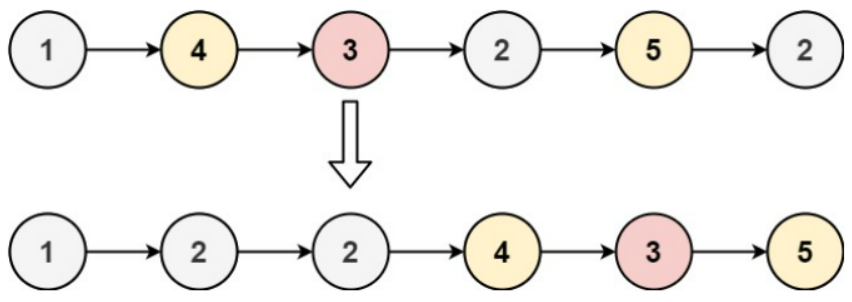
<https://leetcode.com/problems/partition-list/>

Given the head of a linked list and a value x, partition it such that all nodes **less than** x come before nodes **greater than or equal** to x.

You should **preserve** the original relative order of the nodes in each of the two partitions.

**Example 1:**



Input: head = [1,4,3,2,5,2], x = 3

Output: [1,2,2,4,3,5]

**Example 2:**

Input: head = [2,1], x = 2

Output: [1,2]

**Constraints:**

* The number of nodes in the list is in the range [0, 200].
* -100 <= Node.val <= 100
* -200 <= x <= 200

**Attempt 1: 2023-02-19**

**Solution 1:  Priority Queue Solution (30 min)**

/\*\*

\* Definition for singly-linked list.

\* public class ListNode {

\* int val;

\* ListNode next;

\* ListNode() {}

\* ListNode(int val) { this.val = val; }

\* ListNode(int val, ListNode next) { this.val = val; this.next = next; }

\* }

\*/

class Solution {

public ListNode partition(ListNode head, int x) {

ListNode dummy1 = new ListNode();

ListNode dummy2 = new ListNode();

dummy1.next = head;

dummy2.next = head;

ListNode iter1 = dummy1;

ListNode iter2 = dummy2;

while(head != null) {

if(head.val < x) {

iter1.next = head;

iter1 = iter1.next;

} else {

iter2.next = head;

iter2 = iter2.next;

}

head = head.next;

}

// Set to 'null' to avoid: Error Found cycle in the ListNode

iter2.next = null;

iter1.next = dummy2.next;

return dummy1.next;

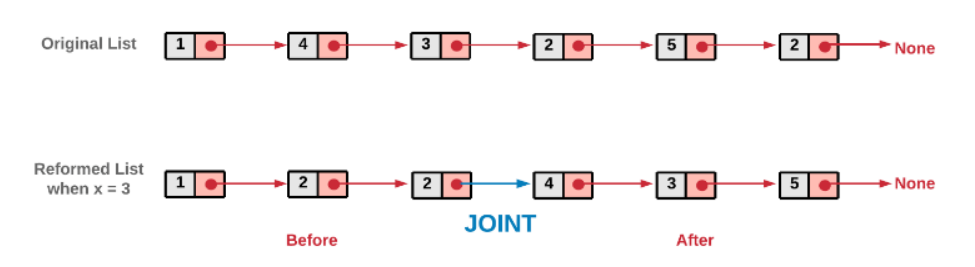
}

}

**Refer to**

<https://leetcode.com/problems/partition-list/solutions/212538/partition-list/>

The problem wants us to reform the linked list structure, such that the elements lesser that a certain value x, come before the elements greater or equal to x. This essentially means in this reformed list, there would be a point in the linked list before which all the elements would be smaller than x and after which all the elements would be greater or equal to x. Let's call this point as the JOINT.



Reverse engineering the question tells us that if we break the reformed list at the JOINT, we will get two smaller linked lists, one with lesser elements and the other with elements greater or equal to x. In the solution, our main aim is to create these two linked lists and join them.

