC: O(1)

Approach2 Do this in single pass TC: O(n) SC: O(1)

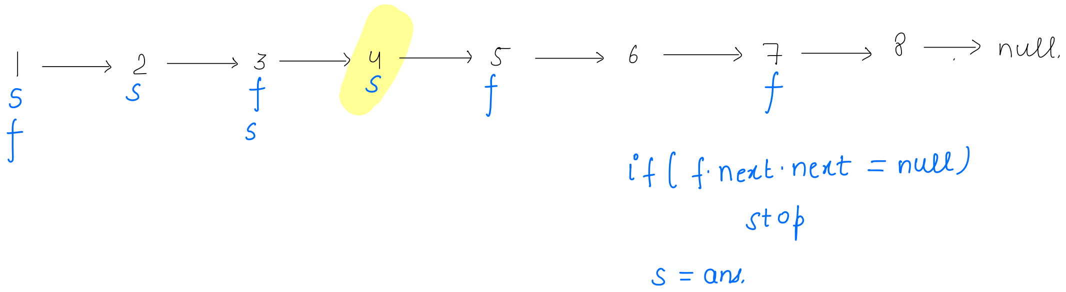
slow = 1 jump

fast = 2 jumps



if (f.next == null)
stop

s = ans.



if (f.next.next = null)
stop

s = ans.

·Pseudocode

```
Node findMiddleNode( Node head)  {
        if (head == null) {
              return null;
        }
        s = head;
        f = head;
        while( f.next != null  &&  f.next.next != null) {
               s =  s.next;
               f =  f.next.next;
        }
        return s;
}
```

Qu2 Given 2 sorted linked lists, merge them into a single sorted linked list.

ip fir = 1 → 2 → 8 → 10

sec = 3 → 5 → 9 → 11

ans: 1 → 2 → 3 → 5 → 8 → 9 → 10 → 11.

fir = 1 → 7 → 8 → 9

sec = 2 → 5 → 10 → 11

ans: 1 → 2 → 5 → 7 → 8 → 9 → 10 → 11.

Arrays approach

fir =

1	7	8	9	
i	i	i	i	i

sec =

2	5	10	11
j	j	j	

1	2	5	7	8	9	10	11

Approach

fir = 1 → 7 → 8 → 9
h1 h1 h1

sec = 2 → 5 → 10 → 11
h2. h2 h2

head = ~~null~~ 1

tail = ~~null~~ 1 → 2. → 5 → 7
↑

h1. data < h2. data [head = h1 tail = h1] h1 = h1.next.
(1) (2)

h1. data > h2. data [tail. next = h2] h2 = h2.next tail = tail. next
(7) (2) 1 → 2

h1. data > h2. data [tail. next = h2] tail = tail. next h2 = h2 . next
(7) (5) 2 → 5

h1. data < h2. data tail . { tail. next = h1 } tail = tail. next h1 = h1. next
7 10 5 5 → 7

h1. data < h2. data tail . { tail. next = h1 } tail = tail. next h1 = h1. next
8 10 7 7 → 8

Pseudocode

```
Node mergeTwoSortedLinkedLists ( Node head1, Node head2) {
        if ( head1 == null && head2 == null) {
            return null;
        }
        if ( head1 == null) {
            return head2;
        }
        if ( h2 ==null) {
            return h1;
        }
        head = null;
        tail = null;
        if ( h1.data <=h2.data) {
            head =h1;
            tail =h1;
            h1= h1.next;
        } else {
            head= h2;
            tail =h2;
            h2 = h2.next;
        }
        while( h1 != null && h2 !=null) {
            if ( h1.data <= h2.data) {
                tail.next = h1;
                tail = tail.next;
                h1= h1.next;
            } else{
                tail.next =h2;
                tail= tail.next;
                h2 = h2.next;
            }
        }
        if ( h1 !=null) {
            tail.next = h1;
        }
        if (h2! =null) {
            tail.next = h2;
        }
    return head;
}
```

Qu3 Sort a linked list using merge sort.

1 ⟶ 2 ⟶ 5 ⟶ 4 ⟶ 3

ans:

1 ⟶ 4 ⟶ 3 ⟶ 2

ans:

Approach and steps

1. Base case. { head == null || head.next == null ⟶ head }
2. find middle node.
3. Split the list into two parts.
4. Recursively sort both parts.
5. Merge 2 sorted halves.

