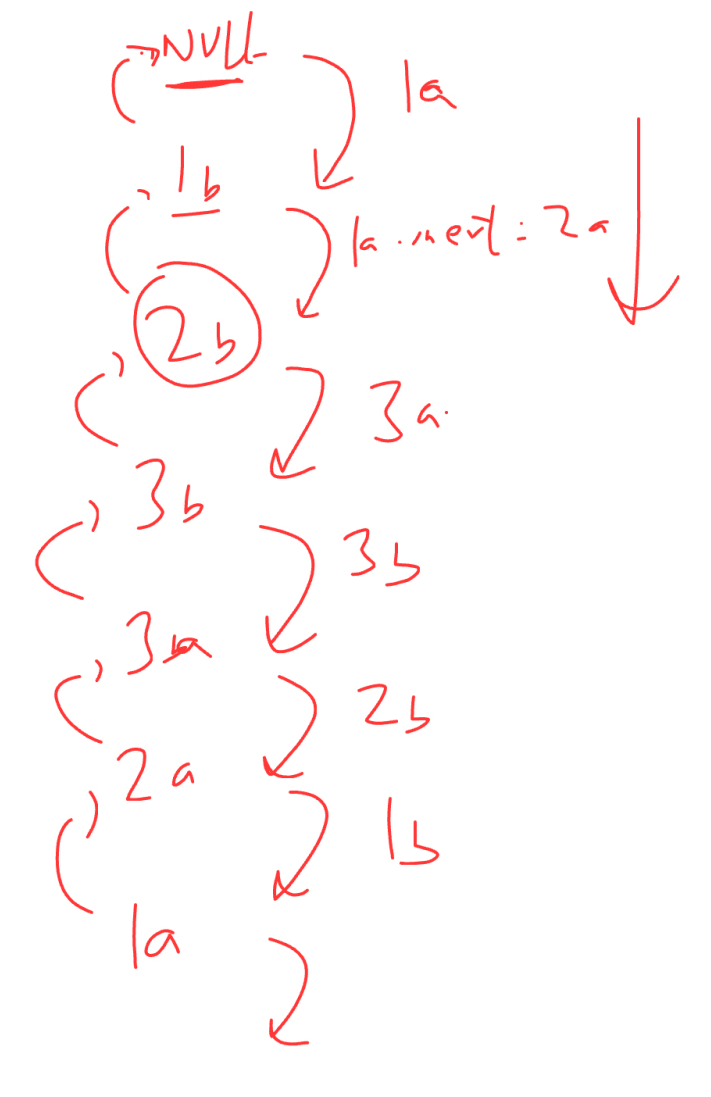
head

1a → 2a → 3a → 3b → 2b → 1b → NULL

mirror


NULL
1a
1b
1a.next = 2a
2b
3a
3b
3b
3a
2b
2a
1b
1a

Rec.

TC: O(n)

AS: O(n). ⟸

1st half: 1a → 2a → 3a → NULL

Reverse of 2nd half: 1b → 2b → 3b → NULL

1a → 2a → 3 → 2b → 1b → NULL

1st half: 1a → 2a → NULL

Rev. of the 2nd half: 1b → 2b → 3 → NULL

Step-1: Break the LL around its centre.

Step-2: Reverse the 2nd half of the LL.

Step-3: Node-by-node comparison.

TC: O(n + n/2 + n) = O(n)

AS: O(1).

