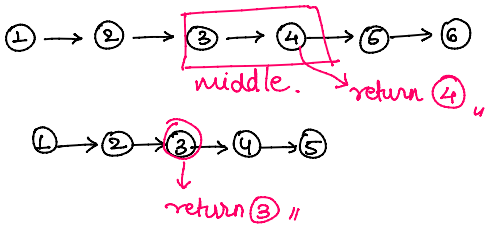
Middle of Linked List

Tuesday, May 3, 2022 11:16 AM

Given



Naive Method

Algo :-

① Iterate over the LL from head to tail maintaining cnt.

② cnt = size of LL.
Again iterate from head till (cnt-1)/2.

③ The last pointing Node is middle node.

T.C = O(n) ; S.C = O(1) = constant

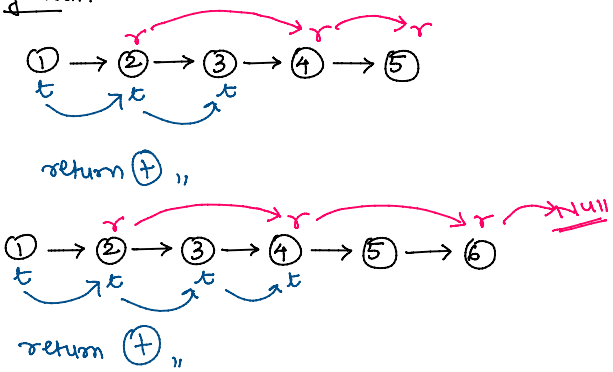
Rabbit - Turtle Algo

*rabbit → fast pointer → 2 steps at a time.
* turtle ⇒ slow pointer → one step at a time.

Base case :-) ① head != NULL
head → next != NULL } If False :- return head;

② head → next → next != NULL } if false return head;

Dry Run :-

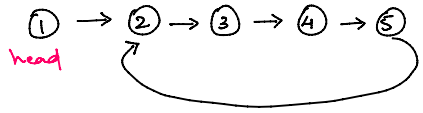


T.C = O(n)
S.C = O(1) = constant.

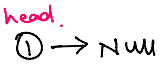
Linked List Cycle

Tuesday, May 3, 2022 11:55 AM

given :-



o/p = 1 → True.



o/p = false = 0

Optimal | Brute force solution :-

① Rabbit – Turtle Algo

* rabbit = head → next : (fast ptr → it takes 2 steps at a time)
* turtle = head : (slow ptr → it takes 1 step at a time)

Algo :-

① Check whether the LL is null or contains only one element
If yes :-

```
if (!head || !(head->next))
    return false;
```

② *rabbit = head → next , *turtle = head
③ continue to iterate till rabbit != Null.
If at any point :-

```
if (rabbit == turtle)
    return true
```

④ Else return false.

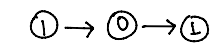
T.C = O(n)
S.C = O(1) = constant

Convert Binary Number in a Linked List to Integer

Tuesday, May 3, 2022 6:15 PM

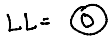
Given :-

LL with binary values.



LL = [1, 0, 1]

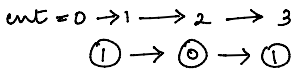
o/p = $(101)_2 = (5)_{10}$.



LL = [0]

o/p = $(0)_2 = (0)_{10}$

Brute force | Naive Method :-



cnt = 0 = size of linked list

```
cnt--;
ptr = head;  int ans = 0;
while (ptr) {
        ans += (1 << cnt) * ptr->value;
        cnt--;
        ptr = ptr->value;
}
return ans;
```

Algo

1) Find the size of the LL.
2) Decrease 1 from the size to follow the binary convention
3) loop the entire LL again while updating answer.

ans = ans + (1 << cnt) * ptr->value

4) Return ans;

T.C = O(n+n) => O(2n) => O(n)
S.C = O(1)

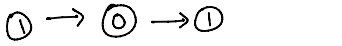
Better Method (using one loop) :-

1) int ans = 0;

①

```
int ans =0;
Ptr = head;
while ( ptr){
   ans = (ans<<1) + ptr -> value;
    ptr = ptr -> next;
}
return ans;
```

Dry Run:-



ans = 0

it : 1 → (ans <<1) + ptr → value

0<<1 = 0 1 ⇒ 0+1 = 1

it : 2 → (ans <<1) + ptr → value

1<<1 = 2 0 ⇒ 2+0 = 2

it : 3 → (ans <<1) + ptr → value

2<<1 = 4 1 ⇒ 4+1 = 5

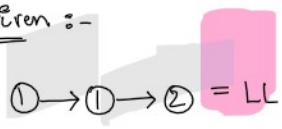
Eg :- $(1101)_2 = (13)_{10}$

ans = 0 → (0<<1)+1 = 1 ⇒ (1<<1)+1 = 3 ⇒ (3<<1)+0 = 6 ⇒ (6<<1)+1 = (13)

Remove Duplicate from the List

Wednesday, May 4, 2022 9:01 AM

Given :-



Removing the duplicates.

