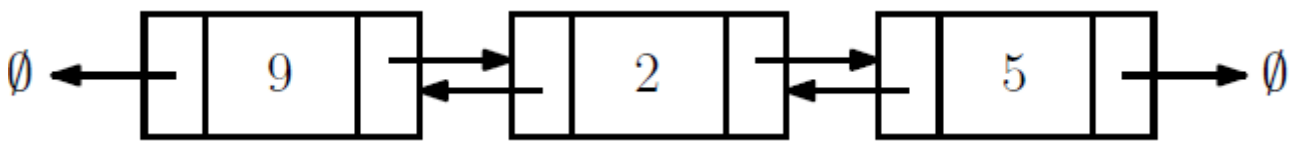
양방향 연결리스트

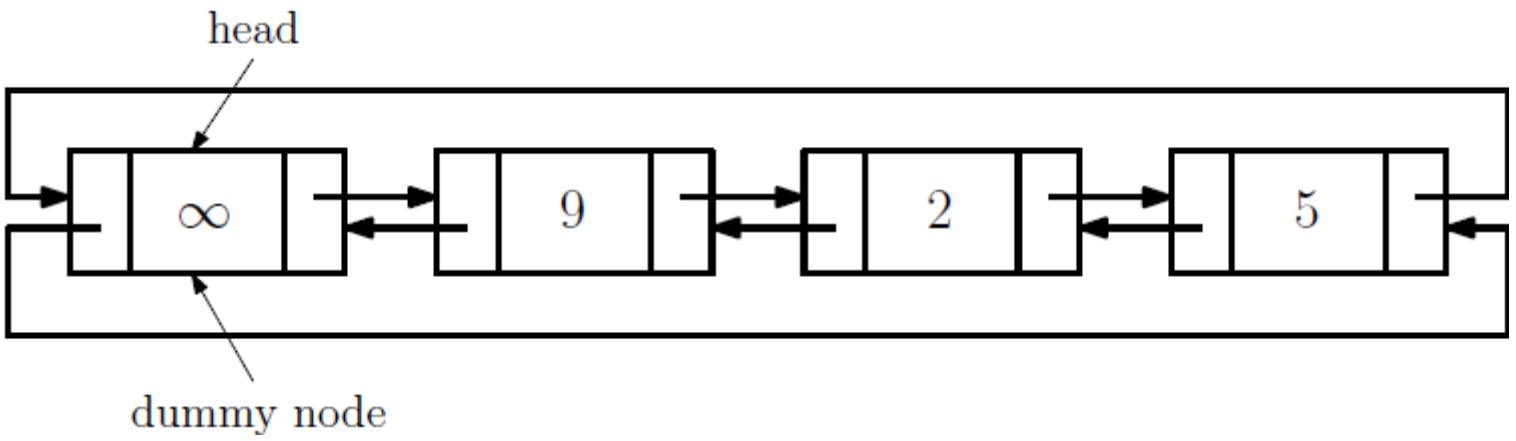
양방향(이중) 연결리스트란?

1. 한쪽 방향 링크만 있으면, 반대 방향의 이웃 노드를 O(1) 시간에 알 수 없다.
    a. 예를 들어, 노드 x를 지울 때, x의 전 노드 prev를 알아야 한다. x에서 prev 노드로 가는 링크가 없기 때문에, head노드부터 링크를 따라가면서 prev를 알아내야 한다. 최악의 경우엔 O(n) 시간이 걸려 매우 비효율적이다.
    b. 가장 간단한 방법은 각 노드가 양 쪽 방향 링크를 가지는 것이다.

```
class Node:
  def __init__(self, key=None, value=None):
    self.key = key
    self.value = value
    self.prev = None
    self.next = None
```



1. 양방향 연결리스틀 루언형으로 연결된 원형 리스트(circular list)로 관리한다. 이 때, 기준 노드 역할을 하는 dummy 노드를 사용하면 실제 구현이 매우 간단해진다.
2. dummy node의 key 값은 특수한 값(예를 들어, None)으로 지정하고, dummy node가 head 노드의 역할을 한다.
3. 한방향 연결리스트에서는 head==none이면 빈 리스트를 의미했지만, 양방향 연결리스트의 경우엔 dummy node 하나로만 구성된 리스트가 빈 리스트가 된다. (이 때, 두 링크는 모두 dummy 노드 자신을 가르킴)
4.



