알고리즘과 자료구조 워크북

교과목명 : 알고리즘과 자료구조
차시명: 8차시 자료구조 개요 및 연결리스트

◈ 담당교수: 신일훈 **(서울과학기술대학교)**

◉ 세부목차
- 자료구조의 개념과 종류
- 자료구조와 프로그램 성능의 관계
- 단방향 연결리스트 개념 및 연산
- 단방향 연결리스트 구현

학습에 앞서

■ 학습개요

자료구조는 여러 데이터를 조직화하는 구조이며, 크게 선형 자료형과 비선형 자료형으로 구분된다. 어떤 자료구조를 사용하여 데이터를 조직화하느냐에 따라 알고리즘과 프로그램의 성능이 달라질 수 있다. 본 강에서는 자료구조의 개념 및 종류를 이해하고, 또한 자료구조와 프로그램 성능과의 관계를 이해한다. 여러 자료구조 중 가장 간단한 형태의 자료구조 중 하나인 단방향 연결리스트의 개념과 연산을 이해하고 이를 파이썬으로 구현한다.

■ 학습목표

1	자료구조의 개념과 종류를 이해한다.
2	자료구조와 프로그램 성능의 관계를 이해한다.
3	단방향 연결리스트 개념 및 연산을 이해한다.
4	단방향 연결리스트를 구현할 수 있다.

■ 주요용어

용어	해설
자료구조	자료를 정리하고 조직화하는 구조로서 다양한 형태의 자료구조가 있다. 자료구조의 종류에 따라 데이터를 검색하고 추가하고 삭제하는 연산의 속도가 다를 수 있다.
연결리스트	데이터를 저장하는 노드(node)와 노드를 연결하는 링크로 구성되며, 모든 노드들이 선형으로 연결된다. 일반적으로 임의의 위치에서 노드의 삭제와 삽입이 가능하다.
단방향 연결리스트	연결리스트의 한 종류로서, 모든 노드들이 한쪽 방향 링크를 통해 선형으로 연결된다. 따라서 가장 앞의 노드를 찾으면 나머지 노드들도 링크를 통해 접근 가능한 특징이 있다.

학습하기

1. 자료구조 개념

가. 개념

- 다량의 자료(데이터)를 정리하고 조직화하는 다양한 구조로서 자료를 검색하거나 새로운 자료를 추가하거나 기존 자료를 삭제하는 작업의 용이성 및 성능에 영향을 미친다.
- 크게 선형 자료구조와 비선형 자료구조로 구분된다.
- 자료구조를 사용하여 자료들의 관리가 필요한 예시
  - 도서관 프로그램: 보관하고 있는 도서 데이터들을 저장하고 추가, 삭제, 검색의 기능을 제공해야 함
  - 휴대폰의 연락처 관리 프로그램: 연락처들을 저장하고 추가, 삭제, 검색의 기능을 제공해야 함
  - 윈도우: 현재 실행중인 프로세스 관련 데이터들을 저장하고 사용자 요청시 이를 출력해야 함.

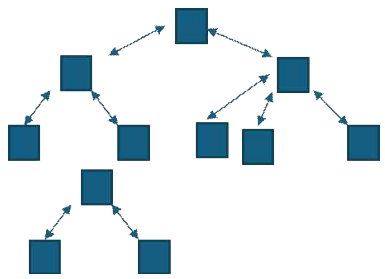
나. 종류

1) 선형 자료구조

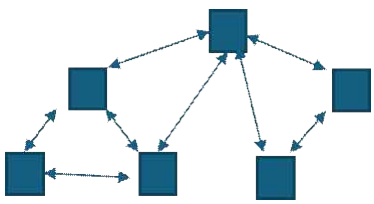
- 각각의 자료(데이터)들을 선형으로 연결하여 관리한다.
- 배열, 연결리스트, 스택, 큐 등이 대표적인 선형 자료구조이다.

