

Linked List

$O(1)$

fast

finite only 3 steps

Singly Linked List → (Integer) (JVM)

Diagram illustrating a singly linked list structure:

- The list consists of three nodes.
- Each node is represented as a box containing an integer value (10, 20, or 30) and a reference to the next node.
- The reference is labeled "D" (next) and points to the next node's address.
- The last node's reference is labeled "tail" and points to "null".
- Annotations show addresses: 20000 for the first node, 30000 for the second, and 32000 for the third.

Node (Data , Pointer(Next))  
 or  
 Address

- \* Each node has some data & the pointer to its next node.
- \* The first node is the head node & the only way to traverse the list.

- \* The last node has no next node, so its next pointer is "null".

Important Operations:

```
class Node {  
    String data;  
    SLL Node next;  
    DLL Node prev;  
}
```

① Insert  
② Delete  
③ Display

Operations and their parameters:

- Insert:
  - @ Head / Front
  - @ Tail / Back
  - After specific
- Delete:
  - @ Head
  - @ Tail
  - Target Node

(newNode)

$n = h$

- Case 1: Single Node

Diagram: A node labeled 1 with a self-loop arrow labeled `next = null`. A blue arrow labeled `h` points to the node.

Case 2: Multiple Nodes

Diagram: Three nodes labeled 1, 2, and 3. Node 1 points to node 2, which points to node 3, which points to `null`. A blue arrow labeled `h` points to node 1. To the right, it says `this.head.next`.

Diagram: Four nodes labeled 1, 2, 3, and 4. Node 1 points to node 2, node 2 points to node 3, and node 3 points to node 4. Node 4 points to `null`. A blue arrow labeled `h` points to node 1. To the right, it says `(h = h.next;)`.

① `h.next == null` ✓ Case 1  
 ② `h = h.next;` ✓ Case 2

- \* Delete the middle node of a linked list .
- \* Detect cycle in a linked list.
- \* Intersection of two linked lists.
- \* Add two numbers with carry using linked lists.
- \* Merge two sorted linked lists.
- \* Reverse a linked list.
- \* Reverse a linked list in groups of k.
- \* Linked List Palindrome.

Diagram illustrating the intersection of two linked lists:

- List 1 (head):** Nodes 1, 2, 3, **null**. Node 3 is labeled **(temp)**.
- List 2 (temp):** Nodes 1, 2, 1, **null**.
- Merged List:** Nodes 1, 2, 1, 1, **null**. Node 3 of List 1 points to node 1 of List 2.

To the right, a stack diagram shows the state of pointers **p1** and **p2**:

2	3
2	2
1	1

Arrows indicate the popping of values from the stack: **p1** starts at 3 and points to 2, then to 1; **p2** starts at 2 and points to 1.

$p_1 \leftarrow \text{null}$     $p_2$  (2)  $\rightarrow$  (4)  $\rightarrow$  (6)  $\rightarrow$  (8)  $\rightarrow$  null  
 $p_1$ : head2    $p_2$     $p_2$     $p_2$

if ( $p_1.\text{data} < p_2.\text{data}$ )  
 move one  $p_1 = p_1.\text{next};$   
 else if ( $p_2.\text{data} < p_1.\text{data}$ )  
 move other  $p_2 = p_2.\text{next};$   
 else (match found)

→ node  
 (2) → (4) → (6) →  
 head  
 tail → →

{ Store in  
new list  
as moveboth }      node.data (p2.data)  
 $p1 = p1.next$   
 $p2 = p2.next$

head → null      head == null  
tail → null

( head = tail = newNode )

The diagram illustrates the insertion of a new node  $t$  into a linked list. It shows two states:

- Initial State:** A node labeled  $2$  is highlighted with a blue circle. A variable  $h$  (highlighted with a blue circle) points to this node. The label  $\text{newNode}$  is written below the node.
- Final State:** The node  $2$  is now highlighted with a red circle. A variable  $t$  (highlighted with a red circle) points to it. The label  $\text{newNode}$  is written below  $t$ . The original node  $2$  is now part of the list, with its original value  $2$  and the label  $t$  below it. The node  $6$  is highlighted with a blue circle. A variable  $\text{tail} = \text{newNode}$  is written above the node  $6$ , and the label  $t$  is written below it.

- \* Middle of linked list
- \* Delete the middle node
- \* Cycle Detection

Two pointer Approach

Fast Poi  
 $\text{prev} = \text{null}$ ,  
 $\text{fast} = \text{head}$ ,  
 $\text{slow} = \text{head}$

while ( $\text{fast} \neq \text{null}$ )  
 $\text{fast}.\text{next}!$

head → **1** → **2** → **3** → **4** → **5** → **6** → null  
 s      s      s p    **s**  
 prev = slow  
 slow = fast  
 fast = f \* n;  
 prev = s.next;  
 slow = null;  
 slow = fast  
 fast = fast  
 }  
 return slow

if ~~slow == fast~~ true or false.

Cycle

fast.next != null  
slow = slow.next  
fast = fast.next