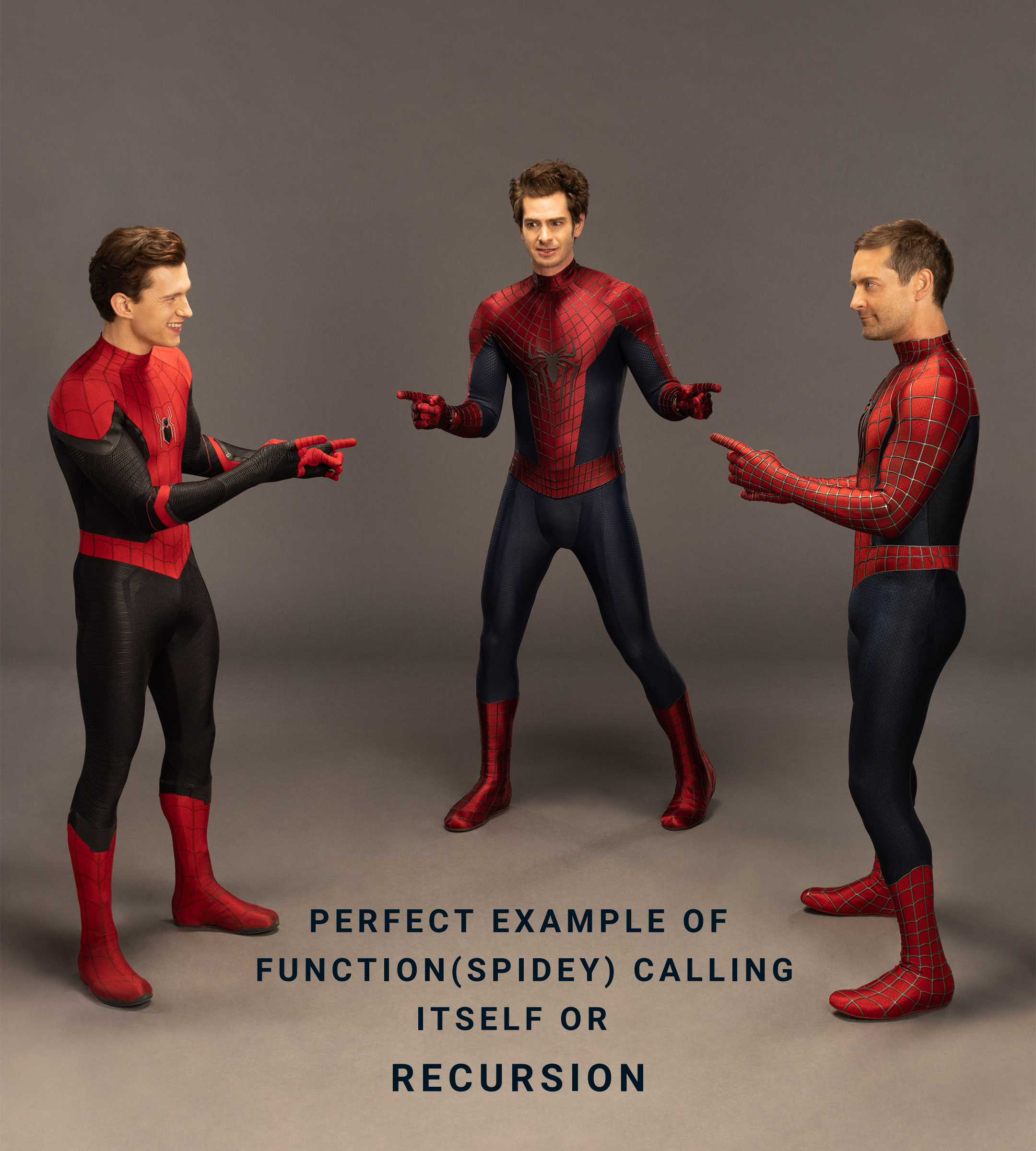

PERFECT EXAMPLE OF
FUNCTION(SPIDEY) CALLING
ITSELF OR
RECURSION

ON A MISSION TO MAKE YOU LOVE DSA

CRASH CAMP

RECURSION

Introduction

BY-

MAYANK SINGH

FINDING

RECURRENCE RELATION

Ruyank: Bhaiyaa, in previous episode you taught me about Recursion I mean what Recursion looks like 'under the hood'