mid h2

1a → 2a → 3a ⨯ 3b → 2b → 1b → NULL

h ↳ 1a → 2a → 3a → NULL

h2 ↳ 3b → 2b → 1b → NULL ⟹ h2 ↳ 1b → 2b → 3b → NULL

h ↳ 1a → 2a → 3a → 3b → 2b → 1b → NULL

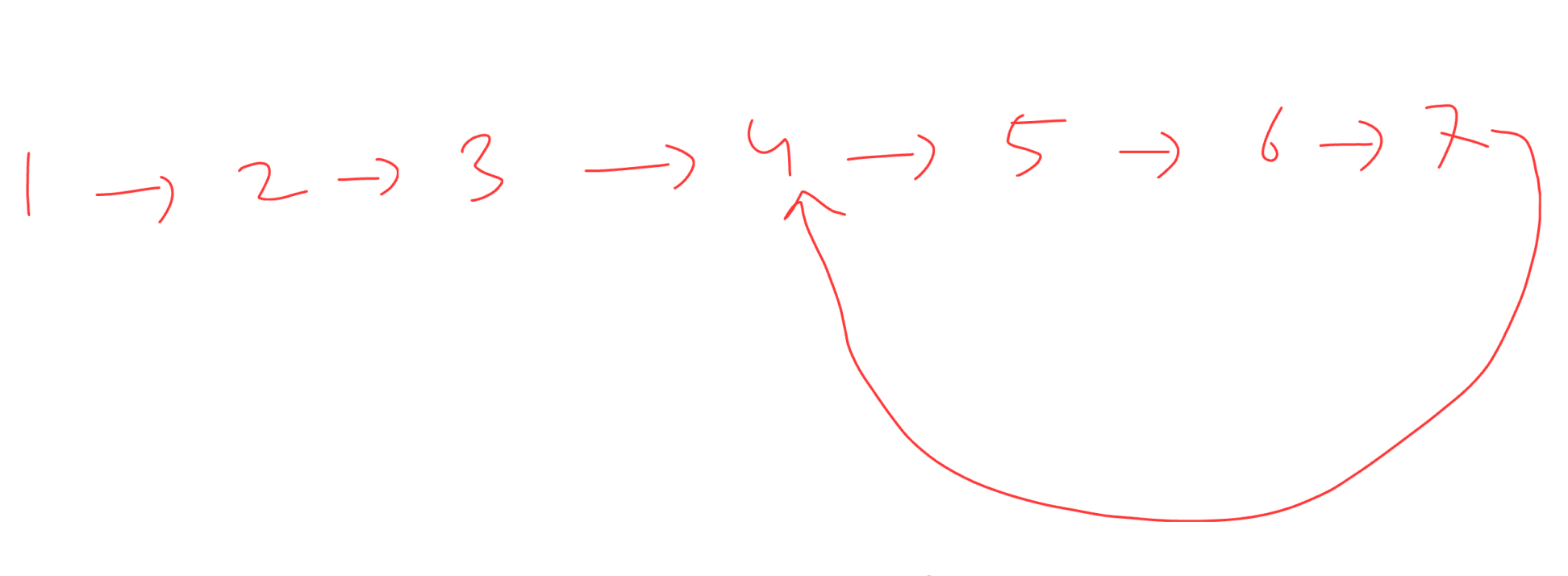
mid

h ↳ 1a → 2a → 3a → 3b → NULL

h2 ↳ 1b → 2b → NULL

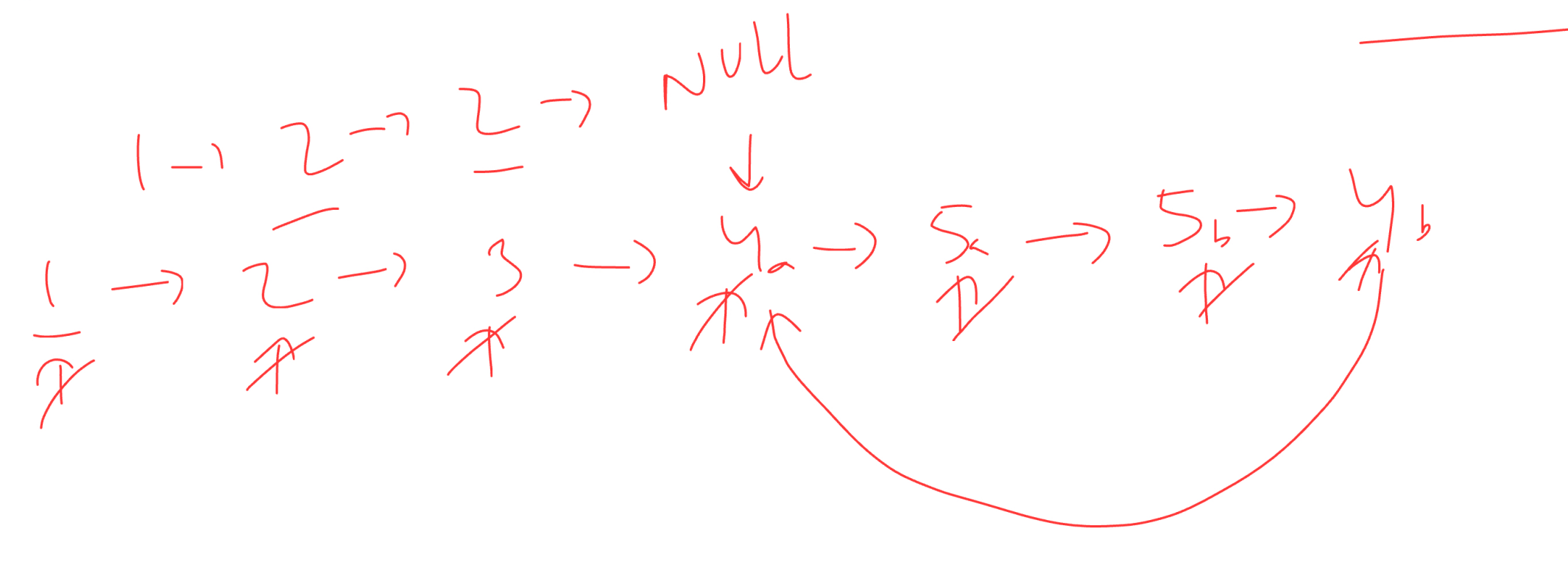
mid

1a → 2a → 3 → 2b → 1b → NULL



O/P: [1, 2, 3, 4, 5, 6, 7]

