03. 연결리스트

발표자: 한혜원

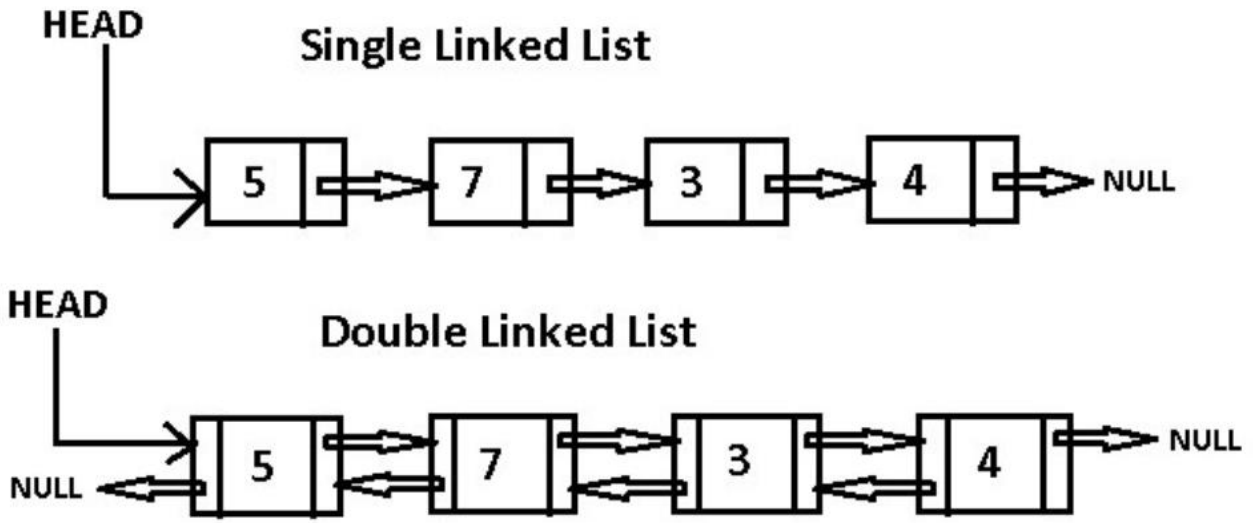
1. 연결리스트란

- 노드가 연결되어 리스트 형태로 구성됨

- 노드는 값과 다음 노드의 주소를 저장

- 고정 크기의 배열 등의 단점을 보완

- 동적으로 크기가 변할 수 있음



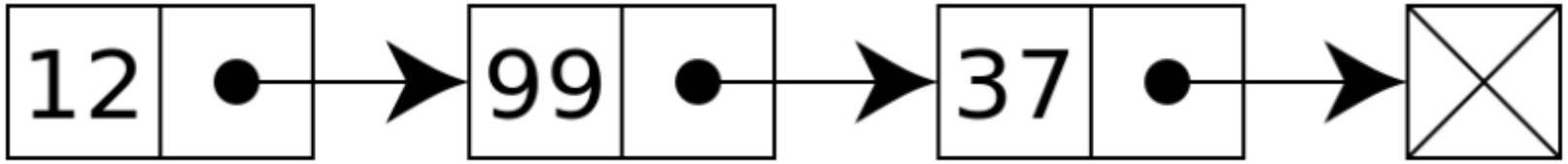
2. 연결리스트 시간복잡도

노드 검색을 할 때 O(n)의 시간 소요

배열도 원소 검색할 때 한번 씩 모두 훑으므로 O(n)의 시간 소요

다만 차이점은 배열은 arr[3]를 검색할 때 O(1)이 소요되지만

연결리스트는 무조건 O(n)의 방식으로 탐색가능. 단점

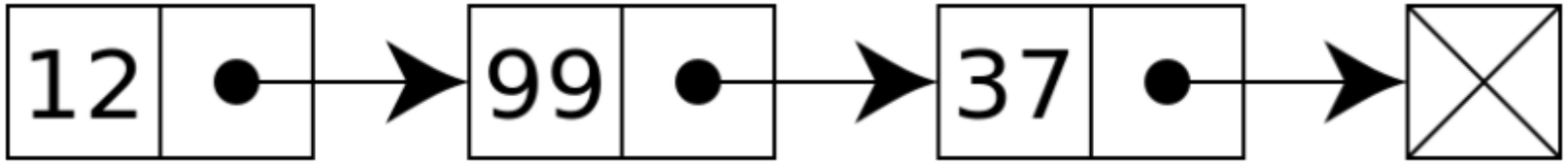


{0,1,2,3,4}

2. 연결리스트 시간복잡도

하지만 삽입 삭제는 O(1)으로 빠름

다음 노드의 주소 값만 바꿔주면 되기 때문. 장점



{0,1,2,3,4}

단순연결리스트 구현해보기

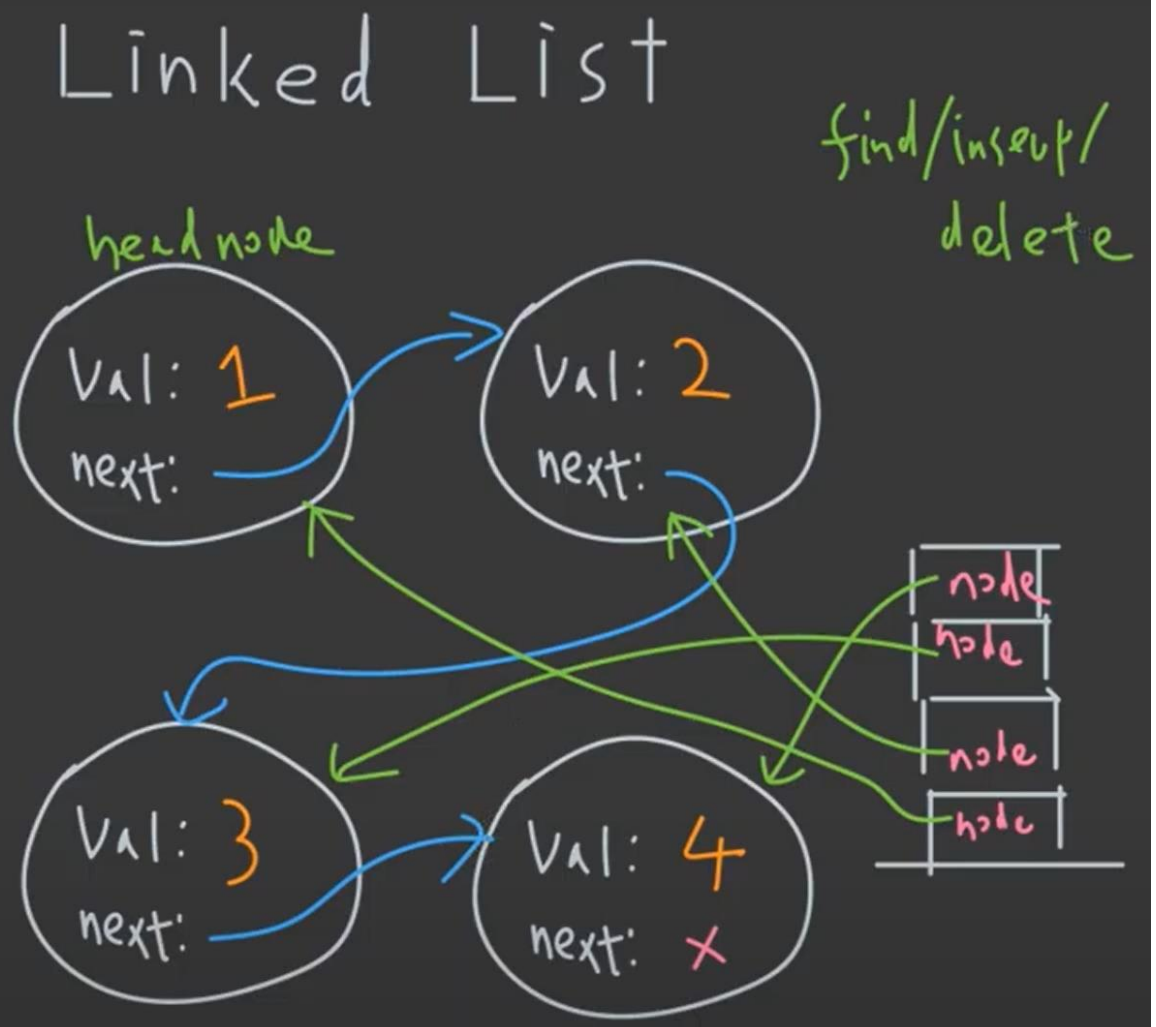
```
[1]  class ListNode:
       def __init__(self,val):
         self.val = val
         self.next = None
```

```
[4]  head_node = ListNode(1)
     head_node.next = ListNode(2)
     head_node.next.next = ListNode(3)
     head_node.next.next.next = ListNode(4)
```

```
     def printNodes(node:ListNode):
       crnt_node = node
       while crnt_node is not None:
         print(crnt_node.val , end= ' ')
         crnt_node = crnt_node.next

     printNodes(head_node)
```

```
     1 2 3 4
```

```
     def printNodesRecur(node:ListNode):
       print(node.val, end=' ')
       if node.next is not None:
         printNodesRecur(node.next)

     printNodesRecur(head_node)
```



```
        print(crnt_node.val, end=' ')
        crnt_node = crnt_node.next
```