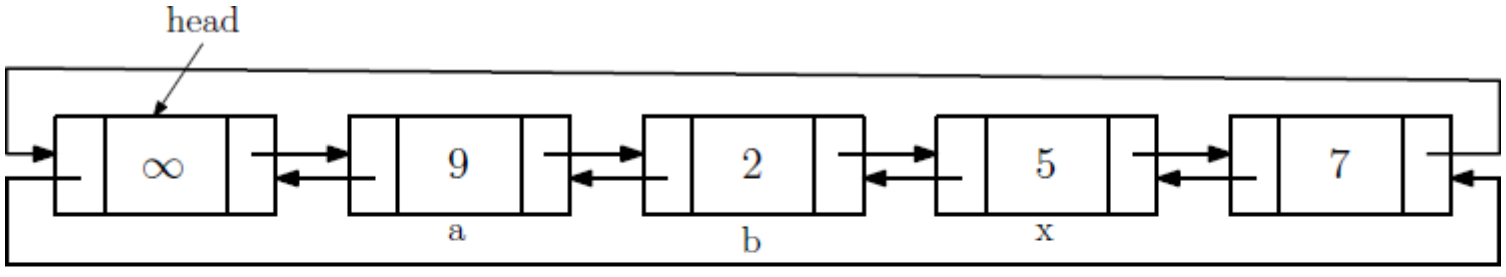
1. Node 클래스의 생성함수를 아래와 같이 바꾼다. prev와 next 링크를 자신을 가르키도록 한다.
2. DoublyLinkedList 클래스의 생성자를 아래와 같이 정의한다.
    a. dummy 노드 하나로만 구성된 빈 리스트가 만들어진다.

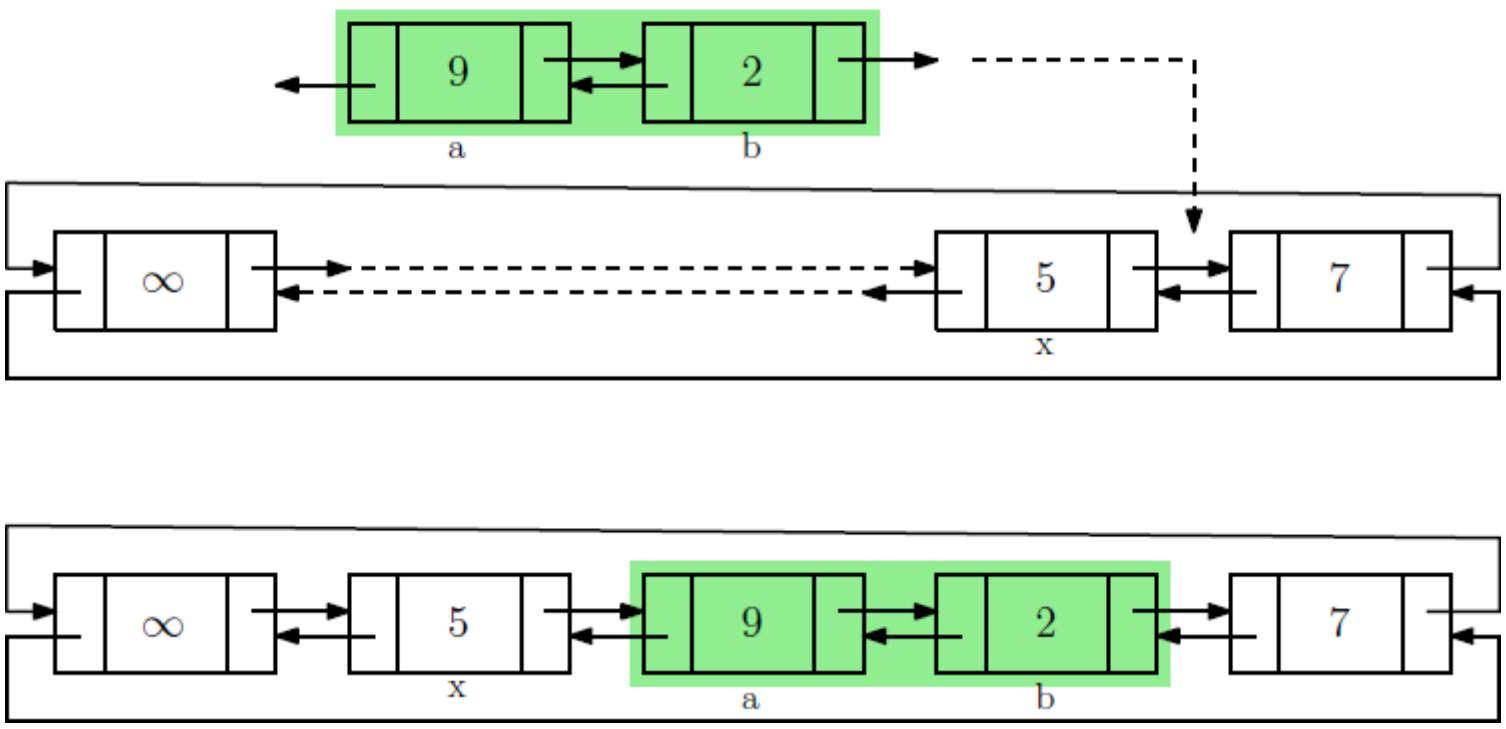
```
class Node:
  def __init__(self, key=None, value=None):
    self.key = key
    self.value = value
    self.prev = self
    self.next = self

class DoublyLinkedList:
  def __init__(self):
    self.head = Node()  # dummy 노드로만 이루이진 빈 리스트
```