set < Node* >


1 → 2 → 2 → NULL
1 → 2 → 3 → 4a → 5a → 5b → 4b

4b
5b
5a
4a
3
2
1

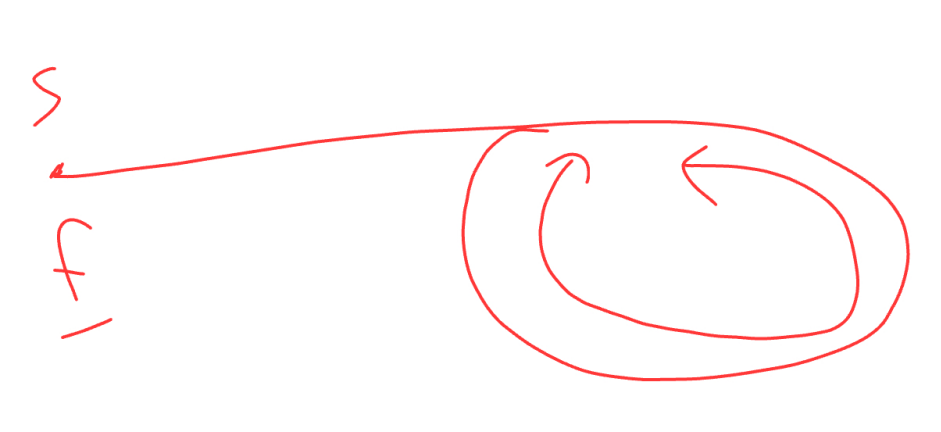
O/P: [1, 2, 3, 4, 5, 5, 4]

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

HashSet-based approach:

TC: O(n)

AS: O(n). ⟸



time t: slow: x pos.
fast: x+1 pos.

time (t-1): slow: (x-1) pos
fast: (x+1) - 2 = (x-1) pos

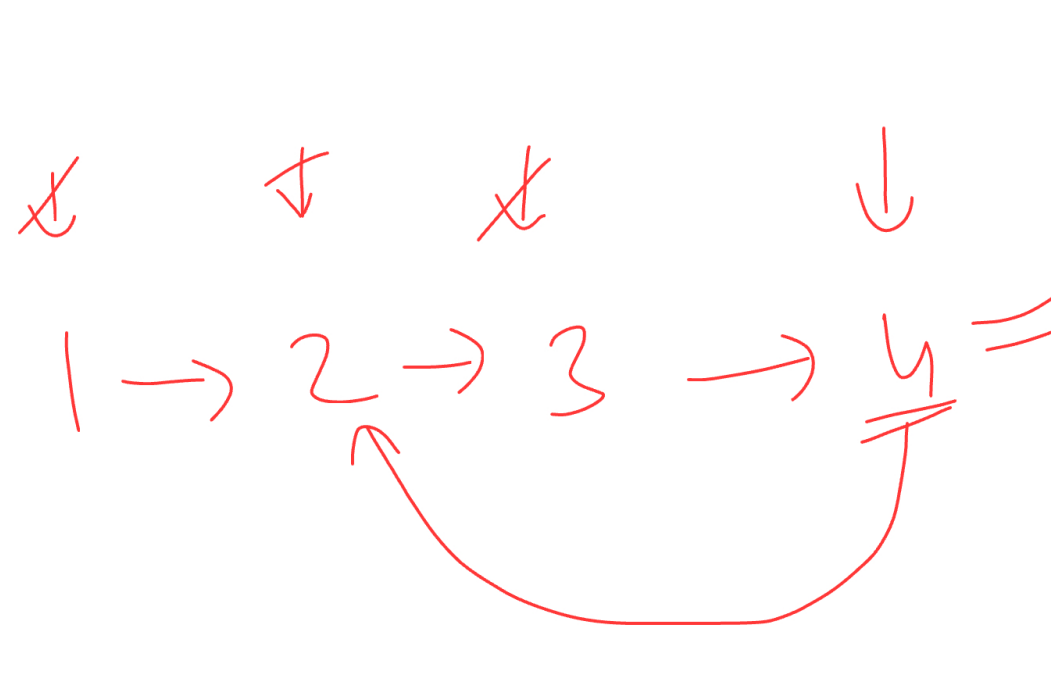
case-1: Linear LL.
⟹ fast ptr will reach the end of the LL.

case-2: Circular LL.
⟹ fast & slow ptrs. will collide at some point.

1?

I/P: 1 → 2 → 3 → 4

O/P: 1 → 2 → 3 → 4 → NULL.



1st node pointing to something which is already visited.