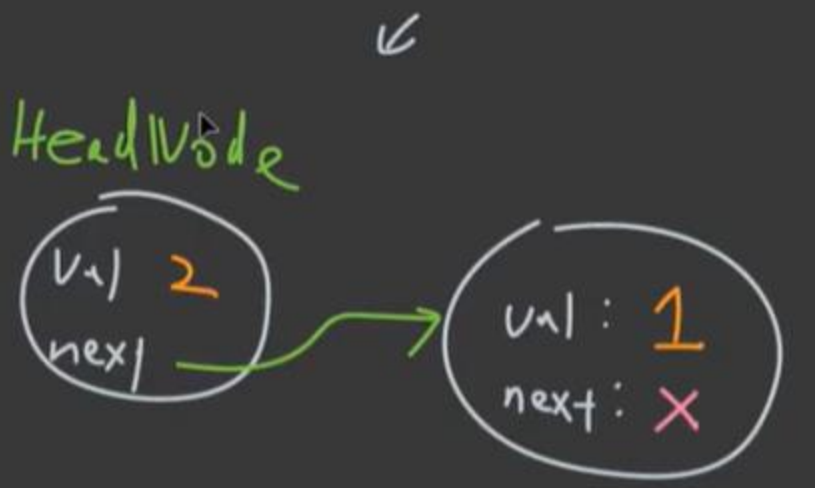
```
[3]  class SLinkedList:
       def __init__(self):
         self.head = None

       def addAtHead(self, val):
         node = ListNode(val)
         node.next = self.head
         self.head = node
```

```
slist = SLinkedList()
slist.addAtHead(1)
slist.addAtHead(2)
printNodes(slist.head)
```

```
2 1
```

```
[ ]
```

Linked List



left addAtHead(1) ←

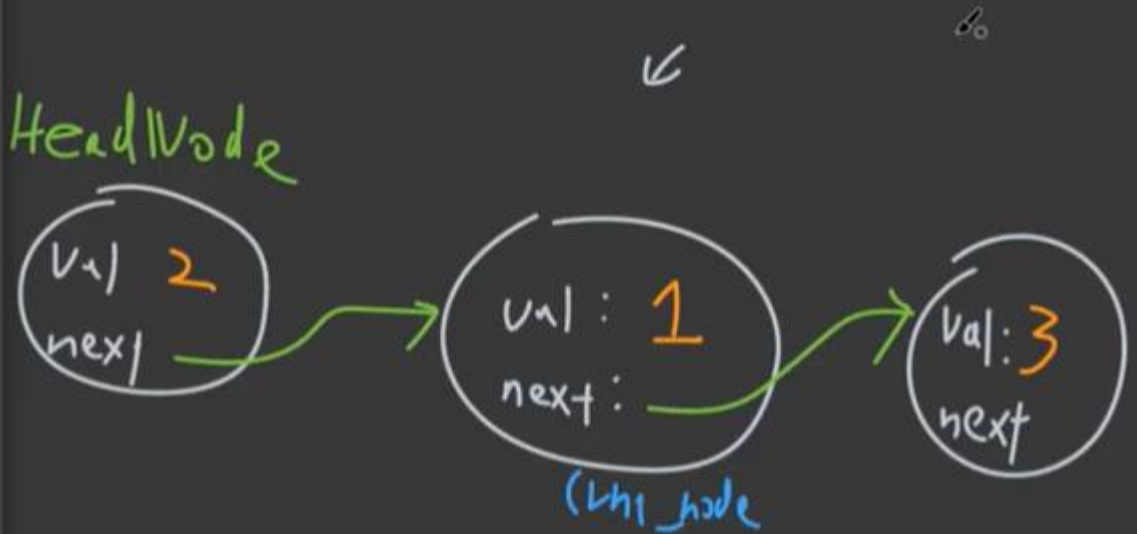
addAtHead(2) ←

```python
class SLinkedList:
  def __init__(self):
    self.head = None

  def addAtHead(self, val):
    node = ListNode(val)
    node.next = self.head
    self.head = node

  #but when the list
  def addBack(self, val):
    node = ListNode(val)
    crnt_node = self.head
    while crnt_node.next:
      crnt_node = crnt_node.next
    crnt_node.next = node
```

```python
[6] slist = SLinkedList()
    slist.addAtHead(1)
    slist.addAtHead(2)
    slist.addBack(3)
    printNodes(slist.head)
```

```
2 1 3
```

