알고리즘과 자료구조 워크북

교과목명 : 알고리즘과 자료구조
차시명: 9차시 양방향 연결리스트

◈ 담당교수: 신일훈 **(서울과학기술대학교)**

◉ 세부목차
- 양방향 연결리스트 개념 및 연산
- 양방향 연결리스트 구현
- 원형 양방향 연결리스트 개념
- 원형 양방향 연결리스트 구현

학습에 앞서

■ 학습개요

양방향 연결리스트는 연결리스트의 한 종류로서 모든 노드들이 이전 노드를 가리키는 prev 변수와 다음 노드를 가리키는 next 변수를 가지고 있다. 본 강에서는 양방향 연결리스트의 개념 및 연산을 이해하고 이를 바탕으로 양방향 연결리스트를 파이썬으로 구현한다. 또한 양방향 연결리스트의 한 종류인 원형 양방향 연결리스트의 개념을 이해하고 이를 파이썬으로 구현한다.

■ 학습목표

1	양방향 연결리스트 개념 및 연산을 이해한다.
2	양방향 연결리스트를 구현할 수 있다.
3	원형 양방향 연결리스트 개념을 이해한다.
4	원형 양방향 연결리스트를 구현할 수 있다.

■ 주요용어

용어	해설
양방향 연결리스트	연결리스트의 한 종류로서, 모든 노드들이 양쪽 방향 링크를 통해 선형으로 연결된다. 따라서 임의의 노드를 찾으면 나머지 노드들도 모두 링크를 통해 접근 가능한 특징이 있다. 일반적으로 가장 앞쪽의 노드의 위치를 기억하며 경우에 따라 마지막 노드의 위치도 기억할 수 있다.
원형 양방향 연결리스트	양방향 연결리스트의 한 종류로서, 모든 노드들이 양쪽 방향 링크를 통해 선형으로 연결된다. 또한 가장 앞의 노드와 가장 마지막 노드도 서로 연결된다. 즉 가장 앞의 노드의 prev 링크는 가장 마지막 노드를 가리키고, 가장 마지막 노드의 next 링크는 처음 노드를 가리킨다. 따라서 가장 앞쪽의 노드의 위치만 기억하더라도 prev 링크를 통해 가장 마지

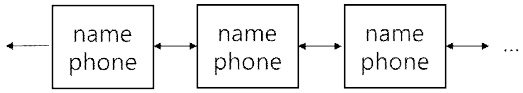
	막 노드도 한번에 접근할 수 있는 장점이 있다.

학습하기

1. 양방향 연결리스트 개념 및 연산

가. 개념

- 데이터를 저장하는 노드(node)와 노드를 연결하는 링크로 구성됨
- 모든 노드들이 양쪽 방향 링크를 통해 선형으로 연결됨



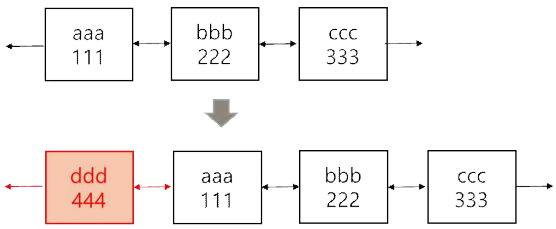
- 임의의 노드를 찾으면 나머지 노드들도 링크를 통해 모두 접근 가능함
- 검색, 삽입, 삭제 연산 등이 가능함

나. 검색 연산

- 검색을 수행하기 위해서는 임의의 노드의 위치를 알아야 함 (일반적으로 가장 앞의 head 노드의 위치를 기억함)
- 이후 타겟 노드를 발견할 때까지, 가장 앞 노드부터 마지막 노드까지 차례로 검사하면 됨.
- 따라서 최악의 경우 모든 노드를 검사해야 하므로 검색 연산의 최악 시간복잡도는 O(N).