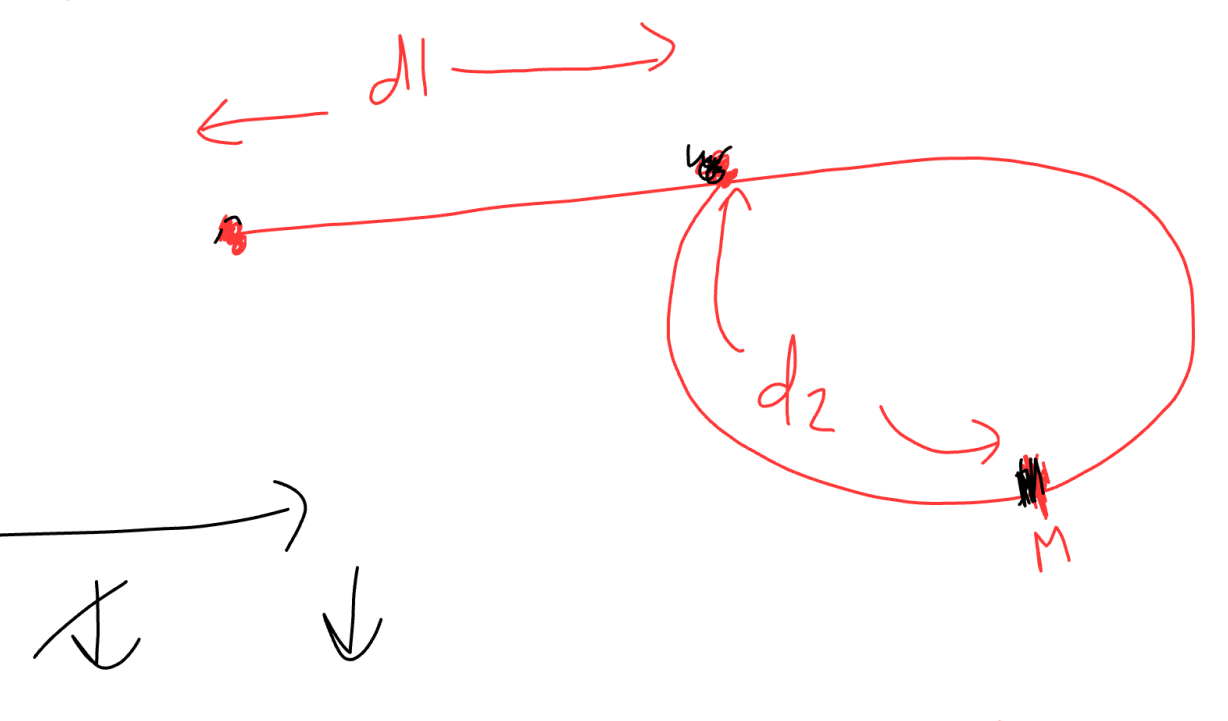
Mark 4->next = NULL

Set < Node* > vis = { 1, 2, 3 }

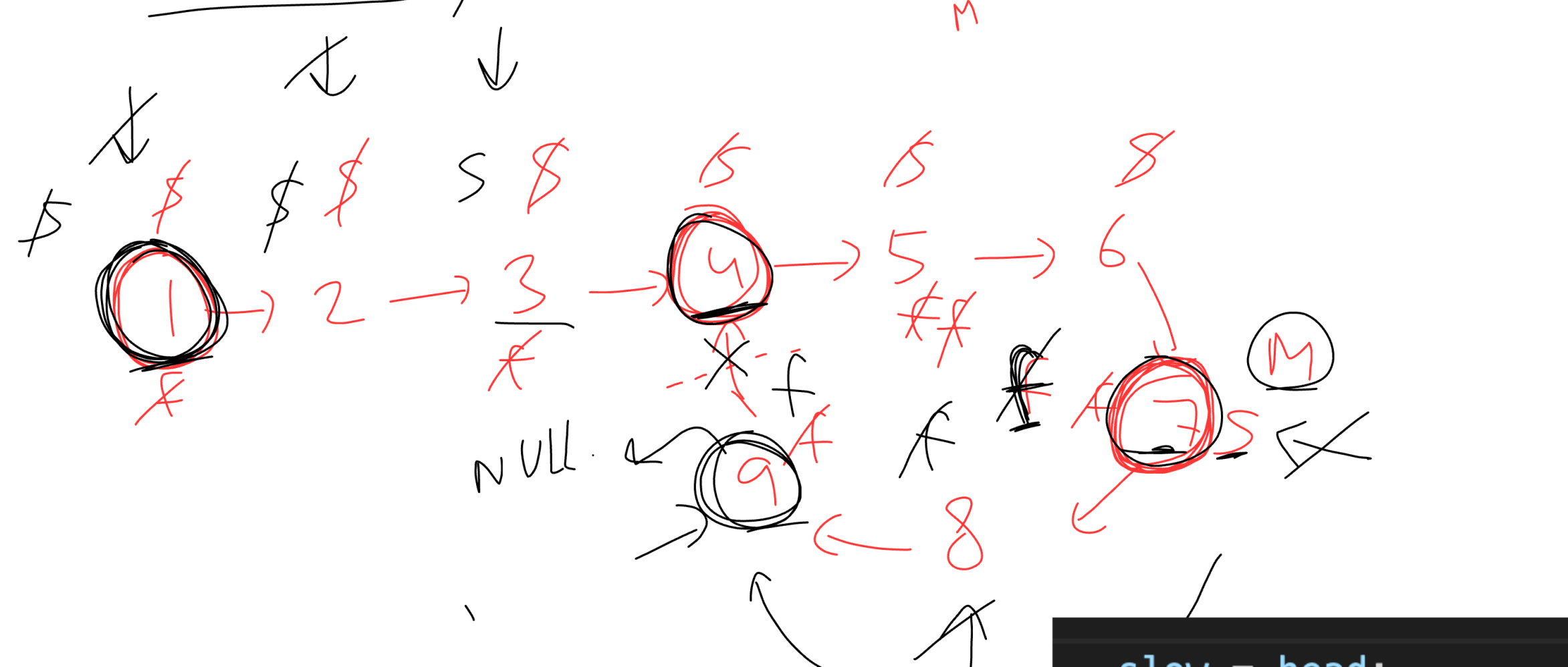
TC: O(n)