Pseudocode

```
Node mergesort( Node head) {
    if (head ==null  ||  head. next == null) {
            return  head;
    }
    middle = find MiddleNode (head);

    // split LL in 2 halves.

    h1 = head ;
    h2 = middle. next;
    middle. next = null;


    left = merge sort(h1);
    right = mergesort(h2);
    return  merge Two Sorted Linked Lists (left, right);
}
```

1 ⟶ 3 ⟶ 8 ⟶ 2 ⟶ 5 ⟶ 7 ⟶ 6 ⟶ null

↑ head. ↑ middle.

h1 = 1 ⟶ 3 ⟶ 8 ⟶ 2 ~~⟶ 5 ⟶ 7 ⟶ 6~~ ⟶ null

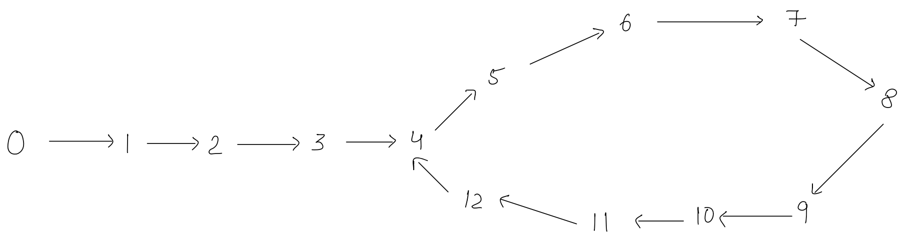
h2 = 5 ⟶ 7 ⟶ 6

TC: O(n logn)

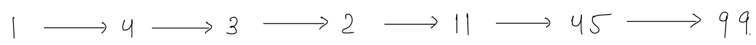
SC: O(logn).

Break: 8:17 – 8:28

Qu. Given a linked list, find whether it contains a cycle

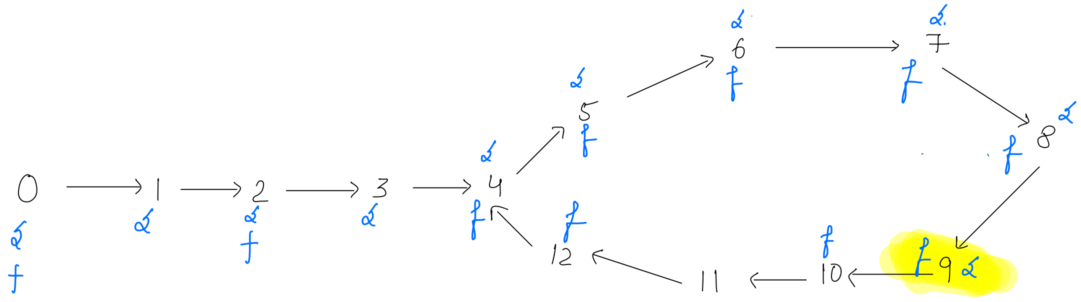


true

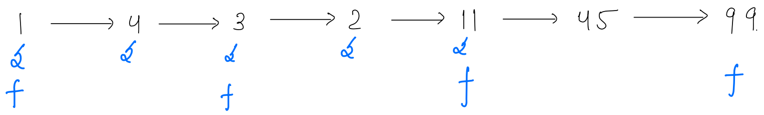


false

Approach



s == f [cycle]



f == null || f.next == null

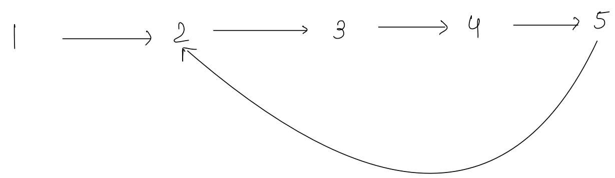
No cycle.

Pseudocode

```
boolean hascycle( Node head) {
        if ( head == null || head.next == null) {
              retum false;
        }
        s = head;
        f = head;
        while( f != null  &&  f.next != null) {
              s = s.next;
              f = f.next.next;
              if( s == f) {
                  return true;
              }
        }
     retum false;
}
```

TC: O(n)
SC: O(1)

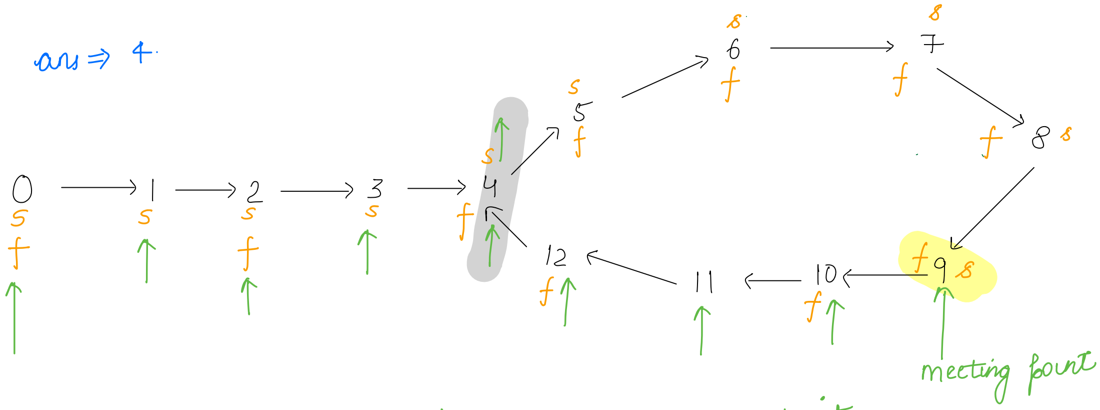
<u>Qu</u> Given a linked list which contains a cycle, find start point of the cycle.

