Let's briefly review the structure definition of a node in the doubly linked list.

Based on this definition, we are going to give you the solution step by step. The solution for the doubly linked list is similar to the one using singly linked list.

**1. Initiate a new linked list: represent a linked list using the head node.**

|  |
| --- |
| class MyLinkedList {  private DoublyListNode head;  /\*\* Initialize your data structure here. \*/  public MyLinkedList() {  head = null;  }  } |

**2. Traverse the linked list to get element by index.**

|  |
| --- |
| /\*\* Helper function to return the index-th node in the linked list. \*/  private DoublyListNode getNode(int index) {  DoublyListNode cur = head;  for (int i = 0; i < index && cur != null; ++i) {  cur = cur.next;  }  return cur;  }  /\*\* Helper function to return the last node in the linked list. \*/  private DoublyListNode getTail() {  DoublyListNode cur = head;  while (cur != null && cur.next != null) {  cur = cur.next;  }  return cur;  }  /\*\* Get the value of the index-th node in the linked list. If the index is invalid, return -1. \*/  public int get(int index) {  DoublyListNode cur = getNode(index);  return cur == null ? -1 : cur.val;  } |

**3. Add a new node.**

|  |
| --- |
| /\*\* Add a node of value val before the first element of the linked list. After the insertion, the new node will be the first node of the linked list. \*/  public void addAtHead(int val) {  DoublyListNode cur = new DoublyListNode(val);  cur.next = head;  if (head != null) {  head.prev = cur;  }  head = cur;  return;  }  /\*\* Append a node of value val to the last element of the linked list. \*/  public void addAtTail(int val) {  if (head == null) {  addAtHead(val);  return;  }  DoublyListNode prev = getTail();  DoublyListNode cur = new DoublyListNode(val);  prev.next = cur;  cur.prev = prev;  }  /\*\* Add a node of value val before the index-th node in the linked list. If index equals to the length of linked list, the node will be appended to the end of linked list. If index is greater than the length, the node will not be inserted. \*/  public void addAtIndex(int index, int val) {  if (index == 0) {  addAtHead(val);  return;  }  DoublyListNode prev = getNode(index - 1);  if (prev == null) {  return;  }  DoublyListNode cur = new DoublyListNode(val);  DoublyListNode next = prev.next;  cur.prev = prev;  cur.next = next;  prev.next = cur;  if (next != null) {  next.prev = cur;  }  } |

Similar to the singly linked list, it takes O(N) time to get a node by index, where N is the length of the linked list. It is different from adding a new node after a given node.

**5. Delete a node.**

|  |
| --- |
| /\*\* Delete the index-th node in the linked list, if the index is valid. \*/  public void deleteAtIndex(int index) {  DoublyListNode cur = getNode(index);  if (cur == null) {  return;  }  DoublyListNode prev = cur.prev;  DoublyListNode next = cur.next;  if (prev != null) {  prev.next = next;  } else {  // modify head when deleting the first node.  head = next;  }  if (next != null) {  next.prev = prev;  }  } |

Similar to the add operation, it takes O(N) time to get the node by the index which is different from deleting a given node. However, it is different to the singly linked list. When we get the node we want to delete, we don't need to traverse to get its previous node but using the "prev" field instead.