다. 전면 삽입 연산

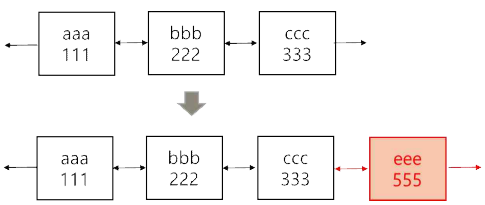
- 가장 앞 노드(head)의 위치를 알고 있다고 전제함.
- 새로운 노드(ddd)를 가장 앞으로 삽입하고 기존의 첫 번째 노드(aaa)는 두 번째 노드가 됨.
- 즉 새로운 노드(ddd)의 next가 기존의 첫 번째 노드(aaa)를 가리키고 기존 첫 번째 노드의 prev가 새로운 노드를 가리키도록 링크 설정함. 새로운 노드의 prev는 가리키는 대상이 없음 (NULL 또는 None).



- 노드의 개수에 관계없이 수행해야 하는 연산의 개수가 일정하므로 전면 삽입 연산의 최악 시간복잡도는 O(1).

라. 후면 삽입 연산

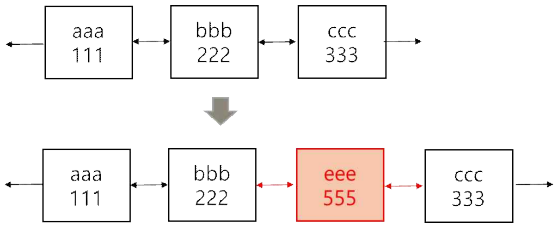
- 가장 앞 노드(head)의 위치를 알고 있다고 전제함.
- 가장 앞 노드부터 링크를 따라가며 가장 마지막 노드(ccc)를 발견해야 함.
- 가장 마지막 노드(ccc)의 next가 새로운 노드(eee)를 가리키고 새로운 노드의 prev가 기존의 가장 마지막 노드를 가리키도록 링크를 설정해야 함. 새로운 노드의 next는 가리키는 대상이 없음 (NULL 또는 None).



- 따라서 가장 마지막 노드를 발견하기 위해 모든 노드들의 링크를 따라가야 하므로 일반적인 후면 삽입 연산의 최악 시간복잡도는 O(N).
- 만약 가장 마지막 노드의 위치를 별도로 관리한다면 모든 노드들의 링크를 따라갈 필요가 없으므로 후면 삽입 연산의 최악 시간복잡도도 전면 삽입 연산과 동일하게 O(1)임.

마. 임의의 위치에 삽입 연산

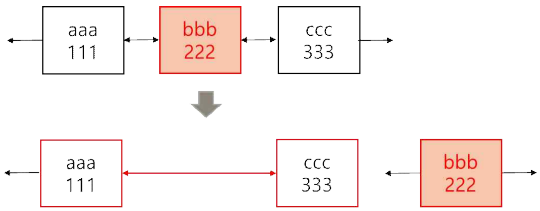
- 가장 앞 노드(head)의 위치를 알고 있다고 전제함.
- 가장 앞 노드부터 링크를 따라가며 삽입을 수행할 노드(bbb)를 발견해야 함.
- 삽입을 수행할 노드(bbb)의 next가 새로운 노드(eee)를 가리키고 새로운 노드의 prev가 삽입을 수행할 노드(bbb)를 가리키도록 링크를 설정해야 함. 또한 새로운 노드의 next는 그 다음 노드(ccc)를 가리킴. 만약 그 다음 노드가 존재하지 않는다면 NULL 또는 None의 값을 갖는다.



- 삽입을 수행할 노드를 발견하기 위해 최악의 경우, 모든 노드들의 링크를 따라가야 하므로 삽입 연산의 최악 시간복잡도는 O(N).

바. 삭제 연산

- 가장 앞 노드(head)의 위치를 알고 있다고 전제함.
- 가장 앞 노드부터 링크를 따라가며 삭제하고자 하는 노드(bbb)를 발견해야 함.
- 삭제할 노드를 발견하면 이전 노드(aaa)의 next가 다음 노드(ccc)를 가리키고 다음 노드의 prev가 이전 노드를 가리도록 링크를 수정해야 함.
- 삭제할 노드는 연결리스트에서 분리되며 이후 필요에 따라 메모리에서 삭제함.