2) 비선형 자료구조

- 트리와 그래프가 있다.
- 트리와 그래프는 모두 데이터를 나타내는 정점(노드)과 정점을 연결하는 간선(링크, 에지)로 구성된다.
- 트리는 아래 그림과 같이 컴퓨터의 폴더 구조와 유사한 계층 구조로 데이터들을 조직화한다. 사각형으로 표현된 것이 하나의 데이터를 저장하는 노드를 나타내며 화살표는 노드들을 연결하는 간선(링크)이다.
- 트리의 경우 일반적으로 최상위 노드가 1개 존재하며 모든 노드들이 최상위 노드로부터 연결되어 있다.
- 트리의 경우 노드들이 연결된 관계에서 사이클(순환)의 관계가 존재하지 않는다.



- 그래프는 간선의 방향성이 없는 무방향 그래프와 간선의 방향성이 있는 방향 그래프 등으로 분류되며 지하철 노선도, SNS 관계도, 도로망 등을 표현하기에 적합하다.
- 트리도 그래프의 일종이며, 차이점은 그래프는 사이클의 관계를 허용한다는 점이다.



2. 자료구조와 프로그램 성능 (예시: 연락처 프로그램)

가. 문제 및 가정

- 개별 엔트리는 이름, 핸드폰, 이메일로 구성된다.
- 총 백만 개의 엔트리로 구성된다.
- 백만 개의 엔트리들을 저장하기 위해 어떤 자료구조를 사용할 것인가?

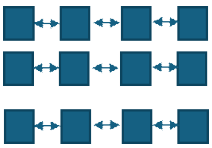
나. 1번 자료구조 후보

- 모든 엔트리들을 순서 없이 선형으로 연결하는 구조

- 특정 연락처를 검색하기 위해서는 첫번째 엔트리부터 마지막 엔트리까지 발견할 때까지 검사해야 함.
- 따라서 최악의 경우에는 백만 개 모두 검사해야 함
- 검색 연산의 최악 시간복잡도: (O(N)).

다. 2번 자료구조 후보

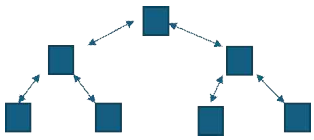
- 모든 엔트리들을 첫 알파벳 별로 다수의 선형 리스트로 연결 (가령 A-Z까지 총 26개의 리스트로 연결)



- 특정 연락처를 검색하기 위해서는 각 알파벳을 연결한 하나의 리스트만 검색하면 됨. 가령 이름이 D로 시작하면 D로 시작하는 리스트만 검색하면 됨.
- 검사해야 하는 연락처의 개수가 평균적으로 1/26으로 감소하므로, 평균적인 검색 시간이 크게 감소함.
- 그러나 최악의 시간복잡도는 여전히 (O(N))이다.

라. 3번 자료구조 후보

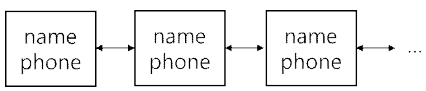
- 모든 엔트리들을 알파벳 순서로 이진트리(binary tree) 구조로 연결



- 특정 연락처를 검색하는 시간이 크게 감소함.
- 균형이진트리가 사용된다면 검색 연산의 최악의 시간복잡도는 O(logN)
- 대신 구현 난이도는 상대적으로 높음

3. 연결리스트 개념

- 데이터를 저장하는 노드(node)와 노드를 연결하는 링크로 구성됨
- 모든 노드들이 선형으로 연결됨
- 일반적으로 임의의 위치에서 삭제와 삽입이 가능함

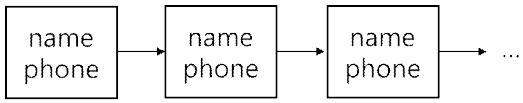


- 단방향 연결리스트와 양방향 연결리스트로 구분됨.
- 단방향 연결리스트는 링크의 방향성이 한쪽 방향으로 되어 있음.
- 양방향 연결리스트는 링크가 양쪽 방향으로 모두 연결되어 있음.