Linked List

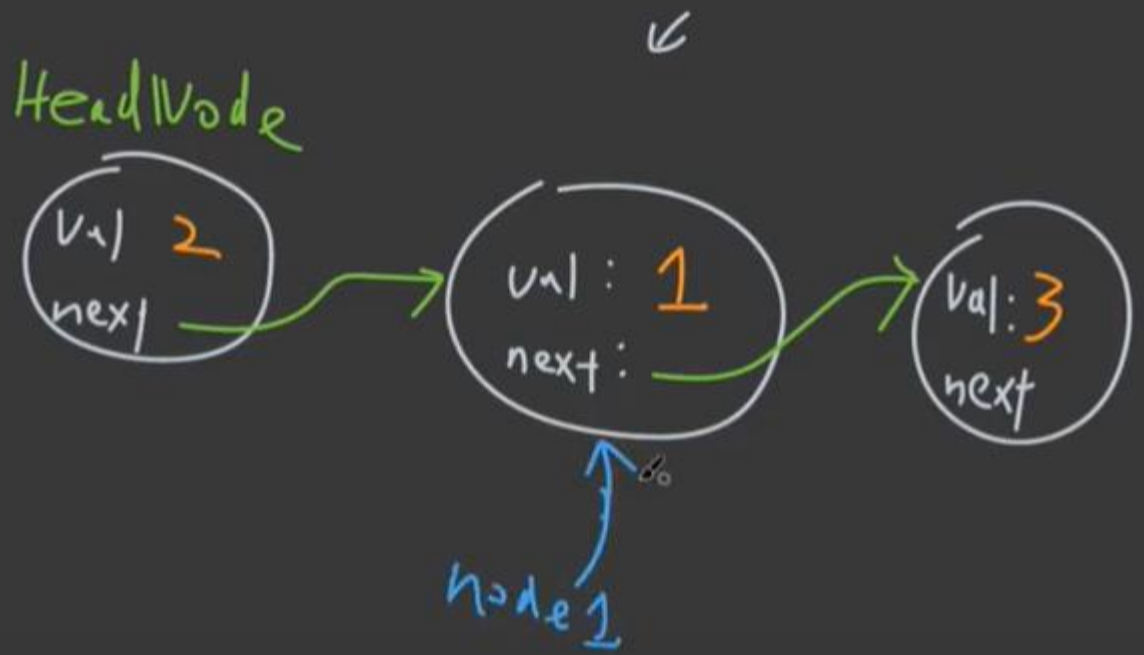


let addBack (3)

```python
    def addAtHead(self, val):
        node = ListNode(val)
        node.next = self.head
        self.head = node

    #but when the list
    def addBack(self, val):
        node = ListNode(val)
        crnt_node = self.head
        while crnt_node.next:
            crnt_node = crnt_node.next
        crnt_node.next = node

    def findNode(self, val):
        crnt_node = self.head
        while crnt_node is not None:
            if crnt_node.val == val:
                return crnt_node
            crnt_node = crnt_node.next
        raise RuntimeError('Node not found')
```

```python
[11] slist = SLinkedList()
     slist.addAtHead(1)
     slist.addAtHead(2)
     slist.addBack(3)
     printNodes(slist.head)

     node1 = slist.findNode(1)
```

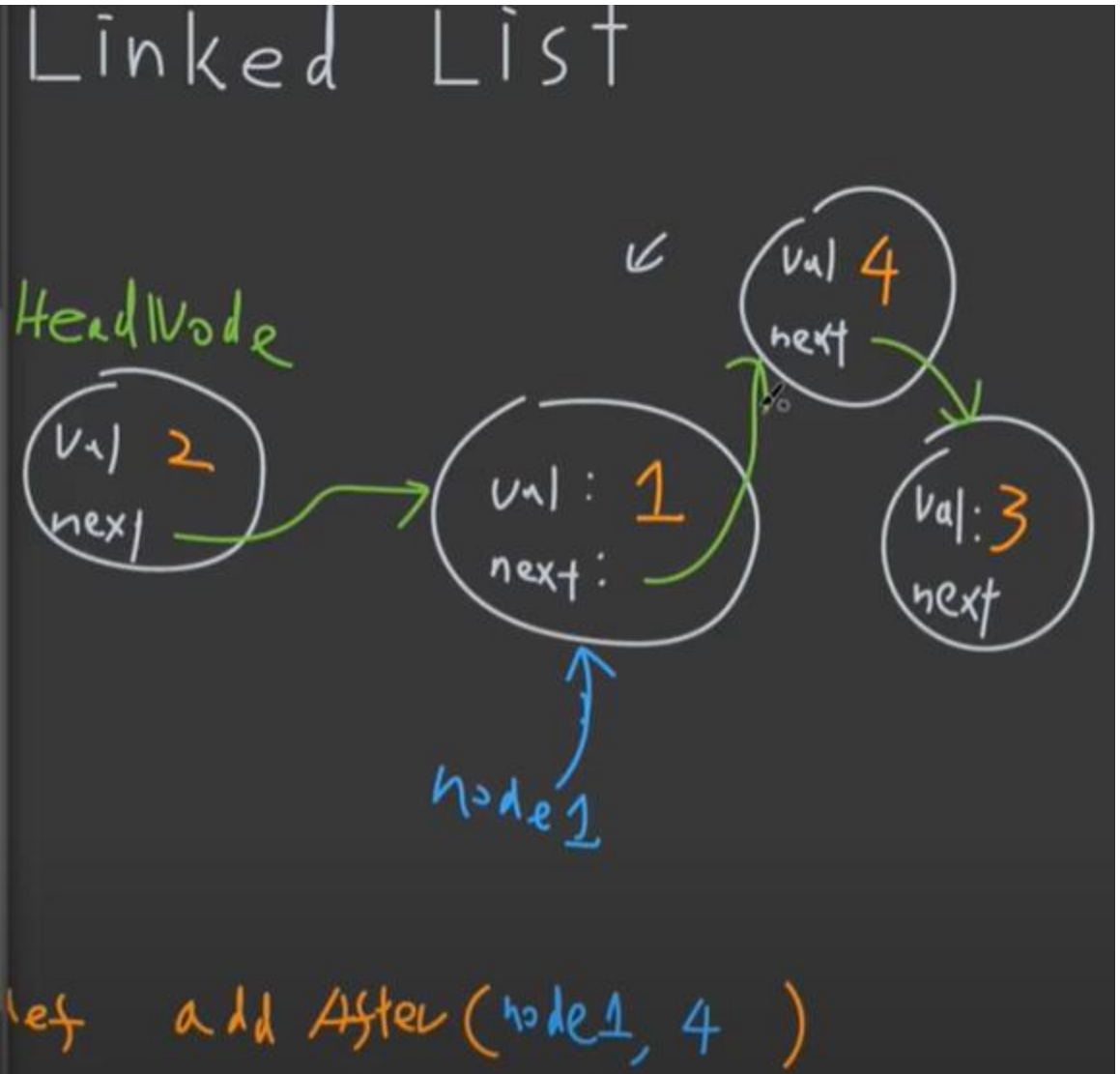
Linked List



def find (1)