- 삭제할 노드를 발견하기 위해 최악의 경우에는 모든 노드들을 검사해야 하므로 삭제 연산의 최악 시간복잡도는 O(N).

2. 양방향 연결리스트 구현

가. 클래스 DNode 정의

- 리스트를 구성하는 하나의 노드에 해당
- 멤버 변수
  - item: 노드의 데이터를 저장
  - next: 다음 노드를 가리키는 링크
  - prev: 이전 노드를 가리키는 링크
- 메서드
  - 생성자

나. 클래스 DNode 코드

```
class DNode:
        def __init__(self, item, prev=None, next=None):
                self.item = item
                self.prev = prev
                self.next = next
```

- 생성자는 세 개의 인자, item, prev, next를 전달받아 이들 값으로 멤버 변수들을 초기화함.

다. 클래스 DList 정의

- 양방향 리스트를 나타냄
- 멤버 변수
  - head: 첫째 노드를 가리킴
- 메서드
  - 생성자
  - insert_front()
  - delete_front()
  - search()
  - print_list()
  - ...

라. 클래스 DList 생성자 코드

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

- 리스트가 생성될 때 비어 있는 상태이므로 head를 None으로 초기화함.

마. 클래스 DList의 insert_front() 메서드 코드

- 전면 삽입을 수행하는 메서드이다.

```
        def insert_front(self, item):
                dnode = DNode(item, None, self.head)
                if (self.head != None):
                        self.head.prev = dnode
                self.head = dnode
```

- 추가할 item을 인자로 전달받고, DNode를 호출하여 item을 저장할 노드를 생성한다. 이 때 생성된 노드의 next는 현재 self.head가 가리키는 값으로 초기화된다. 즉 기존의 첫 번째 노드를 가리키게 된다.
- 기존에 리스트가 비어 있는 경우에는 self.head가 None이었을 것이므로 생성된 노드의 next는 None으로 초기화된다.
- 또한 생성된 노드는 첫 번째 노드이므로 prev는 None으로 초기화한다.
- 기존의 첫 번째 노드가 존재하면 해당 노드의 prev가 생성된 노드를 가리키도록 한다.
- 또한, 생성된 노드가 새로운 첫 번째 노드이므로 self.head가 이를 가리키도록 수정한다.

.

바. 클래스 DList의 print_list() 메서드 코드

- 모든 노드들의 item 값들을 출력하는 메서드이다.

```
    def print_list(self):
        if self.head == None:
            print('empty')
            return
        node = self.head
        while node:
            if (node.next != None):
                print(node.item, ' <=> ', end= ' ')
            else:
                print(node.item)
            node = node.next
```

- self.head가 만약 None이면 리스트가 비어 있으므로 empty를 출력하고 바로 종료한다.
- self.head가 첫째 노드를 가리키므로 이를 시작 노드로 해서 노드가 더 이상 없을 때까지 노드에 저장된 item을 출력한다.
- 출력 후에는 타겟 노드(node)를 다음 노드(node.next)로 수정하여 작업을 반복한다.
- 현재 노드의 다음 노드가 존재할 때는 (if문), item을 출력하고 나서 화살표도 함께 출력한다.

사. 클래스 DList의 delete_front() 메서드 코드

- 첫째 노드를 삭제하는 메서드이다.

```
    def delete_front(self):
        target = self.head;
        second = None
        if (target != None):
            second = target.next
            self.head = second
            del(target)
        if (second) :
            second.prev = None
```

- self.head가 첫째 노드를 가리키므로 이를 target에 저장한다.
- target이 존재하면 (첫째 노드가 존재하면), 이를 리스트에서 분리하기 위해 self.head를 두 번째 노드로 수정한다.
- del() 함수는 메모리를 해제하는 함수이다.
- second(기존의 두 번째 노드)가 존재하면 second의 prev를 None으로 수정한다.

