Imp. Even-Odd Merge

7.10 7/7/22

There are two O(n), O(1) methods

(I) Have a left and right pointers that each points to the ends of the list.

- Then ~~change the~~ make the ~~poo~~ left pointer point to the next even and make the right pt'r point to the ~~left~~ left subsequent node. And make the odd pos. node point to Null.

```
left.next = left.next.next
right.next = left.next
right.next.next = None
```

(II) - Make a dummy head and even and odd pointers. Make the ~~pointers~~ node point to the next even/odd nodes.

- When it reached the end, make the last even node point to the first odd.

(EPI Sol'n)

1/2

$\boxed{7.10}$ Evaluation

- I was moving back and forth w/ the two sol'n method.

- ~~The idea th~~

  $\hookrightarrow$ I eventually stack w/ method $\boxed{\text{II}}$ but couldn't come up w/ the idea of having two separate even and odd dummy node and link them (last even $\rightarrow$ first odd) after the loop was finished.

- Didn't do boundary /edge cases. $\rightarrow$ what if List was empty? one elem? two elm?

- If working w/ even/odd lists, consider separating them.
- Make nodes point to Null each step, then the last elem will point to Null when the loop is finished.

2/2

~~7/7/22~~ $\boxed{7.10}$

I/O : Linked List

- Return a list w/ ~~[illegible]~~ nodes at even position first then odd.
- starts at 0

- I probably have to use a dummy node
- The Naive approach is to keep track of the steps taken by incrementing a variable while traversing through the list.
  - Then checking if a node is in even/odd position by (x % 2 == 0).

⟶ How can I rearrange the list?
  ↳ sentinel /dummy

⟹

1/6

step = 0

$$\begin{tikzcd}
\boxed{\text{Dum}} \arrow[r] & l_0 \arrow[r] & l_1 \arrow[r] & l_2 \arrow[r] & l_3 \arrow[r] & l_4 \arrow[r] & X
\end{tikzcd}$$

$\uparrow$ pointer

Start @ dummy

$\uparrow$ pointer

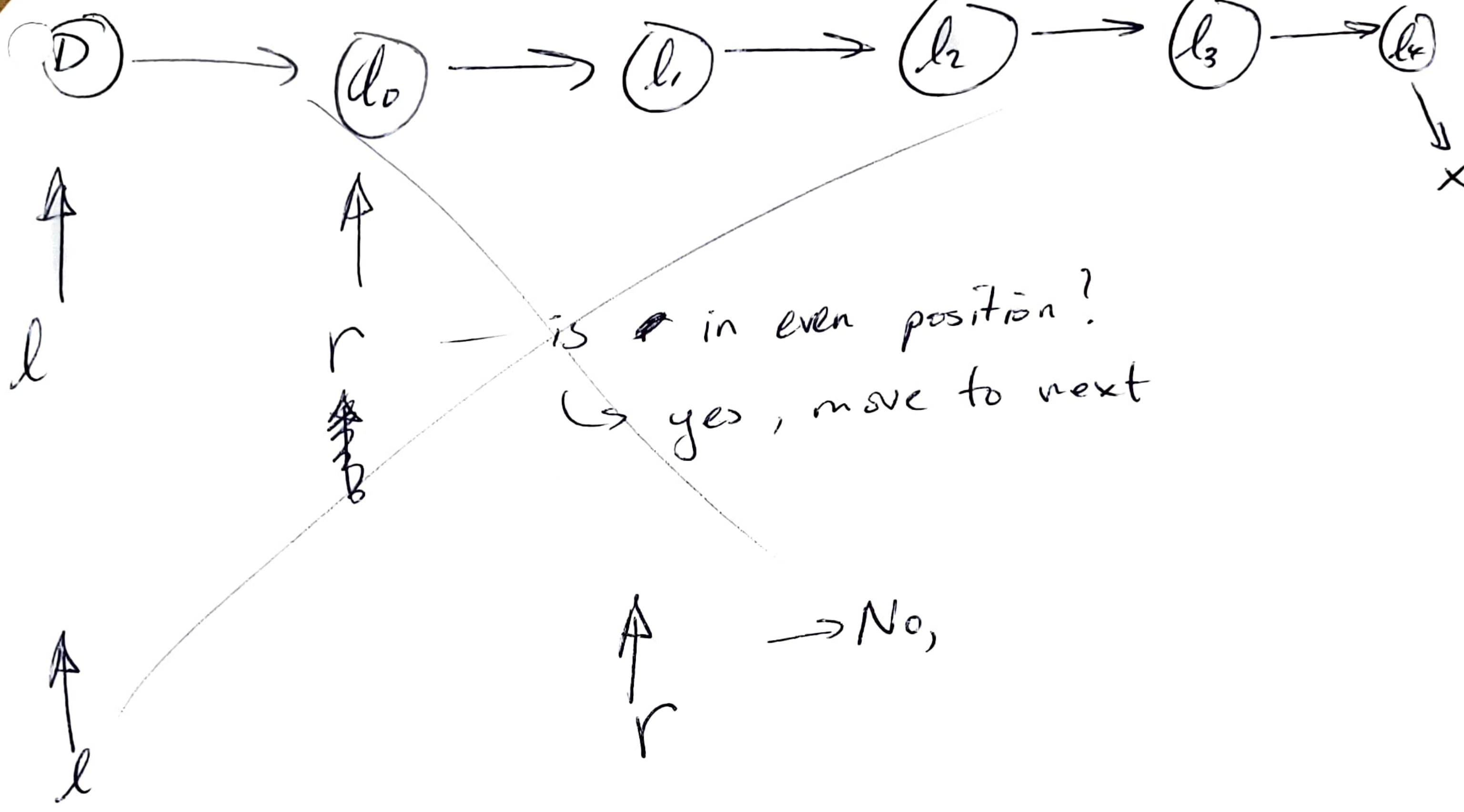
step = 1 $\rightarrow$ if step % 2 $\neq 0$, move to next