Bhaiyaa: Awesome, if anyone of you haven't watched that I'll drop link in comment for Episode 1

Ruyank: I still don't get recurrence relation for most problems

Bhaiyaa: It takes time to get hold of recurrence relations, I'm attaching few of my previous Decks which you can see & get the idea of recurrence relation

Remember, only get idea about finding recurrence relations that's it, except that all other stuffs like setting base case & writing the recursion, we will see in depth later

Ruyank: Sure bhaiyaa, I'll only get idea about recurrence relation & guys let's like, comment & share this post to cheer up bhaiyaa

Leetcode
Daily Challenge

06/03/2022

T.C. o(n)
S.C. o(n)

problem

Count All Valid Pickup and Delivery Options

pre-requisites

recursion, basic maths

difficulty
Hard

est. time
10-15 min

can be asked in...

55%
Accuracy

Statement

Description

- Given n orders, each order has pickup and delivery services.
- An order would be first picked & then delivered vice-versa in not possible.
- Count all valid sequences of pickup-delivery for n oders.

i/p	o/p
n = 2	6

explanation

- All possible orders:
- (P1,P2,D1,D2), (P1,P2,D2,D1), (P1,D1,P2,D2), (P2,P1,D1,D2), (P2,P1,D2,D1) and (P2,D2,P1,D1).
- This is an invalid order (P1,D2,P2,D1) because Pickup 2 is after of Delivery 2.

Intuition

i/p

n = 3

o/p

90

if we have 3 orders then total

3 pickups + 3 deliveries = 6

p1,p2,p3, d1,d2,d3

* * * * * *

we just need to find all possible ways to fill above * such that an order is first picked then delivered.

- let's start by replacing 1st *, out of all orders we can choose any, let's choose p1

p1 * * * * *

- now d1 can occupy any of 5 * i.e (2n-1) positions
- but in place of p1 - p2,p3 can also come so a total of n * (2n-1) combinations are there just for handling 1st order.

Intuition

i/p

n = 3

o/p

90

* * * * * *

- so 1 order generates `n * 2n - 1` combinations.

- now if 1st order is handled, we are remaining with 2 more (n-1)

- we started for calculating all combinations of n orders, now we need for n-1 orders
- say you had a functions `countOrders(int n)`

which gives combinations for n, can we reuse it for n-1 oders as well....

- what's this behaviour...?

Recursion

algorithm

```
int countOrders(int n) {
  return n*2n-1 * countOrders(n-1)
}
```

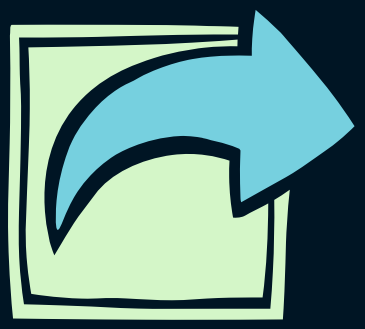
we calculated all combinations for 1st order

give me all combinations for for n-1 orders

```cpp
class Solution {
public:
    #define mod 1000000007
    int countOrders(int n) {

        if(n == 0) {
            return 1;
        }

        long long ans = (long long)n*(2*n-1) % mod;
        ans = ans * countOrders(n-1) % mod;
        return (int) ans;
    }
};
```

Leave a Like

Comment if you love posts like this, will motivate me to make posts like these



Share, with friends

T.C. o(n)
S.C. o(1)

Leetcode
Daily Challenge

09/03/2022

problem
Remove Duplicates from Sorted List II

pre-requisites
linked list

difficulty
Medium

est. time
15-20 min

Let's build Intuition

can be asked in...

