Fast and Slow

Pointers Pattern

slow pointer

fast pointer

Handwritten Notes

by **Dipti Verma**

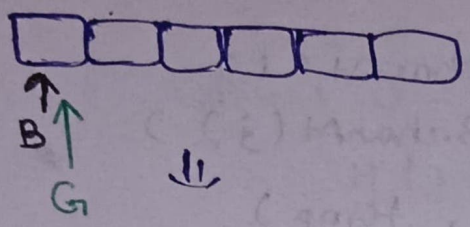
learned from **CTO Bhaiya**

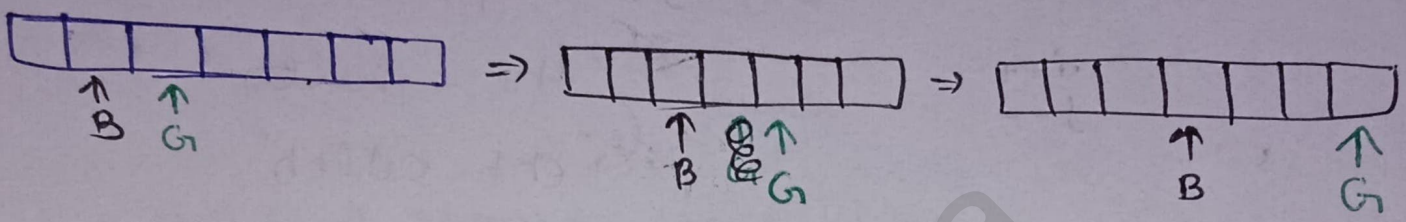
Day-7

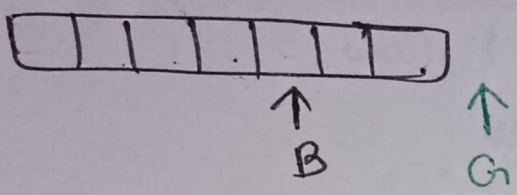
Fast & Slow Pointers Pattern

B = Boy = step 1
G = Girl = step 2

Eg



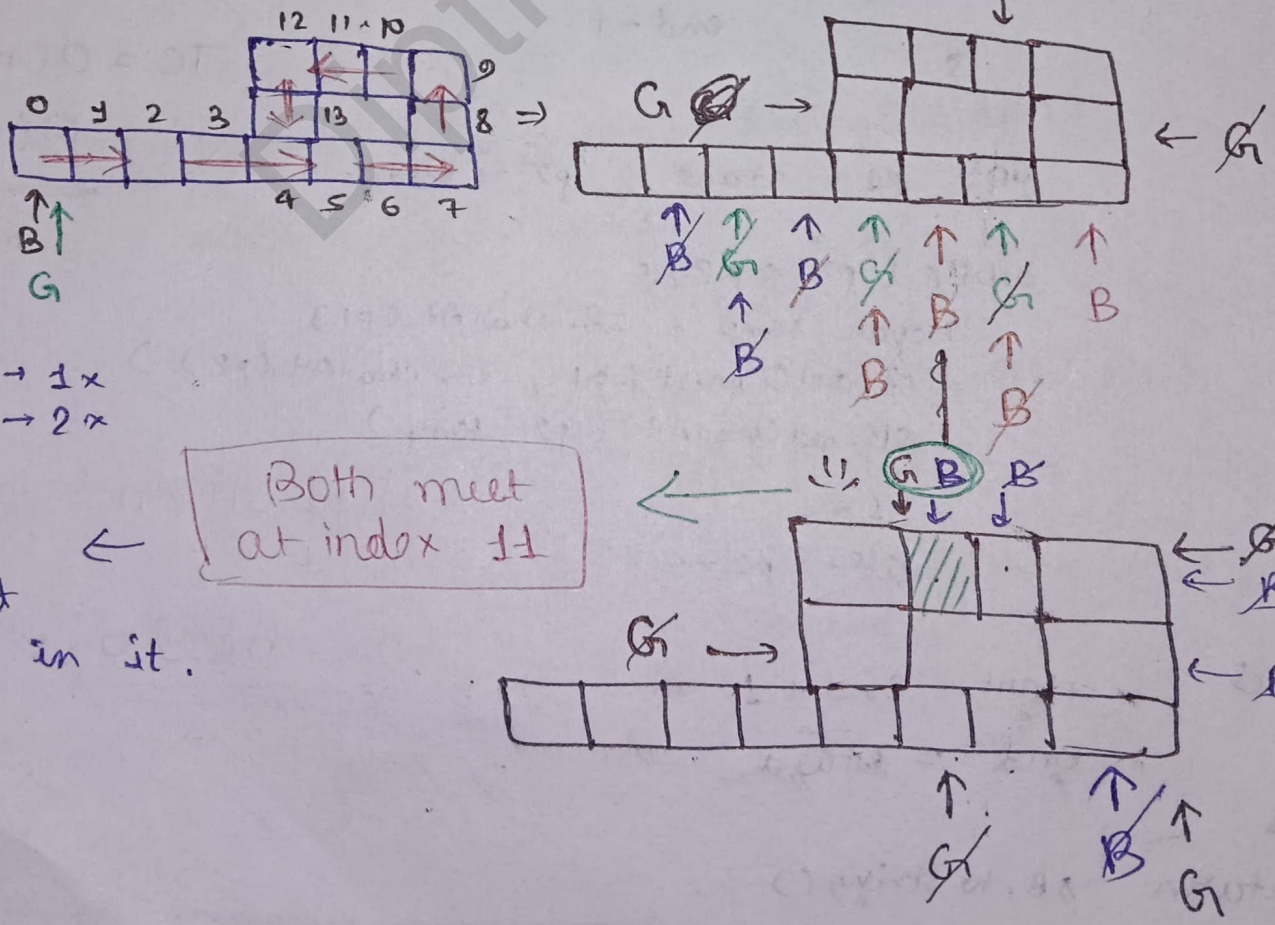




- Linear data structure (array, linked list)

* It mainly helps in detection of cycle and intersection in linked list.

Eg



B → 1x
G → 2x

Both meet at index 11

It detect cycle in it.

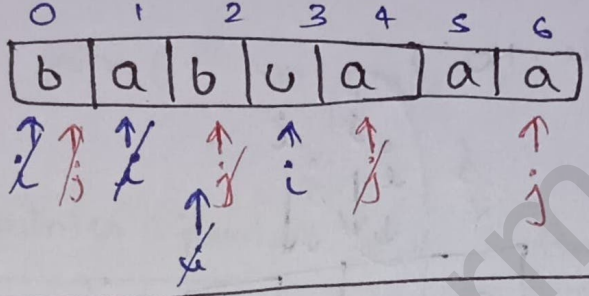
Jyoti Verma

Fast-slow pointer pattern

→ A technique where two variables (pointers) traverse a data structure (like an array, string or linked list) simultaneously from same position.

*(2 pointers start from same / diff position)

→ The slow pointer moves step-by-step, while fast pointer moves faster (usually twice as fast)



Look when j reached at last index
→ i always at mid of the array

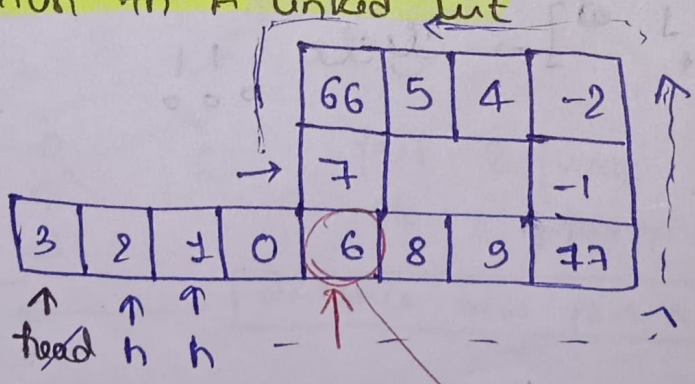
i at 3
j at 6

Useful for analysing the structure of linear data
Commonly used to
→ detect cycles
→ find mid-points
→ determine intersections

0. Cycle Detection in A Linked List

Brute Force →

- Store head → data in set.
- If head ka data, already set mein hai, toh → (cycle)



Set
3, 2, 1, 0,
6, 8, 9, 77,
-1, -2, 4, 5,
66, 7

TC = O(n)
SC = O(n)

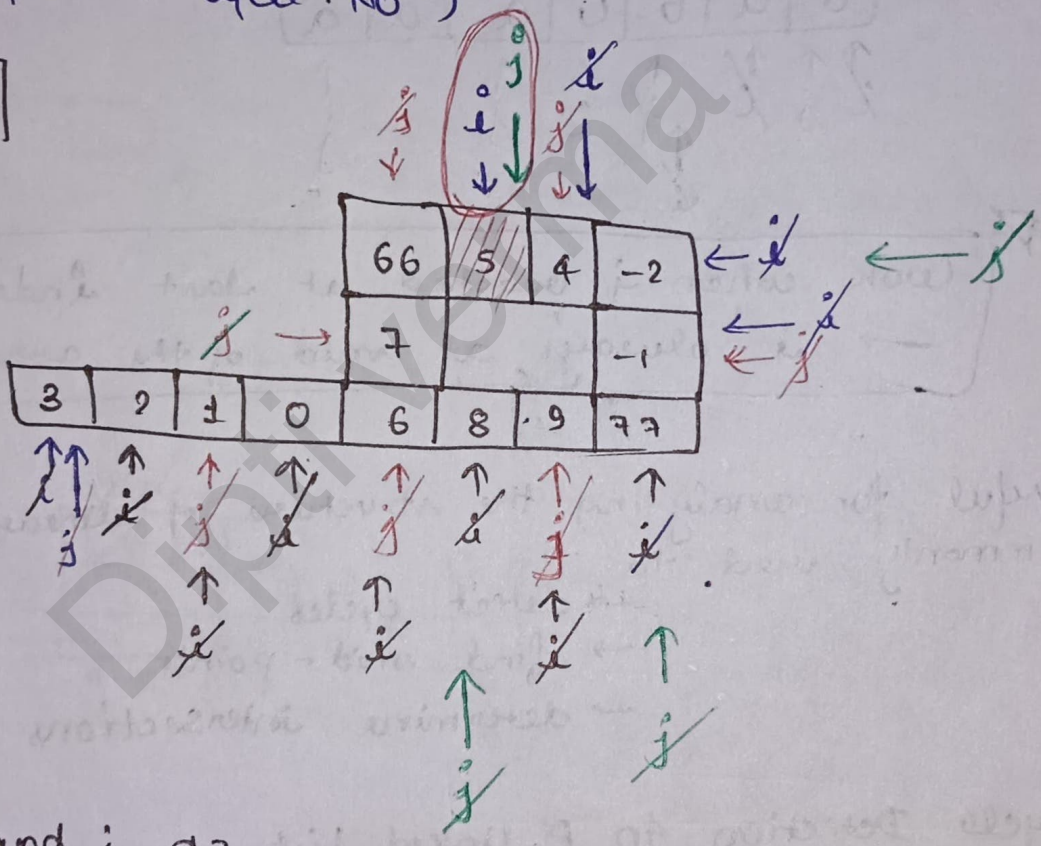
⇒ Cycle.