4. 단방향 연결리스트 개념 및 연산

가. 개념

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



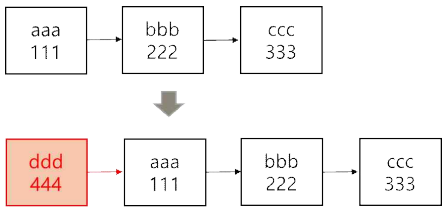
- 가장 앞의 노드를 찾으면 나머지 노드들도 링크를 통해 모두 접근 가능함
- 검색, 삽입, 삭제 연산 등이 가능함

나. 검색 연산

- 검색을 수행하기 위해서는 가장 앞 노드(head)의 위치를 알아야 함
- 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

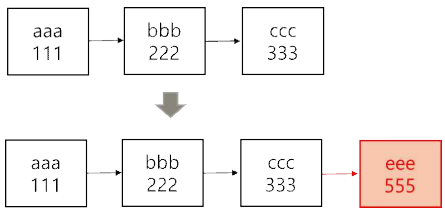
- 가장 앞 노드(head)의 위치를 알고 있다고 전제함.
- 새로운 노드(ddd)를 가장 앞으로 삽입하고 기존의 첫 번째 노드(aaa)는 두 번째 노드가 됨.
- 즉 새로운 노드(ddd)가 기존의 첫 번째 노드(aaa)를 가리키도록 링크 설정함.



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

라. 후면 삽입 연산

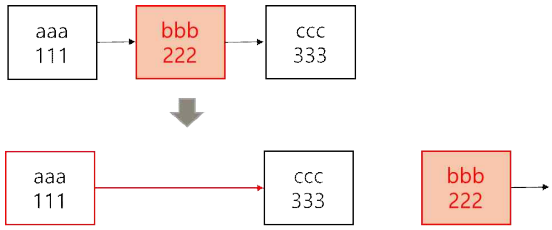
- 가장 앞 노드(head)의 위치를 알고 있다고 전제함.
- 가장 앞 노드부터 링크를 따라가며 가장 마지막 노드(ccc)를 발견해야 함.
- 가장 마지막 노드(ccc)가 새로운 노드(eee)를 가리키도록 링크를 설정해야 함.



- 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. 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)가 다음 노드(ccc)를 가리키도록 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).

5. 단방향 연결리스트 구현

가. 클래스 SNode 정의

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

나. 클래스 SNode 코드

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

- 생성자는 두 개의 인자, item과 next를 전달받아 이들 값으로 멤버 변수들을 초기화함.

다. 클래스 SList 정의

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

라. 클래스 SList 생성자 코드

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

- 리스트가 생성될 때 비어 있는 상태이므로 head를 None으로 초기화함.

마. 클래스 SList의 insert_front() 메서드 코드

- 전면 삽입을 수행하는 메서드이다.

```
        def insert_front(self, item) :
                snode = SNode(item, self.head)
                self.head = snode
                return
```

- 추가할 item을 인자로 전달받고, SNode를 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.
- 생성된 노드가 새로운 첫 번째 노드이므로 self.head가 이를 가리키도록 수정한다.

.

바. 클래스 SList의 print_list() 메서드 코드

- 모든 노드들의 item 값들을 출력하는 메서드이다.

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

- self.head가 첫째 노드를 가리키므로 이를 시작 노드로 해서 노드가 더 이상 없을 때까지 노드에 저장된 item을 출력한다.
- 출력 후에는 현재 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. 클래스 SList의 delete_front() 메서드 코드

- 첫째 노드를 삭제하는 메서드이다.

```
        def delete_front (self) :
                first = self.head
                if first != None:
                        self.head = first.next
                        del(first)
```

- self.head가 첫째 노드를 가리키므로 이를 first에 저장한다.
- first가 존재하면 (첫째 노드가 존재하면), 이를 리스트에서 분리하기 위해 self.head를 두 번째 노드로 수정한다.
- del() 함수는 메모리를 해제하는 함수이다.