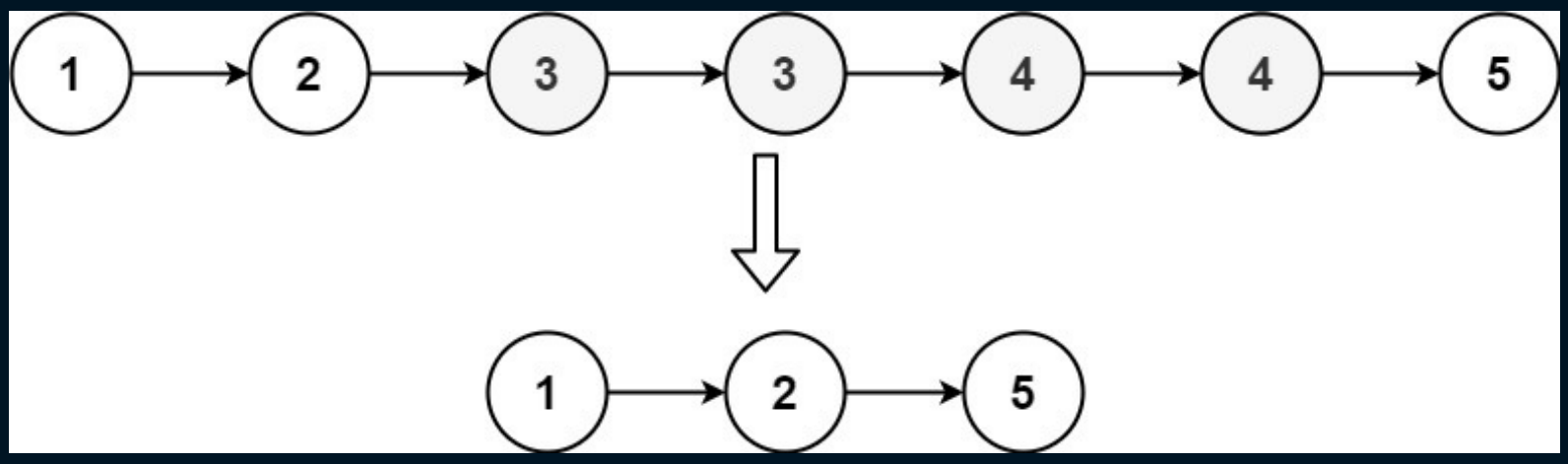
42%
Accuracy

Statement

Description

- Given the head of a sorted linked list, delete all nodes that have duplicate numbers, leaving only distinct numbers from the original list. Return the linked list sorted as well.

i/p	o/p
head = [1,2,3,3,4,4,5]	[1,2,5]



Intuition

- Let's make a function `deleteDuplicates()` which performs following tasks-
  1) Takes Linked List as input.
  2) Deletes the duplicate nodes
  3) Returns a list which has no 'duplicates'

