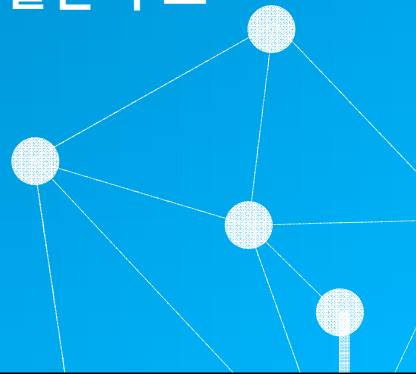


06 CHAPTER

연결된 구조



6장. 학습 목표



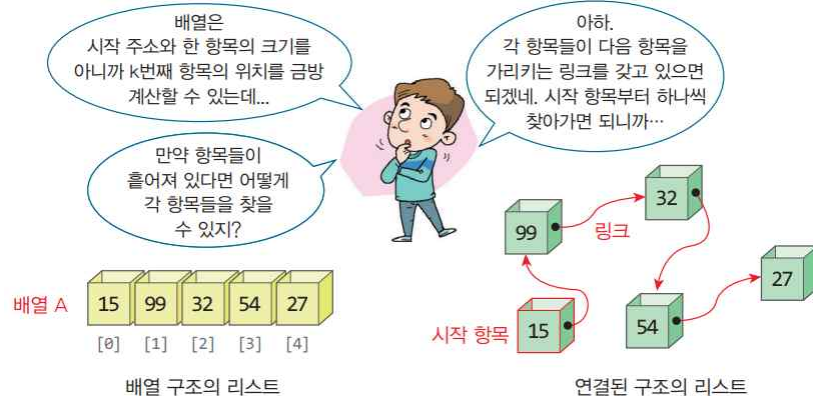
- 배열 구조와 연결된 구조의 특징과 장단점을 이해한다.
- 다양한 연결된 구조의 형태와 특징을 이해한다.
- 파이썬을 이용해 연결된 형태의 자료구조를 구현할 수 있다.
- 단순연결리스트로 스택과 리스트를 구현할 수 있다.
- 원형연결리스트로 큐를 구현할 수 있다.
- 덱을 이중연결리스트로 구현하는 이유를 이해한다.

6.1 연결된 구조란?

- 연결된 구조는 흩어진 데이터를 링크로 연결해서 관리한다.
- 연결된 구조의 특징
- 연결리스트의 구조
- 연결리스트의 종류

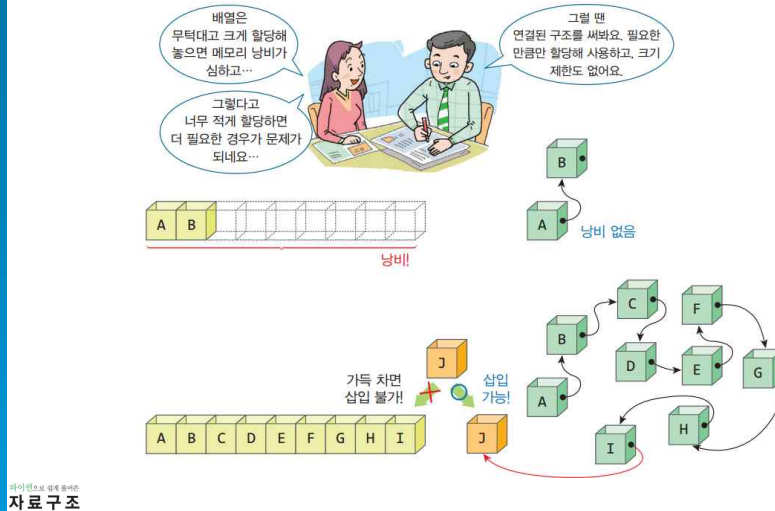
연결된 구조란?