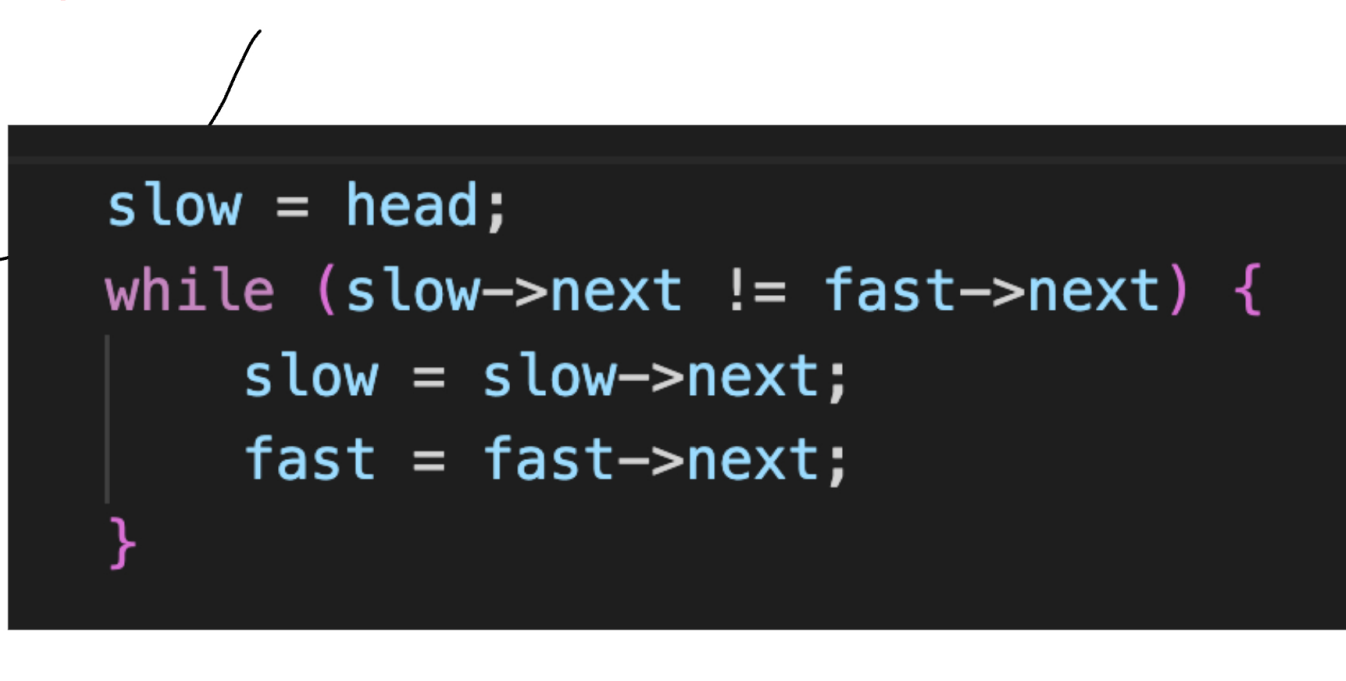
AS: O(n)