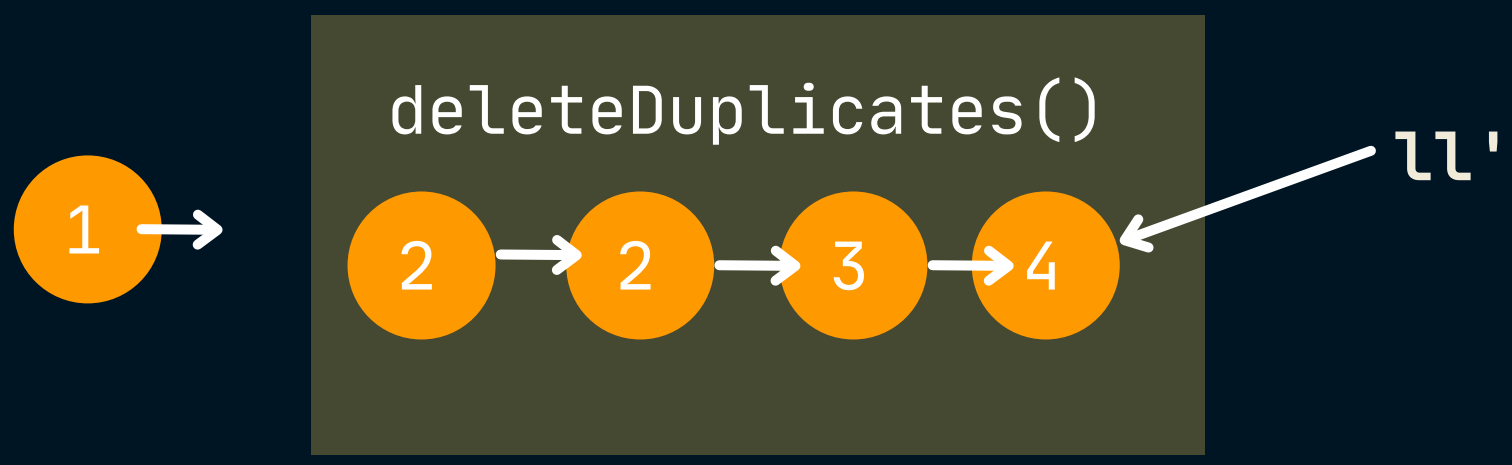
ll

Let's consider above list

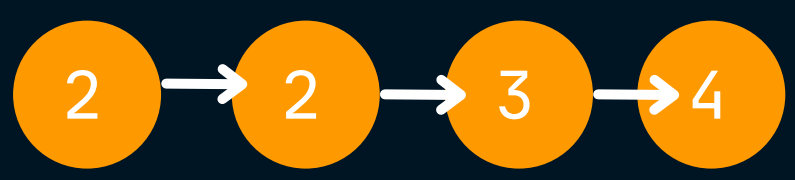
deleteDuplicates()

- we start with 1st node, traverse the list to see if there exists any duplicate of 1
- we don't find any so we keep 1 & ask `deleteDuplicates()` to take the remaining list, delete the duplicates & return remaining list with no dupl.

Intuition

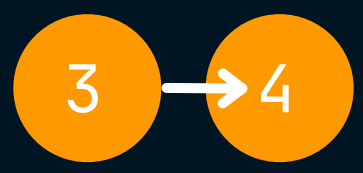


- Now deleteDuplicates() take ll', deletes duplicates & return remaing list which gets attached to node 1's right.
  - let's see ll' in action-



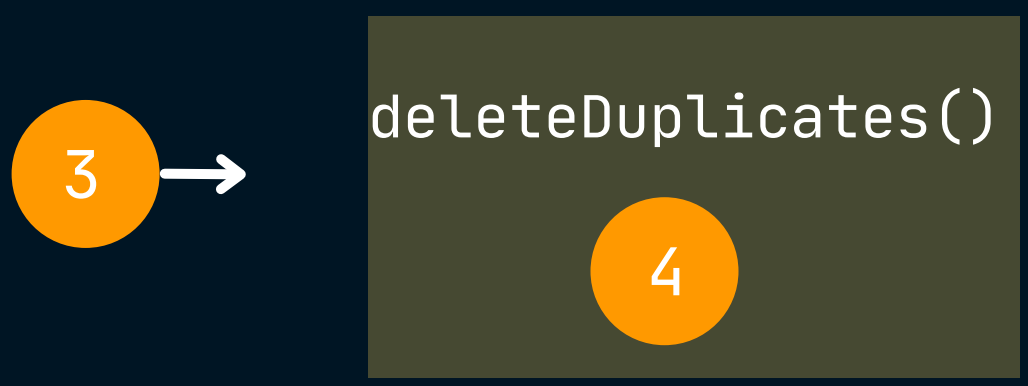
- since 2 is duplicate, deleteDuplicates() will delete these nodes & again check for remaining list
- Now after deleteDuplicates() deleted 2, our list looks like