- 연결된 구조는 흩어진 데이터를 링크로 연결해서 관리



연결된 구조의 특징

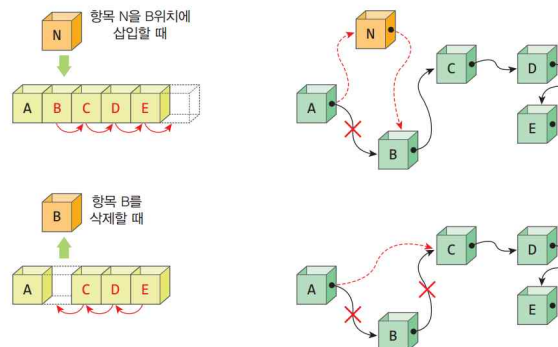
- 용량이 고정되지 않음.



5

연결된 구조의 특징

- 중간에 자료를 삽입하거나 삭제하는 것이 용이



- n 번째 항목에 접근하는데 $O(n)$ 의 시간이 걸림.

자료 구조

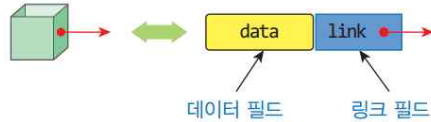
6

연결 리스트의 구조



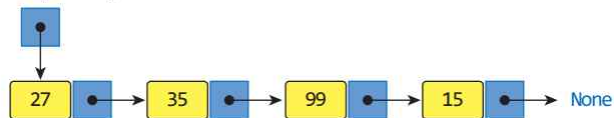
노드 (node)

- 데이터 필드(data field)
- 하나 이상의 링크 필드(link field)



헤드 포인터 (head pointer)

헤드 포인터
(header pointer)



파이썬으로 쉽게 배우는
자료구조

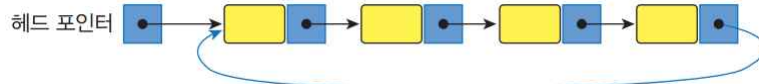
7

연결 리스트의 종류

- 단순 연결 리스트(singly linked list)



- 원형 연결 리스트(circular linked list)



- 이중 연결 리스트(doubly linked list)



파이썬으로 쉽게 배우는
자료구조

8

6.2 단순연결리스트 응용: 연결된 스택

- 삽입 연산
- 삭제 연산
- 모든 노드의 방문

단순연결리스트 응용: 연결된 스택

- 노드 클래스

```
class Node:
    def __init__(self, elem, link=None):
        self.data = elem
        self.link = link
```

단순연결리스트를 위한 노드 클래스
생성자. 디폴트 인수 사용
데이터 멤버 생성 및 초기화
링크 생성 및 초기화

- 연결된 스택 클래스

```
class LinkedStack:
    def __init__(self):
        self.top = None

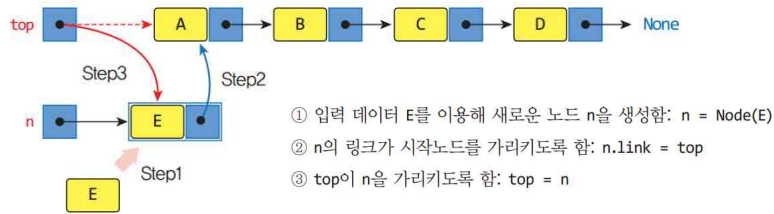
    def isEmpty(self):
        return self.top == None

    def clear(self):
        self.top = None
```

생성자
top 생성 및 초기화
공백상태 검사
스택 초기화

top  None

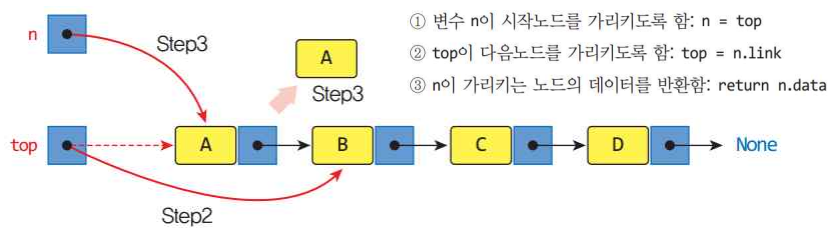
삽입 연산



```
def push( self, item ):
    n = Node(item, self.top)
    self.top = n
```