Dipti Verma

Code:

```
ret = { }

while ( head != null ):

    if ret.has ( head ):
            print ( " Cycle " )
            break
    ret.add ( head )
    head = head.next

print ( " Cycle . NO " )
```

Optimized

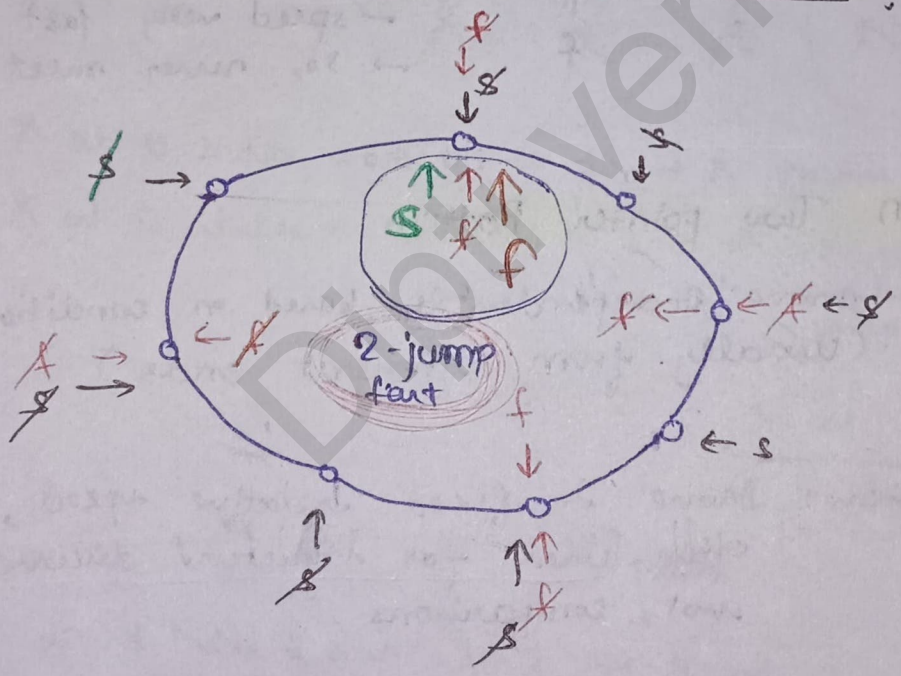


→ i and j @] → cycle !!!
⇒ meet

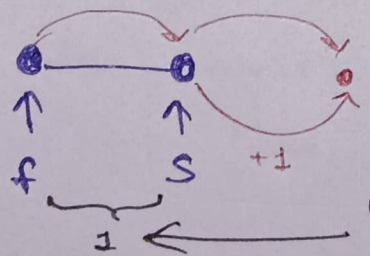
Code:

```
slow = head
fast = head
while ( fast != null && fast.next != null):
    slow = slow.next
    fast = fast.next.next

    if( fast != null && slow == fast):
        print ( "Cycle")
        break

print ( "No Cycle")
```

Why fast pointer jumps only 2 times ?



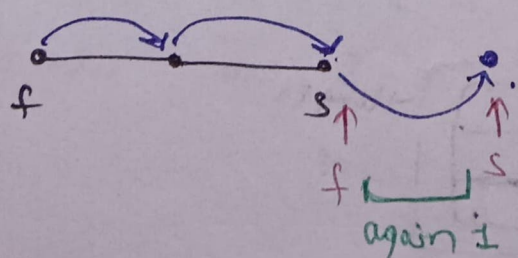
Eg.



fast = 2 jump
slow = 1 jump } meet

distance b/w fast & slow = 1

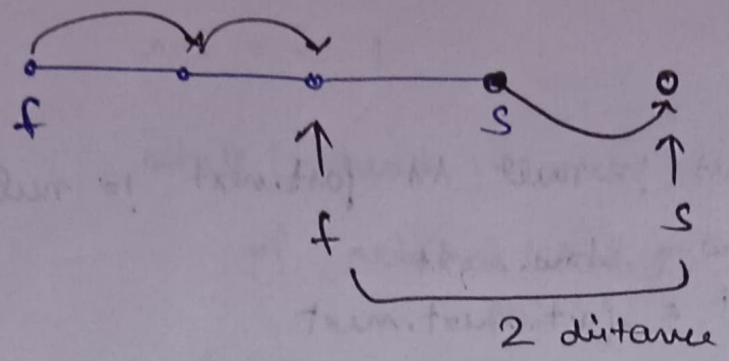
Eg



so → always meet !!!

Divi verma

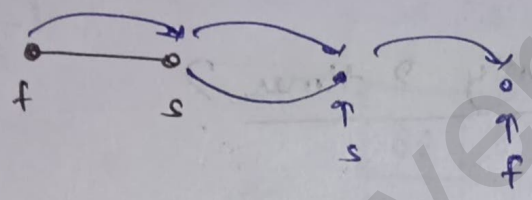
Eg.



✱ Previous, we proof, distance b/w s & f = 2
→ always meet.

Now, jump fast = 3
jump slow = 1

Eg. 1.



(fast cross the slow)
→ speed very fast ...
→ so, never meet.

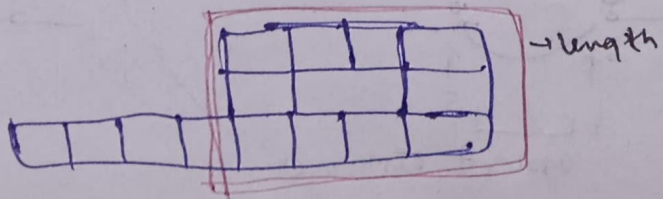
✱ Difference from Two pointer Pattern

→ Two pointers: move independently based on conditions (usually from different ends)

→ Fast & slow pointers: Move in fixed relative speed, often used for structure discovery, not comparisons.

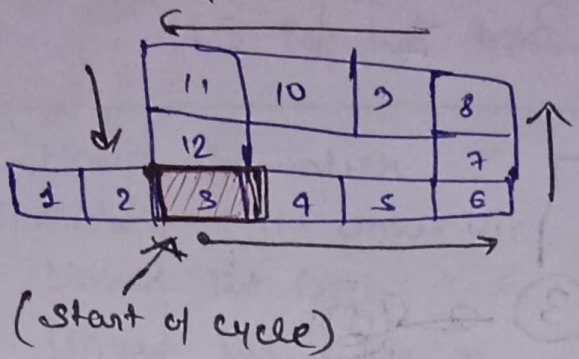
✱ Application

→ Cycle Detection
→ Mid-point finder
→ Finding cycle length



Dipti verma

→ Start of the cycle

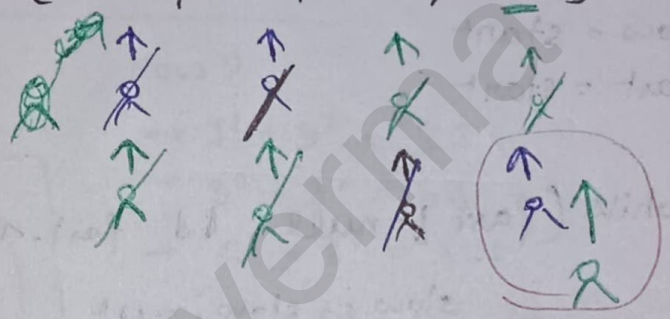


Cycle detection in Arrays

Cycle

arr[i] = tell the index to move

arr = [1, 2, 3, 1]
0 1 2 3



= only 1 step

= only 2 step

meet cycle

- at 0 index = arr[0] = 1 → move at index 1

  at 0 index = arr[0] = 1 → at 1 index then, (step 1)

  arr[1] = 2 → at index 2 (step 2)

  ∴ (fast) at 2nd index

- at 1 index = arr[1] = 2 → at 2

  at 2 index = arr[2] = 3 → I: at 3 index

  Now, arr[3] = 1 → II: at 1 index

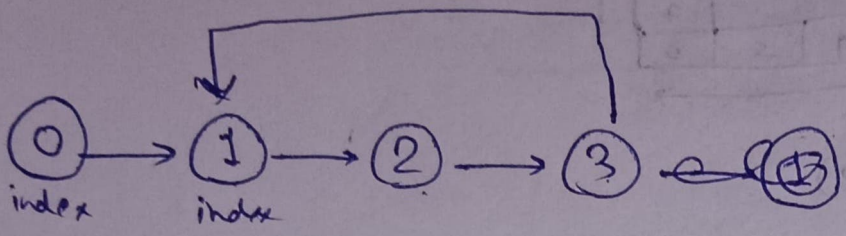
- at 2 index = arr[2] = 3 → at 3 index

  at 1 index = arr[1] = 2 → I: at 2 index

  Now, arr[2] = 3 → II: at 3 index

Dipti Verma

arr = [1, 2, 3, 1]
(0) (1) (2) (3)



⇒ Fast & slow pointer template

```
slow = start
fast = start

while (fast != null && fast.next != null):
    slow = slow.next
    fast = fast.next.next

    if (fast != null && slow == fast):
        # they meet again

return()
```

Diptiverma

Day - 8

Fast & Slow Pointer Pattern

(5 Popular Interview Questions)

202. Happy Number	(LC)	Easy
876. Middle of the Linked List	(LC)	Easy
141. Linked List Cycle	(LC)	Easy
142. Linked List Cycle II	(LC)	Medium
Find length of Loop	(GFG)	Medium

1. Happy Number

Eg n = 19

O/P → True

How?

→ $1^2 + 9^2 = 82$

→ $8^2 + 9^2 = 68$

→ $6^2 + 8^2 = 100$

→ $1^2 + 0^2 + 0^2 = 1$ ⇒ True

Eg n = 2

O/P → False

$2^2 = 4$

$4^2 = 16$

$1^2 + 6^2 = 37$

$3^2 + 7^2 = 9 + 49 = 58$

$5^2 + 8^2 = 25 + 64 = 89$

$8^2 + 9^2 = 64 + 81 = 145$

$1^2 + 4^2 + 5^2 = 42$

$4^2 + 2^2 = 20$

$2^2 + 0^2 = 4$ (again 4)

Brute force

20 ← 42 ←

⇒ 2 → 4 → 16 → 37 → 58 → 89 → 145

(Cycle !!!)