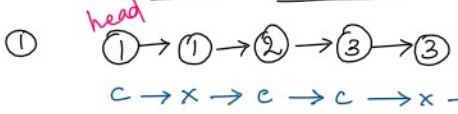
O/P :- 1 → 2 = LL

LL = 1 → 1 → 2 → 3 → 3

Removing the duplicates.

O/P = 1 → 2 → 3 "

Brute force | Naive method :-




```
* curr = head;
while (curr) {
    if ( curr -> next != NULL   and   curr -> value
                                    == curr -> next -> value)
        * nextNode = curr -> next -> next;
        * del = curr -> next;
        delete(del);
        curr -> next = nextNode;
    }
    else {
        curr = curr -> next;
    }
}
```

T.C = O(n)

S.C = O(1) = constant

Algo :-

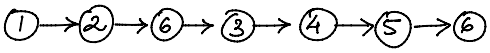
Algo :-

- Make a current pointer pointing towards Head.
- Traverse the LL once.
- if same => del : otherwise = continue.
- Return head.

Remove the Linked List Element

Wednesday, May 4, 2022 12:32 PM

given :-

LL = 

val = 6

o/p ⇒ LL = 

Brute force | Naive Method :-

Algorithm

1) keep a pointer, who's (pointer → next) . value is equal to the value given. and pointer's → next is not null.
2) The problem with the above logic is it always look at (→next → val). What if the head→value is the target value. In that case keep on moving head until we reach a point who's head→val != val

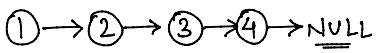
```
while( head != Null and head → val == val ){
        head = head → next;
}
* ptr = head;

while (ptr ){
    if ( ptr → next != Null and ptr → next → val == val ){
        * nextNode = ptr → next → next;
        delete ( ptr → next) ;
        ptr → next = nextNode;
    }
    else{
        ptr = ptr → next;
    }
}
return head;
```

Reverse a Linked List

Friday, May 6, 2022 9:11 AM

Given :-



O/P :- 

Brute Force | Naive Method

① Iterative Method

Algo :-

i) Create 3 pointers.
* prev = Null , * forward = Null , * curr = head;

ii) iterate over the entire Linked list.

iii) Maintain the 3 pointers such that →

```
forward = curr -> next
curr -> next = prev
prev = curr
curr = forward.
```

② Recursive solution :-

i) Base condition

```
if (curr == Null)
    return prev;
```

ii) Logic

```
forward = curr -> next;
curr -> next = prev;
prev = curr
curr = forward
```

iii) Recursive logic

```
return reverseLL (curr, prev, forward)
```

Multiply Two Linked List

Friday, May 6, 2022 10:49 AM

Given:-

L1 = 3 → 2 → NULL

L2 = 2 → NULL

O/P = L3 = 6 → 4 → NULL

Brute force/Naive method :-

L1 = 3 → 2 → X

L2 = 2 → X

Algo

i) Traverse both the list to form a number. first and second namely.

ii) return first * second

TC = O(n)
S.C = O(1) = constant.

Delete Without Head Pointer

Saturday, May 7, 2022 9:10 AM

Given :-

LL = 10 → 20 → 4 → 30 → NULL

Del * = 20

O/P = 10 → 4 → 30 → Null

Brute Force | Naive Method :-

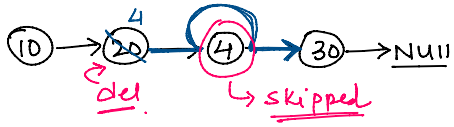
*del = Node that needs to be deleted.

We can't delete the Node without ~~e~~ head node.

∴ del → data = del → next → data ;
del → next = del → next → next ;

TC = O(1) = constant
SC = O(1) = constant.

Dry Run :-



del → data = del → next → data ;
del → next = del → next → next ;

Sort 0's 1's and 2's in linked list

Saturday, May 7, 2022 9:35 AM

given:-

LL = { 1, 1, 2, 0, 1, 2, 0 }

o/p = { 0, 0, 1, 1, 1, 2, 2 }

Brute force | Naive Method :-

Algo:-

count all 0's, 1's & 2's in first traversal then traverse again to fill first N_0 nodes with 0, Next N_1 nodes with 1 and last N_2 nodes with 2.

TC = $O(n+n+n+n) = O(4n) = O(n)$

SC = $O(1)$ = constant.