Floyd's Cycle Detection

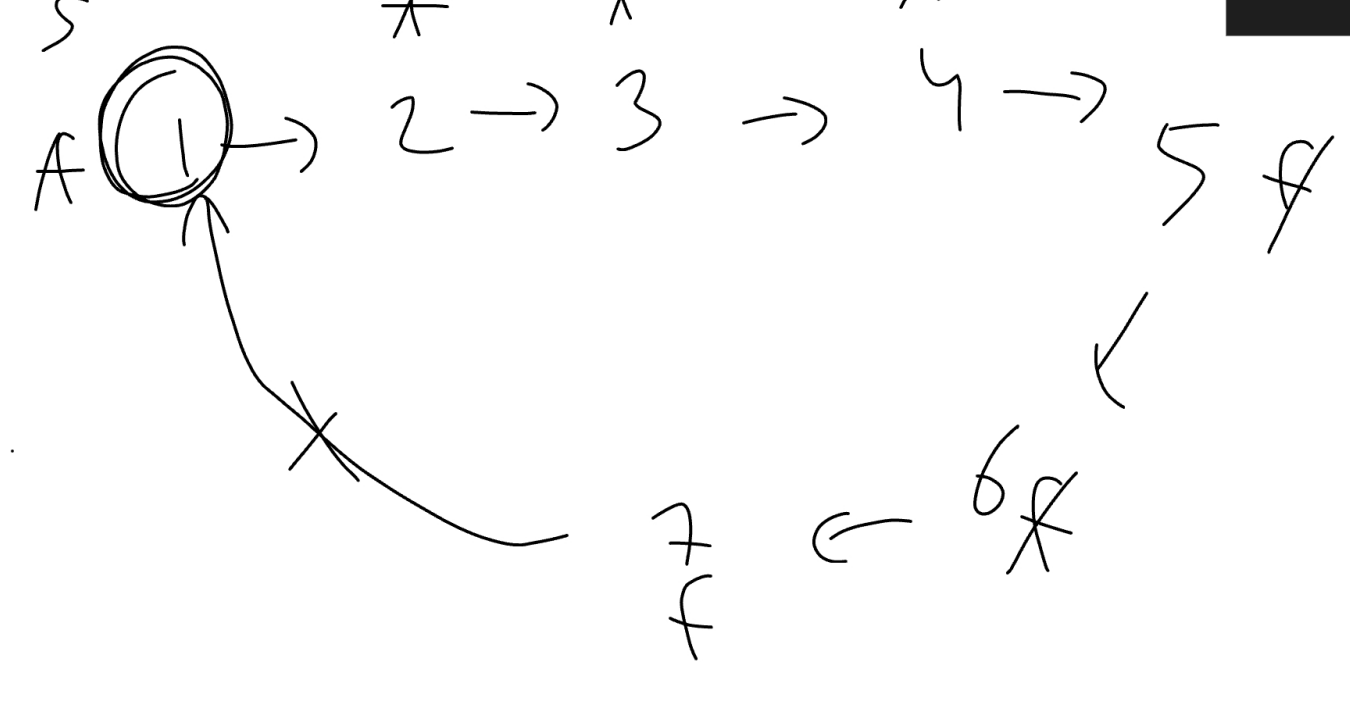


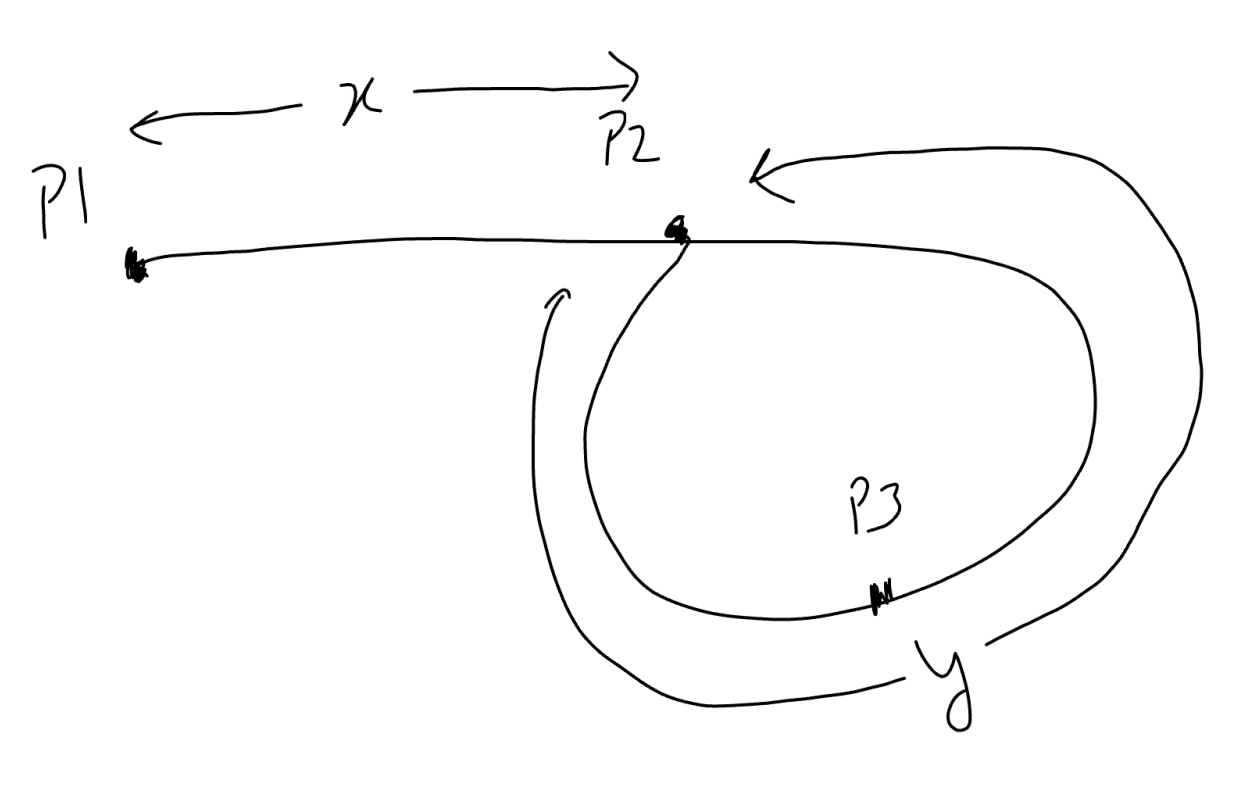
d1 = d2 #


1 → 2 → 3 → 4 → 5 → 6
8
NULL
M



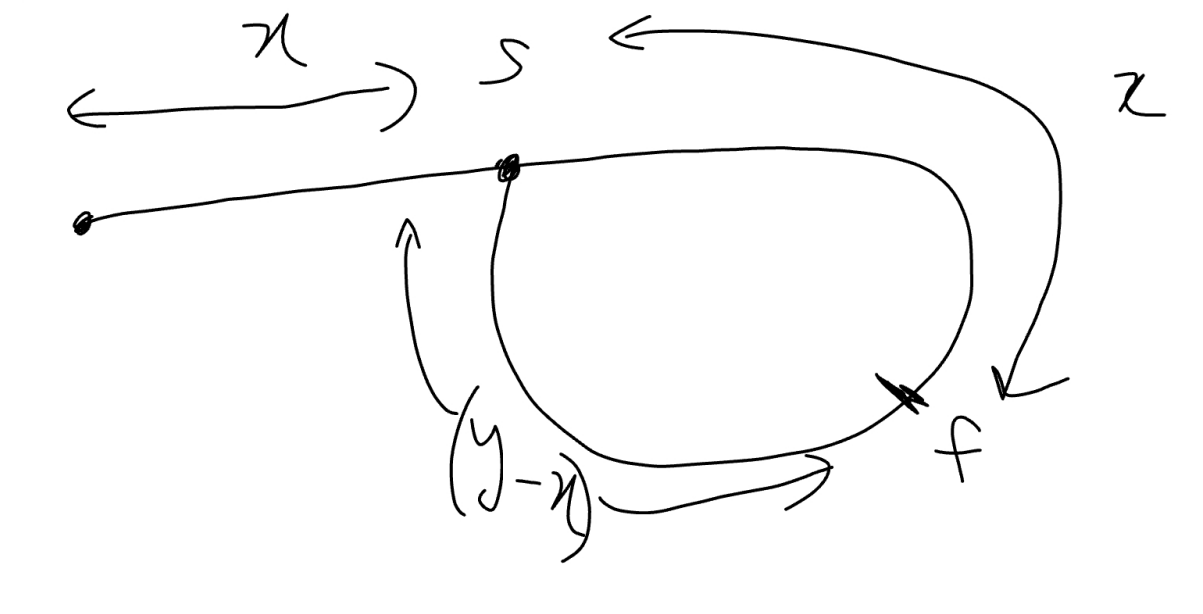
```
slow = head;
while (slow->next != fast->next) {
    slow = slow->next;
    fast = fast->next;
}
```


M
S
1 → 2 → 3 → 4 → 5
6
7
f



y > x

when slow is at P2, it has travelled a dist of x.
⟹ fast has travelled a dist of 2x
⟹ fast is x steps into the loop