- store it in set data structure

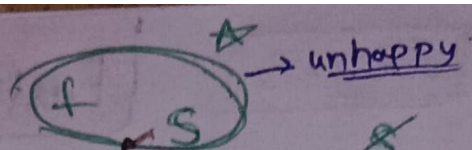
{ 2, 4, 16, 37, 58, 89, 145, 42, 20 }

- then, (4) at set

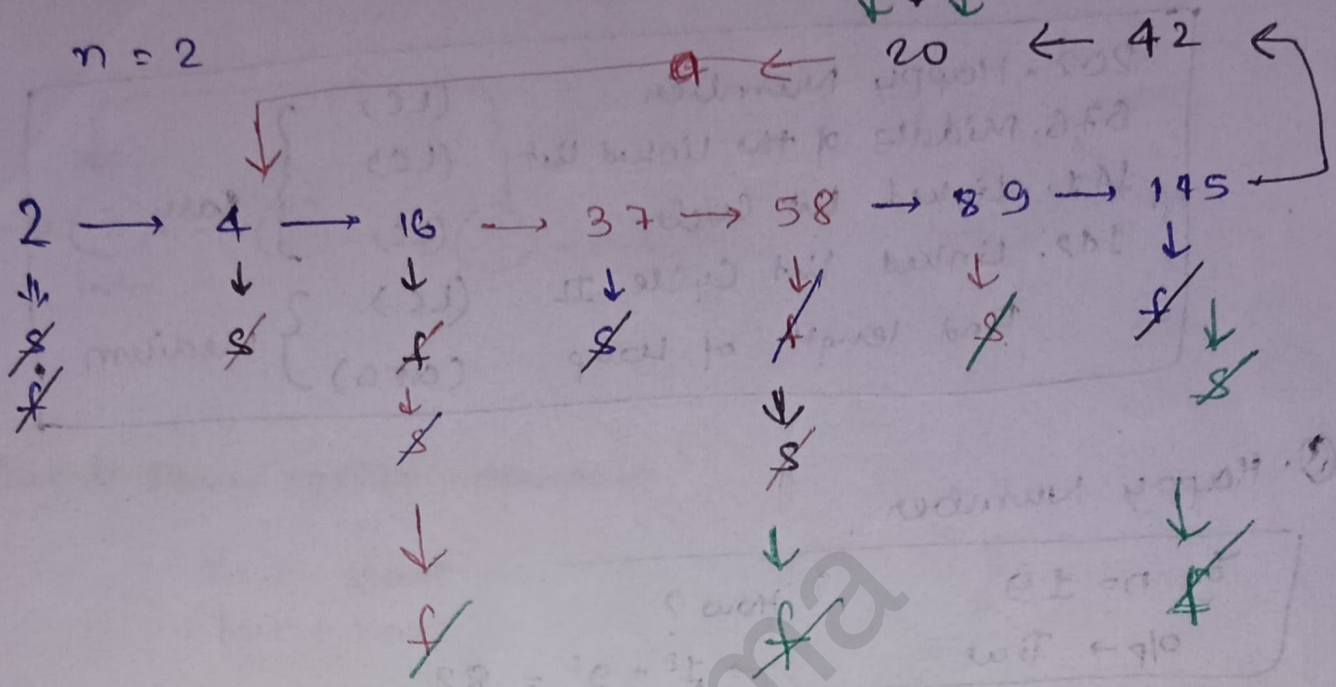
Fake

Dipti Verma

[Solve by fast-slow pointer]



Eg n = 2



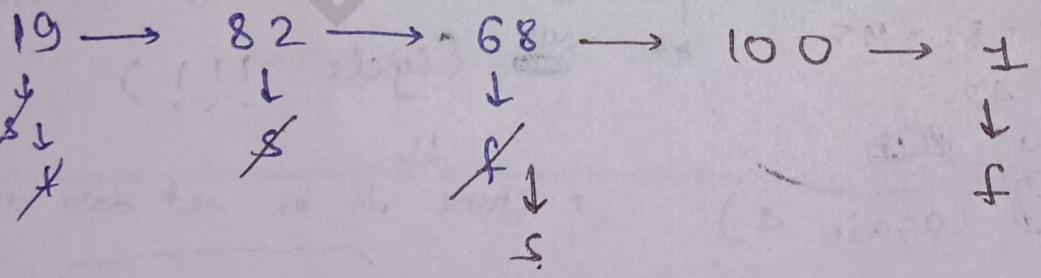
Eg. n = 19

s = ~~19~~ ~~82~~ 68 ...
f = ~~19~~ ~~68~~ 1

If fast == 1

↓

True



Dipti Verma

Code :

```
class Solution {
    public int sumOfSquare( int n){
        int sum = 0
        while ( n > 0 ){
            int dig = n % 10
            sum = sum + ( dig x dig)
            n = n / 10
        }
        return sum
    }

    public boolean isHappy ( int n ){
        int slow = n , fast = n
        while ( fast -> 0 x fast != 1 ){
            slow = sumOfSquare (slow)
            fast = sumOfSquare ( sumOfSquare ( fast) )
            if ( fast == 1 || slow == 1 ){
                return true
            }
            if ( slow == fast && fast != 1 ){
                return false
            }
        }
        return true
    }
}
```

if n = 1
→ True

why ? changes

eg n = 10

10 → 1 → 1
s f

s == f => 1 → True

Dipti Verma

⇒ Interview ask ?

n = 19 cases → ① 1
② Cycle detect
③ 19 → 82 → 5000 → . . . . big big

You have to proof that it has only ① & ② cases :

Given constraint of question ⇒ $1 <= n <= 2^{31} - 1$

↓

2,147,483,647

10 digits

largest 10 digits = 9,999,999,999

↓ square of digits sum

810 ⟶ 3 digits

↓

65

↓

61

↓

⋮

largest 3 digits = 999 ⟶ 243

Mean, koi bhi number lelo 999 se UO 243 se chota hi hoga or equal.

i.e jo 'n' hoga ⊙ yah toh loop mein rahega

1 ↔ 243

⊙ yah toh '1' aa jayega.

Dipti Verma

TC = ?

d → digit

n → work → sumOfDigitSquare(n)

→ loop

$n1 \rightarrow \frac{9,999,999,999}{(81) \times 10}$

$n1 \longrightarrow d$

$f(n1(d)) \rightarrow 81d$

$n1 \rightarrow 81(\log n1)$

$\because f(n1) \leq 81 \cdot (\log n1)$

$n1 \rightarrow n2 \rightarrow n3 \ldots$

$f(n2) \leq 81(\log n2)$

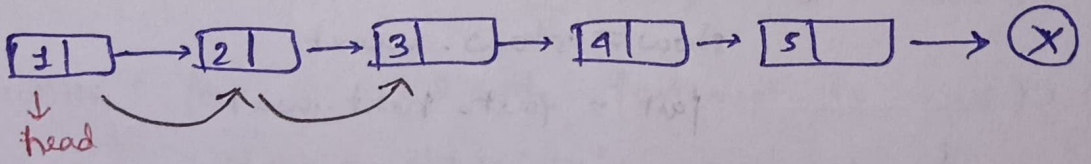
work done: $81 \cdot \log n1 + 81 \cdot \log(81 \log n1) + \ldots$

$\log n + \log(\log n) + \log(\log(\log n)) + \ldots$

$\therefore TC = O(\log n)$ and $SC = O(1)$

Q. Middle of the linked list

→ If 2 middle node ⇒ return 2nd middle node.

Eg. 

length = 5 → $\frac{5}{2} = 2.5 \Rightarrow 2$ ⇒ jump = 2

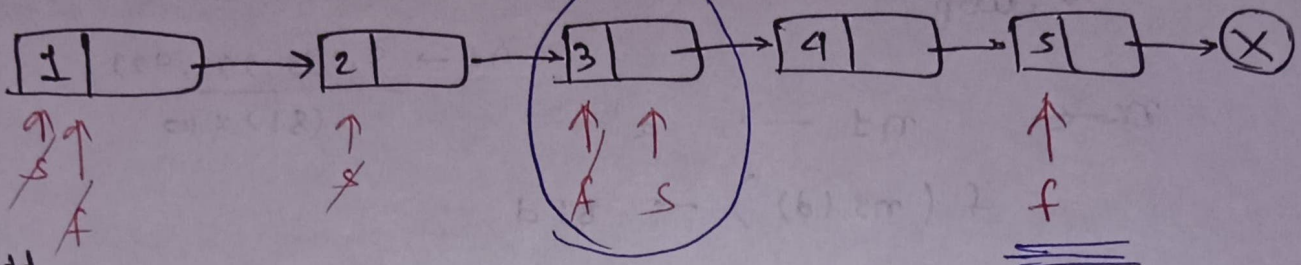
So, 3 = middle node

But in this approach, we use 2 iteration

Dipti verma

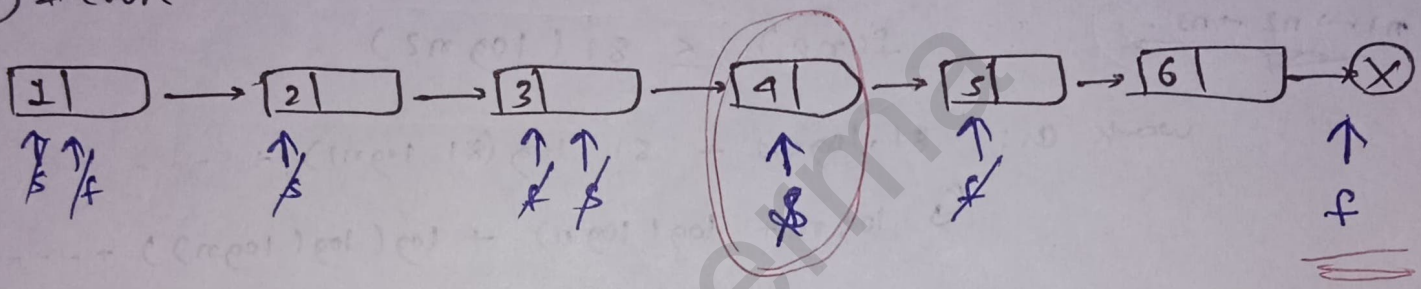
Optimized

Eg



odd

Eg # even



Code :

```
public ListNode middleNode(ListNode head) {
    ListNode slow = head, fast = head;
    while (fast != null && fast.next != null) {
        slow = slow.next
        fast = fast.next.next
    }
    return slow
}
```

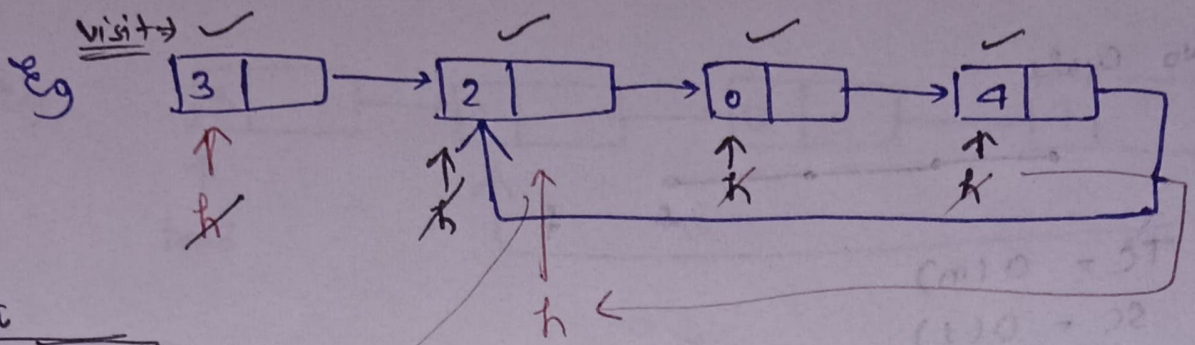
TC = O(n)

SC = O(1)

Dipti verma

Q. 141 → Linked List Cycle

Brute Force



Set

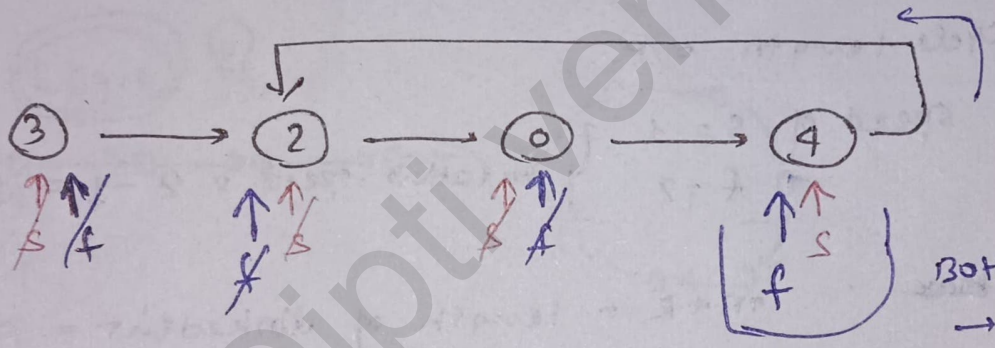
③, ②, ⓪
④,

2 → visited already → loop → True

∴ TC = O(n)
SC = O(n)

Optimized

Eg



Both meet
→ cycle
→ True.