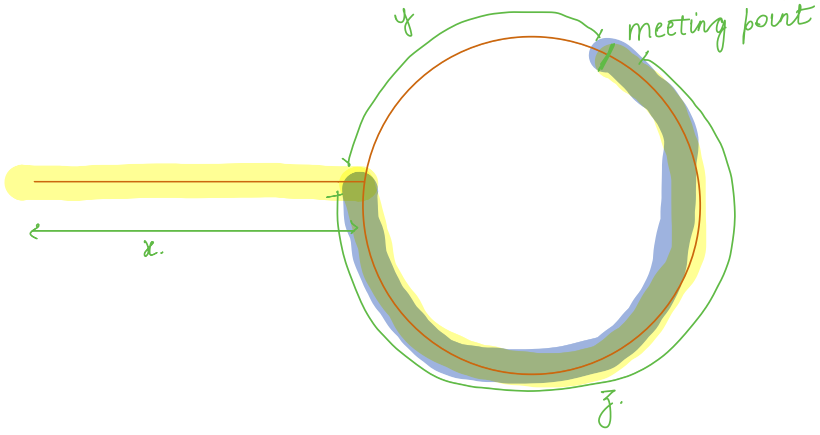


ans = 2.

Approach

ans ⇒ 4.





Distance travelled by :

slow ⇒ $x + y$.

fast ⇒ $x + y + z + y$.

fast pointer = double speed of slow pointer

$$d_f = 2 * d_s$$

$$x + y + z + y = 2(x + y)$$

$$x + y + z + y = 2x + 2y$$

$$x + z + 2\cancel{y} = 2x + 2\cancel{y}$$

$$2x - x = z$$

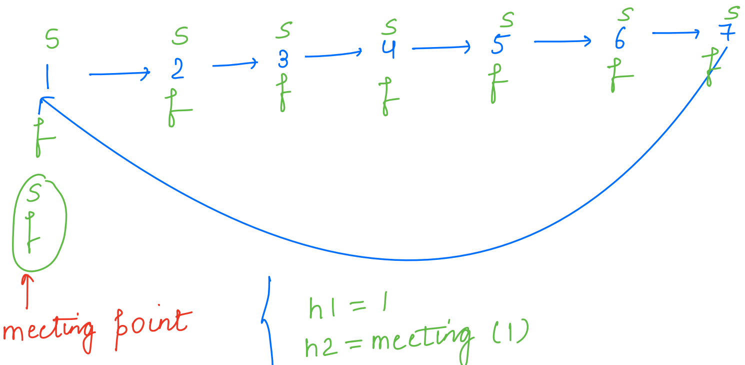
$$\boxed{x = z}$$

Pseudocode

```
Node getStartingPoint( Node head) {
    if (head == null || head.next == null) {
        return null;
    }
    s = h;
    f = h;
    hascycle = false;
    while( f != null && f.next != null) {
        s = s.next;
        f = f. next. next;
        if( s == f) {
            hascycle = true;
            break;
        }
    }
    if( ! hascycle) {
        return null;
    }
    h1 = head;
    h2 = s;
    while ( h1 != h2) {
        h1 = h1.next)
        h2 = h2.next;
    }
    return h1;
}
```

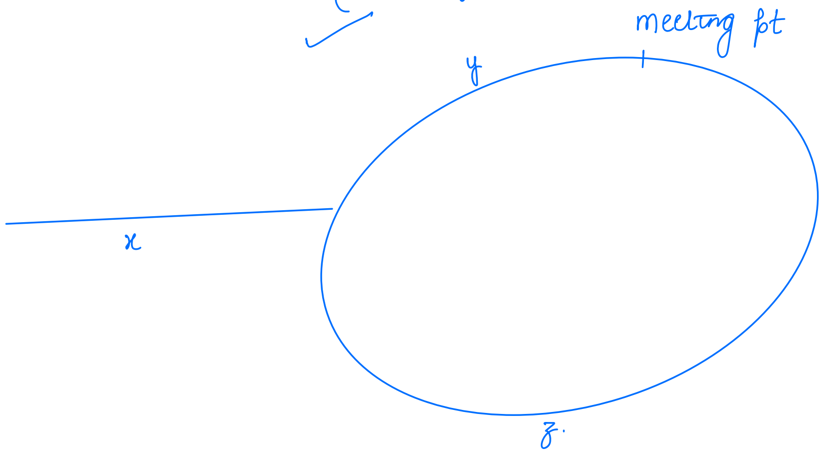
Thanks ☺

Doubts



starting = 1.

h1 = 1
h2 = meeting (1)
starting = h1



$$0 + y + z + y = 2(0 + y)$$

$$\cancel{2y} + z = \cancel{2y}$$

$$\boxed{z = 0}$$