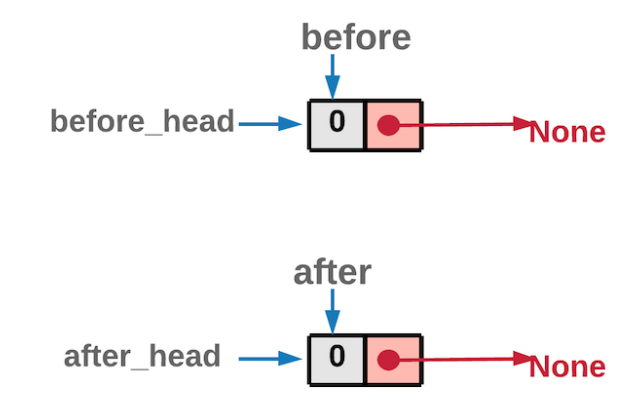
#### **Approach 1: Two Pointer Approach**

**Intuition**

We can take two pointers before and after to keep track of the two linked lists as described above. These two pointers could be used two create two separate lists and then these lists could be combined to form the desired reformed list.

**Algorithm**

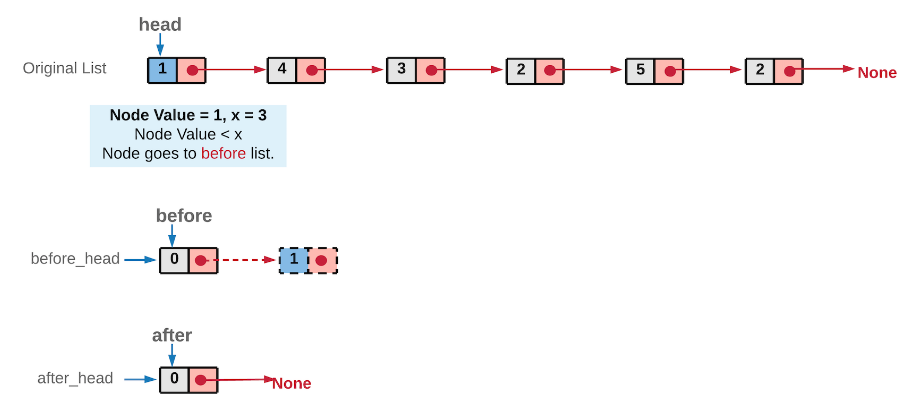
1.Initialize two pointers before and after. In the implementation we have initialized these two with a dummy ListNode. This helps to reduce the number of conditional checks we would need otherwise. You can try an implementation where you don't initialize with a dummy node and see it yourself!



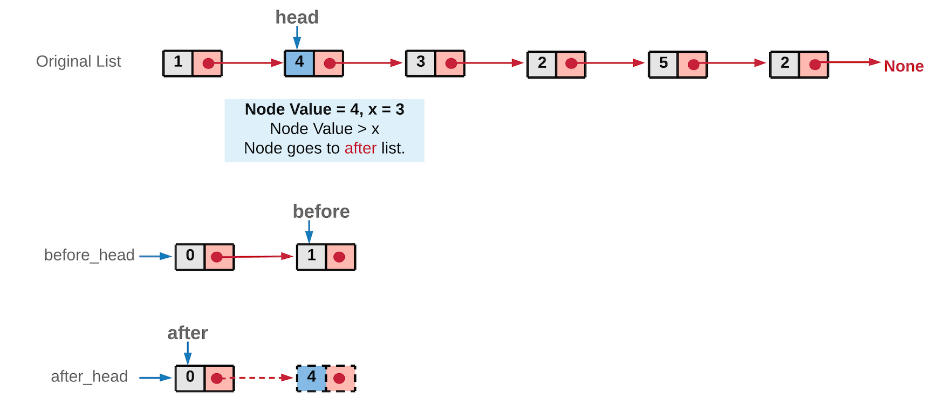
Dummy Node Initialization

2.Iterate the original linked list, using the head pointer.