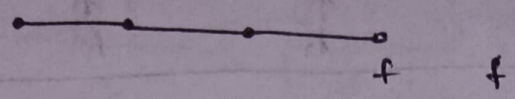
Code:

```
ListNode slow = head, fast = head;

while ( fast != null && fast.next != null ){
    slow = slow.next
    fast = fast.next.next

    if ( slow == fast){
        return true
    }
}
return false
}
```

Diptiverma

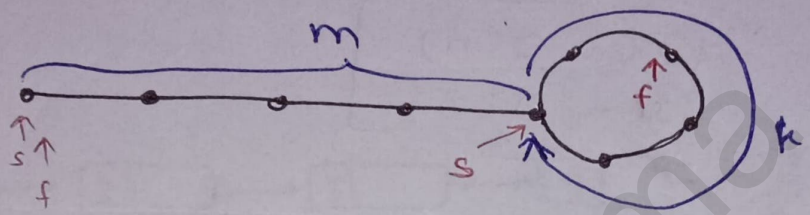
Time Complexity:

① No Cycle



TC = O(n)
SC = O(1)

② Cycle



TC = O(n)
SC = O(1)

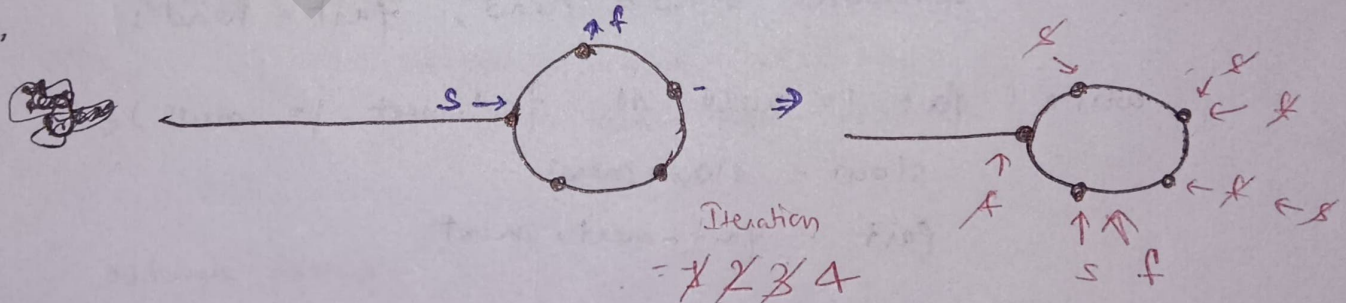
cycle length = K

speed of s = 1
f = 2
} relative speed = 2 - 1 = 1

m + k = length of linkedlist = n

TC = O(n)

Eg.



Iteration = 1 2 3 4

Node = 5 in loop (a)

Iteration = 5 - 1 = 4 (a - 1)

Total time = m + k + 1

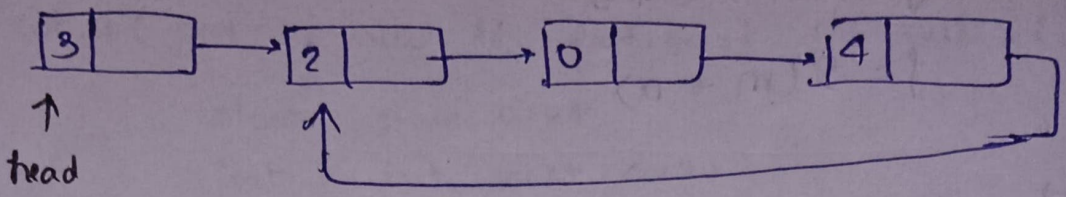
m + k = n

∴ TC = O(n)

Dipti Verma

142. Linked List Cycle II (Medium)

Eg,

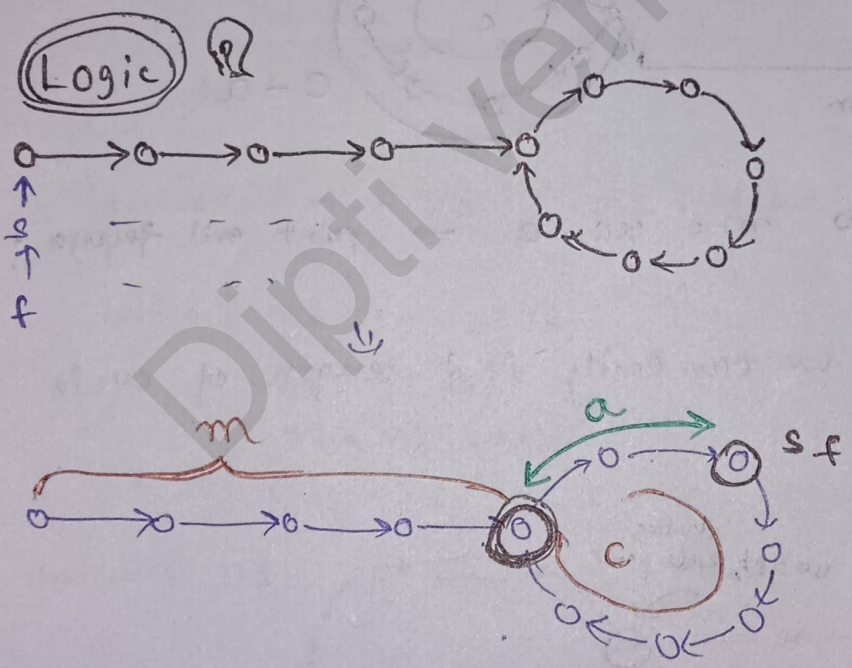


- return the node where the cycle begins.
- If there is no cycle → return null.

I/p → head = [3, 2, 0, -4] , pos = 1

↑ indicate the index of node that tail's next pointer is connected to.

Think Logic



⇒ Slow moves = $m + a$

⇒ fast moves = $m + x \cdot c + a$

a → n times cover cycle (c)

⇒ fast always run 2x slow

So, fast → $2(m + a)$

Dipti Verma

We have,

$$f = m + x \cdot c + a$$

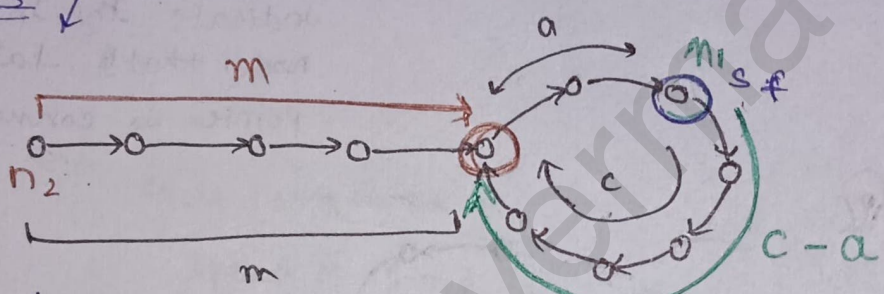
$$f = 2(m + a)$$

$$\Rightarrow \quad m + x \cdot c + a = 2m + 2a$$

$$\Rightarrow \quad xc + e = m + a$$

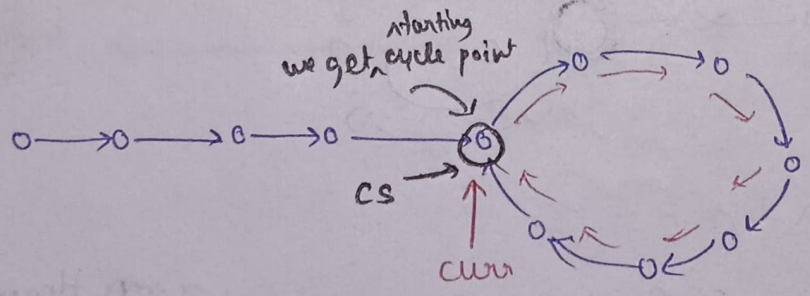
$$\Rightarrow \quad \boxed{m = xc - a}$$

means



Jab $m = xc - a$ → point mil jayega !!!

from this we can easily find length of cycle



curr.next - - - -

traverse current until curr = cs

→ we get length of cycle.

Dipti verma

TC - O(n)
SC - O(1)

Code :

```
ListNode slow = head, fast = head
while ( fast != null && fast.next != null ){
    slow = slow.next
    fast = fast.next.next

    if (slow == fast){
        // cycle
        break
    }
}

if ( fast == null || fast.next == null ){
    // no cycle
    return null
}

ListNode n1 = slow, n2 = head

while ( n1 != n2 ){
    n1 = n1.next
    n2 = n2.next
}

return n1
```

Dipti verma

Q- Find length of the loop (GFG)

Code :

```
Node slow = head , fast = head

while ( fast != null && fast.next != null ){
        slow = slow.next
        fast = fast.next.next

        if ( slow == fast){
            // cycle
            break
        }
}

if ( fast == null || fast.next == null){
        // no cycle
        return 0
}

Node n1 = head , n2 = slow ;
while ( n1 != n2){
        n1 = n1.next
        n2 = n2.next
}

len = 1

Node curr = n1.next
while ( curr != n1){
        len = len+1
        curr = curr.next
}
return len
```

Dipti verma

Day - 9

Fast & Slow Pointers Pattern

(Medium Questions)

Split a LL into two halves (GFG)
287. Find the Duplicate Number
234. Palindrome Linked List (easy)

0 - Split a LL into two halves (GFG)

→ Circular LL

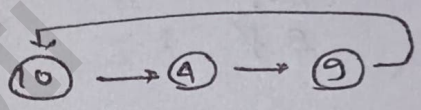
→ Last node connected to first node

→ Task : split into 2 Circular LL

→ If odd no. of nodes ⇒ First list should have one node more than second list.