Intuition



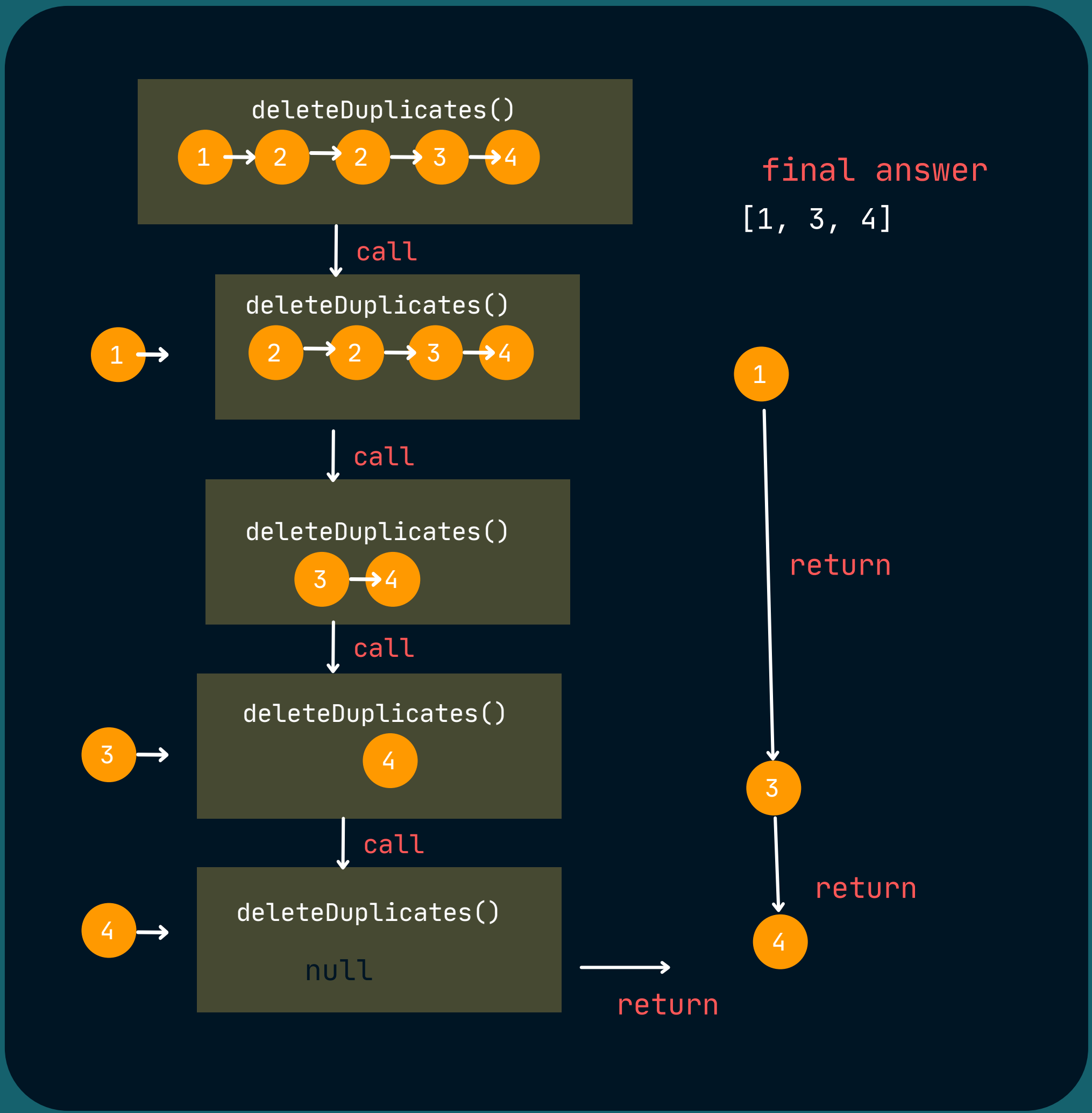
ll''

- Now deleteDuplicates() will work on ll''
- as 3 is not duplicated, so 3 won't be deleted & remaining list will be checked.



- now we are remaining with a single node, which is not repeated, so our last function call ends & we return the lists at end of each call.