연결된 스택의 삽입연산
 # Step1 + Step2
 # Step3

삭제 연산

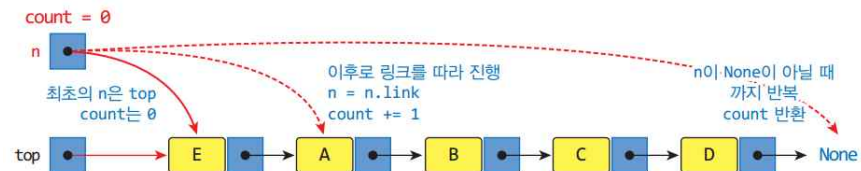


```
def pop( self ):
    if not self.isEmpty():
        n = self.top
        self.top = self.n.link
        return n.data
```

연결된 스택의 삭제연산
 # 공백이 아니면
 # Step1
 # Step2
 # Step3

- 메모리 해제를 신경 쓸 필요 없음!

전체 노드의 방문



```
def size( self ):
    node = self.top
    count = 0
    while not node == None :
        node = node.link
        count += 1
    return count
```

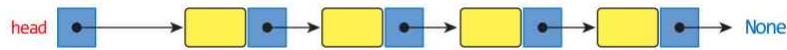
스택의 항목 수 계산
시작 노드
node가 None이 아닐 때 까지
다음 노드로 이동
count 증가
count 반환

6.3 단순연결리스트 응용: 연결 리스트

- 연결 리스트 구조
- 삽입 연산
- 삭제 연산

단순연결리스트 응용: 연결 리스트

- 연결된 리스트 구조



- 노드 클래스: 연결된 스택에서와 동일
- 연결 리스트 클래스

```
class LinkedList:                                # 연결된 리스트 클래스
    def __init__( self ):
        self.head = None

    def isEmpty( self ): return self.head == None    # 공백상태 검사
    def clear( self ) : self.head = None            # 리스트 초기화
    def size( self ) : ...                          # self.top->self.head로 수정. 코드 동일
    def display(self, msg):...                      # self.top->self.head로 수정. 코드 동일
```

파이썬으로 설계한
자료구조

15

연결 리스트 메소드

- pos번째 노드 반환: getNode(pos)

```
def getNode(self, pos) :                        # pos번째 노드 반환
    if pos < 0 : return None
    node = self.head;                          # node는 head부터 시작
    while pos > 0 and node != None :           # pos번 반복
        node = node.link                      # node를 다음 노드로 이동
        pos -= 1                             # 남은 반복 횟수 줄임
    return node                                # 최종 노드 반환
```

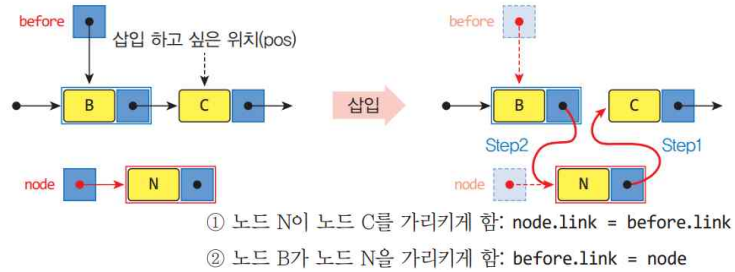
- getEntry(pos), replace(pos,elem), find(val)

```
def getEntry(self, pos) :                      # pos번째 노드의 데이터 반환
    node = self.getNode(pos)                  # pos번째 노드
    if node == None : return None             # 찾는 노드가 없는 경우
    else : return node.data                   # 그 노드의 데이터 필드 반환
```

파이썬으로 설계한
자료구조

16

삽입 연산: insert(pos, elem)



```
def insert(self, pos, elem) :
    before = self.getNode(pos-1)          # before 노드를 찾음
    if before == None :                    # 맨 앞에 삽입하는 경우
        self.head = Node(elem, self.head) # 맨 앞에 삽입함
    else :                                  # 중간에 앞에 삽입하는 경우
        node = Node(elem, before.link)     # 노드 생성 + Step1
        before.link = node                 # Step2
```

파이썬 자료구조

17

삭제 연산: delete(pos)



