③-2 adding to DLL

① Palindrom?
② largest palindrom
③ Doubly linked List
④ delete from DLL
basics, Abstract ← ⑤ LRU cache

P1 check if a given linked list is palindrom. SC: O(1) do not copy to an array!

ex $\overset{\times}{1} \to 3 \to 3 \to 1 \to \overset{\times}{5} \to$ null ans = false

ex $1 \to 3 \to \cancel{5} \to 3 \to 1 \to$ null ans = true

ideas? $3 \leftarrow 1$

1	3	5	3	1

going one back is costly in single LL

2 min

O(n) ← ① find the midle of array & split from midle
+
O(n) ← ② reverse the 2nd half
+
O(n) ← ③ Compare nodes until one is not eq? → false
or one is null → true
true

TC = O(3n) = O(n)

SC: O(1)

code for step 3

```
while (cur1 != null && cur2 != null) {
    if (cur1.data != cur2.data) ret false;   fix()
    cur1 = cur1.next;
    cur2 = cur2.next;
} fix()
ret true;
```

optional fix, revert changes to original state

h1, h2
fix() {
- reverse 2nd half again
- Connect them again
- be aware of odd len case
}

even? your assignment

P2 find the length of the longest odd length palindromic list in the given linked list.

SC: O(1)
do not copy to an array!

ex 1→2→1→1→1→2→3→2→null
ans=5

2 min

idea 1 $O(n^3)$

{1, 2, 1, 1, 1, 2, 3, 2}

```
for i=0 → n-1
    for j=i → n-1
        check( a[i...j] is palindrom) ← O(n)
```

n

idea 2
strings

starting from midle and try to expand
midle ← n
TC of expanding ← n
overall $O(n^2)$ ← optimized

x ← 1 → 2 → 1 → 1 → 1 → 2 → 3 → 2 → null
prev cur next

very similar to reverse

```
ans=0
next=null
prev=null
cur=head
while(cur!=null){
    next=cur.next
    len=expand(prev,next)
    ans=Max(ans,len)
    cur.next=prev
    prev=cur
    cur=next
}
reverse()(?) //optional reverse
ret ans
```

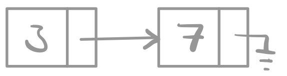
ans
~~1~~
3

```
int expand(prev,next){
    x=prev    y=next
    len=0
    while(x!=null && y!=null){
        if(x.data==y.data) len++
        else break
        x=x.next
        y=y.next
    }
    ret 2*len+1
}
```

next → O(n)
x y

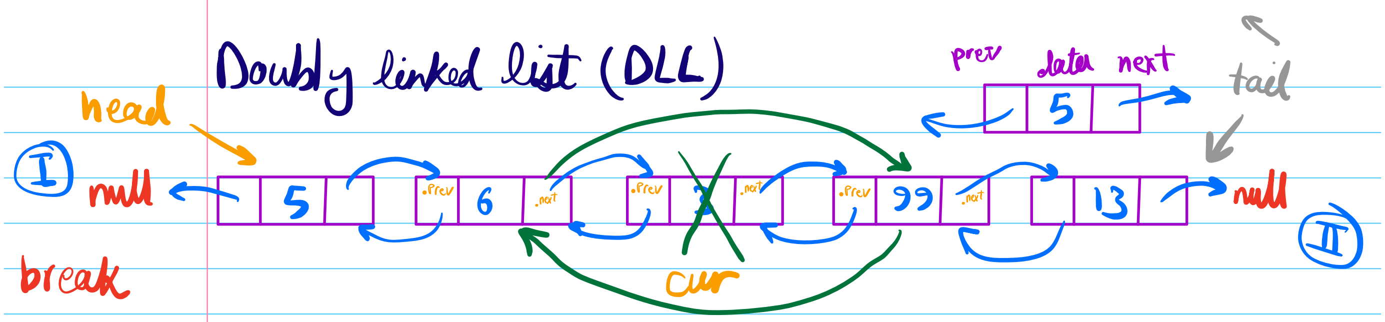
TC: $O(n^2)$
SC: O(1)

simple LL



optional

Doubly linked list (DLL)



P3 Delete the first occurrance of a given data in DLL. → X
if not present do nothing. return head.