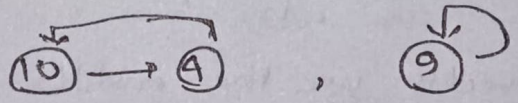
eg. i/p = 10 → 4 → 9

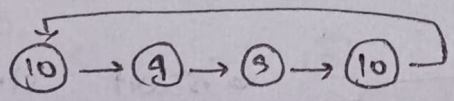
mean,

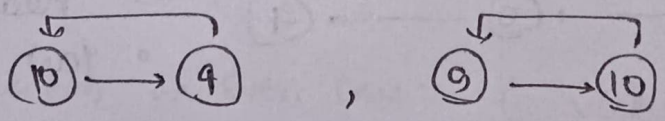


o/p = 10 → 4 , 9

mean,



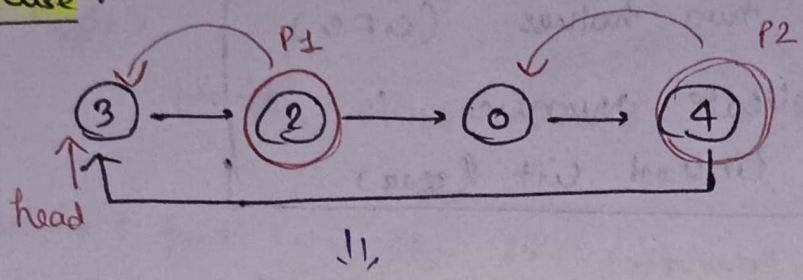
eg, 

o/p 

Dipti Verma

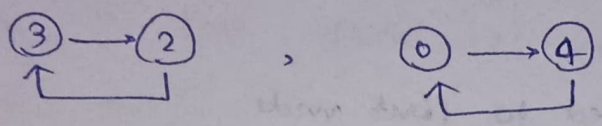
Logic:

Even Case:

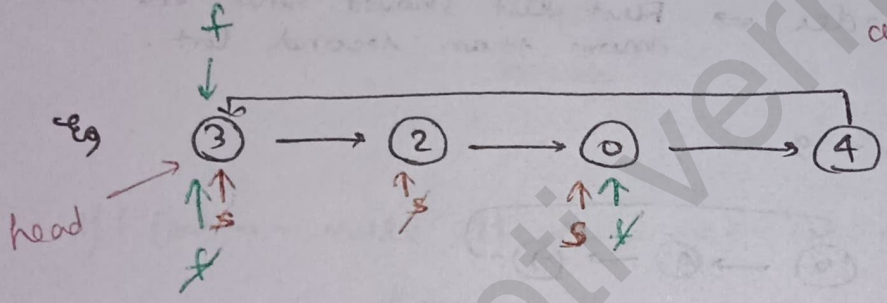


P1 Point to head
P2 Point to P1.next

→ we have to find mid of LL.



Use the concept of middle of LL using fast & slow pointers. we get p1 & p2 points and then connect them.

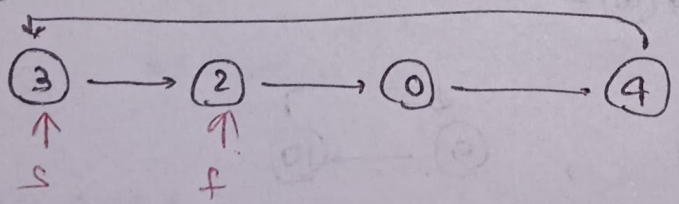


Observe: slow at (0), fast at (3) | But we want slow at (2) & fast at (4)

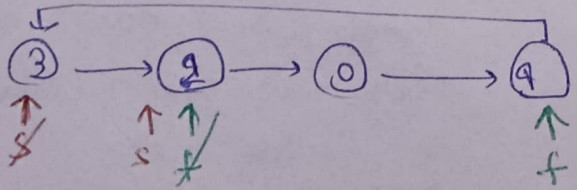
So, we don't directly use the middle of LL concept, we need to little modification in it !!!

Approach

I.



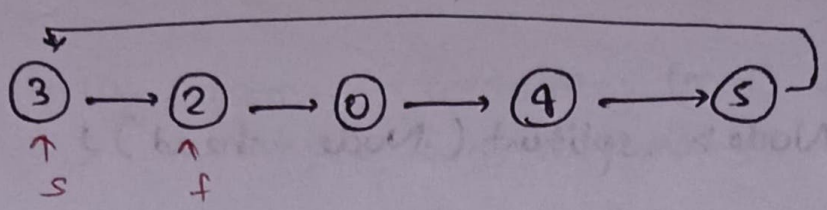
- start slow from head
- fast from head.next

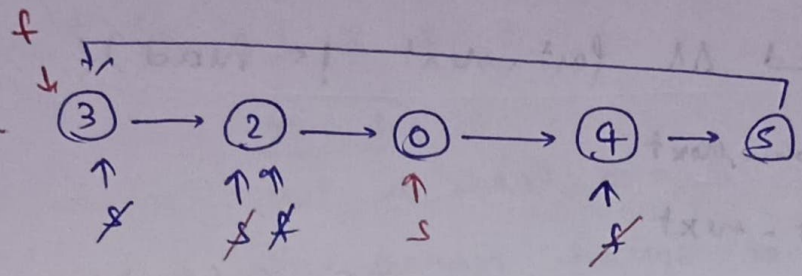


✓ (In this case, we get slow & fast at correct position.)

Dipti Verma

An odd case

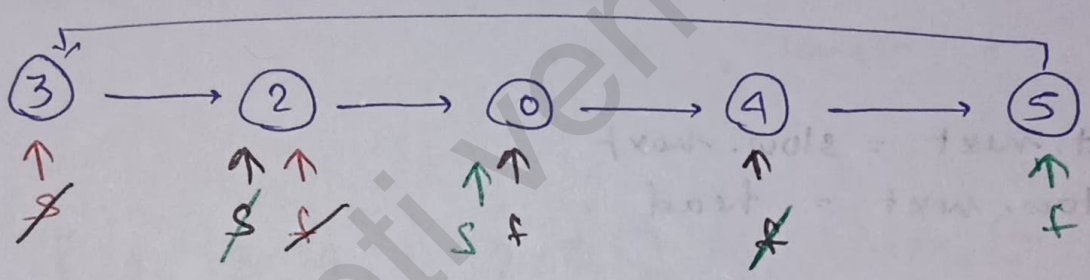




again problem,
slow at ⓪
fast at ③

We do
I condition : fast moves 1 by 1 and check
fast.next != head

See :

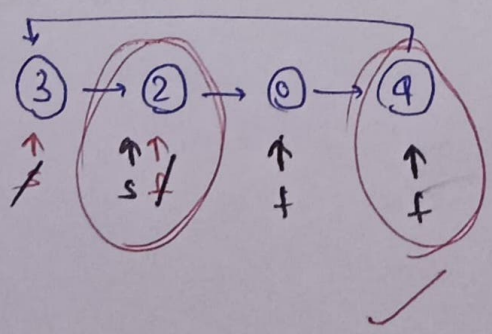


fast ⓕ at ②
ⓕ at ⓪ → move next one ⓕ at ④
ⓕ at ⑤ → now next ⇒ X bcz. fast.next = head.

stop

Now,
slow at ⓪
fast at ⑤

Check this condition in Even case: (Valid)



- f at ②
- f at ⓪ → next ④

stop

- bcz, 4.next = head

Dipti Verma

Code:

```
class Solution {

    public Pair<Node, Node> splitList(Node head){

        Node slow = head, fast = head.next

    while ( fast != head && fast.next != head){
            slow = slow.next
            fast = fast.next

            if ( fast.next != head){
                fast = fast.next
            }
    }

    fast.next = slow.next
     slow.next = head

    return new Pair<Node, Node> ( head, fast.next)

    }

}
```

TC = O(n)
SC = O(1)

Dipti verma

Q. (287) Find Duplicate Number

- Array : nums → containing (n+1) integers where, each integer is in the range [1, n] inclusive.

Mthd:

Eg, $\overset{0}{1}\ \overset{1}{3}\ \overset{2}{4}\ \overset{3}{2}\ \overset{4}{2}$

n = 4

But in real n = 5

and array range ⇒ [1, n] → [1, 4]

But length = 5

So, one number is duplicate

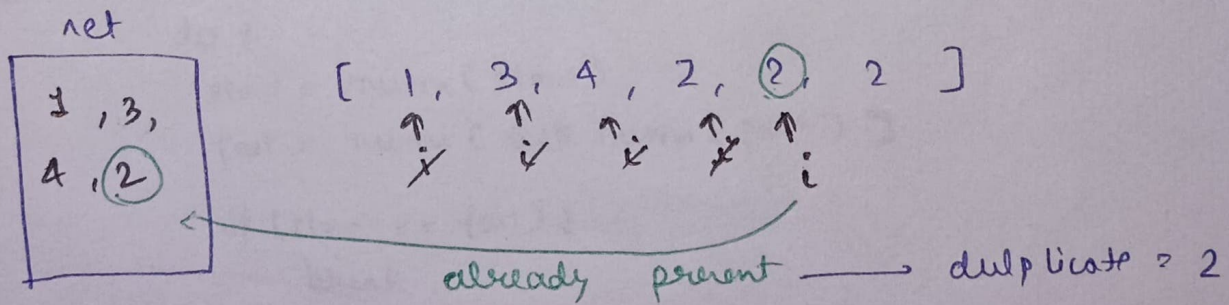
Valid i/p also as [1, 2, 2, 2, 2]

n = 4, length = 5

Eg, $[\overset{0}{1}, \overset{1}{3}, \overset{2}{4}, \overset{3}{2}, \overset{4}{2}, \overset{5}{2}]$ (Brute Force)

Size = 6 = n + 1

So, n = 5 → range = [1, 5]



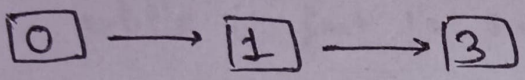
But, this approach use extra space.

Dipti verma

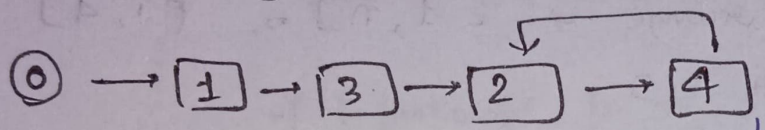
Optimization :

Eg. [1, 3, 4, 2, 2, 5] (indices 0 1 2 3 4 5)

0 index says go to on (1) index

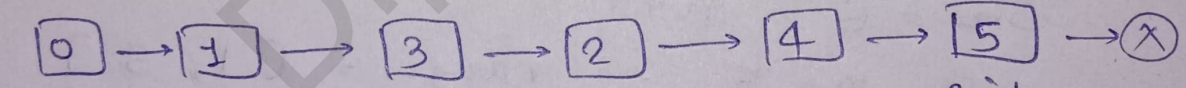


1 index says go to on (3) index



cycle

Eg. No duplicate : [1, 3, 4, 2, 5] (indices 0 1 2 3 4)



entry

exit

Dipti verma

Q.