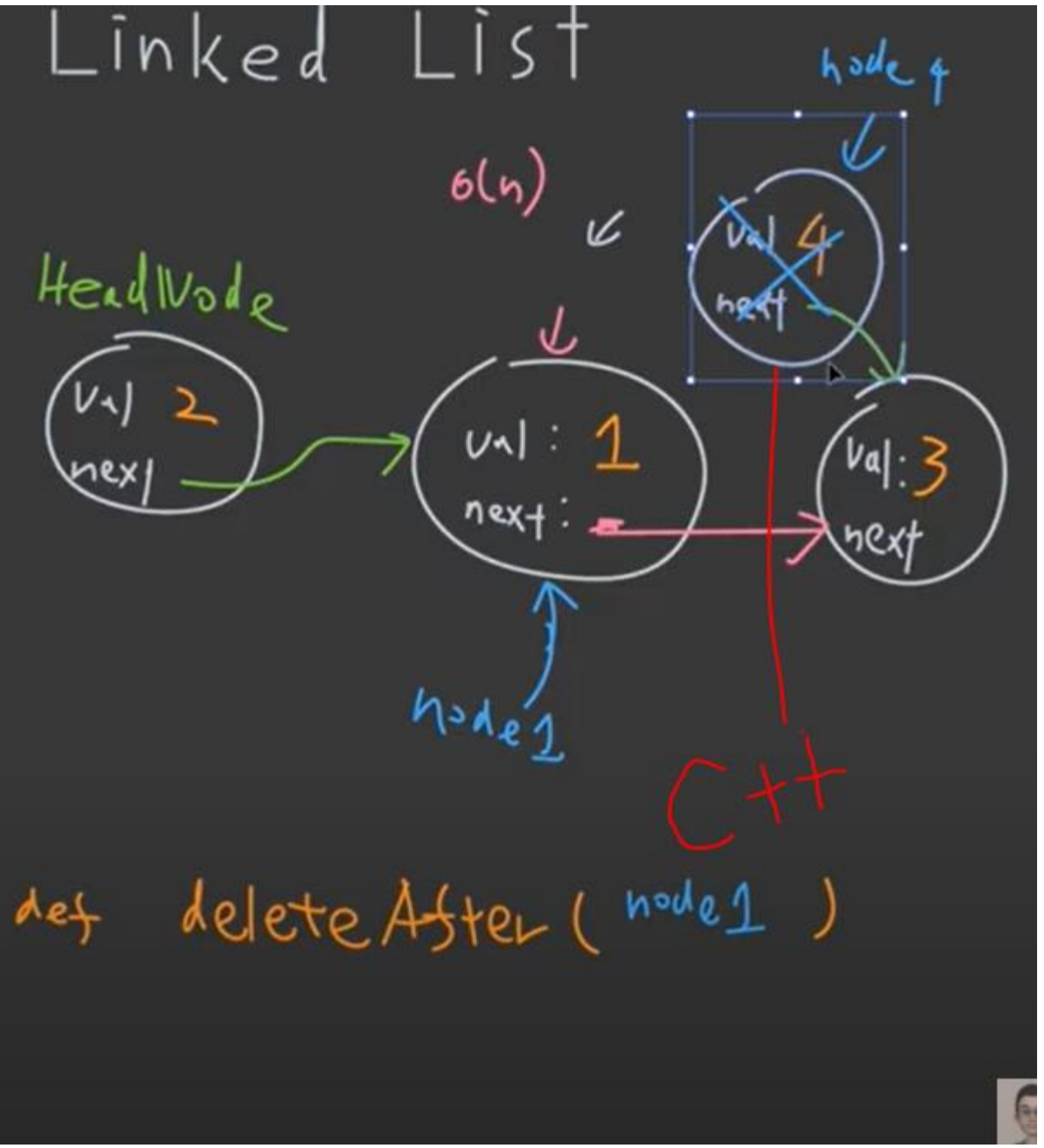
```
    #but when the list
    def addBack(self, val):
        node = ListNode(val)
        crnt_node = self.head
        while crnt_node.next:
            crnt_node = crnt_node.next
        crnt_node.next = node

    def findNode(self, val):
        crnt_node = self.head
        while crnt_node is not None:
            if crnt_node.val == val:
                return crnt_node
            crnt_node = crnt_node.next
        raise RuntimeError('Node not found')

    def addAfter(self, node, val):
        new_node = ListNode(val)
        new_node.next = node.next
        node.next = new_node
```

```
slist = SLinkedList()
slist.addAtHead(1)
slist.addAtHead(2)
slist.addBack(3)

node1 = slist.findNode(1)
slist.addAfter(node1,)
printNodes(slist.head)
```



```
    node = ListNode(val)
    crnt_node = self.head
    while crnt_node.next:
      crnt_node = crnt_node.next
    crnt_node.next = node

  def findNode(self, val):
    crnt_node = self.head
    while crnt_node is not None:
      if crnt_node.val == val:
        return crnt_node
      crnt_node = crnt_node.next
    raise RuntimeError('Node not found')

  def addAfter(self, node, val):
    new_node = ListNode(val)
    new_node.next = node.next
    node.next = new_node

  def deleteAfter(self, prev_node):
    if prev_node.next is not None:
      prev_node.next = prev_node.next.next
```