양방향 연결리스트: splice

잘라낸 후 이동하기 연산: splice(a, b, x)

1. 양방향 연결리스트에서 노드 a와 노드 b 사이에 노드들을 잘라내서 노드 x 뒤에 붙이는 연산
    a. 조건 1: 리스트에서 노드 a 다음에 노드 b가 나와야 함
    b. a와 b 사이에 head 노드가 있으면 안됨
    c. a와 b 사이에 x가 있으면 안됨.






```
def splice(self, a, b, x): # cut [a..b] after x
  if a == None or b == None or x == None:
    return
  # 1. [a..b] 구간을 잘라내기
  a.prev.next = b.next
  b.next.prev = a.prev

  # 2. [a..b]를 x 다음에 삽입하기
  x.next.prev = b
  b.next = x.next
  a.prev = x
  x.next = a
```

양방향 연결리스트: 이동-삽입연산

splice 이용하여 이동 함수 정의하기: moveAfter, moveBefore

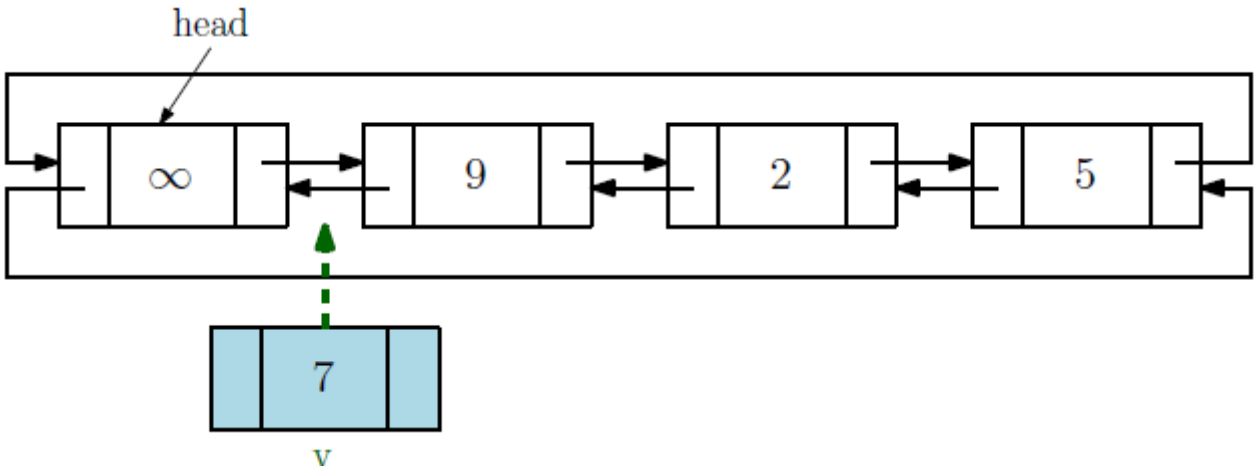
1. moveAfter(a, x): 노드 a를 노드 x 뒤로 이동하기
    a. splice(a, a, x) 호출과 같음
2. moveBefore(a, x): 노드 a를 노드 x 앞으로 이동하기
    a. splice(a, a, x.prev) 호출과 같음