Intuition


deleteDuplicates()
1 2 2 3 4
call
deleteDuplicates()
1
2 2 3 4
call
deleteDuplicates()
3 4
call
deleteDuplicates()
3
4
call
deleteDuplicates()
4
null
return
final answer
[1, 3, 4]
1
return
3
return
4

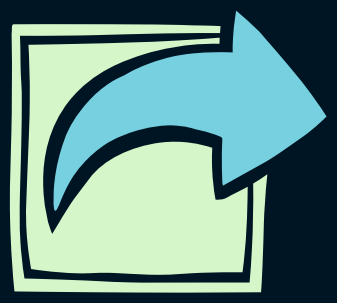
Algorithm

- Can you visualize how 'deleteDuplicates()' is doing-
  1) take list as input.
  2) iterate till you you get a node whose value is not equal to head value.
  3) While iterating if duplicate found, delete duplicates
  4) recur for remaining list.

```cpp
class Solution {
public:


    ListNode* deleteDuplicates(ListNode* head) {


        if(!head || !head->next) {
            return head;
        }

        int val = head->val;
        ListNode* currNode = head->next;

        if(currNode->val != val) {
            head->next = deleteDuplicates(currNode);
            return head;
        } else {
            while(currNode && currNode->val == val) {
            ListNode* dummy = currNode;
            currNode = currNode->next;
            delete dummy;
        }
            delete head;
            return deleteDuplicates(currNode);
        }
    }
};
```

Leave a Like

Comment if you love posts like this, will motivate me to make posts like these



Share, with friends

T.C. o(n)
S.C. o(1)

Leetcode
Daily Challenge

10/03/2022

problem
Add Two Numbers

pre-requisites
linked list, recursion

difficulty
Medium

est. time
15-20 min

Let's build Intuition

can be asked in...

38%
Accuracy

Statement

Description

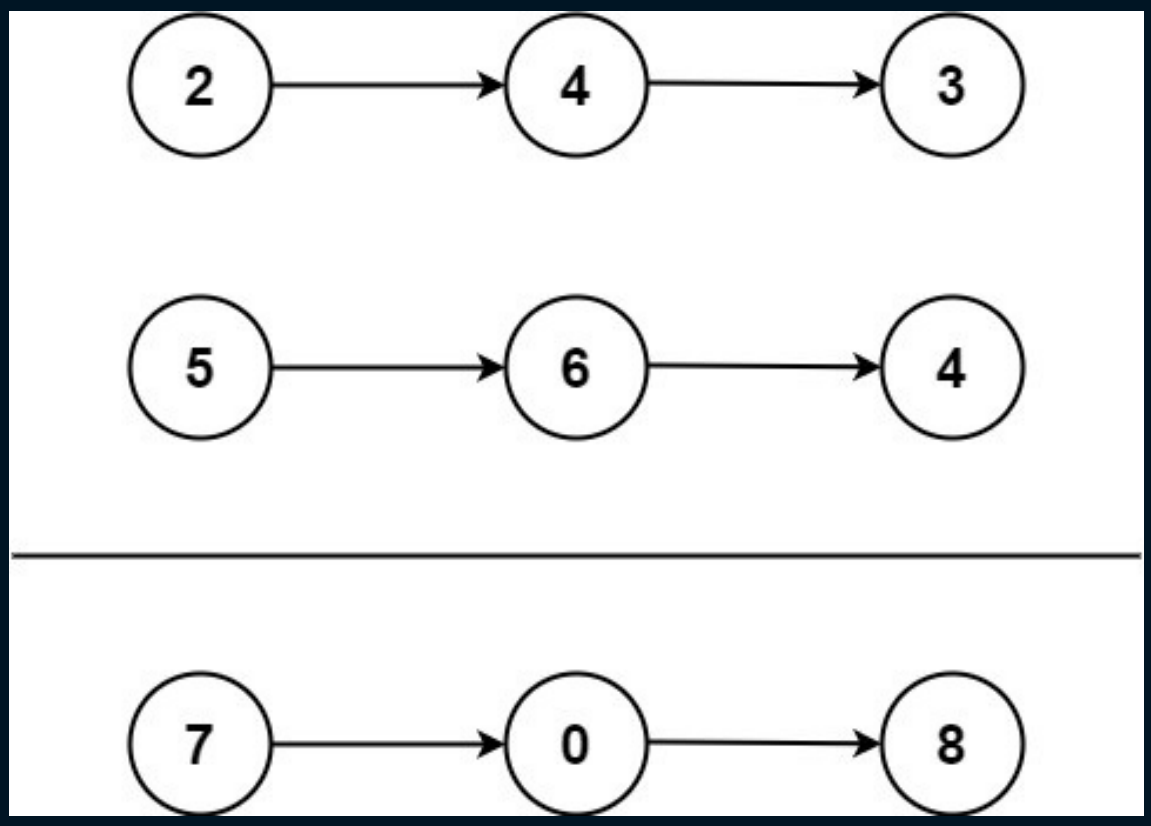
- Given 2 Linked List representing 2 non-negative numbers
- Digits are stored in reverse order
- Add those 2 no. & return the sum as a list