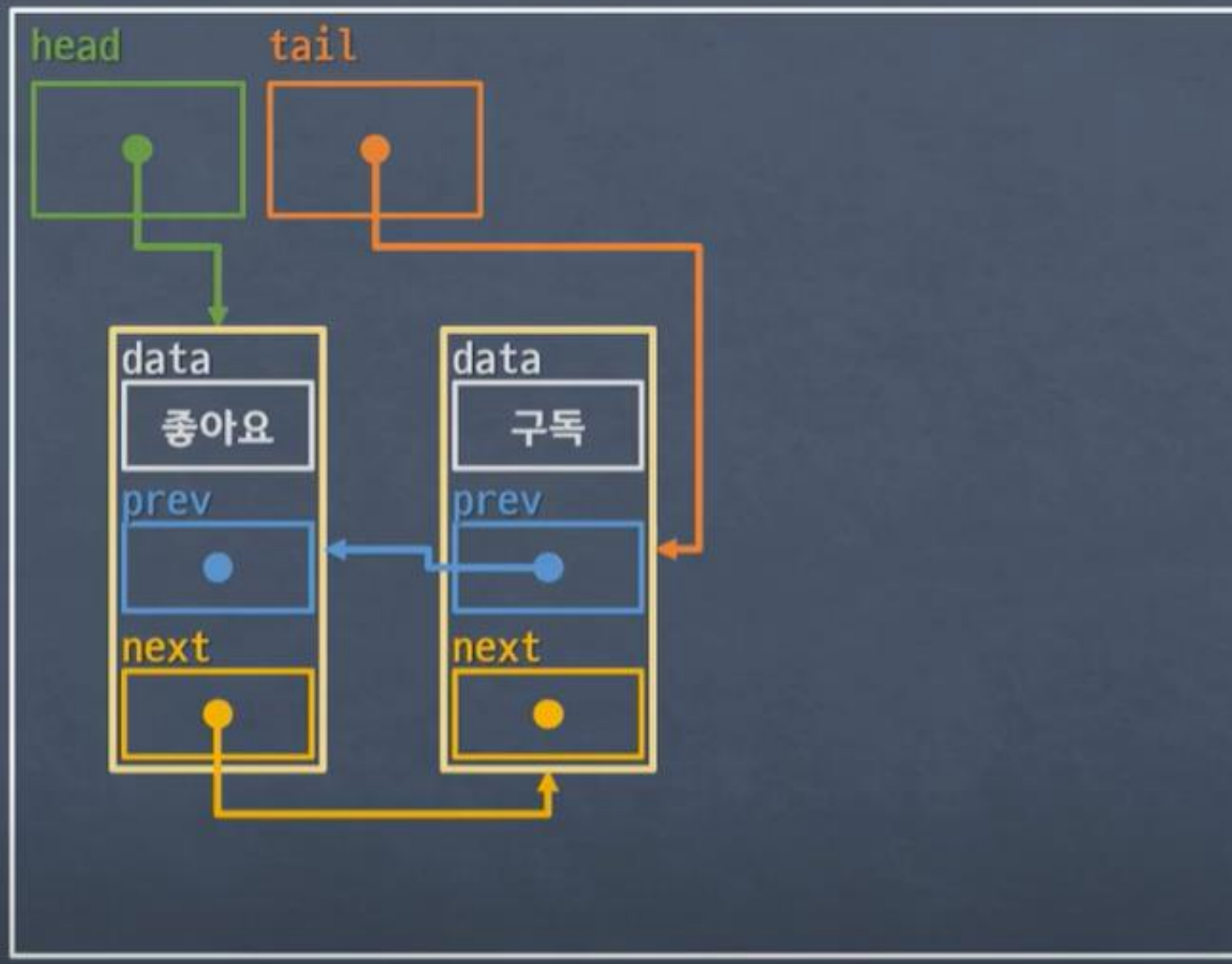
```
[13] slist = SLinkedList()
     slist.addAtHead(1)
     slist.addAtHead(2)
     slist.addBack(3)

     node1 = slist.findNode(1)
     slist.addAfter(node1,4)
     printNodes(slist.head)
```



양방향연결리스트 구현해보기

```
class Dlist:
    def __init__(self):
        self.head = None
        self.tail = None

    class Node:
        def __init__(self, data,
                     p=None, n=None):
            self.data = data
            self.prev = p
            self.next = n

    def add_first(self, data):
        if self.head is None:
            self.head = self.Node(data)
            self.tail = self.head
        else:
            self.head = self.Node(data,
                                  next=self.head)
            self.head.next.prev = self.head
        self.nodeCnt += 1
```