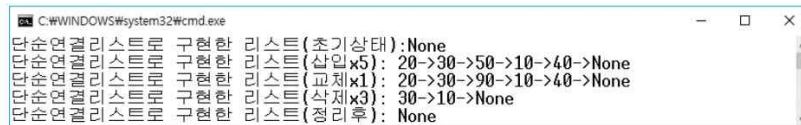
```
def delete(self, pos) :
    before = self.getNode(pos-1)          # before 노드를 찾음
    if before == None :                    # 시작노드를 삭제
        if self.head is not None :         # 공백이 아니면
            self.head = self.head.link    # head를 다음으로 이동
    elif before.link != None :              # 중간에 있는 노드 삭제
        before.link = before.link.link    # Step1
```

파이썬 자료구조

18

테스트 프로그램

```
s = LinkedList()
s.display('단순연결리스트로 구현한 리스트(초기상태):')
s.insert(0, 10);          s.insert(0, 20);          s.insert(1, 30)
s.insert(s.size(), 40);   s.insert(2, 50)
s.display("단순연결리스트로 구현한 리스트(삽입x5): ")
s.replace(2, 90)
s.display("단순연결리스트로 구현한 리스트(교체x1): ")
s.delete(2);   s.delete(s.size() - 1);   s.delete(0)
s.display("단순연결리스트로 구현한 리스트(삭제x3): ")
s.clear()
s.display("단순연결리스트로 구현한 리스트(정리후): ")
```



```
C:\WINDOWS\system32\cmd.exe
단순연결리스트로 구현한 리스트( 초기 상태 ):None
단순연결리스트로 구현한 리스트( 삽입x5 ): 20->30->50->10->40->None
단순연결리스트로 구현한 리스트( 교체x1 ): 20->30->90->10->40->None
단순연결리스트로 구현한 리스트( 삭제x3 ): 30->10->None
단순연결리스트로 구현한 리스트( 정리후 ): None
```

파이썬 자료구조

19

6.4 원형연결리스트의 응용: 연결된 큐

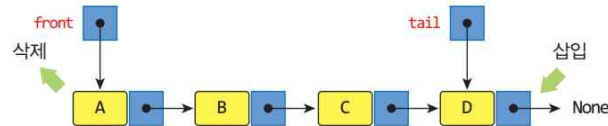
- 구조
- 삽입 연산
- 삭제 연산
- 전체 노드의 방문

파이썬 자료구조

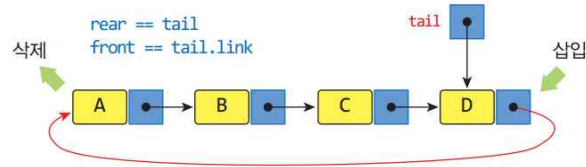
20

원형연결리스트의 응용: 연결된 큐

- 단순연결리스트로 구현한 큐



- 원형연결리스트로 구현한 큐



- tail을 사용하는 것이 rear 와 front에 바로 접근할 수 있다는 점에서 훨씬 효율적

파이썬으로 쉽게 배우는
자료구조

21

연결된 큐 클래스

```
class CircularLinkedList:
    def __init__( self ):
        self.tail = None
        # 생성자 함수
        # tail: 유일한 데이터

    def isEmpty( self ): return self.tail == None
        # 공백상태 검사

    def clear( self ): self.tail = None
        # 큐 초기화

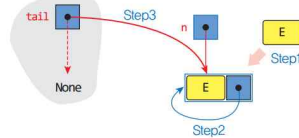
    def peek( self ):
        if not self.isEmpty():
            return self.tail.link.data
        # peek 연산
        # 공백이 아니면
        # front의 data를 반환
```

파이썬으로 쉽게 배우는
자료구조

22

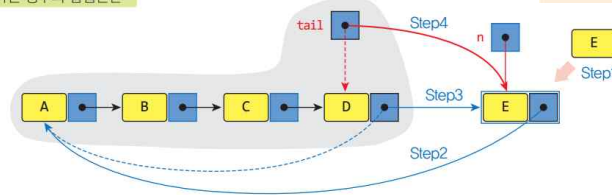
삽입 연산: enqueue()