아. 클래스 SList의 search() 메서드 코드

- 특정 아이템이 존재하는지 탐색하는 메서드이다.

```
        def search(self, item):
            if (self.head == None):
                return False

            cur = self.head
            while cur != None:
                if item == cur.item:
                    return True
                cur = cur.next
            return False
```

- self.head가 None이면 리스트가 비어 있으므로 탐색은 실패한다.
- 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.

```
s = SList()
s.insert_front("mango")
s.insert_front("tomato")
s.insert_front("orange")
s.insert_front("apple")
s.print_list()
```

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

```
In [4]: runfile('D:/data/lecture/
파이썬으로배우는자료구조와알고리즘/code/알고리즘/
untitled1.py', wdir='D:/data/lecture/
파이썬으로배우는자료구조와알고리즘/code/알고리즘')
apple  -> orange  -> tomato  -> mango
```

- delete_front()를 테스트하는 코드이다.

```
s = SList()
s.insert_front("mango")
s.insert_front("tomato")
s.insert_front("orange")
s.insert_front("apple")
s.print_list()
s.delete_front()
s.print_list()
```

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

```
In [5]: runfile('D:/data/lecture/
파이썬으로배우는자료구조와알고리즘/code/알고리즘/
untitled1.py', wdir='D:/data/lecture/
파이썬으로배우는자료구조와알고리즘/code/알고리즘')
apple  -> orange  -> tomato  -> mango
orange  -> tomato  -> mango
```

- search()를 테스트하는 코드이다.

```
s = SList()
s.insert_front("mango")
s.insert_front("tomato")
s.insert_front("orange")
s.insert_front("apple")
print(s.search("apple"))
s.delete_front()
print(s.search("apple"))
```

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

```
In [6]: runfile('D:/data/lecture/
파이썬으로배우는자료구조와알고리즘/code/알고리즘/
untitled1.py', wdir='D:/data/lecture/
파이썬으로배우는자료구조와알고리즘/code/알고리즘')
True
False
```

연습문제

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

정답 :

```
def insert_front(self, item) :
        snode = SNode(item)       # create a new node
        snode.next = self.head    # The new node points to the previous first node
        self.head = snode
        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).
그리고 self.head(연결리스트에서 첫 번째 노드를 가리킬 변수)가 새로운 노드를 가리키도록 하면 된다.

2. 단방향 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. 단방향 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. 자료구조는 자료를 정리하고 조직화하는 구조로서 다양한 형태의 자료구조가 있다. 자료구조의 종류에 따라 데이터를 검색하고 추가하고 삭제하는 연산의 속도가 다를 수 있다.
2. 자료구조는 선형과 비선형 자료구조로 분류된다.
3. 연결리스트는 데이터를 저장하는 노드(node)와 노드를 연결하는 링크로 구성되며, 모든 노드들이 선형으로 연결된다. 일반적으로 임의의 위치에서 노드의 삭제와 삽입이 가능하다.
4. 단방향 연결리스트는 연결리스트의 한 종류로서, 모든 노드들이 한쪽 방향 링크를 통해 선형으로 연결된다. 따라서 가장 앞의 노드를 찾으면 나머지 노드들도 링크를 통해 접근 가능한 특징이 있다. 경우에 따라 가장 마지막 노드를 접근할 수 있는 변수도 함께 사용할 수 있다. 이렇게 하면 insert_back() 메서드의 시간복잡도가 O(1)이 된다.
5. 단방향 연결리스트를 구현할 수 있다.

참고자료

- 파이썬과 함께하는 자료구조의 이해, 양성봉, 2021, 생능출판

다음 차시 예고

- 양방향 연결리스트에 대해 학습한다.