$\rightarrow$ Should I use fast/slow pointer? $\rightarrow$ No

$\uparrow$ ptr

step = 2 $\rightarrow$ if step % 2 = 0,

$\rightarrow$ I should use ~~left~~ left_end pointer and another pointer that iterates through the list.



is ~~[illegible]~~ in even position?
↳ yes, move to next

→ No,

↳ I need to check the next node or prev node

$$\begin{tikzcd}
D \arrow[r] & l_0 \arrow[r] & l_1 \arrow[r] & l_2 \arrow[r] & l_3 \arrow[r] & l_4 \arrow[r] & X
\end{tikzcd}$$

~~Is the prev~~

$$\begin{tikzcd}
D \arrow[r] & l_0 \arrow[r, "\times"] \arrow[rr, bend left] & l_1 \arrow[r] & l_2 \arrow[r] & l_3 \arrow[r] & l_4 \arrow[r] & X \\
 & {} \arrow[u] & {} \arrow[u] & & & &
\end{tikzcd}$$

$$\begin{tikzcd}
D \arrow[r] & l_0 \arrow[rr, bend left] & l_1 \arrow[r] & l_2 \arrow[r, "\times"] \arrow[rr, bend left] & l_3 \arrow[r] & l_4 \arrow[r] & X \\
 & & & {} \arrow[u] & {} \arrow[u] & &
\end{tikzcd}$$

$$\begin{tikzcd}
D \arrow[r] & l_0 \arrow[rr, bend left] & l_1 \arrow[r] & l_2 \arrow[rr, bend left] & l_3 \arrow[r] & l_4 \arrow[r, "\times"] \arrow[llll, bend left] & X \\
 & & & & & {} \arrow[u] & {} \arrow[u]
\end{tikzcd}$$

$$\begin{tikzcd}
D \arrow[r] & l_0 \arrow[rr, bend left] & l_1 \arrow[r, "\times"] \arrow[rr, bend right] & l_2 \arrow[rr, bend left] & l_3 \arrow[r, "\times"] \arrow[rr, bend right] & l_4 \arrow[lll, bend left] & X \\
 & & {} \arrow[u] & {} \arrow[u] & {} \arrow[u] & &
\end{tikzcd}$$

$$\begin{tikzcd}
D \arrow[r] & l_0 \arrow[rr, bend left] & l_1 \arrow[rr, bend right] & l_2 \arrow[rr, bend left] & l_3 \arrow[rr, bend right] & l_4 \arrow[lll, bend left] & X \\
 & & & & {} \arrow[u] & {} \arrow[u] &
\end{tikzcd}$$

$\boxed{17\text{min}}$

4/6

There are two steps to this:

① If the previous node is in even position, make it point to the next even node.

② When all even nodes are found, make the last even node point to the first odd node.

---

① If the curr. node is odd, make the prev. node point to curr.next

↳ ~~take~~ two steps increment?

↳ when curr == None, → $\otimes$

→ I might have to know the length of the list. → $\otimes$

→ I should keep track of the first odd node and traverse from there.

5/6

- When we reach the last even node, make it point to the first odd node.
  - Then make it point to x.next.next

    ↳ What happen when I reach the end?

    ↳ Either the last odd point to Null or the last even.

→ 32 min Quit

Imp. Stack w/ Max API

$\boxed{8.1}$ 7/7/22

Are we working w/ arrays, linked lists?

So the input is LL and output is the max value w/in that list.
This might mean for every values that are popped, keep track of the max by taking an item and comparing w/ all others.
This will take O(n) time O(1) space

```
def f(l):
    max = 0
    while l:
        [illegible]
        if l.pop() > max:
            max = l.pop()
        l.pop()
    return max
```

6 min

→ Had to use OOP for this problem