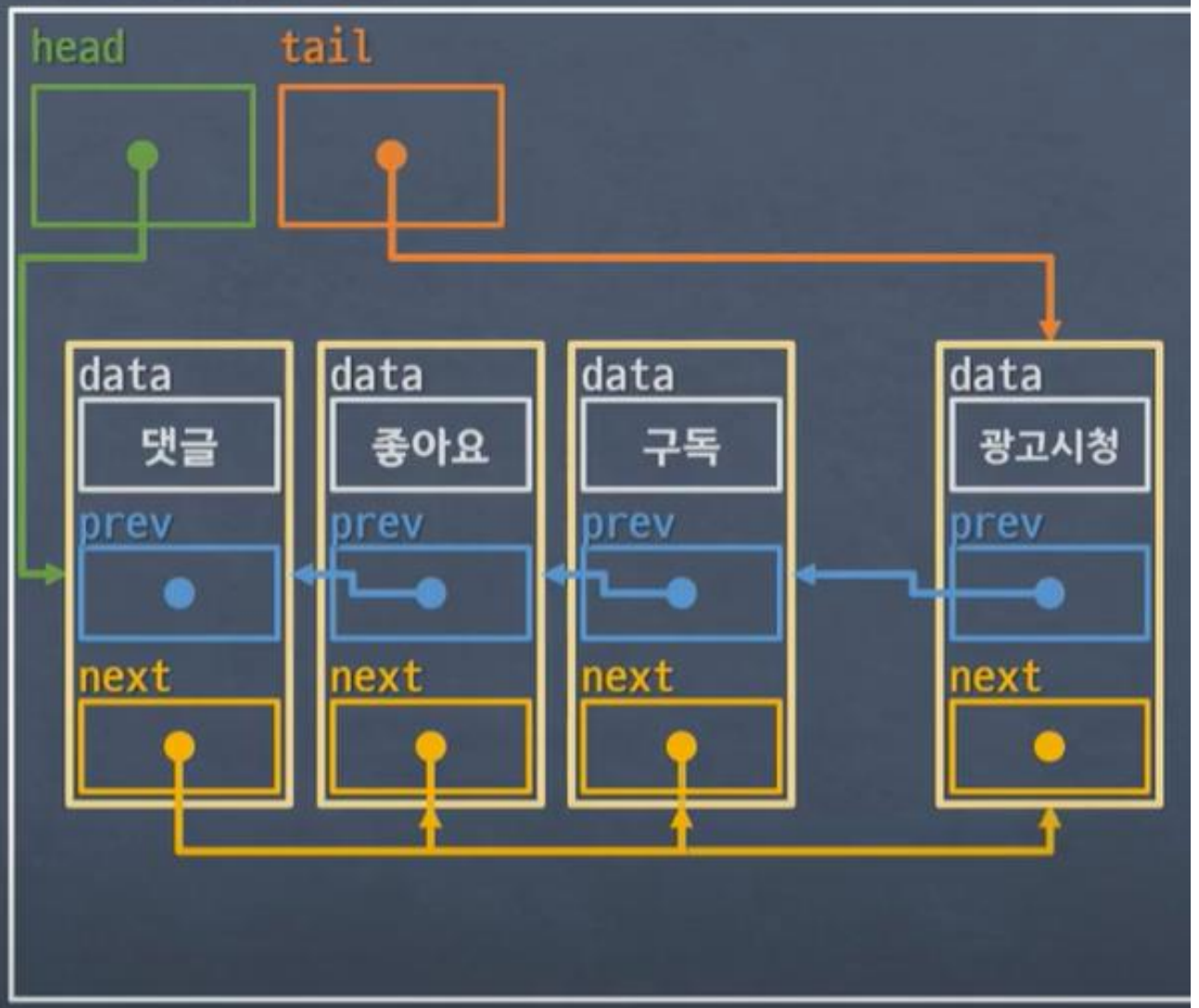
doubleLinkedList



doubleLinkedList.add_first("좋아요")

```python
class Dlist:
    def __init__(self):
        self.head = None
        self.tail = None

    class Node:
        def __init__(self, data,
                     p=None, n=None):
            self.data = data
            self.prev = p
            self.next = n

    def add_last(self, data):
        if self.tail is None:
            self.tail = self.Node(data)
            self.head = self.tail
        else:
            self.tail = self.Node(data,
                                  prev=self.tail)
            self.tail.prev.next = self.tail
        self.nodeCnt += 1
```

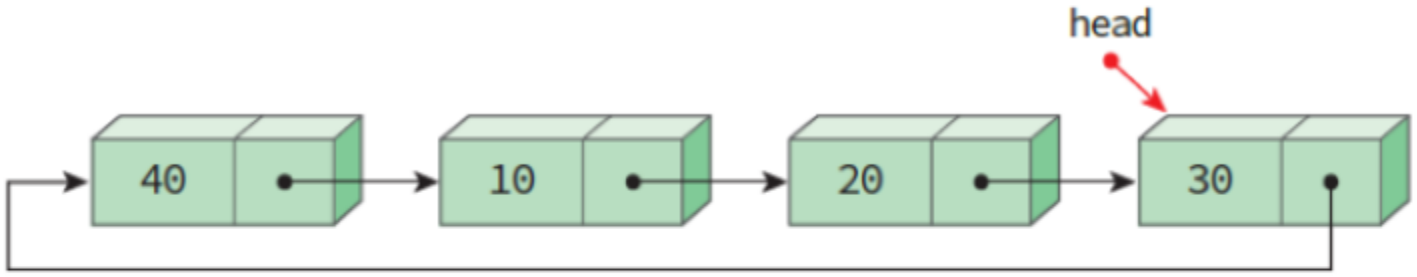
doubleLinkedList



doubleLinkedList.add_last("광고시청")

원형연결리스트란?