Case1: 큐가 공백상태인
경우의 삽입연산



```
def enqueue( self, item ):
    node = Node(item, None)
    if self.isEmpty() :
        node.link = node
        self.tail = node
    else :
        node.link = self.tail.link
        self.tail.link = node
        self.tail = node
```

Case2: 큐가 공백상태가
아닌 경우의 삽입연산

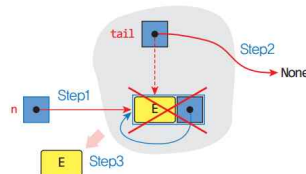


파이썬으로 쉽게 배우는
자료 구조

23

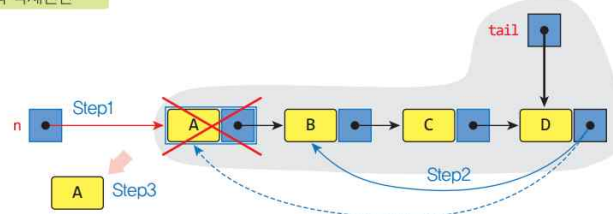
삭제 연산: dequeue()

Case1: 큐가 하나의 항목을
갖는 경우의 삭제연산



```
def dequeue( self ):
    if not self.isEmpty():
        data = self.tail.link.data
        if self.tail.link == self.tail :
            self.tail = None
        else:
            self.tail.link = self.tail.link.link
        return data
```

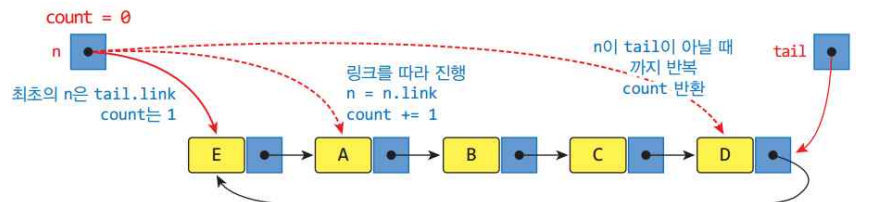
Case2: 큐가 여러 개의 항목을
갖는 경우의 삭제연산



파이썬으로 쉽게 배우는
자료 구조

24

전체 노드의 방문



```
def size( self ):
    if self.isEmpty() : return 0      # 공백: 0반환
    else :                             # 공백이 아니면
        count = 1                     # count는 최소 1
        node = self.tail.link         # node는 front부터 출발
        while not node == self.tail:  # node가 rear가 아닌 동안
            node = node.link           # 이동
            count += 1                 # count 증가
        return count                  # 최종 count 반환
```

파이썬으로 쉽게 배우는
자료구조

25

테스트 프로그램

- 5장 원형 큐 테스트 코드와 동일 (객체 생성만 다름)

```
q = CircularLinkedQueue()    # 연결된 큐 만들기
```

```
C:\WINDOWS\system32\cmd.exe
CircularLinkedQueue:0 1 2 3 4 5 6 7
CircularLinkedQueue:5 6 7
CircularLinkedQueue:5 6 7 8 9 10 11 12 13
```