i/p

```
l1 = [2,4,3],
l2 = [5,6,4]
```

o/p

```
[7,0,8]
```



Intuition

- Idea is to see how we add 2 numbers

148 + 895 = 1043

```
        carry = 1    carry = 1  carry = 0

                1      4      5
            +
carry = 1       8      9      5
        ------------------------
        1       0      4      0
```

we add from right to left, with following steps

- sum = n1 + n2; // see last digits 5+5 = 10
- n = sum % 10; // we keep 10 % 10 = 0 only
- carry = sum / 10; // carry = 10 / 10 = 1

continue above steps till you reach left most digit

Intuition

- Here our LLs are numbers but in reverse order so we need to go from `left->right`

Idea is, we start iterating both lists same time

```
carry = 0
1) Create a new node "node"
2) sum = l1->val + l2->val + carry
3) node->val = sum % 10
4) carry = carry / 10
5) l1 = l1->next, l2 = l2->next;
```

now, repeat these steps till you have some none null nodes

you can also avoid creating new node all the time by reusing same nodes, try that.

```cpp
class Solution {
public:
    ListNode* addTwoNumbers(ListNode* l1, ListNode* l2) {
        ListNode *newHead = new ListNode();
        ListNode *temp = newHead;

        int c = 0;
        while (l1 || l2 || c)
        {
            int sum = 0;
            if(l1)
            {
                sum += l1->val;
                l1 = l1 -> next;
            }
            if(l2)
            {
                sum += l2->val;
                l2 = l2 -> next;
            }
            sum += c;
            c = sum/10;
            ListNode *node = new ListNode(sum%10);
            temp -> next = node;
            temp = temp -> next;
        }
        return newHead -> next;
    }
};
```

Intuition

- Now let's see a "recursive" way
- let's say you write a function

  "add2Numbers(Node* l1, Node* l2, int carry)"