원형 연결리스트



원형 연결리스트에서 head 포인터는 마지막 노드를 가리키게 됩니다. 이러한 방식의 원형 리스트는 head는 마

03. 연결리스트 문제:

https://www.acmicpc.net/problem/1158

요세푸스 문제

시간 제한	메모리 제한	제출	정답	맞은 사람	정답 비율
2 초	256 MB	49059	23837	17011	48.104%

문제

요세푸스 문제는 다음과 같다.

1번부터 N번까지 N명의 사람이 원을 이루면서 앉아있고, 양의 정수 K(≤ N)가 주어진다. 이제 순서대로 K번째 사람을 제거한다. 한 사람이 제거되면 남은 사람들로 이루어진 원을 따라 이 과정을 계속해 나간다. 이 과정은 N명의 사람이 모두 제거될 때까지 계속된다. 원에서 사람들이 제거되는 순서를 (N, K)-요세푸스 순열이라고 한다. 예를 들어 (7, 3)-요세푸스 순열은 <3, 6, 2, 7, 5, 1, 4>이다.

N과 K가 주어지면 (N, K)-요세푸스 순열을 구하는 프로그램을 작성하시오.

입력

첫째 줄에 N과 K가 빈 칸을 사이에 두고 순서대로 주어진다. (1 ≤ K ≤ N ≤ 5,000)

출력

예제와 같이 요세푸스 순열을 출력한다.

예제 입력 1 복사

```
7 3
```

예제 출력 1 복사

```
<3, 6, 2, 7, 5, 1, 4>
```

*언어별 연결리스트

- c++ : std 라이브러리에서 제공하는 list 클래스가 있음

(근데 문제에서 요구하는 경우아니면 벡터가 훨씬 효율적이라고 함)

(승아 궁금하면 https://www.youtube.com/watch?v=R-YqJmmrWhc 이 영상 참고)

- 자바: util 라이브러리에서 제공하는 LinkedList 클래스 있음

- 파이썬: 따악히 똑같은 명칭의 것은 없지만 deque 클래스가 이중연결리스트 구조라고 함.

- 큐 두 개 중 하나를 좌우로 뒤집어서 붙인 구조
- 큐의 양쪽 끝에서 삽입, 삭제 연산을 수행할 수 있도록 확장함

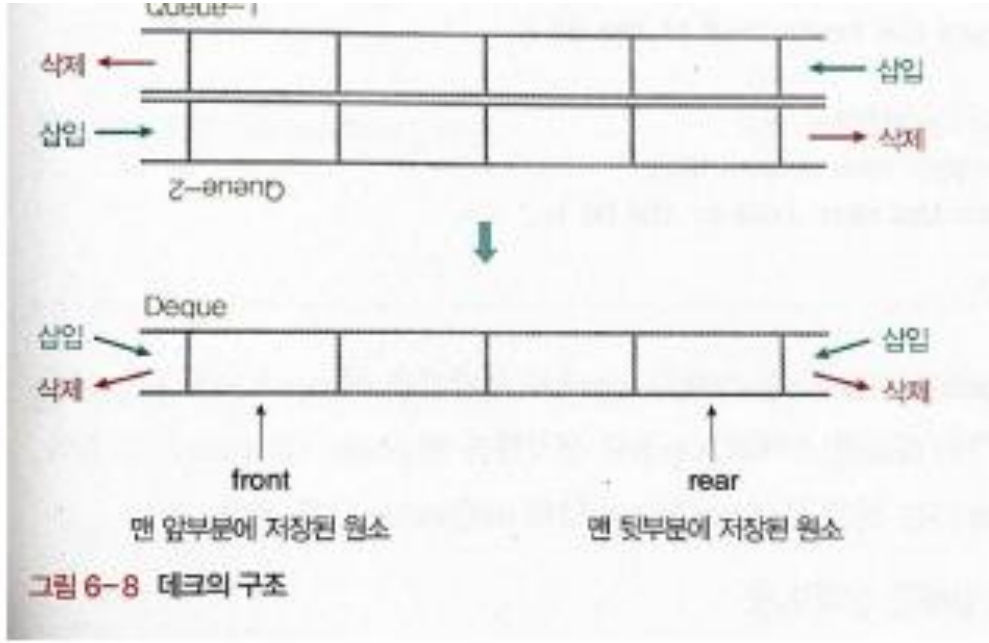


그림 6-8 데크의 구조