- 용량 제한이 없고, 삽입/삭제가 모두 $O(1)$

파이썬으로 쉽게 배우는
자료구조

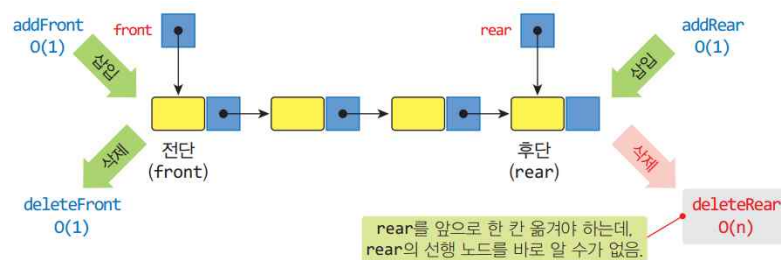
26

6.5 이중연결리스트의 응용: 연결된 덱

- 연결된 덱을 이중연결리스트로 구현하는 이유?
- 이중연결리스트를 위한 노드 클래스
- 연산들의 구현

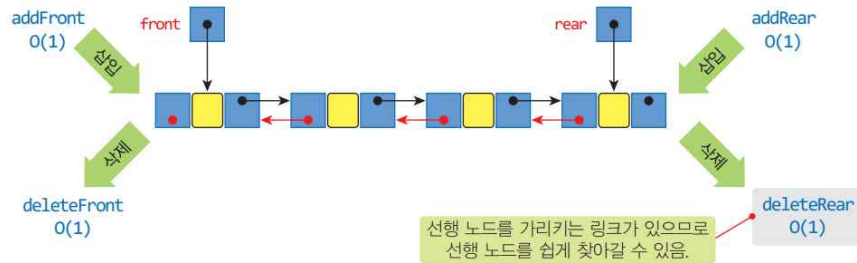
이중연결리스트의 응용: 연결된 덱

- 단순연결리스트로 구현한 덱



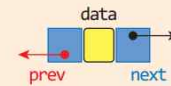
- 해결 방안은?
 - 이중연결리스트 사용

이중연결리스트로 구현한 덱



이중연결리스트를 위한 노드

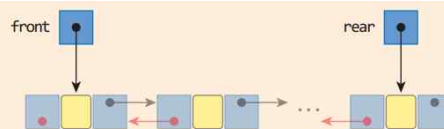
```
class DNode:                                # 이중연결리스트를 위한 노드
    def __init__(self, elem, prev = None, next = None):
        self.data = elem
        self.prev = prev
        self.next = next
```



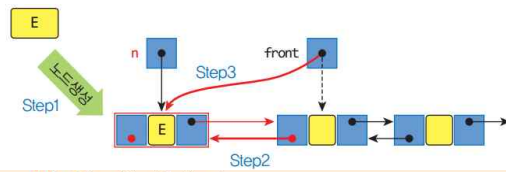
연결된 덱 클래스

```
class DoublyLinkedDeque:
    def __init__(self):
        self.front = None
        self.rear = None

    def isEmpty(self): return self.front == None    # 공백상태 검사
    def clear(self): self.front = self.rear = None    # 초기화
    def size(self): ...    # self.top->self.front로 수정. 코드 동일
    def display(self, msg):...    # self.top->self.front로 수정. 코드 동일
```



addFront(), addRear()



```
def addFront( self, item ):
    node = DNode(item, None, self.front)    # Step1
    if( self.isEmpty() ):                  # 공백이면
        self.front = self.rear = node      # front와 rear가 모두 node
    else :                                # 공백이 아니면
        self.front.prev = node             # Step2
        self.front = node                  # Step3

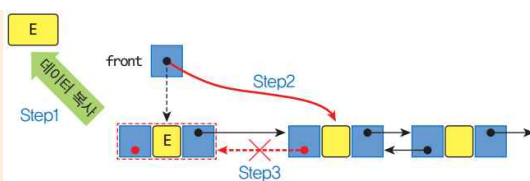
def addRear( self, item ):
    node = DNode(item, self.rear, None)    # Step1
    if( self.isEmpty() ):                  # 공백이면
        self.front = self.rear = node      # front와 rear가 모두 node
    else :                                # 공백이 아니면
        self.rear.next = node              # Step2
        self.rear = node                   # Step3
```

파이썬으로 쉽게 배우는
자료구조

31

deleteFront(), deleteRear()

```
def deleteFront( self ):
    if not self.isEmpty():
        data = self.front.data
        self.front = self.front.next
    if self.front == None :
        self.rear = None
    else:
        self.front.prev = None    # Step3
    return data                   # Step4
```



```
def deleteRear( self ):
    if not self.isEmpty():
        data = self.rear.data      # Step1
        self.rear = self.rear.prev # Step2
    if self.rear == None :         # 노드가 하나 뿐이면
        self.front = None         # front도 None으로 설정
    else:
        self.rear.next = None     # Step3
    return data                    # Step4
```

파이썬으로 쉽게 배우는
자료구조

32

테스트 프로그램



```
dq = DoublyLinkedList() # 연결된 덱 만들기
```

6장 연습문제, 실습문제