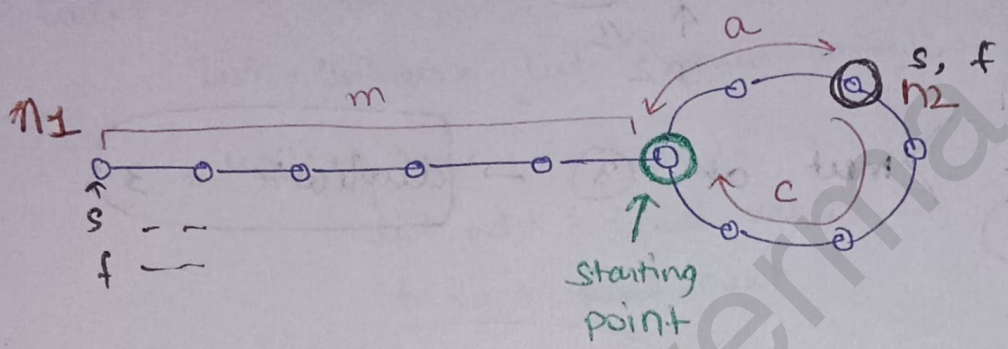
0	1	2	3	4	5
1	3	4	2	2	5

s

f = ∅ → 1 → 3

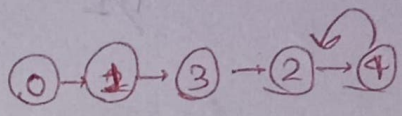
s = ∅ ~~1~~ ~~3~~ ~~4~~ ~~2~~ 4
f = ∅ ~~3~~ ~~1~~ ~~4~~ ~~2~~ 4 same } determine of cycle.

This array simply represent a LL.



[1, 3, 4, 2, 2, 5]

↓



duplicate ?
= 2 →
which starting point of cycle

$m = x \cdot c - a$

n_1 & n_2 → 1 , 1 travers
→ meet (n_1 & n_2)
→ point found.

Code:

```
int slow = 0, fast = 0;
    do {
      slow = nums[slow]
      fast = nums[nums[fast]]

      if (slow == fast) {
          break
      }
    } while (slow != fast)

    int n1 = 0, n2 = slow
    while (n1 != n2) {
        n1 = nums[n1]
        n2 = nums[n2]
    }
    return n1
```

Dipti verma

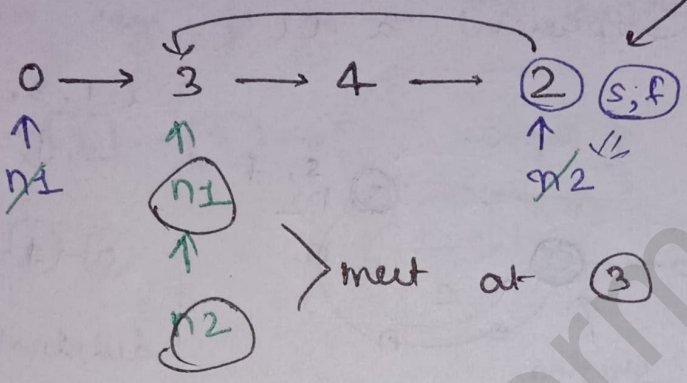
Eg.

0	1	2	3	4
3	1	3	4	2

s = ~~0~~ ~~3~~ ~~4~~ (2)
f = ~~0~~ ~~4~~ ~~3~~ (2)

break

look like



meet at (3) → duplicate = 3

→ Intitution Click ?

TC = O(n)
SC = O(1)

I. n+1 = Integers

mean,

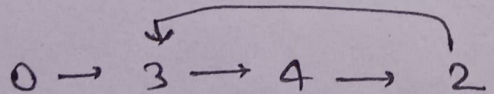
if n+1 = 5

so, n = 4

and range all no. in b/w [1, 4]

and index b/w (0, 4)

II.

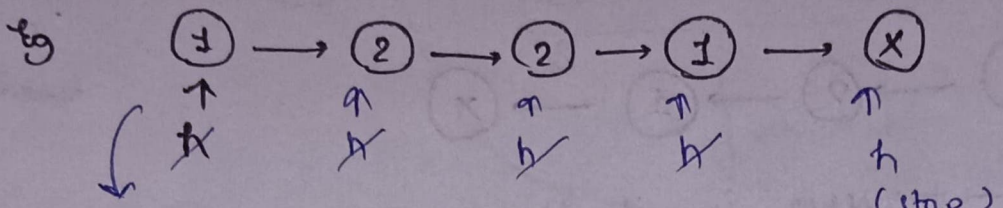


Cycle.

starting point of cycle = duplicate no.

Dipti verma

(234) Palindrome LL



dynamic array = [1, 2, 2, 1]

i j → Palindrome !!!

Code!

```
List<Integer> list = new ArrayList<>();

while (head != null){
    list.add (head.val)
    head = head.next
}

int i=0, j = list.size() - 1

while (i<j) {
    if (list.get(i) != list.get(j)){
        return false
    }

    i++
    j--
}

return true
```

TC = O(n)

SC = O(n)

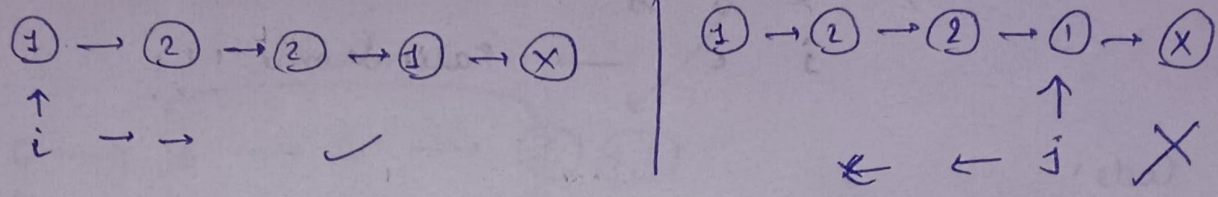
Dipti verma

Optimized !!

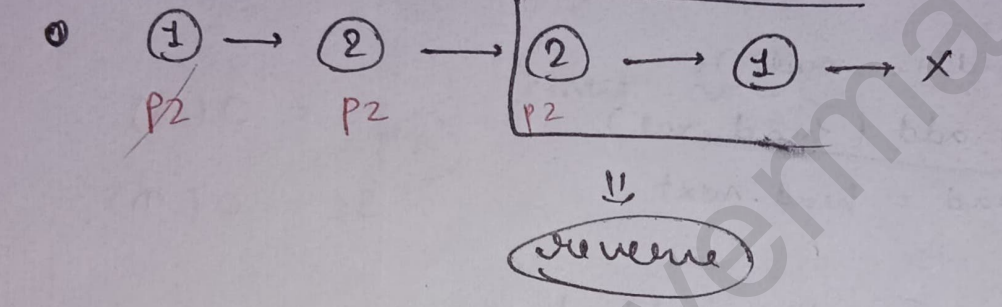
- 1 → 2 → 2 → 1 → X

Problem : LL → traverse in 1 direct

Toh,



- 1 → 2 → | 2 → 1 → X | (Half)



P1 = P2 → ++

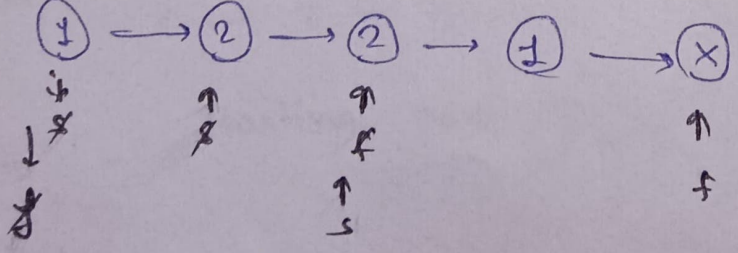
P1 → null → stop

- 1 → 2 → 2 → 1 → X

To reach at the 2nd index (middle)

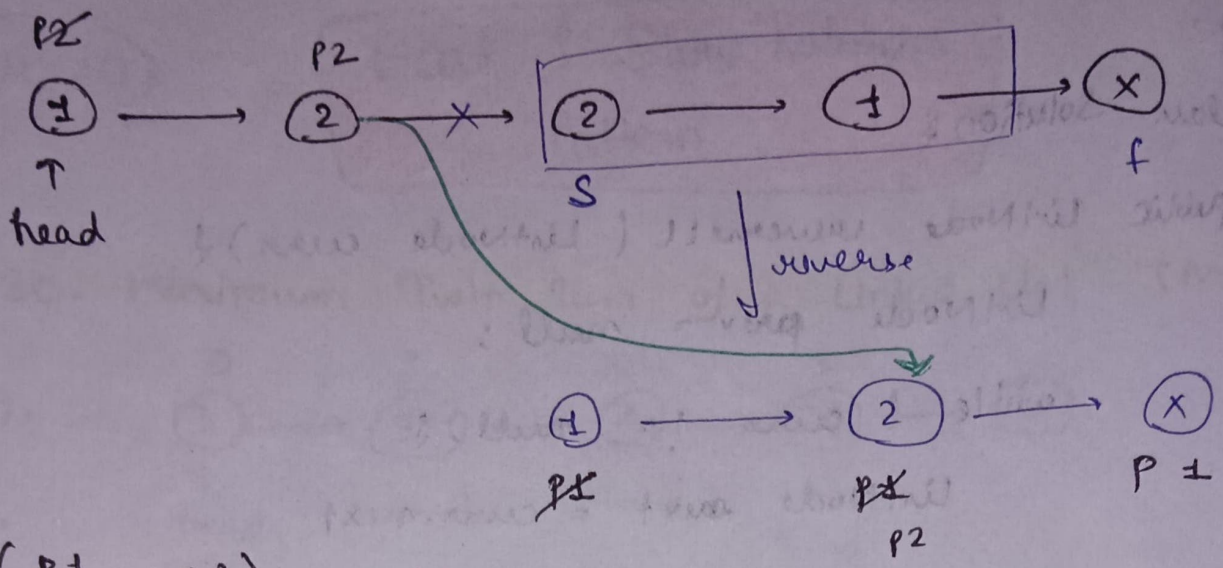
→ use fast & slow pointers

Even



[slow at 2nd index] ✓

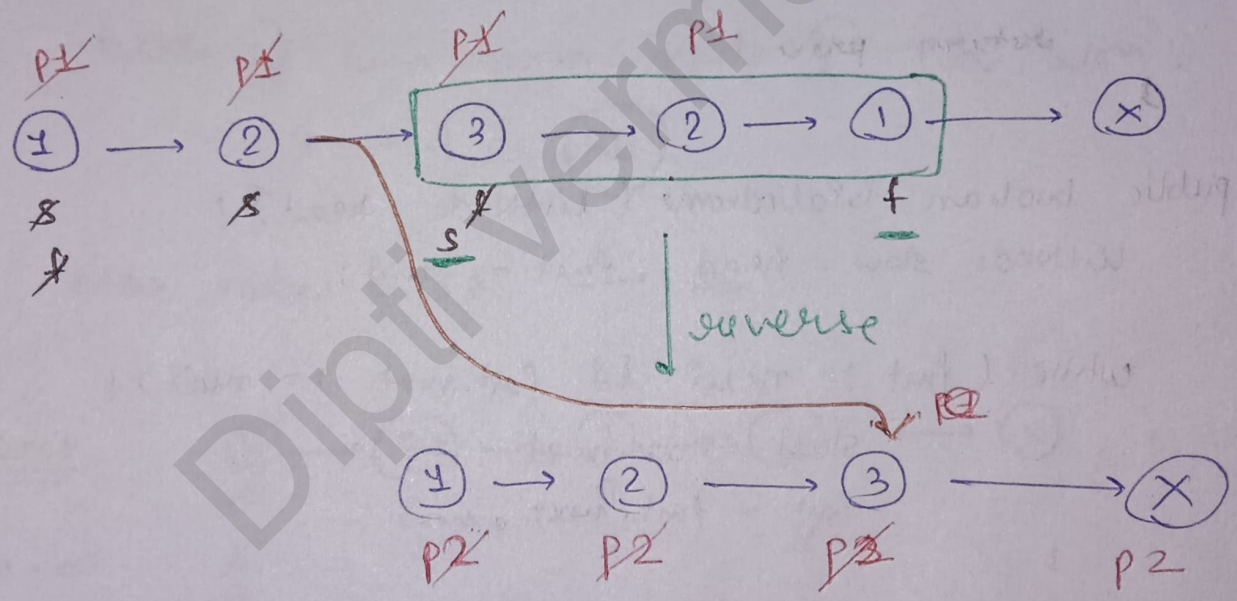
Dipti verma



(p1 == p2) ⟶ ++

⇒ p1 = null ⟶ stop ⟶ True !!

odd



(stop)

true !!