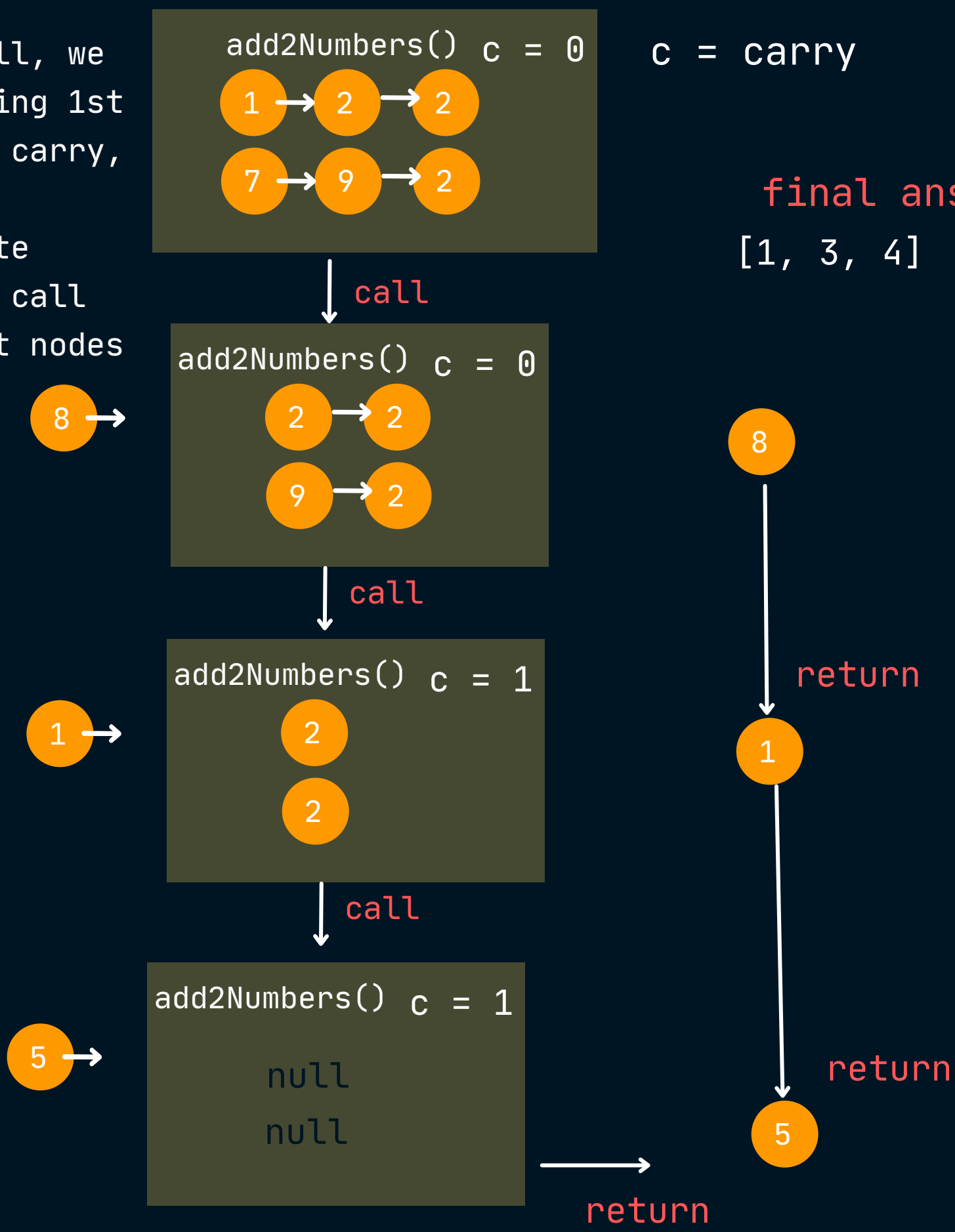
- what above function does in every call
  1) add sum of curr nodes
  2) calculates carry
  3) call for next nodes with obtained carry

I'm providing you a pictorial explanation (step by step), try to come up with full fledged solution on you own.

Pictorial explanation, next →

Intuition

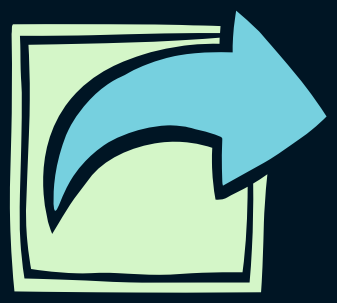
each call, we are adding 1st nodes + carry, then we calculate carry & call for next nodes



```cpp
class Solution {
public:


    ListNode* addNumbersHelper(ListNode* l1, ListNode* l2, int carry) {

        if(!l1 && !l2) {
            if(carry) {
                return new ListNode(carry);
            } else {
                return NULL;
            }
        }

        if(!l1) {
            carry += l2->val;
            l2->val = carry % 10;
            l2->next = addNumbersHelper(l1, l2->next, carry/10);
            return l2;
        }
        if(!l2) {
            carry += l1->val;
            l1->val = carry % 10;
            l1->next = addNumbersHelper(l1->next, l2, carry/10);
            return l1;
        }


        carry += (l1->val + l2->val);
        l1->val = carry % 10;
        l1->next = addNumbersHelper(l1->next, l2->next, carry/10);
        return l1;

    }

    ListNode* addTwoNumbers(ListNode* l1, ListNode* l2) {
        return addNumbersHelper(l1, l2, 0);
    }
};
```

Leave a Like

Comment if you love posts like this, will motivate me to make posts like these



Share, with friends