Merge Two Sorted Linked list

Saturday, May 7, 2022 9:44 AM

Given :-

L_1 = 1 → 2 → 4

L_2 = 1 → 3 → 4

Op :- 1 → 1 → 2 → 3 → 4 → 4

Brute Force :-

(*) Extra Space Algorithm :-

① Create a dummy Node and a * pointer to keep track of dummy node.

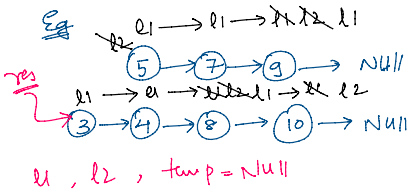
② While both the list exists insert the smaller value into the dummy list.

③ Check if any one list is remaining if 'yes' add their values directly to dummy node cause already sorted.

T.C = $O(n_1+n_2)$

S.C = $O(n_1+n_2)$

(*) Inplace Algorithm :-



l1 , l2 , temp = NULL

l1 = points to node with smaller value.

l2 = points to node with larger value.

temp = points to one prev node of l1

it ①

temp = NULL

temp = 3

temp → next = 5

swap (l1, l2)

it ②

temp = NULL

temp = 7

temp → next = 8

swap (l1, l2)

it ③

temp = NULL

temp → next = 9

swap (l1, l2)

it ④

temp = NULL

temp → next = 10

over

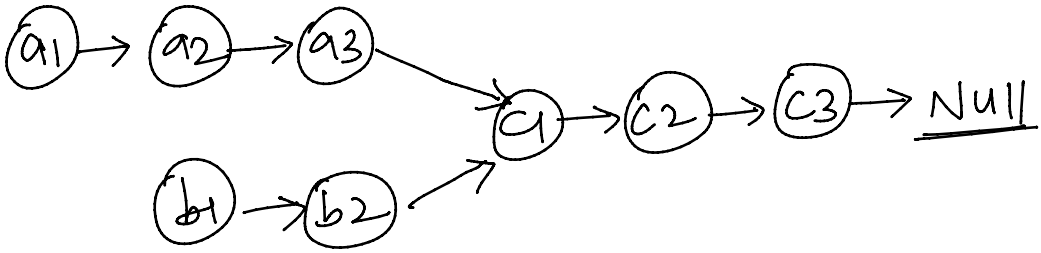
∴ return res

→ [3, 4, 5, 7, 8, 9, 10]

Intersection of Two Linked List

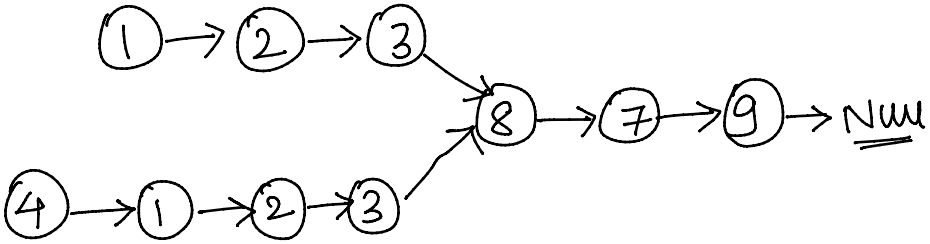
Monday, May 9, 2022 12:10 PM

Given :-



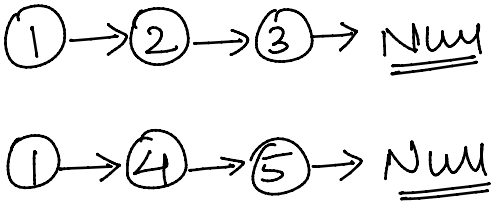
return c1 if exist or return NULL.

Eg:-



O/P = 8

Eg:-



O/P No intersection

⊛, Brute force Approach | Naive Method :-

Use two loops to iterate over each and every

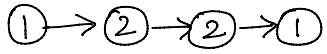
Use two loops to iterate over each and every node. when encountered same return that node other wise return null.

```
while(l1){
        ListNode* p2 = headB;
        while(p2){
            if(l1==p2){
                return p2;
            }
            p2 = p2->next;
        }
        l1 = l1->next;
}
return null;
```

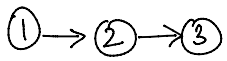
Palindrome in Linked List

Saturday, May 7, 2022 1:09 PM

given:-



o/p = true.



o/p = false.

Brute force

Algo

Use of extra Data-structure. :-

1) Find the size of LL by traversing the LL.
2) Create an array of size of LL
   ∴ arr[cnt];
3) Copy the elements of the LL in the array.
4) Check whether the array is palindrome or not.

TC = O(n+n+n) = O(3n) = O(n)
SC = O(n)

Better Method

1) Find the mid of the linked list.
   let it be called mid.
2) Reverse the linked list after mid.
3) Traverse and check if palindrome or not.

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

Disadvantage → Manipulating original LL