Dipti Verma

Code:

```
class Solution {
    public ListNode reverseLL(ListNode curr) {
        ListNode prev = null;
        while (curr != null) {
            ListNode next = curr.next;
            curr.next = prev;
            prev = curr;
            curr = next;
        }
        return prev;
    }

    public boolean isPalindrome(ListNode head) {
        ListNode slow = head, fast = head;
        while (fast != null && fast.next != null) {
            slow = slow.next;
            fast = fast.next.next;
        }
        ListNode p2 = reverseLL(slow);
        ListNode p1 = head;
        while (p1 != null && p2 != null) {
            if (p1.val != p2.val) {
                return false;
            }
            p1 = p1.next;
            p2 = p2.next;
        }
        return true;
    }
}
```

// eg- 4 → 3 → 3 → 4 → ⊗ (s, f)
// 4 → 3

TC = O(n/2 + n/2 + n/2)
⇒ TC = O(n)
SC = O(1)

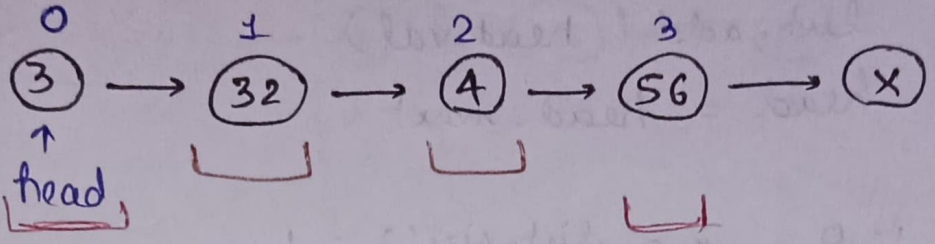
Dipti Verma

Day-10

Fast & Slow Pointers Pattern

2130. Maximum Twin Sum of a Linked List (Medium)

Eg,



Here, n = 4

node (0) twin with node (3) and sum:

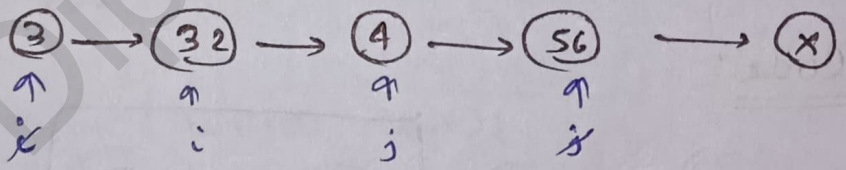
3 + 56 = 59

node (1) twin with node (2) and sum:

32 + 4 = 36

Max value = 59 ⟶ Ans

Brute force



Max = 0 - ∞

- i + j = 3 + 56 = 59 ⟶ Max = 59
  i++ and j--
- i + j = 32 + 4 = 36 ⟶ Max = 59 Ans.
  i++ and j-- break.

But, problem: It is a LL, we traverse in one direction.

So, we use linear data structure, store all the values of LL in the array / arraylist and use two pointers i & j.

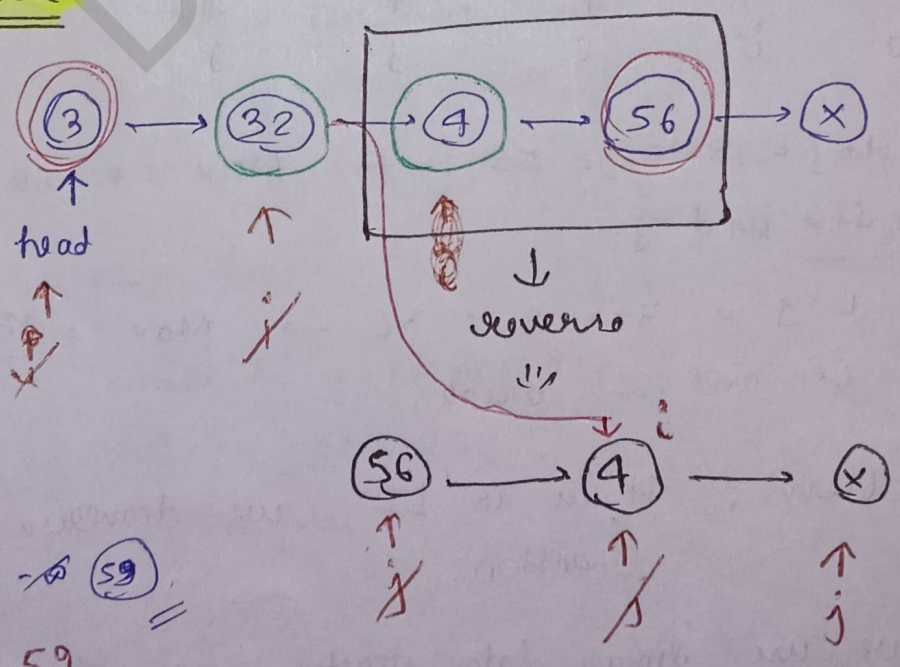
Dipti verma

Code :

```
List<Integer> list = new AL<>();

while ( head != null){
    list.add ( head.val)
    head = head.next
}

int i=0 , j = list.size() - 1
int max = Integer.MIN_VALUE
while ( i < j ){
    int sum = list.get(i) + list.get(j)
    max = Math.max( max, sum)
    i++
    j--
}

return max
```

TC = O(n)
SC = O(n)

Optimized Code :



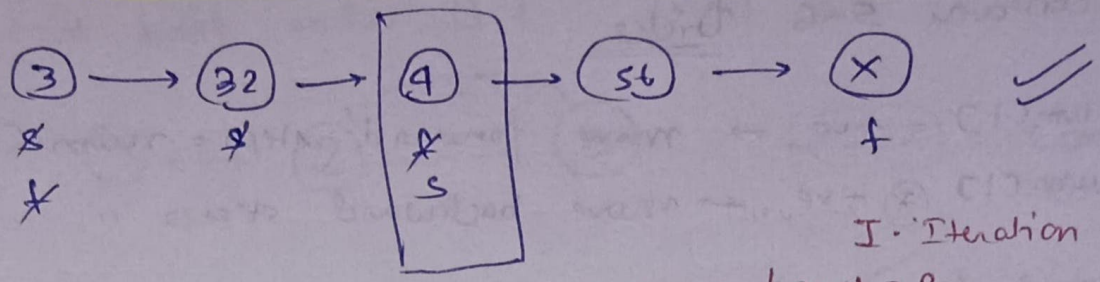
Max = ~~-∞~~ 59

3 + 56 = 59

32 + 4 = 36

Dipti verma

→ For reverse
→ we need middle node
→ Use Fast & Slow pointer



Code:

```
class Solution {
    public ListNode reverseLL(ListNode node) {
        ListNode prev = null
        while (node != null) {
            ListNode next = node.next
            node.next = prev
            prev = node
            node = next
        }
        return prev
    }
    public int pairSum(ListNode head) {
        ListNode slow = head, fast = head
        while (fast != null && fast.next != null) {
            slow = slow.next
            fast = fast.next.next
        }
        ListNode p2 = reverseLL(slow)
        ListNode p1 = head
        int max = Integer.MIN_VALUE
        while (p1 != null && p2 != null) {
            int sum = p1.val + p2.val
            max = Math.max(max, sum)
            p1 = p1.next
            p2 = p2.next
        }
        return max
    }
}
```

node → 1 → 2 → 3 → null
• prev = null
node = 1

I · Iteration
next = 2
• 1 → null
prev = 1
node = 2

II · node = 2
• next = 3
• node.next = prev → 2 → 1 → null
prev = 2
node = 3

III · node = 3
• next = null
• prev → 3 → 2 → 1 → null
prev = 3
node = null

TC = O(n)
SC = O(1)

return prev
3 → 2 → 1 → null

Dipti verma

Day-11

Fast & Slow Pointers Pattern

457. Circular Array Loop (Leetcode)

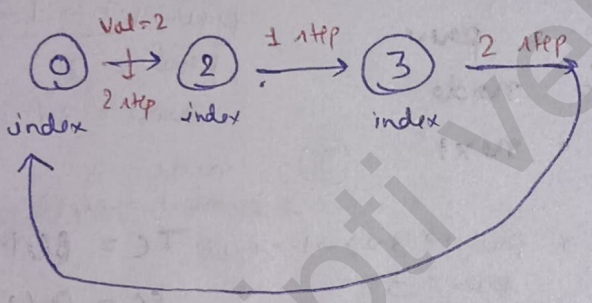
☆ It contain 5-6 tricks

- num[i] = +ve → move forward steps = num[i]
- num[i] = -ve → move backward steps = ||

→ move in circular way

Eg [2 (0), -1 (1), 1 (2), 2 (3), 2 (4)] → True

Condition I: length of cycle (k) > 1



Condition II: Cycle mein movement either +ve or -ve.

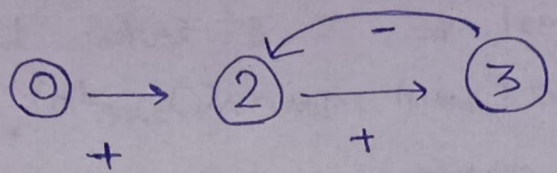
Condition II: Movement in sequence mein.