아. 클래스 DList의 search() 메서드 코드

- 특정 아이템이 존재하는지 탐색하는 메서드이다.

```
    def search(self, target):
        node = self.head
        while node:
            if target == node.item:
                return True
            node = node.next
        return False
```

- self.head가 가리키는 첫째 노드부터 찾고자 하는 아이템을 가지고 있는지를 검사한다. 아이템이 발견되면 탐색을 종료하며 그렇지 않은 경우에는 다음 노드로 계속 진행한다.

- 마지막 노드까지 탐색했음에도 불구하고 찾지 못하면 실패를 반환한다.

자. 테스트 코드 및 실행 결과

- insert_front(), print_list()를 테스트하는 코드이다.

```
d = DList()
d.insert_front('mango')
d.insert_front('orange')
d.insert_front('apple')
d.print_list()
```

- 비어 있는 리스트를 생성한 후, mango, orange, apple을 차례로 삽입한다.
- 리스트를 출력하면 가장 마지막에 추가한 apple이 제일 먼저 나와야 한다.
- 다음은 이에 대한 실행 결과이다.

```
In [7]: runfile('D:/data/lecture/
파이썬으로배우는자료구조와알고리즘/code/알고리즘/
untitled2.py', wdir='D:/data/lecture/
파이썬으로배우는자료구조와알고리즘/code/알고리즘')
apple  <=>  orange  <=>  mango
```

- delete_front()를 테스트하는 코드이다.

```
class DList :
d = DList()
d.insert_front('mango')
d.insert_front('orange')
d.insert_front('apple')
d.print_list()
d.delete_front()
d.print_list()
```

- 비어 있는 리스트를 생성한 후, mango, orange, apple을 차례로 삽입한다.
- 그리고 delete_front()를 호출하면 가장 마지막에 추가된 apple이 삭제된다.
- 따라서 print_list() 함수를 호출하면 orange, mango 순으로 출력되어야 한다.
- 다음은 이에 대한 실행 결과이다.

```
In [8]: runfile('D:/data/lecture/
파이썬으로배우는자료구조와알고리즘/code/알고리즘/
untitled2.py', wdir='D:/data/lecture/
파이썬으로배우는자료구조와알고리즘/code/알고리즘')
apple  <=>  orange  <=>  mango
orange  <=>  mango
```

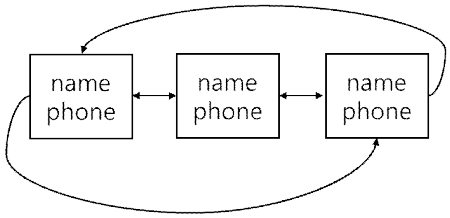
- search()를 테스트하는 코드이다.

```
d = DList()
d.insert_front('mango')
d.insert_front('orange')
d.insert_front('apple')
print(d.search('mango'))
print(d.search('mongo'))
d.delete_front()
print(d.search('mango'))
```

- 비어 있는 리스트를 생성한 후, mango, orange, apple을 차례로 삽입한다.
- mango를 탐색한다. => 성공해야 함
- mongo를 탐색한다. => 실패해야 함
- delete_front()를 호출하여 apple을 삭제한 후, mango를 탐색한다. => 성공해야 함
- 다음은 이에 대한 실행 결과이다.

```
In [10]: runfile('D:/data/lecture/
파이썬으로배우는자료구조와알고리즘/code/알고리즘/
untitled2.py', wdir='D:/data/lecture/
파이썬으로배우는자료구조와알고리즘/code/알고리즘')
True
False
True
```

3. 원형 양방향 연결리스트 개념
- 데이터를 저장하는 노드(node)와 노드를 연결하는 링크로 구성됨
- 모든 노드들이 양쪽 방향 링크를 통해 원형으로 연결됨



