**Name:\_\_\_\_\_\_\_Shuqing Ye\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_UCI NET ID:\_\_\_\_\_\_\_\_shuqingy2\_\_\_\_\_\_\_\_**

|  |  |
| --- | --- |
| Test cases (including the edge cases):  1->2->3->4, i = 1 output: 1->2->3  1->2->3->4, i = 2 output: 1->2->4  1->2->3->4, i = 4 output: 2->3->4  1->null, i = 1 output: null | time complexity: O(n) // n is the length of linkedlist  space complexity: O(n) for solution 1, O(1) for solution 2 |

|  |  |
| --- | --- |
| // Solution 1, using a single pointer and a hashMap  public linked\_list remove\_i\_th\_element(linked\_list head, int i) {  if (head == null || i == 0)  return null;  Map<Integer, linked\_list > map = new HashMap<>();  Int cnt = 0;  linked\_list tmp = head;  while (tmp != null ) {  map.put(cnt, tmp);  tmp = tmp.next;  cnt++;  }  linked\_list target = map.get(cnt - i);  // if target is not the head  if (cnt – i > 0) {  linked\_list pre = map.get(cnt – i - 1);  pre.next = target.next;  }  else head = head.next;  return head;  } | // Solution 2, using two pointers (fast and slow)  public linked\_list remove\_i\_th\_element(linked\_list head, int i) {  if (head == null || i == 0)  return null;  linked\_list fast = head;  while( i-- > 0) {  fast = fast.next;  }  linked\_list slow = head;  while (fast != null && fast.next != null) {  fast = fast.next;  sl  ow = slow.next;  }  if (fast == null) // fast has left the linkedlist  head = head.next; // head has to be removed  else  slow.next = slow.next.next;  return head;  } |