splice를 이용하여 삽입함수 정의하기: insertAfter, insertBefore, pushFront, pushBack

- splice(head, head, x)라고 호출하면, 잘라내기 부분은 어떻게 동작하나?

1. insertAfter(a, key): key값을 갖는 새 노드 b를 만들어 a 뒤에 삽입
2. insertBefore(a, key): key값을 갖는 새 노드 b를 만들어 a 앞에 삽입
3. pushFront(key): key값을 갖는 새 노드 b를 만들어 맨 앞(head 앞)에 삽입
    a. insertAfter()를 이용할 것
4. pushBack(key): key값을 갖는 새 노드 b를 만들어 맨 뒤(tail 앞)에 삽입
    a. insertBefore()를 이용할 것

pushFront



양방향 연결리스트: 삭제연산

양방향 연결리스트에서 삭제 연산: deleteNode, popFront, popBack

1. deleteNode(x): 노드 x를 제거한다. (단, x는 head가 아니여야 함)
2. popFront(): 리스트의 가장 앞 노드(head.next)의 값을 리턴하고 해당 노드 제거
3. popBack(): 리스트의 가장 뒷 노드(head.prev)의 값을 리턴하고 해당 노드 제거

```
def deleteNode(self, x): # delete x
    if x == None or x == self.head:
      return
    # 노드 x를 리스트에서 분리해내기
    x.prev.next, x.next.prev = x.next, x.prev
```

```
def popFront(self):
  if self.head.next == self.head:
    return None
  key = self.head.next.key
  self.deleteNode(self.head.next)
  return key

def popBack(self):
  if self.head.prev == self.head:
    return None
  key = self.head.prev.key
  self.deleteNode(self.head.prev)
  return key
```

양방향 연결리스트: 기타연산

양방향 연결리스트에서 기타 연산: search, isEmpty, firtst, last, join, split

1. search(key): key 값이 있으면 해당 노드를 리턴하고 없으면 None을 리턴
2. isEmpty(): 리스트가 비어있다면 True, 아니면 False 리턴

3. first(): 리스트의 가장 앞 노드(self.haed)를 리턴, 빈 리스트면 None 리턴
4. last(): 리스트의 가장 뒷 노드(self.head.prev)를 리턴, 빈 리스트면 None 리턴
5. join(list): 현재 리스트 뒤에 list를 이어 붙임. (두 리스트를 연결하는 함수)
    - splice() 함수를 이용할 것
6. split(x): 리스트의 노드 x부터 마지막 노드까지를 떼어내 새로운 리스틀르 만들어 리턴. (하나의 리스트를 두 개로 분할하는 함수)
    - splice() 함수를 이용할 것

```
def search(self, key):
  v = self.head
  while v.next != v:
    if v.key == key:
      return key
    v = v.next
    return None

def isEmpty(self):
  if self.size == 0:
    return True
  return False
```

```
def first(self):
  if self.size == 0:
    return None
  return self.head

def last(self):
  if self.size == 0:
    return None
  return self.head.prev
```