- 가장 마지막 노드인 tail 노드의 next 링크는 첫 번째 노드인 head 노드를 가리킴.
- head 노드의 prev 링크는 tail 노드를 가리킴
- 임의의 노드를 찾으면 나머지 노드들도 링크를 통해 모두 접근 가능함
- 검색, 삽입, 삭제 연산 등이 가능함

4. 원형 양방향 연결리스트 구현

가. 클래스 CNode 정의
- 리스트를 구성하는 하나의 노드에 해당
- 멤버 변수
  - item: 노드의 데이터를 저장
  - next: 다음 노드를 가리키는 링크
  - prev: 이전 노드를 가리키는 링크
- 메서드
  - 생성자

나. 클래스 CNode 코드

```
class CNode:
        def __init__(self, item, prev=None, next=None):
                self.item = item
                self.prev = prev
                self.next = next
```

- 생성자는 세 개의 인자, item, prev, next를 전달받아 이들 값으로 멤버 변수들을 초기화함.

다. 클래스 CList 정의
- 양방향 리스트를 나타냄
- 멤버 변수

  - head: 첫째 노드를 가리킴
- 메서드
  - 생성자
  - insert_front()
  - delete_front()
  - search()
  - print_list()
  - ...

라. 클래스 CList 생성자 코드

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

- 리스트가 생성될 때 비어 있는 상태이므로 head를 None으로 초기화함.

마. 클래스 CList의 insert_front() 메서드 코드

- 전면 삽입을 수행하는 메서드이다.

```
        def insert_front(self, item):
                cnode = CNode(item, None, None)
                if (self.head == None):
                        cnode.next = cnode
                        cnode.prev = cnode
                        self.head = cnode
                        return

                first = self.head
                last = first.prev
                cnode.next = first
                cnode.prev = last
                first.prev = cnode
                last.next = cnode
                self.head = cnode
```

- 추가할 item을 인자로 전달받고, CNode를 호출하여 item을 저장할 노드를 생성한다.
- 기존에 리스트가 비어 있는 경우에는 self.head를 생성한 노드로 설정하고 생성한 노드의 next, prev는 모두 자신을 가리키도록 하고 종료한다.
- 리스트가 비어있지 않은 경우에는 생성한 노드의 next가 기존의 첫째 노드(first)를 가리키도록 한다.
- 생성한 노드의 prev는 기존의 마지막 노드(last)를 가리키도록 한다.
- first의 prev는 생성한 노드를 가리키고, last의 next도 생성한 노드를 가리키도록 한다.
- 생성된 노드가 새로운 첫 번째 노드이므로 self.head가 이를 가리키도록 수정한다.

.

바. 클래스 CList의 print_list() 메서드 코드

- 모든 노드들의 item 값들을 출력하는 메서드이다.

```
        def print_list(self):
            if self.head == None:
                print('empty')
                return
            p = self.head
            while p.next != self.head :
                print(p.item, ' <=> ', end= ' ' )
                p = p.next
            print(p.item)
            return
```

- self.head가 만약 None이면 리스트가 비어 있으므로 empty를 출력하고 바로 종료한다.
- self.head가 첫째 노드를 가리키므로 이를 시작 노드로 해서 노드가 더 이상 없을 때까지 노드에 저장된 item을 출력한다.
- 출력 후에는 타겟 노드(p)를 다음 노드(p.next)로 수정하여 작업을 반복한다.
- 현재 노드의 다음 노드가 존재할 때는 (if문), item을 출력하고 나서 화살표도 함께 출력한다.