2 min

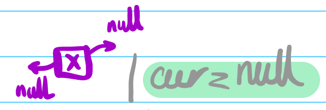
```
cur = head
while(cur != null){
    if (cur.data == X) break
    cur = cur.next
}
⚠1 if (cur==null) ret head
   // here and after I know cur.data == x
⚠2 if( cur.prev==null && cur.next==null) ret null
⚠3 else if (cur.prev==null){
    ① cur.next.prev = null
    ② head = head.next
    ret head
}
⚠4 else if(cur.next== null){
    ① cur.prev.next = null
       tail = tail.prev
    ret head
}
happy path! else {
    ① cur.prev.next = cur.next
    ② cur.next.prev = cur.prev
    ret head
}
```

search

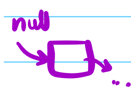
both empty list or data not found

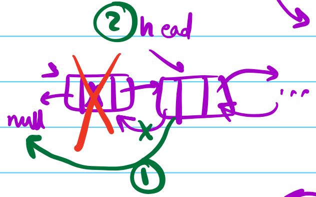
if data not found or empty list

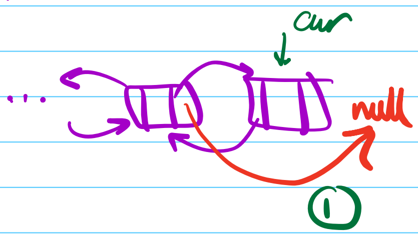
⚠
1 empty list ✓
2 only one node ✓
3 del head ✓
4 del last node ✓



optional free memory before ret







p = cur.prev
n = cur.next

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

what is a cache?
what is LRU?
what are other cache polices?

LLD+HLD

Least recently used

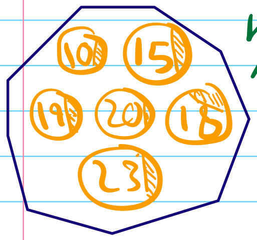
P4 LRU Cache

Abstract Given a running stream of integers & a fixed memory size 100 = M [i.e. any Data structure with SC=O(M)]

for each int intake, the DS should have most recent M items.

ex 10, 15, 19, 20, 18, 23, 20, 19, 17, 17, 10

O(1) lookup



hash Map ← ① look up op. if(new data exist) ?

② LRU ⟵ traking linear D.S.

like Q → linked List

head → DLL ← tail

for each new item

O(1) look up

19 already exist
O(1) ① delete,
② add to end

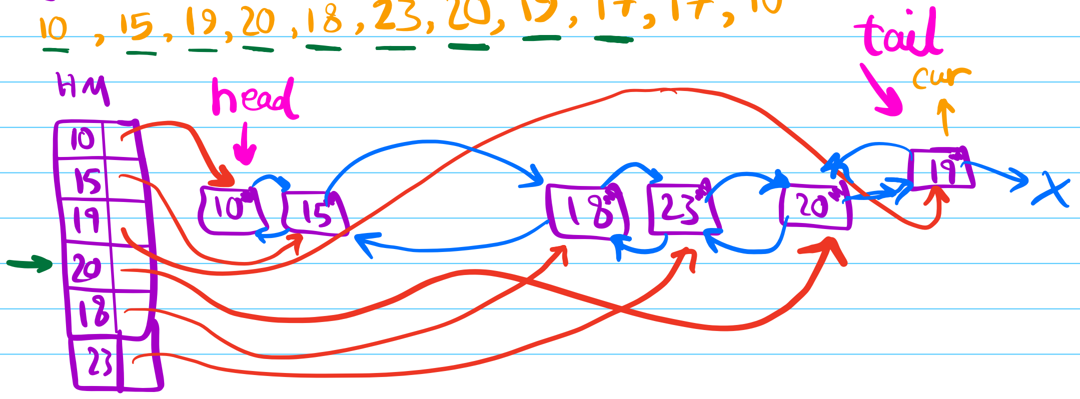
doesn't exist
① if full delete first node → Q O(1)
garanteed to be least recently used ←
② Insert to the end

HM<int, Node>

M=6

10, 15, 19, 20, 18, 23, 20, 19, 17, 17, 10

O(1)



3 letters

$\underline{abaab}$ $O(n^2) \leftarrow$ $\underbrace{aba}_{①}\underbrace{cb}_{①\,①}$

$n/_2 \geq n_1$

$\overline{aaa \ldots b \ldots aaa} \, \underline{c \ldots b \ldots c}$ $O(n^3)$

$O(n_1^2)$ $n/_2 \geq n_2$

abc

finding largest palindrom

$\left\{ \begin{array}{l} \text{hash map} \\ \text{DP} \end{array} \right.$ $O(n^2)$