Dipti Verma

Logic

Eg [2, -1, 1, -1, 2]
(0 1 2 3 4)

① Start with index 0 :

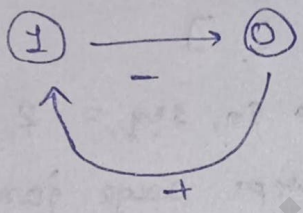


+ + - Condition break

Yah toh ++++... or ----. ✓ (valid)

~~Invalid~~ → +-+++--- or, --+++- ✗

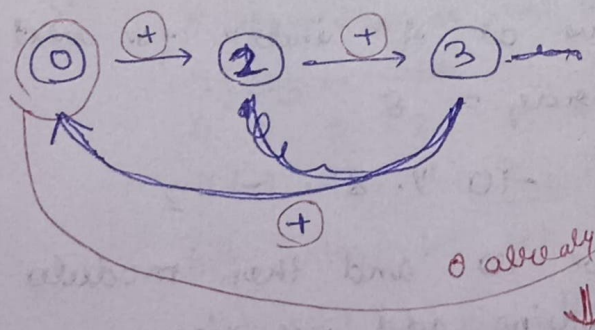
② Start with index 1 :



→ ✗ condition break (+ - +) ✗

Eg. [2, -1, 1, 2, 2]
(0 1 2 3 4)

Cycle → idx ⇒ visiting again.



Set: 0, 2, 3

0 already in set

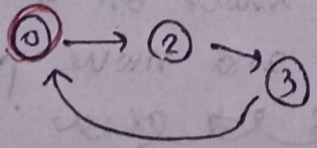
↓ yes

Means, cycle

+++ ✓
cycle ✓

Length of cycle > 1 ✓

How check ?



check (0.next != 0)

→ length > 1 //

Dipti verma

Code:

public boolean circularArrayLoop(int nums[]){

// seq, k > 1, all +ve or all -ve

// check for all indexes

class Solution {

public int calcNextIdx(int nums[], int curr){

int next = curr

int seq = nums[curr]

if(seq > 0){

next = (next + seq) % nums.length

// [2, -1, -1, 2, 2]

// curr index = 3 → so, seq = 2

// means, hame 2 steps aage jana hai and in circular way.

TC = (n²)

SC = (n)

}else{

// mod with -ve in java

// Let index = 0 1 2 3 4 5 6 7

// array = - (-5) - - - - - -

// we have to move at 4th index for next step

// So, length of array = 8

// in java, eg -10 % 8 = -2

// How? let -10 be 10 and then modulo

// and after the solving add -ve sign

// so, 10 % 8 = 2 → -10 % 8 = -2

// Let, we have seq = -5 .

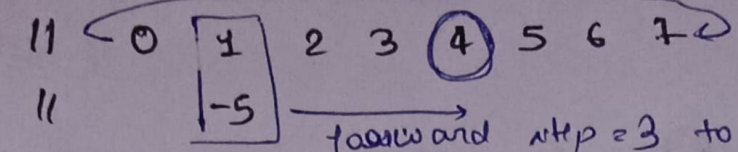
// So, we have to move on 4th index and

// array's length = 8 and move forward 3 steps

// so,

Dipti Verma

// How 3 steps ? backward steps = 5

// 

// So, -5 + x = 3 ⇒ x = 8

// and what '8' ? ⇒ length of array

// So, move next -ve no. in forward direction as

// (req ~~to the of add~~ % len of array)

```
int mod = req % nums.length
int forward = nums.length + mod
next = (curr + forward) % nums.length
```

}

return next

}

public boolean circularArrayLoop (int nums[]) {

// req, k > 1, all +ve or all -ve

// check for all indexes

for (i=0 ; i < nums.length ; i++) {

// set → indexes that we have visited so far

// flag → isPos → Check +ve or -ve

// 0 1 2 3 4

// [2 , -1 , -1 , 2 , 2]

Set<Integer> set = new HashSet<>()

set.add (i); // add the index

boolean isPos = nums[i] > 0

int curr = i (curr → jump from index to index)

.

Diti verma

```
// cycle detection
//  0   1   2   3   4
// [ 2, -1,  1,  2,  2 ]
// set => {0, 2, 3}
while ( true ) {

    int next = calcNextIdx ( nums, curr)

    if ( isPos ) {      // no = +ve
        if ( nums[next] < 0 ) {     // next no. = -ve
            break
        } else {
            if ( set.contain(next) ) {   // set -> contain -> cycle
                // cycle is there
                // condition : K > 1 -> valid
                if ( curr != next ) {    // cycle length > 1
                    return true
                } else {
                    break     // check the next idx
                }
            }
            set.add(next)
        }
    } else {
        if ( nums[next] > 0 ) {
            break
        } else {
            if ( set.contains(next) ) {
                if ( curr != next ) {
                return true
                } else {
                    break
                }
            }
            set.add(next)
        }
    }
    curr = next
}
```

```
   0   1   2   3   4
[  2  -1   1   2   2 ]
   ↑       ↑   ↑
  curr     next next
   ↑       ↑   ↑
  next     curr curr
{ 0, 2, 3 }
   ↑
curr at 3
next at 0
length > 1
```

if curr == next
length = 1

Dipti verma

* How this code handle -ve steps that means backward steps ?

⇒ if movement is -ve (seq < 0)

Actually, java's (%) with -ve no. gives -ve results.

Eg, $-2 \% 5 = -2$

So, we have handle it manually.

I. Take modulo of -ve no.

[mod = seq % nums.length]

Eg, seq = -2, length = 5 → mod = -2

II : Convert this -ve steps into -ve forward movement.

forward = nums.length + mod

forward = 5 + (-2) = 3

Eg, [2, -1, -1, 2, 2] (0, 1, 2, 3, 4)

↑ curr

next = curr

seq = nums[curr] = 2

(seq > 0) → next = (0 + 2) % 5 = 2

[2, -1, -1, 2, 2]

↑ next

next = 2

seq = -1

seq = nums[curr] = -1

Forward step = 4

(seq < 0) → mod = -1 % 5 = -1

[2, -1, -1, 2, 2] ⇒ How get 4 ?

1 step backward

-1 + x = 4

x = 5

and 5 = nums.length !!!

Dipti verma

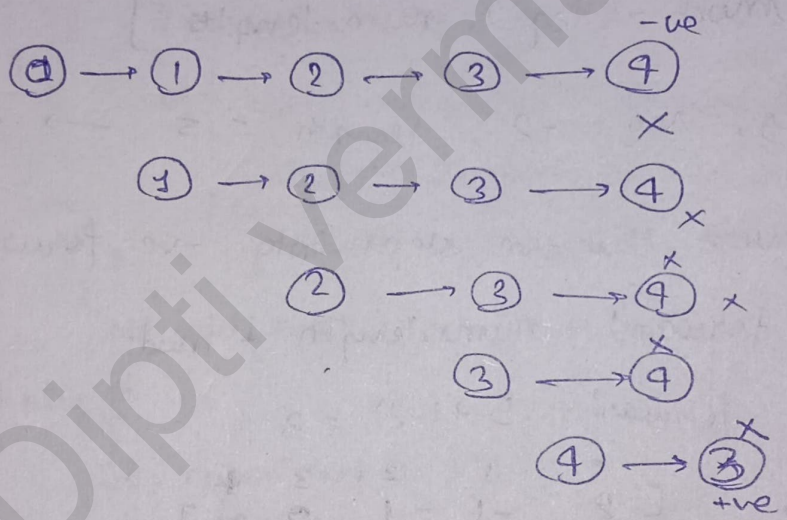
II. Calculate nxt Idx

next = (curr + forward) % nums.length.

TC = O(n²) ???

eg.

0	1	2	3	4
1,	1,	1,	1,	-1
↑	↑	↑	↑	↑
i	i	i	i	i



O(n²)

But, we see if cycle is not found b/w index 0 to 3 as all are +ve

that mean, : 1 to 3
2 to 3
3 to ...

} NO cycle found

So, why we iterate again on them ?

→ we have to optimized this

Dipti verma

Let hame '0' index pe cycle wahi mila toh,

0	1	2	3	4
1	1	1	1	-1

we mark 0 to 3 as '0' as all are +ve

0	1	2	3	4
0	0	0	0	-1
↑ i	↑ i	↑ i	↑ i	

(1) → no step so, i++

(4) → (3) → no step

Code:

```
for(int i = 0 ; i< nums.length ; i++){
    if(nums[i] == 0){
        continue
    }
Set<Integer> set = new HS<>();
set.add(i);
boolean iPos = nums[i] > 0
int curr = i;
while(true){
    :
    :
}
curr = i
if(iPos){
    while( nums[curr] >0 ){
        int next = calcNextIdx(nums, curr)
        nums[curr] = 0
        curr = next
    }
}else{
```

Dipti verma

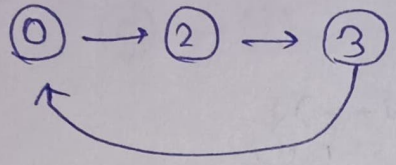
```
while ( nums[curr] < 0){
    int next = calcNextIdx( nums, curr)
    next(curr) = 0
    curr = next
}
}
```

$T(n) = O(n)$

$S(n) = O(n)$

Optimized :

0	1	2	3	4
2,	-1,	1,	2,	2]

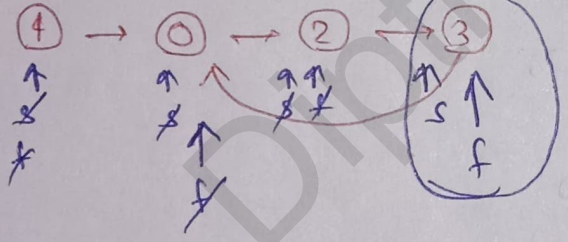


Cycle detection

↓

always think ⇒ fast & slow pointer

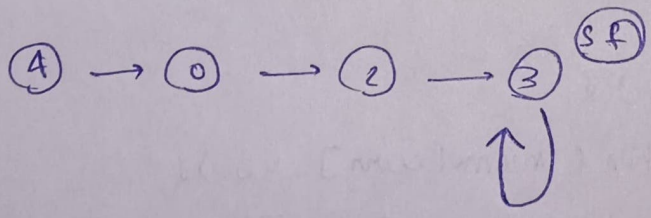
↓



meet → cycle !!

How check the length of cycle ?

is cycle length == 1 ? →



In this case slow index == fast next

Dipti verma

Main code :

```
public boolean circularArrayLoop ( int[] nums){
  for( int i=0; i< nums.length ; i++ ){
      if( nums[i] ==0 ){
          continue
      }
      boolean isPos = nums[i] > 0
      int slow = i , fast = i
      do{
        slow = calcNextIdx ( nums, slow)
        fast = calcNext Idx ( nums, fast)
        if ( isPos ){
              if( nums[fast] < 0){
                  break
              }
        }else{
              if (nums[fast] > 0){
                  break
              }
        }
        fast = calcNextIdx ( nums , fast)
        if( isPos ){
              if (nums[fast] < 0){
                  break
              }
        }else{
              if (nums[fast] >0){
                  break
              }
        }
```

Dipti verma

```
if ( slow == fast) {
        if ( slow != calcNextIdx ( nums, slow )) {
            return True
        }
        break
    }
} while ( slow != fast);

int curr = i

if (isPos) {
    while ( nums[curr] > 0 ) {
        int next = calcNextIdx ( nums, curr)
        nums[curr] = 0
        curr = next
    }
} else {
    while ( nums[curr] < 0 ){
        int next = calcNextIdx ( nums, curr).
        nums[curr] = 0
        curr = next
    }
}
}
return false
}
```

Dipti verma