사. 테스트 코드 및 실행 결과

- insert_front(), print_list()를 테스트하는 코드이다.

```
class CList:
c = CList()
c.insert_front('mango')
c.insert_front('orange')
c.insert_front('apple')
c.print_list()
```

- 비어 있는 리스트를 생성한 후, mango, orange, apple을 차례로 삽입한다.
- 리스트를 출력하면 가장 마지막에 추가한 apple이 제일 먼저 나와야 한다.
- 다음은 이에 대한 실행 결과이다.

```
In [14]: runfile('D:/data/lecture/
파이썬으로배우는자료구조와알고리즘/code/알고리즘/
untitled1.py', wdir='D:/data/lecture/
파이썬으로배우는자료구조와알고리즘/code/알고리즘')
apple  <=>  orange  <=>  mango
```

연습문제

1. 양방향 연결리스트 클래스의 insert_front() 메서드의 알고리즘을 의사코드로 표현하시오.

정답 :

```
def insert_front(self, item):
        dnode = DNode(item)              # create a new node
        dnode.prev = None
        dnode.next = self.head  # The new node points to the previous first node
        if (self.head != None):
                self.head.prev = dnode
        self.head = dnode
        return
```

해설 : 먼저 추가할 item으로 새로운 노드를 만들고, 이 노드의 next 링크가 기존의 첫 번째 노드를 가리키도록 한다. (기존에 비어 있는 리스트였다면 None(NULL)을 가리키면 됨). 새로운 노드가 첫 번째 노드가 되므로 새로운 노드의 prev 링크는 None(NULL)을 가리킨다.
만약 기존에 비어있는 리스트가 아니었다면, 기존에 첫 번째 노드의 prev 링크는 새로 추가된 노드를 가리키도록 한다. 마지막으로 self.head(연결리스트에서 첫 번째 노드를 가리킬 변수)가 새로운 노드를 가리키도록 하면 된다.

2. 양방향 연결리스트가 변수 head를 통해 첫 번째 노드만 직접 접근할 수 있을 때, 연결 리스트의 마지막에 새로운 노드를 추가하는 insert_back() 메서드의 최악의 시간복잡도를 구하시오.

정답 : O(N)

해설 : 처음 노드부터 링크를 순서대로 따라가며 마지막 노드를 찾고, 마지막 노드에 새로운 노드를 추가해야 한다. 따라서 스캔하는 연산의 반복 횟수가 노드의 개수에 비례한다. 따라서 시간복잡도는 O(N)이다.

3. 양방향 연결리스트가 변수 head를 통해 첫 번째 노드만 직접 접근할 수 있을 때, 연결 리스트의 처음에 새로운 노드를 추가하는 insert_front() 메서드의 최악의 시간복잡도를 구하시오.

정답 : O(1)

해설 : 처음 노드를 head를 통해 직접 접근할 수 있으므로 노드의 개수에 상관없이 한번에 처음 노드를 발견할 수 있다. 따라서 시간복잡도는 O(1)이다.

정리하기

1. 양방향 연결리스트는 연결리스트의 한 종류로서, 모든 노드들이 양쪽 방향 링크를 통해 선형으로 연결된다. 따라서 임의의 노드를 찾으면 나머지 노드들도 링크를 통해 접근 가능한 특징이 있다. 일반적으로 가장 앞 노드의 위치를 저장하는 변수를 사용하며, 경우에 따라 가장 마지막 노드를 접근할 수 있는 변수도 함께 사용할 수 있다.
2. 양방향 연결리스트를 구현할 수 있다.
3. 원형 양방향 연결리스트는 양방향 연결리스트의 한 종류로서, 모든 노드들이 양쪽 방향 링크를 통해 선형으로 연결된다. 또한 가장 앞의 노드와 가장 마지막 노드도 서로 연결된다. 즉 가장 앞의 노드의 prev 링크는 가장 마지막 노드를 가리키고, 가장 마지막 노드의 next 링크는 처음 노드를 가리킨다. 따라서 가장 앞쪽의 노드의 위치만 기억하더라도 prev 링크를 통해 가장 마지막 노드도 한번에 접근할 수 있는 장점이 있다.
4. 원형 양방향 연결리스트를 구현할 수 있다.

참고자료

- 파이썬과 함께하는 자료구조의 이해, 양성봉, 2021, 생능출판

다음 차시 예고

- 선형자료구조 중 하나인 큐에 대해 학습한다.