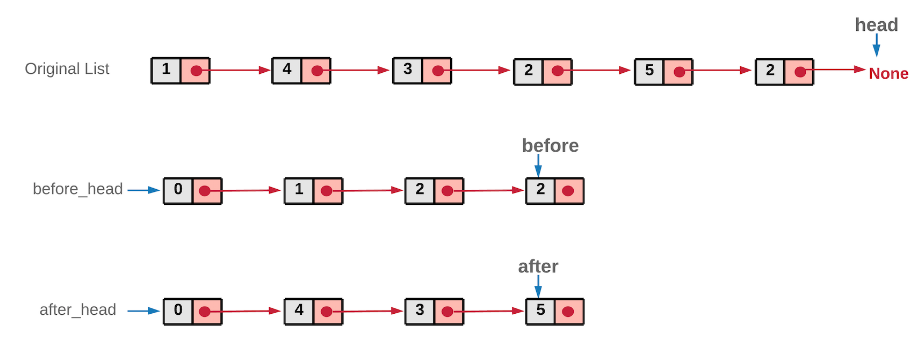
3.If the node's value pointed by head is *lesser* than x, the node should be part of the before list. So we move it to before list.



4.Else, the node should be part of after list. So we move it to after list.

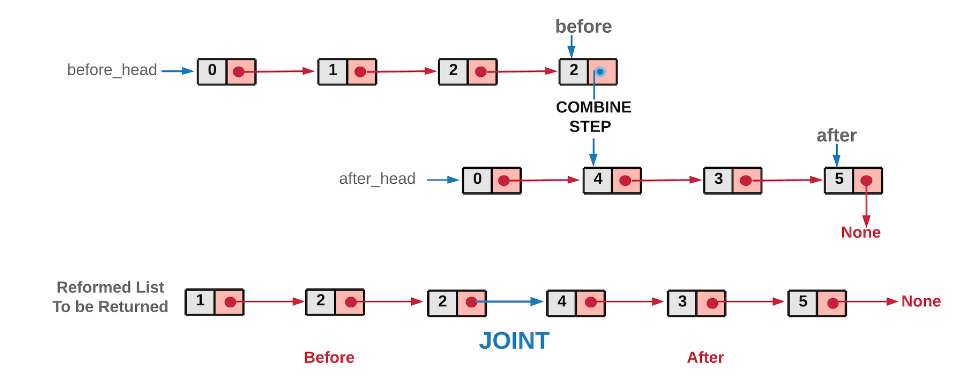


5.Once we are done with all the nodes in the original linked list, we would have two list before and after. The original list nodes are either part of before list or after list, depending on its value.



*Note: Since we traverse the original linked list from left to right, at no point would the order of nodes change relatively in the two lists. Another important thing to note here is that we show the original linked list intact in the above diagrams. However, in the implementation, we remove the nodes from the original linked list and attach them in the before or after list. We don't utilize any additional space. We simply move the nodes from the original list around.*

6.Now, these two lists before and after can be combined to form the reformed list.



We did a dummy node initialization at the start to make implementation easier, you don't want that to be part of the returned list, hence just move ahead one node in both the lists while combining the two list. Since both before and after have an extra node at the front.

class Solution {

public ListNode partition(ListNode head, int x) {

// before and after are the two pointers used to create the two list

// before\_head and after\_head are used to save the heads of the two lists.

// All of these are initialized with the dummy nodes created.

ListNode before\_head = new ListNode(0);

ListNode before = before\_head;

ListNode after\_head = new ListNode(0);

ListNode after = after\_head;

while (head != null) {

// If the original list node is lesser than the given x,

// assign it to the before list.

if (head.val < x) {

before.next = head;

before = before.next;

} else {

// If the original list node is greater or equal to the given x,

// assign it to the after list.

after.next = head;

after = after.next;

}

// move ahead in the original list

head = head.next;

}

// Last node of "after" list would also be ending node of the reformed list

after.next = null;

// Once all the nodes are correctly assigned to the two lists,

// combine them to form a single list which would be returned.

before.next = after\_head.next;

return before\_head.next;

}

}

**Complexity Analysis**

Time Complexity: O(N), where N is the number of nodes in the original linked list and we iterate the original list.

Space Complexity: O(1), we have not utilized any extra space, the point to note is that we are reforming the original list, by moving the original nodes, we have not used any extra space as such.