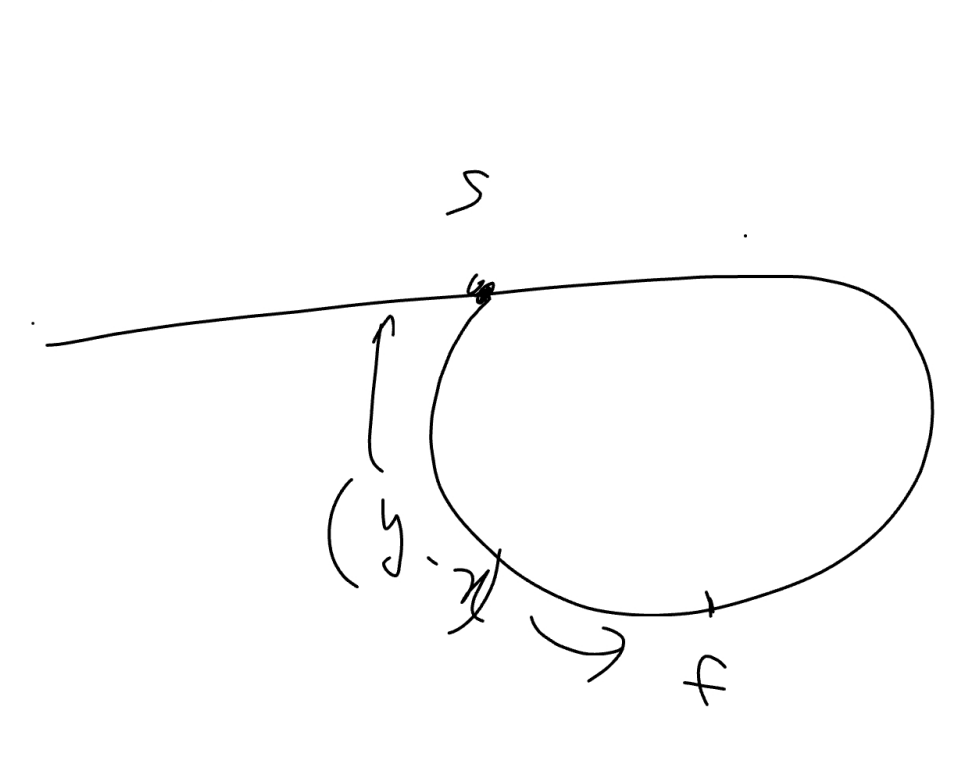


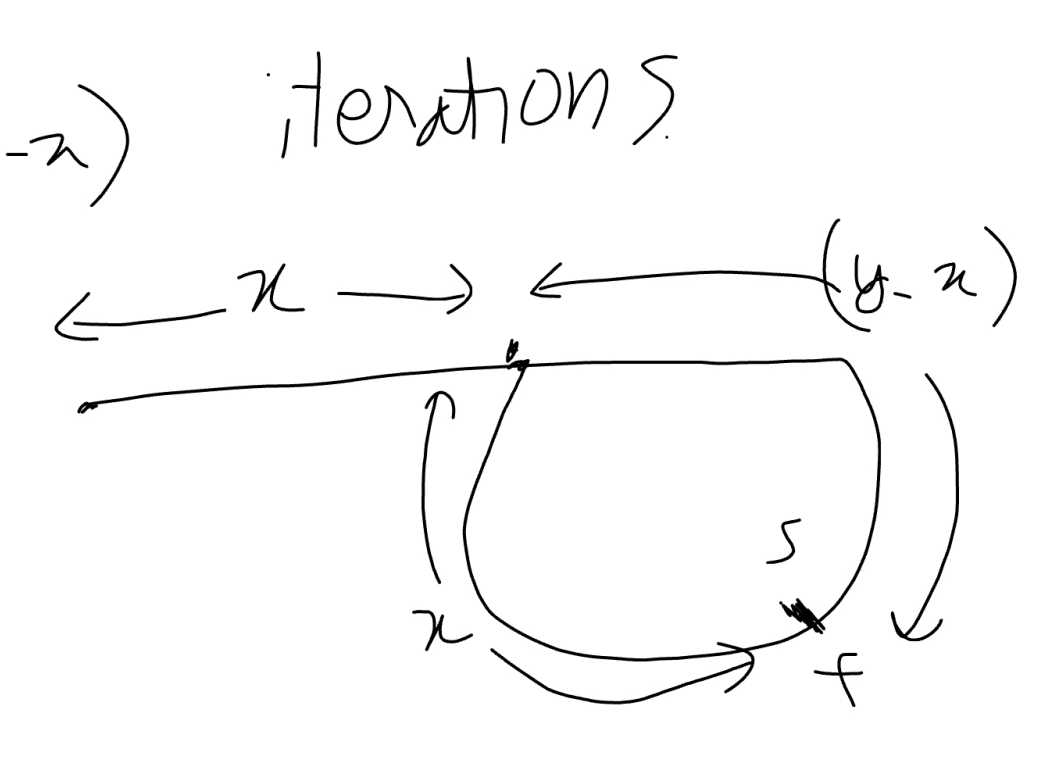
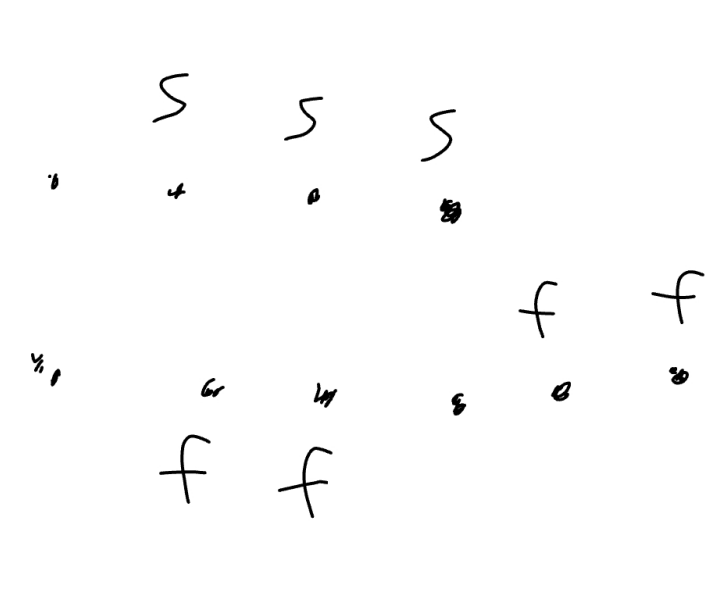
Now, slow is x steps behind fast.
⟹ fast is (y-x) steps behind slow #

fast is catching up slow at the rate of 1 step per iteration.

⟹ Initial dist b/w fast & slow = (y-x)
fast is catching up 1 step/iter
⟹ After (y-x) iterations, slow & fast will be at the same point

Initial:



After (y-x) iterations